

SAPIENZA
UNIVERSITÀ DI ROMA

LUISS



The Mirage of Honesty in Cryptography: Secure Multi-Party Computation with Untrusted Devices

PhD in Cybersecurity (XXXVIII cycle)

Lorenzo Magliocco
ID number 1794969

Advisor
Prof. Daniele Venturi

Academic Year 2024/2025

Dissertation defended on the 29th of January, 2026
in front of a Board of Examiners composed by:

Prof. Roberto Caldelli - CNIT / Università Mercatorum

Prof. Pierluigi Perri - Università di Milano Statale

Prof. Valerio Schiavoni - University of Neuchâtel

External Reviewers:

Prof. Oriol Farràs Ventura - Universitat Rovira i Virgili

Prof. Luisa Siniscalchi - Technical University of Denmark

The Mirage of Honesty in Cryptography: Secure Multi-Party Computation with Untrusted Devices

PhD Dissertation. Sapienza University of Rome and LUISS

© 2026 Lorenzo Magliocco. Creative Commons Attribution-NonCommercial-ShareAlike 4.0.



This dissertation has been typeset using LaTeX. This frontpage is based on the Sapthesis class.

Author's email: {lastname} (dot) {firstname} (at) gmail (dot) com

Abstract

Secure Multi-Party Computation (MPC) is a widely acknowledged framework enabling the design of multi-party protocols that preserve the privacy of parties' inputs while ensuring the correct evaluation of the desired functionality. Crucially, these security guarantees should hold even in the presence of external entities who are empowered with some adversarial capabilities, such as controlling the communication channels used throughout the protocol run or forcing a subset of the parties to behave arbitrarily (so-called "malicious" or "byzantine" corruptions). Concretely, a user can instantiate secure MPC protocols on a device to carry out computations involving sensitive information with other untrusted parties.

Despite capturing very general classes of real-world threats, one limitation of "traditional" MPC lies in assuming at least one "honest" party who, throughout the protocol run, behaves exactly as per the theoretical specification of the protocol itself. For several practical settings this may be unrealistic, as the devices used to run the protocol are themselves exposed to a plethora of threats, such as attacks on software or hardware components. Moreover, the security guarantees provided by secure MPC could be voided if a protocol is found to be faulty, be it from cryptographic assumptions falling short or from an incorrect formalization of the protocol itself.

In this composition, we explore more expressive frameworks that enable the design of secure MPC protocols and cryptographic primitives even in the presence of untrusted devices.

We first consider cryptographic reverse firewalls: lightweight devices that sanitize a party's traffic while preserving the correctness of the protocol. These objects were originally introduced by Mironov and Stephens-Davidowitz (EUROCRYPT'15) and later embedded in the framework of subversion-resilient Universal Composability (srUC) due to Chakraborty et al. (EUROCRYPT'22). Under the srUC framework, it is possible to design protocols that provide meaningful security guarantees even if the devices of honest parties have been tampered with in an undetectable manner with the goal of exfiltrating information (so-called "specious subversion attacks"). In particular, we focus on the design of protocols for Password-Authenticated Key Exchange (PAKE): a cryptographic primitive that enables two parties to mutually authenticate by establishing a shared high-entropy key leveraging exclusively some (possibly low-entropy) pre-shared password.

- Our first contribution focuses on sanitizing the PAKE protocol from Oblivious Transfer (OT) due to Canetti et al. (PKC'12). For that, we design and instantiate novel cryptographic primitives with sanitation-friendly properties that may be of independent interest, including sanitizable variants of oblivious transfer, dual-mode cryptosystems, and signature schemes. As an additional contribution, we formalize the unauthenticated setting in the srUC framework by extending the framework of split-authentication due to Barak et al. (CRYPTO'05, JoC'07). This is the first PAKE protocol ever designed in the srUC framework.
- Our second contribution consists of sanitizing the PAKE protocol from trapdoor smooth-projective hashing due to Benhamouda and Pointcheval (CRYPTO'13). The sanitation requires non-trivial modifications to the original protocol, whose security relies on a CCA-secure encryption scheme - an inherently non-malleable primitive. Along the way, we bring advances to the field of malleable smooth-projective hash functions, originally introduced by Chen et al. (ASIACRYPT'16), and coin the notion of malleable trapdoor smooth-projective hashing. Our resulting PAKE protocol has better communication and round complexity compared to the aforementioned PAKE-from-OT.

We then shift our attention to t-out-of-n robust combiners: constructions that take as input n candidate instantiations of some cryptographic primitive to securely realize the same primitive, as long as at least t of the candidates are secure. These objects were first formalized by Harnik et al. (EUROCRYPT'05), where robustness is characterized by explicitly forbidding combiners from re-implementing the desired primitive from scratch. Here, we focus on Non-Interactive Zero-Knowledge (NIZK): a cryptographic primitive that allows a prover to convince a verifier of the veracity of some NP-statement by using a single message (commonly referred to as a "proof").

- Our third contribution provides a comprehensive characterization of robust combiners for NIZK. We show the first formal definition of these objects, and prove that no robust NIZK combiner exists for $t \leq \lfloor n/2 \rfloor$ unless the polynomial hierarchy collapses. To complement our negative results, we provide three incomparable constructions:
  - A black-box combiner for *homomorphic* NP languages, where n, t are polynomial and $t > \lfloor n/2 \rfloor$.
  - A non-black-box combiner for any NP language, where n, t are constant and $t > \lfloor n/2 \rfloor$.
  - A non-black-box combiner for any NP language, where n, t are polynomial and $t > \lfloor 2n/3 \rfloor$.

Contents

1 Introduction **1**
- 1.1 Traditional threat models in Secure Multi-Party Computation . . . 2
- 1.2 Our contributions . . . 3
- 1.3 Cryptographic reverse firewalls . . . 4
  - 1.3.1 Warm-up: sanitizing a leaky random value . . . 4
  - 1.3.2 Password-Authenticated Key Exchange . . . 5
  - 1.3.3 Subversion-resilient UC PAKE from Oblivious Transfer . . . 6
  - 1.3.4 Subversion-resilient UC PAKE from Smooth-Projective Hashing . . . 6
- 1.4 Robust Combiners . . . 7
  - 1.4.1 Warm-up: a robust combiner for Collision-Resistant Hashing . . . 8
  - 1.4.2 Non-Interactive Zero-Knowledge . . . 8
  - 1.4.3 Robust Non-Interactive Zero-Knowledge Combiners . . . 9
- 1.5 Organization . . . 10

I Preliminaries **11**

2 Basic background **13**
- 2.1 Notation . . . 13
- 2.2 Hardness assumptions . . . 14
  - 2.2.1 The Discrete Logarithm assumption . . . 15
  - 2.2.2 The Computational Diffie-Hellman assumption . . . 15
  - 2.2.3 The Decisional Diffie-Hellman assumption . . . 16
- 2.3 Setup assumptions . . . 16
  - 2.3.1 The Common Reference String model . . . 16
  - 2.3.2 The Random Oracle model . . . 16
  - 2.3.3 The Public-Key Infrastructure model . . . 17
- 2.4 Public-key encryption . . . 17
  - 2.4.1 Defining correctness and IND-CPA security . . . 17
  - 2.4.2 Randomized algorithms and efficient adversaries . . . 18
  - 2.4.3 An instantiation: the ElGamal cryptosystem . . . 19
- 2.5 Digital signature schemes . . . 21
  - 2.5.1 Defining correctness and EUF-CMA security . . . 21
  - 2.5.2 An instantiation: the BLS signature scheme . . . 23
- 2.6 Other useful cryptographic primitives . . . 23
  - 2.6.1 One-Way Functions . . . 23
  - 2.6.2 Pseudo-Random Permutations . . . 23
  - 2.6.3 Non-Interactive Zero-Knowledge . . . 24
  - 2.6.4 Commitment schemes . . . 26
  - 2.6.5 Threshold secret sharing . . . 26

3 Secure Multi-Party Computation **28**
- 3.1 The real-ideal paradigm . . . 28
- 3.2 Secure MPC in the presence of eavesdroppers . . . 28
- 3.3 Secure MPC in the presence of malicious parties . . . 31
- 3.4 Universal Composability . . . 33
- 3.5 An example: Universally Composable Oblivious Transfer . . . 35
  - 3.5.1 Dual-Mode Encryption . . . 35
  - 3.5.2 Instantiating $\mathcal{F}_{\mathsf{OT}}$ from Dual-Mode Encryption . . . 37
  - 3.5.3 The multi-session oblivious transfer ideal functionality $\hat{\mathcal{F}}_{\mathsf{OT}}$ . . . 38

II PAKE From Oblivious Transfer **39**

4 Background **41**
4.1 Communication channels as functionalities . . . 41
4.1.1 Authenticated channels . . . 41
4.1.2 Split authentication . . . 43
4.2 Password-Authenticated Key Exchange . . . 46
4.2.1 The Randomized Equality functionality $\mathcal{F}_{\mathsf{RE}}$ . . . 46
4.2.2 A protocol realizing $\mathcal{F}_{\mathsf{RE}}$ in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model . . . 47
4.2.3 The PAKE functionality $\mathcal{F}_{\mathsf{PAKE}}$ . . . 48
4.2.4 From $\mathcal{F}_{\mathsf{RE}}$ to $\mathcal{F}_{\mathsf{PAKE}}$ . . . 49
4.3 Subversion-resilient UC with Cryptographic Reverse Firewalls . . . 50
4.3.1 Specious corruptions . . . 50
4.3.2 Modelling parties . . . 51
4.3.3 Ideal functionalities . . . 51
4.3.4 Communication channels . . . 52
4.3.5 Strong sanitation of firewalls . . . 53
4.3.6 On input-triggering attacks . . . 54

5 Subversion-Resilient PAKE from Oblivious Transfer **56**
5.1 Summary . . . 56
5.1.1 Technical overview . . . 56
5.1.2 Related work . . . 59
5.1.3 Organization . . . 60
5.2 Sanitizing Oblivious Transfer . . . 61
5.2.1 The sanitizable oblivious transfer functionality $\mathcal{F}_{\mathsf{sOT}}$ . . . 61
5.2.2 Sanitizable Homomorphic Dual-Mode Encryption . . . 61
5.2.3 A generic instantiation of $\mathcal{F}_{\mathsf{sOT}}$ . . . 63
5.3 Sanitizing Randomized Equality . . . 64
5.4 Subversion-resilient Split Functionalities . . . 64
5.4.1 Building link initialization . . . 65
5.4.2 Multi-realizing any ideal functionality in the wsrUC model . . . 69
5.4.3 Realizing generic split functionalitites . . . 70
5.4.4 Putting it all together . . . 71
5.5 Sanitizing PAKE . . . 71
5.5.1 From $\mathcal{F}_{\mathsf{RE}}$ to $\mathcal{F}_{\mathsf{PAKE}}$ . . . 71
5.5.2 A hand-wavy performance comparison . . . 72

III PAKE From Smooth-Projective Hashing **73**

6 Background **75**
6.1 Preliminaries . . . 75
6.1.1 An alternative description of $\mathcal{F}_{\mathsf{PAKE}}$ . . . 75
6.1.2 Controlled-malleable NIZKs . . . 76
6.2 Smooth-projective hash functions . . . 77
6.2.1 SPHFs for DH tuples and ElGamal ciphertexts . . . 78
6.2.2 A SPHF for linearly representable languages . . . 79
6.3 Trapdoor SPHFs . . . 80
6.3.1 A TSPHF for linearly representable languages . . . 82
6.3.2 TSPHFs for DH tuples and ElGamal ciphertexts . . . 82
6.4 Universally composable PAKE from TSPHFs . . . 84
6.5 Malleable SPHFs . . . 85
6.5.1 A MSPHF for a restricted class of linearly representable languages . . . 87
6.5.2 A MSPHF for DH tuples . . . 87

7 Subversion-resilient PAKE from Smooth-Projective Hashing **89**
7.1 Summary . . . 89
7.1.1 Technical overview . . . 89
7.2 Malleable trapdoor smooth-projective hash functions . . . 91
7.2.1 A MSPHF for ElGamal ciphertexts . . . 91
7.2.2 Defining malleable trapdoor SPHFs . . . 92
7.2.3 A MTSPHF for ElGamal . . . 93
7.3 Subversion-resilient PAKE from MTSPHFs . . . 94

IV Robust NIZK Combiners **96**

8 Robust Non-Interactive Zero-Knowledge Combiners **98**
8.1 Summary . . . 98
8.1.1 Technical overview . . . 99
8.1.2 Related work . . . 104
8.2 Defining NIZK combiners . . . 105
8.2.1 Formalizing correctness . . . 106
8.2.2 Security breaks for soundness and ZK . . . 106
8.2.3 Combiners for soundness and ZK . . . 107
8.2.4 Combiners for knowledge soundness and ZK . . . 109
8.3 Impossibility Results for $t \leq \lfloor \frac{n}{2} \rfloor$ . . . 109
8.4 Combiner for Homomorphic Languages . . . 112
8.4.1 Preliminaries . . . 113
8.4.2 Combiner description . . . 114
8.4.3 Proof sketch for the (ZK, Sound) case . . . 115
8.4.4 Proof sketch for the (ZK, AoK) case . . . 116
8.5 Combiner from recursive proofs . . . 117
8.6 Combiner from MPC-in-the-head . . . 118
8.6.1 Preliminaries . . . 118
8.6.2 Combiner description . . . 119
8.6.3 From NIZK candidates to non-interactive commitments . . . 120

V Conclusions **122**

9 Conclusions **123**

VI Appendix **131**

A Subversion-resilient PAKE from Oblivious Transfer **133**
A.1 Security proofs . . . 133
A.1.1 Proof of Theorem 5.2 . . . 133
A.1.2 Proof of Theorem 5.3 . . . 135
A.1.3 Proof of Lemma 5.1 . . . 136
A.1.4 Proof of Theorem 5.6 . . . 136
A.1.5 Proof of Lemma 5.2 . . . 138
A.1.6 Proof of Lemma 5.3 . . . 138
A.1.7 Proof of Theorem 5.8 . . . 142
A.2 Discussing input-triggering attacks in the protocols of [CMMV24] . . . 144
A.2.1 Sanitizable Oblivious Transfer . . . 144
A.2.2 Randomized Equality . . . 146
A.2.3 Sanitizable Split-Authenticated communication . . . 148

B Subversion-resilient PAKE from Smooth-Projective Hashing **150**

B.1 Complete proof of the PAKE protocol . . . 150
B.2 A MTSPHF for ElGamal supporting the original syntax of malleability . . . 154
B.2.1 A malleable SPHF for ElGamal ciphertexts . . . 154
B.2.2 A different TSPHF . . . 154
B.2.3 A malleable TSPHF for ElGamal ciphertexts . . . 155

C Robust NIZK Combiners **157**
C.1 A comparison with [HKNY23] . . . 157
C.2 Combiners for knowledge soundness and zero-knowledge . . . 157
C.2.1 Security break for argument-of-knowledge . . . 157
C.2.2 Combiners for argument-of-knowledge . . . 158
C.3 Non-black-box ZK/AoK and related security breaks . . . 159
C.4 Concrete instantiations of homomorphic languages and $\overline{\texttt{reconstruct}}_{\mathbb{H}}$ . . . 160
C.5 Combiner for homomorphic languages: formal treatment . . . 162
C.5.1 C_{HL} is a (ZK, Sound)-combiner . . . 162
C.5.2 C_{HL} is a (ZK, AoK)-combiner . . . 168
C.6 Combiner from recursive proofs: formal treatment . . . 171
C.6.1 $\mathsf{C}_{\mathsf{PoP}}$ is a (ZK, AoK)-combiner . . . 171
C.7 Combiner from MPC-in-the-head: formal treatment . . . 175
C.7.1 $\mathsf{C}_{\mathsf{MPC}}$ is a (ZK, Sound)-combiner . . . 175
C.7.2 $\mathsf{C}_{\mathsf{MPC}}$ is a (ZK, AoK)-combiner . . . 181

1 Introduction

Over the last twenty years, the dependency of modern society on digital technologies has experienced an exponential growth. Owning a smartphone with access to messaging applications and social medias is expected in almost any social context; remote working has seen a significant rise in adoption, especially following the COVID-19 pandemic; entire enterprise infrastructures have been outsourced to the cloud. As a commonality of these (and plenty other) examples, users supply confidential information to some device, which carries out the desired task by engaging in a protocol run with external entities. Who is to say that these entities will not attempt to violate the confidentiality of the users' inputs, or disrupt the functionality of the protocol itself?

Fortunately, there are multiple active research fields aiming at enhancing the security of multi-party protocols in the presence of adversarial behaviour. One such example consists of modelling these protocols as distributed systems, where parties interact with each other by means of some communication channel with the goal of jointly carrying out some computation. Under this abstraction, it is possible to design protocols that are resilient to the arbitrary compromise of a subset of parties (so-called "malicious" or "byzantine" corruptions) or communication channels. With basic cryptographic primitives (such as encryption), we can send messages across the internet without worrying about the presence of eavesdroppers who might sniff our traffic. With more advanced frameworks (such as Secure Multi-Party Computation), we can carry out arbitrary computations with other (potentially malicious) entities while preserving the confidentiality of our inputs and the correctness of the protocol.

Even though these solutions are powerful enough to provide security against very general classes of adversaries, they often assume that at least one party remains "honest" throughout the protocol run. If we do not trust our peers, this implies that our device should work *exactly* as per the theoretical specification of a *secure* protocol, meaning that all its software and hardware components should also behave "honestly". Given the several thousands of newly-discovered vulnerabilities on a yearly basis (*e.g.*, CVEs), 0-days, and backdoors purposefully introduced in software and hardware, it would already appear that these frameworks require assumptions that are too strong to model any concrete application.

To complicate matters further, even if we could rule out all the aforementioned attacks, the protocol we wish to run might itself turn out to be insecure. This may occur, for instance, due to an incorrect formalization of the protocol (caused, *e.g.*, by bugs in security proofs), due to computational assumptions falling short (*e.g.*, following the advances of quantum computers), or due to the protocol being purposefully designed to be insecure under specific conditions. Unfortunately, this last item is not a mere product of paranoia. In the early 2000s, a pseudo-random generator (PRG) was standardized by the NIST (Dual_EC_DRBG) despite multiple researchers pointing out that a specific initialization of this cryptosystem would embed a backdoor capable of nullifying its security guarantees. In 2013, Edward Snowden revealed plans from the NSA to covertly leverage this vulnerability to breach the security of any protocol relying on that PRG. Concretely, such a backdoor could be used to decrypt messages of a secure messaging application unbeknownst to users, who would still observe their protocol running "correctly".

While we can only make educated guesses over whether similar secretive programs are already in place for existing standards, we do not have to look far to find proposals with a similar scope being *publicly* discussed by governments. A particularly relevant recent example is the "EU Chat Control", which aims at weakening end-to-end encryption within the European Union to combat online child sexual abuse material. Aside from the questionable effectiveness in achieving the declared goals, the implications of such a measure on the security and privacy of society would be dire, as stressed in multiple joint statements cosigned by hundreds of researchers (the most recent being [Sta25]). Indeed, backdooring end-to-end encryption would also enable undetectable and indiscriminate mass surveillance, be it from the intended governments or from some adversarial actor who manages to obtain access to said backdoors.

In light of an ever bleaker future, throughout this composition we consider stronger frameworks that provide remediations to the aforementioned threats by distributing the "honesty" of parties across multiple devices. In particular, we will see that it is possible to design protocols that provide meaningful security guarantees even if a subset of the devices used by a party turns out to be insecure.

1.1 Traditional threat models in Secure Multi-Party Computation

Multi-party protocols involve a set of parties, each with their own local input, exchanging messages in order to jointly evaluate some functionality f. To be concrete, let us consider our most beloved fictional characters, Alice and Bob, who want to compute the addition of their local inputs. A two-party protocol Π could mandate them meeting at a bar, exchanging their inputs verbally, and computing the sum by means of a calculator. In this case, Π evaluates the functionality $f(x, y) = x + y$, where x (resp. y) represents the input supplied by Alice (resp. Bob).

Secure Multi-Party Computation (MPC) is a framework that allows the design of multi-party protocols that, in the presence of a threat model and a set of assumptions, retain (at least) two fundamental properties: (i) *correctness* of the evaluation of the functionality, and (ii) *privacy* of the parties' inputs with respect to some adversarial entity, except for what the adversary might infer by observing the output of the protocol[1].

As a warm-up, let us consider the most optimistic threat model possible: one where no adversary exists! In this situation Alice and Bob mutually trust each other, while additionally having access to a communication channel that cannot be eavesdropped or tampered with by any external entity. We model this setting in Figure 1.1, providing each party with access to some device that will be used to run the protocol. Of course, we cannot expect Alice and Bob to go to the bar every time!

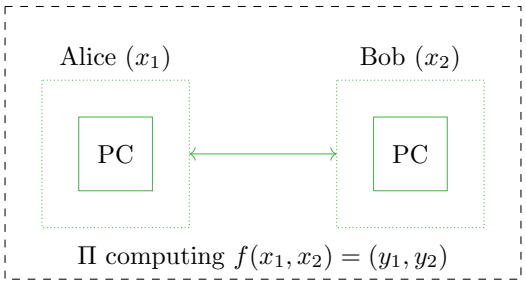


Figure 1.1: Alice and Bob, upon input x_1 and x_2 respectively, evaluate $f(x_1, x_2) = (f_1(x_1, x_2), f_2(x_1, x_2)) = (y_1, y_2)$ over a secure communication channel by running protocol Π.

It is clear that, under this ideal threat model, Alice and Bob can securely and correctly evaluate *any* functionality by simply exchanging their inputs over the communication channel. Unsurprisingly, such a protocol would find limited applicability in the real world, given the very strong trust assumptions required to ensure its security. For that, "traditional" secure MPC protocols typically account for two general threat models: (i) adversaries eavesdropping the communication channel, and (ii) adversaries compromising a subset of the parties running the protocol.

Eavesdroppers. Let us say that Alice and Bob trust each other, and they wish to securely evaluate f over the internet by exchanging their inputs in the clear. How can they be sure that such a communication channel can be fully trusted? Indeed, the traffic they exchange would be exposed to a plethora of entities external to the protocol run itself: some may be connected to the LAN of either party, while others may be in charge of forwarding the traffic between the endpoints. It suffices for any of these entities to have an interest in unveiling the secrecy of their exchange to jeopardize the protocol's confidentiality. Fortunately, standard cryptographic techniques (such as encryption) may be employed to prevent this adversarial vector from causing any harm. The resulting model is represented in Figure 1.2.

Compromised parties. Eavesdroppers are definitely adversaries that should be accounted for while deploying real-world protocols (and that's why HTTP got phased out!). But what if Alice needs to evaluate

[1]This may occur if the adversary is allowed to participate to the protocol run on behalf of a party. Concretely, if the adversary evaluates $f(x, y) = x + y$ acting on behalf of Bob, it may retrieve Alice's input by computing $x = f(x, y) - y$.

f jointly with Bob without necessarily trusting him (Figure 1.3)? Recalling the examples outlined in the introduction, it could be the case that Alice wants to delegate a computationally-expensive task to some potentially compromised cloud provider while ensuring correctness of the computation and privacy of her inputs. This setting is widely studied in the secure MPC literature, in which results span from completeness results (*i.e.*, protocols realizing *any arbitrary* functionality[2], such as [CLOS02]) to much more efficient functionality-specific protocols. Several of these results also hold under *universal composition*, meaning that such protocols would retain their security even when deployed in arbitrary environments (containing, *e.g.*, concurrent protocols running on the same device).

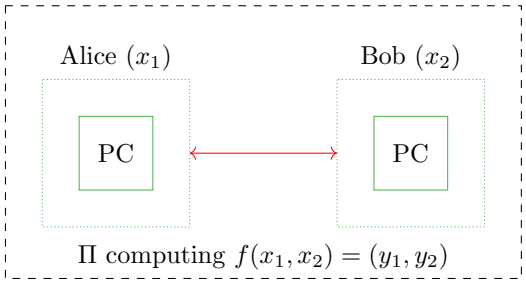


Figure 1.2: Alice and Bob evaluate $f(x_1, x_2)$ over an *authenticated* communication channel (susceptible to eavesdropping) by running protocol Π.

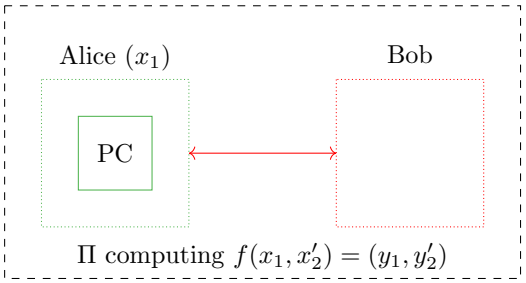


Figure 1.3: Alice securely evaluates $f(x_1, x_2')$ over an *authenticated* communication channel by running protocol Π with (a potentially malicious) Bob.

1.2 Our contributions

As remarked in the introduction, the adversarial strategies handled by the above solutions may still be unsatisfactory whenever Alice's device cannot be fully trusted. In this composition, we focus on two models that allow the design of protocols that provide meaningful security guarantees even in the presence of honest parties that are partially compromised. More specifically:

- We discuss *cryptographic reverse firewalls*: objects that sit on the communication channel of the device they are tasked to protect and sanitize traffic "in the domain of cryptography". Originally introduced in [MS15], cryptographic reverse firewalls enable the design of secure MPC protocols that provide formal guarantees of security even if the implementation of an honest device gets replaced with a *subverted* implementation that attempts to exfiltrate information in an undetectable manner (so-called "specious subversion attacks"). In particular, we focus on the subversion-resilient universally composable (srUC) framework of [CMNV22], by which secure protocols retain the aforementioned security guarantees even under *arbitrary composition* with other protocols.
- We explore the field of *robust combiners*: constructions that combine different candidate instantiations of some cryptographic primitive to securely instantiate the same primitive, as long as at least a subset of the candidates is secure.

Our contributions lie in bringing advances to the above models while focussing on the design of constructions for cryptographic primitives of wide practical interest. More specifically:

- We instantiate Password-Authenticated Key Exchange (PAKE) in the srUC framework. In particular, we present the first PAKE protocol ever designed in the srUC framework, as well as a follow-up construction geared towards efficiency. Along the way, we design new sanitation-friendly building blocks and primitives, as well as modelling extensions, that may be of independent interest for further research.

[2]There are limitations to these completeness results - for instance, *fairness* and *guaranteed output delivery* cannot be achieved for all functionalities, unless the majority of the parties is honest. One such example is coin tossing [Cle86].

- We provide the first comprehensive characterization of robust combiners for Non-Interactive Zero-Knowledge (NIZK), providing definitions, impossibility results, candidate instantiations, as well as novel proof techniques.

In Section 1.3, we provide a succinct introduction to cryptographic reverse firewalls and PAKE, alongside a summary of our two contributions on srUC PAKE. In Section 1.4, we summarize robust combiners and NIZK, providing an overview of our characterization of robust NIZK combiners.

1.3 Cryptographic reverse firewalls

As outlined above, (cryptographic reverse) firewalls assist honest parties in providing resilience against exfiltration of information even in the presence of devices subject to some undetectable corruption of their implementation. These newly-introduced objects and adversarial capabilities can be embedded in "traditional" MPC settings, yielding the model depicted in Figure 1.4.

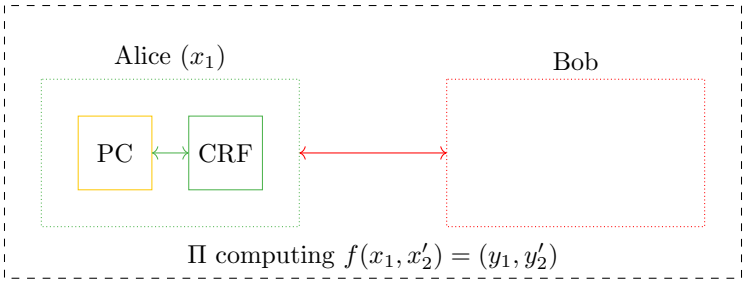


Figure 1.4: Alice securely evaluates $f(x_1, x_2')$ over an *authenticated* communication channel by using a (potentially subverted) device protected by a cryptographic reverse firewall by running protocol Π with (a potentially malicious) Bob.

Of course, firewalls could themselves be compromised, and have to be carefully modelled in order to ensure that attacking a firewall does not jeopardize the security of the (potentially honest) device we wish to protect. For that, these objects must sanitize traffic *without* accessing their party's inputs.

Additionally, firewalls should communicate with their respective device via a *secure* channel. This is necessary, as otherwise the adversary could simply observe the traffic output by a (subverted) device, nullifying any sanitation operated from its firewall. In a real-world deployment, one could delegate the evaluation of the functional component of a protocol and its respective sanitation to different physical devices on the same local network.

1.3.1 Warm-up: sanitizing a leaky random value

Let us consider a simple protocol Π by which Alice, who stores some secret s on her device, samples a random value r and forwards it to a potentially malicious Bob. If we assume Alice to be honest, it is clear that Π would not reveal any information about s.

Let us now consider a tampered implementation $\overline{\Pi}$ that always replaces the first bit of r with the first bit of s. Clearly, this implementation leaks information about s. However, this adversarial strategy can also be trivially detected by Alice. Indeed, by running $\overline{\Pi}$ multiple times, the resulting random strings will always have the same leading bit matching that of the secret (rather than being random). This would prompt Alice to investigate further into her device, foiling the adversary's exfiltration attempt.

Unfortunately, even if we assume Alice to be diligent enough in checking the behaviour of her device, the adversary can come up with a more crafty exfiltration strategy that is undetectable *in a provable manner*, leveraging the same cryptography that is supposed to enhance our security! In particular, let us consider a tampered implementation $\overline{\Pi}$ that has hard-wired some key k and initializes a counter cnt to 0. Instead of sampling r at random, $\overline{\Pi}$ computes r as $P_k(s||cnt)$, where P_k is some pseudo-random permutation (*i.e.*,

a function that is hard to invert and whose output is indistinguishable from random, unless one knows k). After that, it increases the counter (ensuring a different pseudo-random value for each run). In this case, if Alice simply observes the output of her device, she cannot determine that the subversion took place due to the security of the pseudo-random permutation. On the other hand, upon receiving r, Bob simply evaluates $P_k^{-1}(r)$ to retrieve the secret.

In Figure 1.5, we show a protocol run with a cryptographic reverse firewall that destroys the side-channel by XORing the (leaky) r with some fresh randomness $\tilde{r}$. The functionality of the protocol is preserved, as Bob still receives a random value.

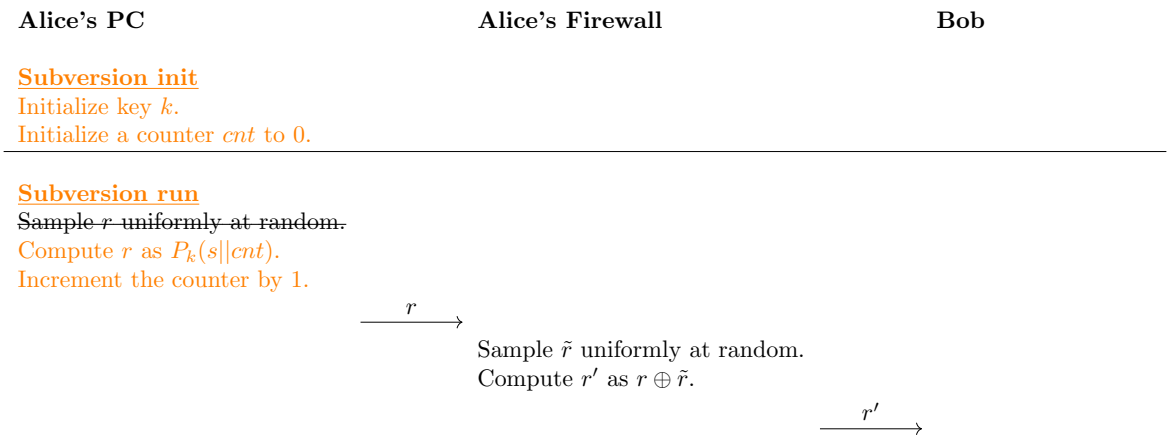


Figure 1.5: A sanitation of a subverted dummy protocol in which Alice, holding a secret s on her PC, sends a random value to Bob.

Is this a realistic attack? While the aforementioned example may appear contrived, it highlights the possibility for a subverted device to exfiltrate information via *any* outbound message that relies on randomness - a class of messages that is ubiquitous in cryptographic protocols. Concretely, in a secure messaging application, a subverted device could exfiltrate information via an encryption of a message!

1.3.2 Password-Authenticated Key Exchange

Authenticated-Key Exchange (AKE) is a cryptographic primitive that allows parties to establish some shared high-entropy key as long as an authentication procedure is successful. This authentication step may be carried out, *e.g.*, by leveraging a Public-Key Infrastructure (PKI) (*i.e.*, a trusted entity that allows parties to bind their identity to some publicly-available information), or by relying on some password shared off-band.

Often times, the key output by a successful AKE run is used to carry out fundamental cryptographic tasks, such as authenticating or encrypting messages flowing between the authenticated parties (*e.g.*, in TLS). As one may expect, this use case finds an enormous reach in real-world deployments, being a foundational building block for cryptographic protocols that guarantee the privacy and security of billions of users on a daily basis. Indeed, an adversarial entity who successfully attacks an AKE protocol could in principle impersonate trusted websites, banks, or messaging peers.

In light of the foundational nature of this primitive, AKE protocols are often designed in traditional MPC settings, ensuring that only honest parties can successfully authenticate with each other. Concretely, if an adversarial entity acts on behalf of one of the communicating peers by following any arbitrary strategy, its authentication attempt will fail altogether.

Within this composition, we focus on PAKE: an AKE where parties mutually authenticate only if they engage in the protocol run using the same password. A core feature of this primitive lies in the possibility of designing PAKE protocols without introducing further trust assumptions. Indeed, a PAKE protocol may

be deployed whenever a PKI is not accessible (or is not deemed to be trustworthy!), and provides formal guarantees of security as long as the password is kept secret among the parties.

Subversion-resilient PAKE. Traditional PAKE protocols typically leverage randomness, and are therefore susceptible to subversion attacks that exfiltrate sensitive information (*e.g.*, an honest party's password) in an undetectable manner. By designing PAKE protocols under the subversion-resilient UC framework, it is possible to provide security even against this stronger threat model, laying down the foundations for a fully-fledged subversion-resilient protocol stack.

1.3.3 Subversion-resilient UC PAKE from Oblivious Transfer

Our first contribution is inspired by the work of [CDVW12], in which the authors design a universally composable PAKE protocol that retains security in the presence of a malicious party in two steps. First, they design a PAKE protocol *on authenticated channels* from oblivious transfer[3]. Then, they remove authenticated channels by invoking a generic transformation for so-called "split functionalities" (as defined in [BCL+05])[4]. This latter step is essential, as it makes little sense to design a PAKE protocol over channels that are already authenticated.

In [CMMV24], we extend the above blueprint to the srUC framework. In spite of its conceptual simplicity, this porting induces the introduction of non-trivial extensions to the srUC framework, as well as the design of multiple sanitizable building blocks and new cryptographic primitives, which we briefly summarize in what follows:

- We introduce and instantiate an ideal functionality capturing "sanitizable" oblivious transfer ($\mathcal{F}_{\mathsf{sOT}}$), which upgrades the traditional oblivious transfer ideal functionality to ensure compatibility with the srUC framework. In particular, $\mathcal{F}_{\mathsf{sOT}}$ allows firewalls to blind the sender's inputs, as these inputs could very well be used to exfiltrate information.
- We extend the srUC framework to the unauthenticated setting by porting the notion of split authentication of [BCL+05], alongside a transformation that allows one to remove authenticated channels from suitable protocols. In the process, we formally define the notion of sanitizable split-authenticated communication by means of an ideal functionality ($s\mathcal{F}_{\mathsf{SAT}}$), and show a matching construction.
- We define sanitizable variants of Dual-Mode Encryption (DME) and EUF-CMA signature schemes. The former is used to instantiate $\mathcal{F}_{\mathsf{sOT}}$ by sanitizing the OT-from-DME protocol due to [PVW08], whereas the latter is used to instantiate $s\mathcal{F}_{\mathsf{SAT}}$ in a similar fashion to [BCL+05].

By combining the aforementioned novelties, we exhibit a highly modular instantiation of srUC PAKE with comparable efficiency to the seminal construction of [CDVW12]. This is the first PAKE protocol ever designed in the srUC framework.

1.3.4 Subversion-resilient UC PAKE from Smooth-Projective Hashing

The PAKE protocol of our first contribution can be regarded as an intermediate step between a feasibility result and an efficient protocol. Despite being more performing compared to general-purpose solutions, its main drawback stems from having a communication complexity scaling linearly with the security parameter (as one has to run sanitizable oblivious transfer for each bit of the password). Moreover, the protocol has a constant but sub-optimal round complexity, primarily due to the requirement of initializing the split-authenticated channel prior to the protocol run.

In this work, we aim at providing a much more concrete srUC PAKE protocol, whose communication complexity is constant in the security parameter while requiring only a single round. To do that, we consider

[3] Oblivious transfer is a two-party functionality in which one party provides two inputs, whereas its peer receives only the input it asks for (*i.e.*, $f_{\mathsf{OT}}((x_1, x_2), \sigma) = (\bot, x_\sigma)$, where σ is a bit).

[4] Roughly, these functionalities capture the best possible security one may obtain in the unauthenticated setting without further setup assumptions: before the protocol run, the adversary either (i) allows two parties to establish an authenticated channel, or (ii) impersonates each party's peer (leading to two parallel protocol runs).

as a starting point the (much more efficient) protocol of [BP13], which relies on trapdoor smooth-projective hash functions (TSPHFs)[5] and encryption schemes secure against chosen-ciphertext attacks (CCA).

In order to sanitize the protocol of [BP13], we first bring forward advances to the field of malleable SPHFs by generalizing the syntax of [CMY+16] and designing constructions for the language of ElGamal ciphertexts. With that, we formalize the notion of "malleable" TSPHFs, and build on top of our novel MSPHFs for ElGamal to obtain MTSPHFs for the same language: a core building block for our srUC PAKE protocol. Then, we tackle the issue of sanitizing the CCA-secure encryption scheme - an inherently non malleable primitive. We achieve this goal by introducing, in its place, a weaker rerandomizable encryption scheme secure against chosen-plaintext attacks (CPA) plus a controlled-malleable NIZK proof system. Intuitively, these two building blocks allow us to preserve the functionality and security of the protocol while introducing the minimal amount of malleability possible to ensure that firewalls can correctly sanitize messages.

1.4 Robust Combiners

From a high-level standpoint, a t-out-of-n combiner is a construction combining n candidate instantiations of some cryptographic primitive $\mathcal{P}$ to itself securely realize $\mathcal{P}$, provided at least t of these candidates securely instantiate $\mathcal{P}$. Following [HKN+05], we additionally say that a combiner is "robust" if its security derives *exclusively* from the security of its candidates. This is an essential restriction that prevents the combiner from creating security on its own, *e.g.*, by discarding the candidate instantiations and re-implementing the desired primitive from scratch.

In the context of secure MPC, Alice can leverage combiners to deploy multiple devices that contribute towards running the desired protocol, as shown in Figure 1.6. In particular, the trust assumption of having an honest party is replaced by assuming that only a subset of the implementations of the protocol is honest, plus the implementation of the combiner. The non-triviality of this setting stems from the robustness of the combiner: even though we assume the combiner to be honest, it cannot implement the desired primitive on its own. Hence, the resulting assumption is weaker compared to a traditional honest party.

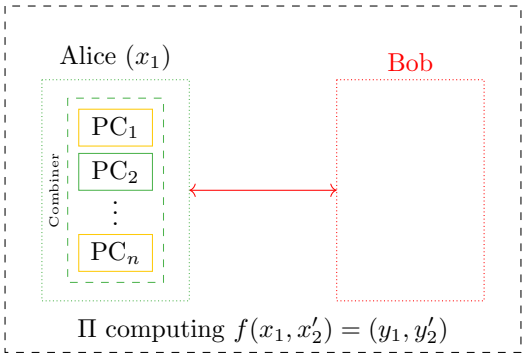


Figure 1.6: Alice securely evaluates a functionality $f(x_1, x_2')$ by running a protocol Π with (a potentially malicious) Bob over an *authenticated* channel leveraging robust combiners, while the adversary compromises a subset of her devices.

Combining protocols finds applicability in a wide range of real-world scenarios. For instance, one could combine protocols relying on classical security assumptions (*e.g.*, Discrete Logarithm-like assumptions) with protocols relying on post-quantum assumptions (*e.g.*, Learning With Errors-like assumptions), tapping into the benefits of both worlds. Indeed, classical assumptions would become insecure as soon as quantum computers become practical, but until then their security would stem from widely-scrutinized problems that

[5] Informally, a smooth-projective hash function allows the evaluation of the hash of a value x (typically a statement for an NP language) either by knowing a *hashing key*, or by knowing a *projection key* and some additional information about x (typically a valid witness for x). A trapdoor SPHF allows the evaluation of the hash by knowing a projection key and a *trapdoor*.

stood the test of time. On the other hand, post-quantum assumptions would provide resilience against quantum adversaries, at the cost of being more susceptible to new attacks.

Another example consists of combining different implementations of the same primitive, each coming from a different (potentially untrustworthy) vendor. Here, deploying combiners would be helpful whenever exhaustively analyzing these implementations in a white-box manner may be too costly.

Finally, should the desired primitive require some setup assumption (*e.g.*, NIZK), one could leverage combiners to distribute the trust placed on (potentially different) setup assumptions. For instance, if the candidates rely on a common reference string, a combiner would retain its security even if a portion of these common reference strings were generated maliciously. Likewise, if the candidates are defined in random oracle model, the combiner would retain its security even if some of the instantiations of the random oracles turn out to be flawed while deploying the respective protocols with real hash functions.

1.4.1 Warm-up: a robust combiner for Collision-Resistant Hashing

Let us consider collision-resistant hash functions: functions for which it is computationally unfeasible to find *a collision*, *i.e.*, two inputs for which the function yields the same output. Following [BB06], a simple 1-out-of-n robust combiner C for n candidate hash functions $H_1, \ldots, H_n$ may be obtained by running each candidate on the input of the combiner and concatenating the resulting hashes. More formally:

$$\mathsf{C}^{H_1,\ldots,H_n}(x) = H_1(x)||H_2(x)||\ldots||H_n(x).$$

This combiner is *robust*, as its security stems *exclusively* from the security of its candidates. In particular, the only way one can find a collision for the combiner is by finding collisions *for each individual candidate*. Hence, as long as at least one of the candidates is collision-resistant, the combiner will be also collision-resistant.

1.4.2 Non-Interactive Zero-Knowledge

In the context of robust combiners, our contributions focus on non-interactive zero-knowledge. This primitive lies in the family of *proof systems* - two-party protocols in which a *prover* wants to convince a *verifier* of the veracity of a (public) statement x with respect to some language $\mathcal{L}$. To do that, in the non-interactive setting, the prover sends a single message π (*i.e.*, a proof) which may be used by the verifier to ascertain the veracity of x. A proof system is said to be *complete* if, for all $x \in \mathcal{L}$, the honest prover always convinces the honest verifier about the veracity of the statement[6].

One widely-studied use case is $\mathcal{L} = \mathsf{NP}$: the verifier cannot determine on its own whether x is in the language, whereas the prover is able to generate a convincing proof π by leveraging a witness w (as long as x in the language). Combined with the security properties we are about to describe, these proof systems can be deployed in a plethora of practical settings in which the witness represents confidential information.

Soundness and zero-knowledge. From a security standpoint, the prover and the verifier might not trust each other. In particular, a verifier may want to ensure that, whenever it accepts a proof, it really means that the statement x is in the language. Conversely, a prover may want to ensure the (potentially sensitive) witness remains hidden while interacting with the verifier. These properties are captured by the notions of *soundness* and *zero-knowledge*, respectively. Informally, the former states that no efficient (malicious) prover can produce an accepting proof for a false statement, whereas the latter states that no efficient (malicious) verifier can retrieve any information about the witness from an honestly-generated accepting proof.

Knowledge soundness. For certain languages, soundness is trivial to achieve, as no false statements exist (*e.g.*, DLOG). Moreover, even though soundness ensures that no false statements can be proved, it could still be possible for a prover to generate accepting proofs for true statements *without* the knowledge of a witness.

[6]Throughout this composition we implicitly refer to NIZK in the common reference string model - *i.e.*, both the prover and the verifier have access to some public information generated by a trusted entity. Indeed, it is impossible to have NIZK for non-trivial languages without any setup assumption [Ore87].

The stronger *knowledge soundness* property prevents this latter scenario from occurring, as it informally states that accepting proofs may only be generated by knowing a valid witness for a statement.

Statistical properties. The informal definitions outlined above only capture efficient adversaries. By lifting this requirement, one obtains the stronger notions of *statistical* soundness, *statistical* zero-knowledge, and *proof*-of-knowledge. Whenever soundness holds against unbounded adversaries, we refer to *proof* systems. Whenever soundness holds only against efficient adversaries, we refer to *argument* systems. For simplicity, throughout informal discussions we typically refer to "proofs" and "arguments" interchangeably.

Robust NIZK Combiners. In mission-critical settings, deploying robust combiners assists in providing fault tolerance against failing cryptographic assumptions, compromised instantiations, as well as broken setup assumptions. This is especially true for NIZK: a cryptographic primitive characterized by a rapidly-evolving implementation landscape. Moreover, instantiations of NIZK are often designed in the random oracle model: an inherently heuristic approach whose security cannot be formally guaranteed by any real-world instantiation.

1.4.3 Robust Non-Interactive Zero-Knowledge Combiners

In [CMVX25], we put forth the first comprehensive characterization of *robust* NIZK combiners, including definitions, impossibility results, and constructions.

In characterizing robustness for NIZK, we follow the rationale of the seminal work of [HKN+05] by requiring that the security of the combiner derives from at least t of the candidates it is combining[7]. Our definitions are general enough to capture both computational and statistical properties of NIZK, as well as the multi-property nature of this primitive: in order for the combiner to be secure, it suffices to have t sound candidates and t zero-knowledge candidates, but these sets may differ. Throughout the composition, we also discuss knowledge soundness in place of soundness.

With our definitions in place, we then show that no t-out-of-n NIZK robust combiner exists if $t \leq \lfloor n/2 \rfloor$, unless the polynomial hierarchy collapses. We prove this is the case by exhibiting a set of candidates for which such a combiner would always output an unconditionally sound and unconditionally zero-knowledge scheme. Due to [Ps05], no NIZK for any NP language can have these properties simultaneously, unless the polynomial hierarchy collapses; hence, no t-out-of-n robust combiner is able to provide a functional NIZK for NP for all possible input candidates, whenever $t \leq \lfloor n/2 \rfloor$.

Finally, we complement the above impossibility result with three incomparable constructions:

- A succinctness-preserving black-box[8] combiner working for a subset of NP-languages denoted as *homomorphic* languages, for all n, t polynomial in the security parameter and $t > \lfloor n/2 \rfloor$. This combiner works by extending the homomorphic properties of the homomorphic language to Shamir's secret sharing scheme. The resulting construction is used alongside the candidates to generate a proof.
- A succinctness-preserving non-black-box combiner working for all of NP, where n, t are constant and $t > \lfloor n/2 \rfloor$. Here, the core idea consists of using the candidates to compute proofs recursively.
- A non-black-box combiner working for all of NP, where n, t are polynomial in the security parameter and $t > \lfloor 2n/3 \rfloor$. Here, the prover runs an MPC protocol locally ("in the head"), and uses the resulting transcript alongside the candidates to generate a proof.

In proving the security of the above constructions, we introduce novel proof techniques that may be of interest for future research.

[7] This notion ensures that the combiner cannot reimplement NIZK from scratch, and is formally captured by asking that, if there exists an adversary successfully attacking the security of the combiner, there must exist reductions successfully attacking the security of at least $n - t + 1$ distinct candidates (*i.e.*, $n - t$ potentially broken candidates plus at least one secure candidate).

[8] Informally, a combiner is succinctness-preserving if it preserves the succinctness property of its candidates (*i.e.*, proofs are "small" compared to the size of statements and witnesses). A combiner is black-box if it accesses its candidates in a black-box fashion.

1.5 Organization

In Part I, we provide an overview of the cryptographic primitives of interest for this composition, as well as an introduction to the secure MPC framework in the authenticated setting. Along the way, we introduce salient technical tools that are used to formally prove the security of protocols and constructions, such as hardness assumptions, security games, complexity reductions, and the real-ideal paradigm. In Part II and Part III, we explore universally-composable secure MPC in the unauthenticated setting in the presence of subversion, designing two PAKE protocols in the subversion-resilient UC framework of [CMNV22] that thwart subversion attacks by leveraging cryptographic reverse firewalls. The protocol in Part II relies on oblivious transfer and split functionalities, whereas the protocol in Part III relies on smooth-projective hashing. In Part IV, we comprehensively discuss robust combiners for NIZK. Finally, in Part V, we conclude the composition by discussing promising directions for further research. The structure of Part II stems from [CMMV24] (published at ASIACRYPT'24). Most of the contents of Part III stems from an unpublished manuscript (currently under reviewing). The majority of Part IV stems from [CMVX25] (presented at ArcticCrypt'25; to appear at EUROCRYPT'26).

Remark. *To ensure better readability, and in order to keep the presentation consistent with the aforementioned research papers, several of the technical details of Part II, III, and IV are deferred to Part VI.*

Co-authors. All the aforementioned works are co-authored with Daniele Venturi (Sapienza University of Rome). The works in Part II and Part III are also co-authored with Suvradip Chakraborty (VISA Research). Finally, the work in Part II was additionally co-authored with Bernardo Magri (University of Manchester; Primev); the work in Part III with Behzad Abdolmaleki, Nahid Roustaeifar (both from the University of Sheffield), Shahram Khazaei, Behzad Vahdani (both from Sharif University of Technology); the work in Part IV with Michele Ciampi, Yu Xia (both from the University of Edinburgh).

Part I

Preliminaries

Part Overview

In the introduction, we broadly discussed about the possibility of designing multi-party protocols that provide meaningful security guarantees even in the presence of an adversarial entity who may compromise a subset of the parties or the communication channels. Before diving deeper in the threat models considered within the contributions of this composition, it is essential to recap the formal tools that allow us to model basic cryptographic primitives as well as "traditional" secure multi-party computation.

Organization. In Section 2, we provide a succinct introduction to the fundamentals of cryptography of interest for this composition. Concretely, we summarize notation, hardness assumptions (*e.g.*, the Discrete Logarithm assumption), and setup assumptions (*e.g.*, Common Reference Strings and Random Oracles). Then, we formally recap basic cryptographic primitives, such as public-key encryption and digital signatures, providing folklore constructions with their related security proofs. Finally, we report further well-known cryptographic primitives pertinent to the remainder of the composition.

In Section 3, we introduce the formal techniques required to establish the security of protocols under the framework of secure multi-party computation. In particular, we focus our attention on canonical threat models considered in the literature, including eavesdroppers and malicious parties, by discussing the real-ideal paradigm - a central modelling artifact used within security proofs. We conclude the section by summarizing the framework of universal composability, exhibiting, as a concrete example, a protocol that instantiates Oblivious Transfer.

2 Basic background

Cryptography is a powerful tool used to achieve fundamental security properties, such as confidentiality and authenticity, in a plethora of real-world deployments. This is especially true in light of the conceptual accessibility of several cryptographic primitives of wide use. Indeed, whenever we leverage an encryption scheme (say, in a secure messaging application), we expect only our intended recipient to be able to decrypt ciphertexts. Likewise, in using a digital signature scheme (say, to sign an official document), we expect that no other entity should be able to produce valid signatures on our behalf.

In spite of the simple intuition, ensuring that an instantiation effectively provides the security guarantees we expect requires designing formal proofs of security, whose rigour is anchored to mathematical objects and formal techniques. This section serves as a succinct introduction to these tools, as well as a summary of the primitives and constructions that will be referred to throughout the composition. For a more comprehensive treatment and further references, we refer the reader to resources such as [KL14, Gol01, Gol04].

In Section 2.1, we summarize relevant notation, including essential mathematical objects. In Sections 2.2 and 2.3, we discuss, respectively, hardness and setup assumptions, highlighting examples of interest for our contributions. In Sections 2.4 and 2.5, we recap public-key encryption schemes and digital signatures, alongside folklore properties, constructions, and related formal proofs of security. In Section 2.6, we provide a summary of other relevant cryptographic primitives.

2.1 Notation

In what follows, we report standard definitions pertaining negligibility of functions, as well as groups and pairings.

Definition 2.1 (Negligible function)**.** *Let $f : \mathbb{N} \to [0,1]$ be a function. We say f is negligible if, for all polynomials p, there exists $\lambda_0 \in \mathbb{N}$ such that:*

$$\forall \lambda \geq \lambda_0, f(\lambda) < \frac{1}{p(\lambda)}.$$

Definition 2.2 (Non-negligible function)**.** *Let $f : \mathbb{N} \to [0,1]$ be a function. We say f is non-negligible if there exist a polynomial p and $\lambda_0 \in \mathbb{N}$ for which the following holds:*

$$\forall \lambda \geq \lambda_0, f(\lambda) \geq \frac{1}{p(\lambda)}.$$

Whenever a function f is negligible (resp. non-negligible), we use the notation $f \in \mathsf{negl}(\lambda)$ (resp. $f \in \mathsf{nonnegl}(\lambda)$). A few examples follow:

- $f(\lambda) = \frac{1}{2^\lambda}$ is a negligible function. Indeed, since f is exponential, it is always possible to find a sufficiently-large $n_0 \in \mathbb{N}$ such that, $\forall \lambda \geq n_0$, $f(\lambda) < \frac{1}{p(\lambda)}$. For instance, if $p(\lambda) = 2\lambda$, $\frac{1}{2^\lambda} < \frac{1}{2\lambda}$ for any $\lambda > 2$.
- $f(\lambda) = \frac{1}{2\lambda}$ is a non-negligible function. Indeed, it is easy to find a $p(\lambda)$ and an $n_0 \in \mathbb{N}$ for which $\frac{1}{\lambda} \geq \frac{1}{p(\lambda)}$, for all $n \geq n_0$. For instance, if $p(\lambda) = \lambda^2$, $\frac{1}{2\lambda} \geq \frac{1}{\lambda^2}$ for any $\lambda \geq 2$.

Finally, the following useful facts hold:

- **Sum of negligible functions.** Let $f, g \in \mathsf{negl}(\lambda)$. Then, $f + g \in \mathsf{negl}(\lambda)$.
- **Product of a negligible function with a polynomial function.** Let $f \in \mathsf{negl}(\lambda)$ and $g \in \mathsf{poly}(\lambda)$. Then, $f \cdot g \in \mathsf{negl}(\lambda)$.
- **Product of a non-negligible function with a constant function.** Let $f \in \mathsf{nonnegl}(\lambda)$ and $c \in O(1)$. Then, $f \cdot c \in \mathsf{nonnegl}(\lambda)$.

Definition 2.3 (Group)**.** *Let $\mathbb{G}$ be a non-empty set, and let $\star : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ (i.e., $\star$ is a binary operator that takes as inputs two elements of $\mathbb{G}$ and outputs an element in $\mathbb{G}$). We say $\mathbb{G}$ is a group if the following properties hold:*

- ***Associativity.*** $\forall g_1, g_2, g_3 \in \mathbb{G}$, $(g_1 \star g_2) \star g_3 = g_1 \star (g_2 \star g_3)$.
- ***Existence of the identity.*** $\exists e \in \mathbb{G}$ *such that,* $\forall g \in \mathbb{G}$, $g \star e = e \star g = g$.
- ***Existence of the inverse.*** $\forall g \in \mathbb{G}$, $\exists! g^{-1} \in \mathbb{G}$ *such that* $g \star g^{-1} = g^{-1} \star g = e$.

Definition 2.4 (Cyclic group of order q)**.** *Let $\mathbb{G}$ be a group, as per Definition 2.3. We say $\mathbb{G}$ is a cyclic group of order q if the following holds: $\exists g \in \mathbb{G}$ such that, for $i \in [q]$, $\mathbb{G} = \{g^0, g^1, \ldots, g^i\}$.*

A frequently-used cyclic group of order $q = p - 1$ is $\mathbb{Z}_p^*$, where p is a prime. For example, for $p = 5$, $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$. 3 is a generator in $\mathbb{Z}_5^*$, as $\mathbb{G} = \{3^0, 3^1, 3^2, 3^3\} = \{1, 3, 4, 2\}$.

Definition 2.5 (Pairing)**.** *Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of prime order q. Let g_1 be a generator for $\mathbb{G}_1$ and g_2 be a generator for $\mathbb{G}_2$. A bilinear pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ satisfies the following properties:*

- ***Bilinearity.*** $\forall g_1 \in \mathbb{G}_1, \forall g_2 \in \mathbb{G}_2$, $\forall a, b \in \mathbb{Z}_\mathsf{q}$, $\hat{e}(g_1^a, g_2^b) = \hat{e}(g_1, g_2)^{ab} \in \mathbb{G}_T$.
- ***Non-degeneracy.*** $\forall g_1 \in \mathbb{G}_1, \forall g_2 \in \mathbb{G}_2$, $\hat{e}(g_1, g_2) \neq 1$.
- ***Efficiency.*** $\forall g_1 \in \mathbb{G}_1, \forall g_2 \in \mathbb{G}_2$, $\hat{e}$ *is efficiently computable.*

Further notation. All randomized algorithms are represented either by having their randomness r implicit (*e.g.*, $\mathsf{Alg}(\ldots)$) or explicit (*e.g.*, $\mathsf{Alg}(\ldots; r)$). If a randomized algorithm is called without explicitly denoting the randomness, we assume such randomness to be randomly sampled from the respective randomness space. We denote with $\mathscr{R}_\mathcal{L}(x, w)$ the NP-relation determining membership of statements with respect to an NP-language $\mathcal{L}$. Further section-specific notation will be introduced along the way. Table 2.1 provides a summary of the symbols used throughout the composition.

Symbol	Description
$a \leftarrow b$	Assign value b to a.
$a \leftarrow\!\!\$\ \mathcal{D}$	Randomly sample a from a distribution $\mathcal{D}$.
$a = b$	Equality between a and b.
$a == b$	Equality check between a and b.
$\mathsf{negl}(\lambda)$	Negligible function in λ.
$\mathsf{nonnegl}(\lambda)$	Non-negligible function in λ.
$\mathsf{poly}(\lambda)$	Polynomial function in λ.
$a[i]$	i-th bit of a.
$\mathbb{Z}_\mathsf{q}$	Set of elements $\{0, \ldots, q-1\}$.
$\mathbb{Z}_\mathsf{p}^*$	Set of elements $\{1, \ldots, p-1\}$, p prime.
$\mathcal{A}$	Adversarial entity.
$\mathcal{S}$	Simulator.
$\mathcal{E}$	Extractor.
$\mathcal{Z}$	Environment.
$\hat{e}$	Pairing.
$\forall$	For all.
$\exists$	Exists.
$\exists!$	Exists and is unique.
$\equiv$	Equivalency of distributions.
$\approx_c$	Computational indistinguishability of distributions.

Table 2.1: Notation.

2.2 Hardness assumptions

Often times, the security of cryptographic protocols derives from problems that are believed to be hard to be efficiently solved. In this subsection, we introduce some of the problems of interest for this composition in the form of *security games*.

2.2.1 The Discrete Logarithm assumption

The discrete logarithm assumption (DLOG) is arguably one of the most known cryptographic assumptions which finds applications in real-world protocols of wide use (*e.g.*, in the RSA cryptosystem). Informally, for certain cyclic groups (*e.g.*, $\mathbb{Z}_{\mathsf{p}}^*$), it is believed that computing the discrete logarithm of a value $y = g^x$ is computationally unfeasible. In other words, it is unfeasible for any *efficient* adversary who only observes y to retrieve the respective discrete logarithm x.

One common approach that is used in the field of cryptography to formally capture hardness assumptions consists of modelling a *security game*, which features a *challenger*, responsible of setting up the experiment capturing the desired hardness assumption, and an *adversary*, who interacts with the challenger with the goal of mounting a successful attack. We often refer to the challenger with the symbol $\mathcal{C}$, and to the adversary with the symbol $\mathcal{A}$.

In what follows, we formalize the discrete logarithm assumption as the security game $\texttt{DLOG}_{\mathcal{A}}$[9], which represents the set of steps performed by the challenger interacting with the adversary $\mathcal{A}$, alongside a matching security definition. We assume the existence of an algorithm $\mathsf{GroupGen}$ that, upon input a security parameter $\lambda \in \mathbb{N}$, returns the description of a cyclic group $\mathbb{G}$ of order $q \geq \log_2(\lambda)$ and a generator $g \in \mathbb{G}$.

Game $\texttt{DLOG}_{\mathcal{A}}$		
1	$(\mathbb{G}, g, q) \leftarrow\!\$\ \mathsf{GroupGen}(1^\lambda)$	*Select a cyclic group consistent with the security parameter.*
2	$x \leftarrow\!\$\ \mathbb{Z}_{\mathsf{q}}$	*Sample a value in* $\mathbb{Z}_{\mathsf{q}}$ *at random.*
3	$y \leftarrow g^x$	*Exponentiate* g *to* x*.*
4	$x' \leftarrow \mathcal{A}(\mathbb{G}, g, q, y)$	*Provide information of the group and* y *to the adversary* $\mathcal{A}$*.* $\mathcal{A}$ *returns a guess* x' *for the discrete logarithm of* y*.*
5	$b \leftarrow\!\$\ \{0, 1\}$	*Sample a random "challenge bit"* b*.*
6	Output $x == x'$	*Output whether the guess is correct.*

Game 1: Description of the DLOG game $\texttt{DLOG}_{\mathcal{A}}$.

Definition 2.6 (Discrete Logarithm assumption (DLOG))**.** *Let* $\mathbb{G}$ *be a cyclic group. We say the DLOG assumption holds in* $\mathbb{G}$ *if, for all PPT adversaries* $\mathcal{A}$*:*

$$\Pr[\texttt{DLOG}_{\mathcal{A}} = 1] \leq \mathsf{negl}(\lambda).$$

2.2.2 The Computational Diffie-Hellman assumption

A stronger variant of the DLOG assumption is the Computational Diffie-Hellman assumption (CDH), in which one is instead asked to complete an incomplete Diffie-Hellman tuple $(\mathbb{G}, g, q, g^x, g^y)$ with the correct g^{xy}, where g is a generator of some cyclic group $\mathbb{G}$ of order q. As a concrete example, we will see that this assumption can be used to design a secure digital signature scheme of interest for this composition (the BLS signature scheme).

Game $\texttt{CDH}_{\mathcal{A}}$		
1	$(\mathbb{G}, g, q) \leftarrow\!\$\ \mathsf{GroupGen}(1^\lambda)$	*Select a cyclic group consistent with the security parameter.*
2	$(x, y) \leftarrow\!\$\ \mathbb{Z}_{\mathsf{q}}$	*Sample two values in* $\mathbb{Z}_{\mathsf{q}}$ *at random.*
3	$(u, v, w) \leftarrow (g^x, g^y, g^{xy})$	*Compute a Diffie-Hellman tuple.*
4	$w' \leftarrow \mathcal{A}(\mathbb{G}, g, q, u, v)$	*Provide an incomplete DH tuple to* $\mathcal{A}$*, who attempts to compute* g^{xy}*.*
5	Output $w == w'$	*Output whether the guess is correct.*

Game 2: Description of the CDH game $\texttt{CDH}_{\mathcal{A}}$.

[9] In the remainder of the composition, if the adversary is clear from the context, we omit the footnote from the label of the respective security game to not overburden the notation.

Definition 2.7 (Computational Diffie-Hellman assumption (CDH))**.** *Let $\mathbb{G}$ be a cyclic group. We say the CDH assumption holds in $\mathbb{G}$ if, for all PPT adversaries $\mathcal{A}$:*

$$\Pr[\mathtt{CDH}_{\mathcal{A}} = 1] \leq \mathsf{negl}(\lambda).$$

2.2.3 The Decisional Diffie-Hellman assumption

An even stronger variant of DLOG is the Decisional Diffie-Hellman assumption (DDH), in which one is instead asked to distinguish $(\mathbb{G}, g, q, g^x, g^y, g^{xy})$ from $(\mathbb{G}, g, q, g^x, g^y, g^r)$ for some $r \neq xy$, where g is a generator of some cyclic group $\mathbb{G}$ of order q. Again, this assumption can be used to instantiate a secure encryption scheme of interest for this composition (the ElGamal cryptosystem), which will be summarized in the next section.

Game $\mathtt{DDH}_{\mathcal{A}}$		
1	$(\mathbb{G}, g, q) \leftarrow\!\!\$\ \mathsf{GroupGen}(1^\lambda)$	*Select a cyclic group consistent with the security parameter.*
2	$(x, y, r) \leftarrow\!\!\$\ \mathbb{Z}_{\mathsf{q}}$	*Sample three values in $\mathbb{Z}_{\mathsf{q}}$ at random.*
3	$b \leftarrow\!\!\$\ \{0, 1\}$	*Sample a random "challenge bit" b.*
4	If $b = 0$, $(u, v, w) \leftarrow (g^x, g^y, g^{xy})$	*Compute a Diffie-Hellman tuple.*
	If $b = 1$, $(u, v, w) \leftarrow (g^x, g^y, g^r)$	*Compute a non-Diffie-Hellman tuple.*
5	$b' \leftarrow \mathcal{A}(\mathbb{G}, g, q, u, v, w)$	*Provide the tuple to $\mathcal{A}$, who guesses whether it is a DH tuple.*
6	Output $b == b'$	*Output whether the guess is correct.*

Game 3: Description of the DDH game $\mathtt{DDH}_{\mathcal{A}}$.

Definition 2.8 (Decisional Diffie-Hellman assumption (DDH))**.** *Let $\mathbb{G}$ be a cyclic group. We say the DDH assumption holds in $\mathbb{G}$ if, for all PPT adversaries $\mathcal{A}$:*

$$\Pr[\mathtt{DDH}_{\mathcal{A}} = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Remark. *The DDH assumption is stronger than the CDH assumption, which is in turn stronger than the DLOG assumption. This means that, if the DDH holds in some group $\mathbb{G}$, both the CDH and DLOG assumptions also hold in $\mathbb{G}$. On the other hand, if the DLOG assumption is found out to be broken in $\mathbb{G}$, it implies that both the CDH and DDH assumptions are also broken in $\mathbb{G}$.*

2.3 Setup assumptions

In addition to hardness assumptions, several functionalities with convenient real-world properties (*e.g.*, enhanced efficiency or security under arbitrary composition) require a trusted setup phase that initializes objects accessible to all parties throughout a protocol run. In this subsection, we present widely-acknowledged setup assumptions that will be referred to throughout the composition.

2.3.1 The Common Reference String model

A common reference string (CRS) is a publicly-accessible string containing setup parameters for the protocol that may either be unstructured (*i.e.*, a random string) or structured. As a concrete example, in the latter case the CRS may contain a public key for an encryption scheme, with the respective secret key being known only to the trusted entity running the setup algorithm. This private information is typically referred to as a "trapdoor", and is often times used within security proofs.

2.3.2 The Random Oracle model

A rather successful heuristic in the field of cryptography, especially in the context of practical deployments, consists of designing protocols by providing parties with access to a "random oracle", which acts as a truly

random function. This enables parties to query the oracle on any input of arbitrary length and obtain a random value in return. Subsequent queries to the same inputs will yield the same output.

It is clear that a random oracle cannot be instantiated in the real world for sufficiently large (and realistic) input and output domains. Indeed, representing such a function requires exponential space in the sizes of the input and output domains, as the evaluation of every input has to be individually associated to a random output.

In order to deploy a protocol under this model in practice, one typically instantiates the random oracle with a hash function (*e.g.*, SHA3), which provides a reasonable approximation of the unpredictability provided by a random oracle. Of course, this implies that the security for protocols lying in this model is only heuristic, given that the real-world protocol can never match its theoretical counterpart. Indeed, there are some (contrived) schemes that are secure in the random oracle model that can never be securely instantiated by *any* hash function [CGH98].

2.3.3 The Public-Key Infrastructure model

Public-key primitives fundamentally rely on *asymmetric* key pairs constituted of *public* and *private* keys. A party would typically generate a key pair and publish the public key to allow other parties to carry out some unprivileged operation (*e.g.*, computing a ciphertext or verifying a digital signature), while locally leveraging the respective secret key to carry out some privileged operation (*e.g.*, decrypting a ciphertext or generating a digital signature). In this context, parties may have an interest in *reliably* sharing public keys with their peers. This is particularly useful, *e.g.*, while designing provably-secure protocols that realize authenticated channels (*i.e.*, channels that cannot be tampered with by a man-in-the-middle). One frequently-adopted solution towards achieving this goal consists of allowing parties to interact with a trusted entity which *binds* public keys to identities of parties. This is often times captured with the notion of *public-key infrastructure* (PKI).

2.4 Public-key encryption

Public-key encryption is typically used to allow two (honest) parties to exchange messages *privately*. In essence, the receiving party generates an *asymmetric* pair of keys: a public key to *encrypt* messages, and a secret key to *decrypt* encrypted messages. The public key is revealed to the sender, who uses it to compute a ciphertext of the secret message. The receiver may then use the respective secret key to retrieve the original message.

2.4.1 Defining correctness and IND-CPA security

In order to make the above informal description actionable, we need to exhibit a formal definition that captures public-key encryption schemes in a satisfactory manner. As a first step, we define the formal structure of this cryptographic primitive in terms of algorithmic behaviour.

Definition 2.9 (Public-key encryption)**.** *Let $\mathcal{K}$, $\mathcal{M}$, $\mathcal{C}$ be key, message, and ciphertext spaces respectively. Let $\mathcal{R}_{\mathsf{KeyGen}}$ and $\mathcal{R}_{\mathsf{Enc}}$ be randomness spaces for algorithms KeyGen and Enc, respectively. Let $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a tuple of algorithms defined as follows:*

- $\mathsf{KeyGen}(1^\lambda; r)$*: Upon input a unary description of the security parameter $\lambda \in \mathbb{N}$ and a randomness $r \in \mathcal{R}_{\mathsf{KeyGen}}$, output a secret-key public-key tuple $(\mathsf{sk}, \mathsf{pk}) \in \mathcal{K}$.*
- $\mathsf{Enc}(\mathsf{pk}, m; r)$*: Upon input a public key pk, a message $m \in \mathcal{M}$, and a randomness $r \in \mathcal{R}_{\mathsf{Enc}}$, output a ciphertext $c \in \mathcal{C}$.*
- $\mathsf{Dec}(\mathsf{sk}, c)$*: Upon input a secret key sk and a ciphertext $c \in \mathcal{C}$, output a message $m \in \mathcal{M}$.*

We say Π is a correct public-key encryption scheme if, for all $\lambda \in \mathbb{N}$ and for all $m \in \mathcal{M}$, the following holds:

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}, c) = m \;\middle|\; \begin{array}{l}(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda) \\ c \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pk}, m)\end{array}\right] = 1.$$

By itself, Definition 2.9 is insufficient to capture what we expect from a secure encryption scheme. Indeed, the following cryptosystem satisfies Definition 2.9 while providing no security whatsoever:

Definition 2.10 (Correct public-key encryption scheme with no security)**.** *A trivial correct public-key encryption scheme that provides no security consists of the following algorithms:*

- $\mathsf{KeyGen}(1^\lambda)$*: Output* $(\mathsf{pk}, \mathsf{sk}) \leftarrow (0, 0)$.
- $\mathsf{Enc}(\mathsf{pk}, m; r)$*: Output* $c \leftarrow m$.
- $\mathsf{Dec}(\mathsf{sk}, c)$*: Output* $m \leftarrow c$.

For that, we need to augment our Definition 2.9 with some security property. Intuitively, in order for a public-key encryption scheme to be secure, we expect that no entity obtaining ciphertexts generated under some public key pk should be able to retrieve any information about the respective plaintext without having access to the matching secret key sk. This requirement can be conveniently captured with the notion of "Indistinguishability against Chosen-Plaintext Attacks" (IND-CPA). Roughly, if a protocol is IND-CPA secure, the adversary cannot tell whether a ciphertext c encrypting either a message m_0 or a message m_1 is an encryption of m_0 or m_1 (with respect to some public key pk). This holds even if the adversary is allowed to choose m_0 and m_1 arbitrarily and has access to pk. This is formalized with the $\texttt{IND-CPA}_{\mathcal{A}}$ game, alongside its related security definition.

Game $\texttt{IND-CPA}_{\mathcal{A}}$		
1	$(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda)$	*Sample a keypair at random.*
2	$(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{pk})$	*Give* pk *to the adversary* $\mathcal{A}$*, who picks two messages.*
3	$b \leftarrow\!\!\$\ \{0, 1\}$	*Sample a "challenge bit"* b *at random.*
4	Compute $c^* \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pk}, m_b)$	*Compute the "challenge ciphertext"* c^* *by encrypting* m_b*.*
5	$b' \leftarrow \mathcal{A}(c^*)$	*Give* c^* *to* $\mathcal{A}$*, who guesses which message was encrypted.*
6	Output $b == b'$	*Output whether the guess is correct.*

Game 4: Description of the IND-CPA game $\texttt{IND-CPA}_{\mathcal{A}}$.

Definition 2.11 (Indistinguishability against Chosen-Plaintext Attacks (IND-CPA))**.** *Let* Π *be a correct public-key encryption scheme, as per Definition 2.9. We say* Π *enjoys indistinguishability against chosen-plaintext attacks if, for all PPT adversaries* $\mathcal{A}$*, the following holds:*

$$\Pr[\texttt{IND-CPA}_{\mathcal{A}} = 1] \leq 1/2 + \mathsf{negl}(\lambda).$$

Discussion. In essence, the challenger sets up the public-key encryption scheme by providing the adversary with the strongest capabilities we wish to capture with our notion of security. In this case, the adversary learns pk and may supply the challenger with any two messages of its choice. What Definition 2.11 tells us is that, as long as the challenger follows the specification of Game 4, it will be impossible to design any (efficient) adversarial strategy that retrieves any information about the plaintext hidden under a ciphertext (except with negligible probability). This security property may seem unnecessarily strong at first: if we consider an eavesdropper observing some ciphertext onto a communication channel, it may not even know which message was encrypted among the entire message space! Nonetheless, several efficient real-world encryption schemes satisfy this property, and we will see such an example in Section 2.4.3.

2.4.2 Randomized algorithms and efficient adversaries

In Definition 2.9, keypairs and ciphertexts are computed by means of *randomized* algorithms KeyGen and Enc, respectively. Moreover, Definition 2.11 restricts adversaries to be computationally bounded. We briefly explore why these two assumptions are essential in the context of IND-CPA security.

On the usage of randomness. In what follows, we formally prove that, whenever either KeyGen or Enc is *deterministic*, the resulting public-key encryption scheme cannot be IND-CPA secure.

Theorem 2.1. *Let* Π *be a correct public-key encryption scheme, as per Definition 2.9. If* KeyGen *or* Enc *are deterministic algorithms,* Π *is not IND-CPA secure.*

Proof. Let us consider a deterministic KeyGen algorithm (*i.e.*, upon input the same security parameter 1^λ, KeyGen always outputs the same keypair). In this case, we design in Figure 2.1 an adversary who wins the IND-CPA game with probability 1.

Adversary $\mathcal{A}$ attacking IND-CPA of Π (deterministic KeyGen)		
1	$\mathsf{pk} \leftarrow \mathcal{C}^{\texttt{IND-CPA}}$	*Obtain the public key of* Π *from the challenger.*
2	$(m_0, m_1) \leftarrow \mathcal{M}$	*Pick two random messages.*
3	$c^* \leftarrow \mathcal{C}^{\texttt{IND-CPA}}(m_0, m_1)$	*Obtain a ciphertext of either message from the challenger.*
4	$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$	*Recompute the deterministic keypair.*
5	If $\mathsf{Dec}(\mathsf{sk}, c^*) = m_0$ output 0.	*Decrypt the challenge ciphertext with* sk.
	If $\mathsf{Dec}(\mathsf{sk}, c^*) = m_1$ output 1.	

Figure 2.1: Attack against IND-CPA of any correct public-key encryption scheme with deterministic KeyGen.

Likewise, a deterministic Enc algorithm would always output the same ciphertext upon input the same tuple (pk, m). Again, we can design an adversary who wins the IND-CPA game with probability 1 (Figure 2.2).

Adversary $\mathcal{A}$ attacking IND-CPA of Π (deterministic Enc)		
1	$\mathsf{pk} \leftarrow \mathcal{C}^{\texttt{IND-CPA}}$	*Obtain the public key of* Π *from the challenger.*
2	$(m_0, m_1) \leftarrow \mathcal{M}$	*Pick two random messages.*
3	$c^* \leftarrow \mathcal{C}^{\texttt{IND-CPA}}(m_0, m_1)$	*Obtain a ciphertext of either message from the challenger.*
4	$c_0 \leftarrow \mathsf{Enc}(\mathsf{pk}, m_0)$	*Recompute the deterministic encryption for both messages.*
5	$c_1 \leftarrow \mathsf{Enc}(\mathsf{pk}, m_1)$	
6	If $c^* = c_0$ output 0.	*Compare the challenge ciphertext against the locally-generated ones.*
	If $c^* = c_1$ output 1.	

Figure 2.2: Attack against IND-CPA of any correct public-key encryption scheme with deterministic Enc.

□

Theorem 2.1 suggests that randomness is a key component in IND-CPA secure encryption schemes. But how much randomness does a public-key cryptosystem need in order to achieve IND-CPA security? By following the same approach of the proof of Theorem 2.1, one can show that $|\mathcal{R}_{\mathsf{KeyGen}}|, |\mathcal{R}_{\mathsf{Enc}}| \in \omega(\lambda)$ (*i.e.*, the size of the randomness spaces for KeyGen and Enc must be super-polynomial in the security parameter). Indeed, if $|\mathcal{R}_{\mathsf{KeyGen}}|, |\mathcal{R}_{\mathsf{Enc}}| \in \mathsf{poly}(\lambda)$, any efficient adversary could exhaustively enumerate all the possible keys (resp. ciphertexts) and mount the same attacks presented in Theorem 2.1.

Efficient adversaries. Finally, the definition of IND-CPA security considers only *efficient* adversaries. This is also a necessary restriction, as any unbounded adversary would be trivially able to mount the same attacks described in Theorem 2.1 even by assuming randomness spaces of size super-polynomial in the security parameter. We describe such an adversary in Figure 2.3.

2.4.3 An instantiation: the ElGamal cryptosystem

Next, we summarize the ElGamal cryptosystem [ElG85], showing that it satisfies IND-CPA security under the DDH assumption. Along the way, we provide an introduction to security proofs and complexity reductions - essential tools to ensure that a protocol formally *realizes* a security property.

Adversary $\mathcal{A}$ attacking IND-CPA of Π		
1	$\mathsf{pk} \leftarrow \mathcal{C}^{\texttt{IND-CPA}}$	*Obtain the public key of* Π *from the challenger.*
2	$(m_0, m_1) \leftarrow \mathcal{M}$	*Pick two random messages.*
3	$c^* \leftarrow \mathcal{C}^{\texttt{IND-CPA}}(m_0, m_1)$	*Obtain a ciphertext of either message from the challenger.*
4	$\forall r$, check $\mathsf{Enc}(\mathsf{pk}, m_0; r) == c^*$.	*Compute all possible ciphertexts of* m_0.
5	If any check above succeeds, output 0. If all checks above fail, output 1.	

Figure 2.3: A computationally unbounded attack against IND-CPA of any correct public-key encryption scheme.

Construction 2.1 (ElGamal PKE [ElG85])**.** *The ElGamal cryptosystem consists of the following algorithms:*

- $\mathsf{KeyGen}(1^\lambda)$*: Compute* $(\mathbb{G}, g, q) \leftarrow\$ \mathsf{GroupGen}(1^\lambda)$*. Compute* $x \leftarrow\$ \mathbb{Z}_\mathsf{q}$*, and evaluate* $h \leftarrow g^x$*. Output a tuple* $(\mathsf{sk}, \mathsf{pk})$*, where* $\mathsf{sk} \leftarrow (\mathbb{G}, g, q, x)$ *and* $\mathsf{pk} \leftarrow (\mathbb{G}, g, q, h)$*.*
- $\mathsf{Enc}(\mathsf{pk}, m)$*: Parse* vk *as* $(\mathbb{G}, g, q, h)$*. Output a ciphertext* $c \leftarrow (g^r, m \cdot h^r) = (c_1, c_2)$*, where* $r \leftarrow\$ \mathbb{Z}_\mathsf{q}$*.*
- $\mathsf{Dec}(\mathsf{sk}, c)$*: Parse* sk *as* $(\mathbb{G}, g, q, x)$*. Output* $m = c_2 / c_1^x$*.*

Theorem 2.2. *Construction 2.1 is an IND-CPA secure encryption scheme, assuming the DDH assumption holds.*

While the security proof of this scheme is folklore, it is instructive to discuss its details to introduce complexity reductions in the context of cryptography. More specifically, in order to prove Theorem 2.2, one can exhibit a proof *by contradiction*. In particular, if the theorem statement is false (*i.e.*, the cryptosystem in Construction 2.1 is not IND-CPA secure), it would be possible to violate an assumption that is believed to hold (*i.e.*, the DDH assumption in $\mathbb{G}$). This is also referred to as a *complexity reduction*: by leveraging the hardness of the DDH assumption in an appropriate group (which is believed to be hard by assumption), it is possible to prove that the ElGamal cryptosystem is IND-CPA secure (something we cannot otherwise tell a priori). We exhibit a verbose security proof in what follows.

Proof (Theorem 2.2, verbose). Assume, by contradiction, that the cryptosystem in Construction 2.1 is not IND-CPA secure in a group $\mathbb{G}$ for which the DDH assumption holds. This implies the existence of a PPT adversary $\mathcal{A}^{\texttt{IND-CPA}}$ that wins the IND-CPA game with non-negligible advantage with respect to a random guess, *i.e.*, $\Pr[\texttt{IND-CPA}_{\mathcal{A}^{\texttt{IND-CPA}}}] \geq \frac{1}{2} + \mathsf{nonnegl}(\lambda)$. Given the existence of $\mathcal{A}^{\texttt{IND-CPA}}$, we show how to design a PPT adversary (reduction) $\mathsf{Red}^{\mathsf{DDH}}$ that, by using $\mathcal{A}^{\texttt{IND-CPA}}$, wins the DDH game with non-negligible advantage with respect to a random guess.

Intuitively, $\mathsf{Red}^{\mathsf{DDH}}$ sets up $\texttt{IND-CPA}$ against adversary $\mathcal{A}^{\texttt{IND-CPA}}$ by relying on the information received from the challenger of $\texttt{DDH}$ (denoted as $\mathcal{C}^{\mathsf{DDH}}$). More specifically, as per Game 3, $\mathsf{Red}^{\mathsf{DDH}}$ receives a tuple $(\mathbb{G}, g, q, u, v, w)$ from $\mathcal{C}^{\mathsf{DDH}}$, where (u, v, w) is either a DH-tuple or a non-DH tuple.

Reduction against DDH $\mathsf{Red}^{\mathsf{DDH}}$		
1	$(\mathbb{G}, g, q, u, v, w) \leftarrow \mathcal{C}^{\mathsf{DDH}}$	*Obtain a tuple from the challenger of the* $\texttt{DDH}$ *game.*
2	$\mathsf{pk} \leftarrow (\mathbb{G}, g, q, u)$	*Derive a public key from the challenge tuple.*
3	$(m_0, m_1) \leftarrow \mathcal{A}^{\texttt{IND-CPA}}(\mathsf{pk})$	*Initialize* $\mathcal{A}^{\texttt{IND-CPA}}$ *with* pk*.*
4	$b \leftarrow\$ \{0, 1\}$	*Randomly sample a "challenge bit"* b*.*
5	Compute $c^* \leftarrow (v, w \cdot m_b)$	*Compute the "challenge ciphertext"* c^* *by encrypting* m_b*.*
6	$b' \leftarrow \mathcal{A}^{\texttt{IND-CPA}}(c^*)$	*Give* c^* *to* $\mathcal{A}^{\texttt{IND-CPA}}$*, who guesses which message was encrypted.*
7	If $b = b'$, send 0 to $\mathcal{C}^{\mathsf{DDH}}$	*If* $\mathcal{A}^{\texttt{IND-CPA}}$ *wins the* $\texttt{IND-CPA}$ *game, guess "DH tuple".*
	If $b \neq b'$, send 1 to $\mathcal{C}^{\mathsf{DDH}}$	*If* $\mathcal{A}^{\texttt{IND-CPA}}$ *loses the* $\texttt{IND-CPA}$ *game, guess "non-DH tuple".*

Figure 2.4: Description of the reduction $\mathsf{Red}^{\mathsf{DDH}}$, which acts simultaneously as the adversary for the game $\texttt{DDH}$ and as the challenger for the game $\texttt{IND-CPA}$.

Analysis. In order to reach a contradiction, we need to argue that $\mathsf{Red}^{\mathsf{DDH}}$ wins $\mathtt{DDH}$ with non-negligible advantage with respect to a random guess. Recall that, by definition, $\mathcal{C}^{\mathsf{DDH}}$ generates (u, v, w) either as a DH tuple or a non-DH tuple. Let us consider the two cases separately:

- $\mathcal{C}^{\mathsf{DDH}}$ generates (u, v, w) as a DH tuple (*i.e.*, $(u, v, w) \leftarrow (g^x, g^y, g^{xy})$). In this case, c^* is computed as $(v, w \cdot m_b)$, *i.e.*, $c^* \leftarrow (g^y, g^{xy} \cdot m_b) = (g^y, h^y \cdot m_b)$, where $h = u$. This implies that $\mathsf{Red}^{\mathsf{DDH}}$ emulates the behaviour of a challenger $\mathtt{IND\text{-}CPA}$ against $\mathcal{A}^{\mathtt{IND\text{-}CPA}}$ (in particular, note that u and (v, w) are sampled by $\mathcal{C}^{\mathsf{DDH}}$ exactly as KeyGen and Enc would in the ElGamal PKE). Since $\mathcal{A}^{\mathtt{IND\text{-}CPA}}$ wins $\mathtt{IND\text{-}CPA}$ with non-negligible advantage, $\mathsf{Red}^{\mathsf{DDH}}$ retains the same advantage against $\mathcal{C}^{\mathsf{DDH}}$, *i.e.*:

$$\Pr[\mathtt{DDH}_{\mathsf{Red}^{\mathsf{DDH}}} | \text{DH Tuple}] = \Pr[\mathtt{IND\text{-}CPA}_{\mathcal{A}^{\mathtt{IND\text{-}CPA}}}] \geq \frac{1}{2} + \mathsf{nonnegl}(\lambda).$$

- $\mathcal{C}^{\mathsf{DDH}}$ generates (u, v, w) as a non-DH tuple (*i.e.*, $(u, v, w) \leftarrow (g^x, g^y, g^r)$). In this case, c^* is computed as $(v, w \cdot m_b)$, *i.e.*, $c^* \leftarrow (g^y, g^r \cdot m_b)$. Since this is not a well-formed ElGamal encryption with respect to public key $(\mathbb{G}, g, q, u)$, $\mathcal{A}^{\mathtt{IND\text{-}CPA}}$ is unable to win $\mathtt{IND\text{-}CPA}$ with non-negligible advantage. This, in turn, means that $\mathsf{Red}^{\mathsf{DDH}}$ is unable to win $\mathtt{DDH}$ with non-negligible advantage, *i.e.*:

$$\Pr[\mathtt{DDH}_{\mathsf{Red}^{\mathsf{DDH}}} | \text{Non-DH Tuple}] = \frac{1}{2}.$$

By combining the above, we have that:

$$\begin{aligned}
\Pr[\mathtt{DDH}_{\mathsf{Red}^{\mathsf{DDH}}}] &= \Pr[\mathtt{DDH}_{\mathsf{Red}^{\mathsf{DDH}}} \wedge \text{DH Tuple}] &&+ \Pr[\mathtt{DDH}_{\mathsf{Red}^{\mathsf{DDH}}} \wedge \text{Non-DH Tuple}] \\
&= \Pr[\mathtt{DDH}_{\mathsf{Red}^{\mathsf{DDH}}} | \text{DH Tuple}] \cdot \Pr[\text{DH Tuple}] &&+ \Pr[\mathtt{DDH}_{\mathsf{Red}^{\mathsf{DDH}}} | \text{Non-DH Tuple}] \cdot \Pr[\text{Non-DH Tuple}] \\
&\geq (\frac{1}{2} + \mathsf{nonnegl}(\lambda)) \cdot \frac{1}{2} &&+ \frac{1}{2} \cdot \frac{1}{2} \\
&= \frac{1}{4} + \frac{1}{2} \cdot \mathsf{nonnegl}(\lambda) &&+ \frac{1}{4} \\
&= \frac{1}{2} + \frac{1}{2} \cdot \mathsf{nonnegl}(\lambda)
\end{aligned}$$

Putting everything together, we have that $\Pr[\mathtt{DDH}_{\mathsf{Red}^{\mathsf{DDH}}}] \geq \frac{1}{2} + \frac{1}{2} \cdot \mathsf{nonnegl}(\lambda)$. Since $\frac{1}{2} \cdot \mathsf{nonnegl}(\lambda) \in \mathsf{nonnegl}(\lambda)$, $\mathsf{Red}^{\mathsf{DDH}}$ retains a non-negligible advantage over a random guess in winning $\mathtt{DDH}$. This means that $\mathsf{Red}^{\mathsf{DDH}}$ would be able to break the DDH assumption in $\mathbb{G}$. But this is a contradiction, as the DDH assumption is believed to hold in $\mathbb{G}$. □

2.5 Digital signature schemes

In the previous subsection we described a primitive that aids towards preserving *confidentiality* of information. Here, we focus on *authenticity* instead. Namely: how can Bob ensure that some message he receives was indeed originating from Alice, and not an adversarial entity impersonating Alice? In the context of public-key cryptography, this goal may be achieved by leveraging *digital signatures*. To do so, Alice generates a signing key to *sign* messages, and a verification key to *verify* signatures. Alice would then compute a signature of a message by using her signing key, and forward the signature, the message, and the verification key to Bob. Bob will then use the verification key to ascertain whether the message is authentic by verifying its respective signature[10].

2.5.1 Defining correctness and EUF-CMA security

Similarly to public-key encryption, we formally capture the algorithmic behaviour of digital signature schemes in what follows.

[10] Of course, this strategy works as long as Bob receives the correct verification key from Alice. We discuss further the implications of the presence of a man-in-the-middle onto the communication channels in Section 4.1.

Definition 2.12 (Digital signature scheme)**.** *Let $\mathcal{K}$, $\mathcal{M}$, and Σ be key, message, and signature spaces respectively. Let $\mathcal{R}_{\mathsf{KeyGen}}$ and $\mathcal{R}_{\mathsf{Sign}}$ be randomness spaces for algorithms* KeyGen *and* Enc*, respectively. Let $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a tuple of algorithms defined as follows:*

- $\mathsf{KeyGen}(1^\lambda; r)$*: Upon input a unary description of the security parameter $\lambda \in \mathbb{N}$ and a randomness $r \in \mathcal{R}_{\mathsf{KeyGen}}$, output a signing-key verification-key tuple $(\mathsf{sk}, \mathsf{vk}) \in \mathcal{K}$.*
- $\mathsf{Sign}(\mathsf{sk}, m; r)$*: Upon input a signing key* sk *and a message m, and a randomness $r \in \mathcal{R}_{\mathsf{Sign}}$, output a signature $\sigma \in \Sigma$.*
- $\mathsf{Vrfy}(\mathsf{vk}, m, \sigma)$*: Upon input a verification key* vk*, a message m, and a signature σ, output $b \in \{0, 1\}$.*

We say Π is a correct digital signature scheme if, for all $\lambda \in \mathbb{N}$ and for all $m \in \mathcal{M}$, the following holds:

$$\Pr\left[\mathsf{Vrfy}(\mathsf{vk}, m, \sigma) = 1 \;\middle|\; \begin{array}{l}(\mathsf{sk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda) \\ \sigma \leftarrow\!\!\$\ \mathsf{Sign}(\mathsf{sk}, m)\end{array}\right] = 1.$$

Once again, Definition 2.12 only ensures that a signature scheme is "well-structured". From a security standpoint, Alice would like to avoid any other entity producing accepting signatures on arbitrary messages on her behalf. This requirement can be captured by the notion of "Existential Unforgeability against Chosen-Message Attacks" (EUF-CMA). Here, even if an adversarial entity is allowed to obtain signatures of any arbitrary message, it still cannot *forge* an accepting signature for a message that was not previously signed. We provide the respective security game in what follows:

Game $\texttt{EUF-CMA}_{\mathcal{A}}$		
1	$(\mathsf{sk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda)$	*Sample a keypair at random.*
2	$M \leftarrow \emptyset$.	*Initialize the set of queried messages M with an empty set.*
3	$(m^*, \sigma^*) \leftarrow \mathcal{A}(\mathsf{vk})^{\mathsf{Sign}_{\mathsf{sk}}(\cdot)}$	*Give* vk *to $\mathcal{A}$, who outputs a signature σ^* for a message m^*. $\mathcal{A}$ has oracle access to* $\mathsf{Sign}_{\mathsf{sk}}$.
4	Output $\mathsf{Vrfy}(\mathsf{vk}, m^*, \sigma^*) == 1 \wedge m \notin M$	*Output 1 if a (forged) signature on a new message is valid.*
Oracle $\mathsf{Sign}_{\mathsf{sk}}(m)$		
1	Add m to the set of messages M.	
2	Output $\sigma \leftarrow\!\!\$\ \mathsf{Sign}(\mathsf{sk}, m)$	

Game 5: Description of the EUF-CMA game $\texttt{EUF-CMA}_{\mathcal{A}}$.

Definition 2.13 (Existential Unforgeability against Chosen-Message Attacks (EUF-CMA))**.** *Let Π be a correct digital signature scheme, as per Definition 2.12. We say Π enjoys existential unforgeability against chosen-message attacks if, for all PPT adversaries $\mathcal{A}$, the following holds:*

$$\Pr[\texttt{EUF-CMA}_{\mathcal{A}} = 1] \leq \mathsf{negl}(\lambda).$$

Introducing oracles. Within this game, the challenger exposes an *oracle* to the adversary. In particular, the adversary can query the oracle on any arbitrary input, and the challenger will return the evaluation of that algorithm. In this case, the adversary is allowed to ask for signatures on any arbitrary message. Upon receiving a signature query for some message, the challenger will return a signature for that message by evaluating the Sign algorithm using sk (with sk remaining hidden to the adversary throughout the game).

Discussion. Once again, the above game captures a plethora of real-world adversarial capabilities. Indeed, even though the adversary is allowed to obtain signatures for any message of its choice with respect to some verification key vk, it still cannot forge a valid signature under vk for a non-queried message (*i.e.*, without interacting with the signer who holds sk) with non-negligible probability. Similarly to the case of IND-CPA, this security notion might be overkill for certain real-world applications (*e.g.*, an entity signing trusted drivers would not sign any arbitrary message upon request from a third party). Nonetheless, several natural (and efficient) schemes can be proven to be secure under this notion.

Remark. *In the context of a EUF-CMA secure signature scheme, it is still possible for the adversary to forge accepting signatures for already-queried messages. Intuitively, this means that any party holding exclusively* vk *can maul any accepting* (m, σ) *tuple to a new accepting* (m, σ') *tuple, with* $\sigma' \neq \sigma$*. While this capability allows the adversary to craft accepting signatures (albeit only for previously-signed messages), we will show later in the composition that this slack can be leveraged as a security enhancement.*

2.5.2 An instantiation: the BLS signature scheme

We are now ready to introduce the EUF-CMA secure BLS signature scheme [BLS01], which is designed in the random-oracle model and relies on the CDH assumption. We refer to [BLS01] for the complete proof of security.

Construction 2.2 (BLS signature scheme)**.** *Let* $\mathbb{G}$ *be a cyclic group in which the CDH assumption is believed to hold, and let* H *be a random oracle. The BLS signature scheme consists of the following algorithms:*

- $\mathsf{KeyGen}(1^\lambda)$*: Compute* $(\mathbb{G}, g, q) \leftarrow\!\!\$\ \mathsf{GroupGen}(1^\lambda)$*. Compute* $x \leftarrow\!\!\$\ \mathbb{Z}_q$*, and evaluate* $h \leftarrow g^x$*. Output a tuple* $(\mathsf{sk}, \mathsf{pk})$*, where* $\mathsf{sk} \leftarrow (\mathbb{G}, g, q, x)$ *and* $\mathsf{pk} \leftarrow (\mathbb{G}, g, q, h)$*.*
- $\mathsf{Sign}(\mathsf{sk}, m)$*: Parse* sk *as* $(\mathbb{G}, g, q, x)$*. Output* $H(m)^x$*.*
- $\mathsf{Vrfy}(\mathsf{vk}, (m, \sigma))$*: Parse* vk *as* $(\mathbb{G}, g, q, h)$*. Output* $\hat{e}(\sigma, g) == \hat{e}(H(m), h)$*.*

Theorem 2.3 ([BLS01])**.** *Construction 2.2 is an EUF-CMA secure signature scheme, assuming the CDH assumption holds and that* H *acts as a random oracle.*

2.6 Other useful cryptographic primitives

In what follows, we recap cryptographic primitives that will be referred to throughout the remainder of the composition. Within this subsection, we use more compact game-based definitions that are equivalent to the more explicit counterparts we presented in previous sections.

2.6.1 One-Way Functions

Informally, a one-way function (OWF) is a function that may be efficiently evaluated while being hard to invert.

Definition 2.14 (One-way function)**.** *Let* $f : \{0,1\}^* \to \{0,1\}^*$ *be an efficiently-computable function. We say* f *is one-way if, for all PPT adversaries* $\mathcal{A}$*, the following holds:*

$$\Pr\left[x = x' \;\middle|\; \begin{array}{l} x \leftarrow\!\!\$\ \{0,1\}^\lambda; y \leftarrow f(x) \\ x' \leftarrow \mathcal{A}(y) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

2.6.2 Pseudo-Random Permutations

A pseudo-random permutation (PRP) is a keyed family of permutations that is hard to evaluate and invert without knowledge of the key k. More formally:

Definition 2.15 (Pseudo-random permutation)**.** *A pseudo-random permutation (PRP) is a keyed family of permutations* $P_k : \{0,1\}^n \to \{0,1\}^n$ *indexed by a key* $k \in \{0,1\}^\lambda$ *for which the following holds:*

- $\forall k \in \{0,1\}^\lambda$*,* $P_k(x)$ *is a bijection from* $\{0,1\}^n$ *to* $\{0,1\}^n$*.*
- $\forall k \in \{0,1\}^\lambda$*,* $\forall x \in \{0,1\}^n$*,* $P_k(x)$ *and* $P_k^{-1}(x)$ *are efficiently computable;*
- *For all PPT adversaries* $\mathcal{A}$*, the following holds:*

$$\Pr\left[\mathcal{A}^{P_k(\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{R(\cdot)}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda),$$

where $k \leftarrow\!\!\$\ \{0,1\}^\lambda$ *and* $R : \{0,1\}^n \to \{0,1\}^n$*, with* R *being a random function.*

2.6.3 Non-Interactive Zero-Knowledge

Next, we expand on the high-level introduction to Non-Interactive Zero Knowledge reported in Section 1.4.2 by formally capturing correctness and salient security properties. For convenience, we restrict our attention to efficient adversaries, although one can obtain equivalent statistical properties by lifting this requirement.

Completeness. From a functional standpoint, a non-interactive zero-knowledge proof system is a non-interactive complete protocol, defined as follows:

Definition 2.16 (Set of Non-Interactive Complete Protocols $\mathcal{F}_{\mathsf{NI}}$ for the language $\mathcal{L}$)**.** *Let* $\Pi = (\mathsf{setup}, \mathcal{P}, \mathcal{V})$ *be a tuple of algorithms defined as follows:*

- $\mathsf{setup}(1^n, 1^\lambda)$*: Upon input a unary description of the length of the supported statements* n *and a unary description of the security parameter* λ*, output public parameters* pp*.*
- $\mathcal{P}(pp, x, w)$*: Upon input public parameters* pp*, a statement* x*, and a witness* w*, output a proof* π*.*
- $\mathcal{V}(pp, x, \pi)$*: Upon input public parameters* pp*, a statement* x *and a proof* π*, output* $b \in \{0, 1\}$*.*

Then, $\Pi \in \mathcal{F}_{\mathsf{NI}}$ *if, for all* $\lambda \in \mathbb{N}$ *and all* $(x, w) \in \mathscr{R}_{\mathcal{L}}$*, where* $\mathcal{L}$ *is an* NP *language, it holds that:*

$$\Pr\left[\mathcal{V}(pp, x, \mathcal{P}(pp, x, w)) = 1 \mid pp \leftarrow\$ \mathsf{setup}(1^{|x|}, 1^\lambda)\right] \geq 1 - \mathsf{negl}(\lambda).$$

Throughout this composition, setup will be an algorithm setting up a common reference string.

Soundness. Next, we formalize soundness, which informally states that no adversary can produce an accepting proof for a false statement.

Definition 2.17 (Soundness)**.** *Let* $\Pi \in \mathcal{F}_{\mathsf{NI}}$*. We say* Π *is sound if, for all PPT algorithms* $\mathcal{A}$*, the following holds:*

$$\Pr\left[\mathcal{V}(pp, x, \pi) = 1 \wedge x \notin \mathcal{L} \mid pp \leftarrow\$ \mathsf{setup}(1^{|x|}, 1^\lambda); (x, \pi) \leftarrow\$ \mathcal{A}(pp)\right] \leq \mathsf{negl}(\lambda).$$

Zero-knowledge. Zero-knowledge roughly states that no malicious verifier can retrieve any information about a witness from an accepting proof. Formally, this requirement is captured by constructing an efficient simulator that, without knowledge of the witness, is able to output accepting proofs that are indistinguishable from real proofs to any (malicious) verifier. This is typically achieved by granting the simulator additional power compared to the real-world execution, such as the capability of rewinding the adversary or generating the setup required by the protocol alongside some trapdoor information.

Definition 2.18 (Black-box Non-Interactive Zero Knowledge)**.** *Let* $\Pi \in \mathcal{F}_{\mathsf{NI}}$*. Let* $\mathtt{REAL}$ *and* $\mathtt{IDEAL}$ *be two experiments defined as follows:*

- $\mathtt{REAL} = \{(pp, x, w, \pi) \mid pp \leftarrow\$ \mathsf{setup}(1^{|x|}, 1^\lambda);\ (x, w) \leftarrow\$ \mathcal{A}(pp);\ \pi \leftarrow\$ \mathcal{P}(pp, x, w)\}_{\lambda \in \mathbb{N}}$
- $\mathtt{IDEAL}_{\mathcal{S}} = \{(pp, x, w, \pi) \mid (pp, \tau) \leftarrow\$ \mathcal{S}^0(1^{|x|}, 1^\lambda);\ (x, w) \leftarrow\$ \mathcal{A}(pp);\ \pi \leftarrow\$ \mathcal{S}^1(pp, \tau, x)\}_{\lambda \in \mathbb{N}}$.

We say that Π *satisfies black-box zero-knowledge if there exists a PPT algorithm* $\mathcal{S} = (\mathcal{S}^0, \mathcal{S}^1)$ *such that, for all PPT algorithms* $\mathcal{A}$*, and for all* $(x, w) \in \mathscr{R}_{\mathcal{L}}$*, the following holds:*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{l} b \leftarrow\$ \{0,1\} \\ b' \leftarrow \mathcal{A}(pp, x, w, \pi) \end{array}; (pp, x, w, \pi) \leftarrow\$ \begin{cases} \mathtt{REAL} & \text{if } b = 0 \\ \mathtt{IDEAL}_{\mathcal{S}} & \text{if } b = 1 \end{cases}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Intuitively, in the experiment $\mathtt{REAL}$ the challenger produces a proof for a statement in the language by using its witness to run the prover algorithm $\mathcal{P}$. In the experiment $\mathtt{IDEAL}$, instead, the challenger produces a proof without knowledge of the witness by running the simulator (using some trapdoor τ). Note that $\mathtt{IDEAL}$ is completely independent of the witness; hence, no adversary can ever retrieve any information about it. If the adversary is unable to distinguish the two experiments, it means that it cannot retrieve any information from $\mathtt{REAL}$ either.

Knowledge soundness. Knowledge soundness strengthens soundness by ensuring that computing an accepting proof implies knowledge of a witness for a statement in the language. This is formalized by means of an efficient extractor algorithm which can retrieve the witness w from any accepting proof π. Similarly to the simulator of zero-knowledge, the extractor may be empowered with the capability of rewinding or generating the setup required by the protocol alongside some trapdoor information (which should remain secret, as otherwise anyone could run the extractor, violating zero-knowledge). The security property states that no (malicious) prover can produce an accepting proof that makes the extractor fail.

Definition 2.19 (Black-box argument-of-knowledge)**.** *Let $\Pi \in \mathcal{F}_{\mathsf{NI}}$. We say that Π satisfies black-box argument-of-knowledge if there exists a PPT algorithm $\mathcal{E} = (\mathcal{E}^0, \mathcal{E}^1)$ such that, for all PPT algorithms $\mathcal{A}$, the following holds:*

$$\Pr\left[\mathcal{A}(pp) = 1 \mid pp \leftarrow\!\$\ \mathsf{setup}(1^{|x|}, 1^\lambda)\right] \approx_c \Pr\left[\mathcal{A}(pp) = 1 \mid (pp, \xi) \leftarrow\!\$\ \mathcal{E}^0(1^{|x|}, 1^\lambda)\right],$$

and

$$\Pr\left[(x, w) \in \mathscr{R}_{\mathcal{L}}, \mathcal{V}(pp, x, \pi) = 1 \;\middle|\; \begin{array}{l}(pp, \xi) \leftarrow\!\$\ \mathcal{E}^0(1^n, 1^\lambda); (x, \pi) \leftarrow \mathcal{A}(pp) \\ w \leftarrow \mathcal{E}^1(pp, \xi, x, \pi)\end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

The first condition ensures that no adversary can detect the introduction of a trapdoor setup by the extractor, whereas the second states that the extractor successfully extracts a valid witness from any accepting adversarially-generated proof with overwhelming probability.

Remark. *The aforementioned definitions only consider "black-box" properties, meaning that the simulator (resp. the extractor) works for all possible adversaries (*i.e.*, $\exists\mathcal{S}\forall\mathcal{A}$). Nonetheless, it is perfectly reasonable to capture these properties in a "non-black-box" manner, allowing the simulator (resp. the extractor) to depend on the adversary (*i.e.*, $\forall\mathcal{A}\exists\mathcal{S}$). We discuss these variants in Appendix C.3.*

Simulation soundness. In proving security of protocols, the adversary might be able to gain access to simulated proofs - a capability which could potentially be leveraged to attack soundness. Simulation soundness strengthens soundness to ensure that no adversary can output an accepting proof for a false statement even if it is allowed to query the simulator to obtain accepting proofs for statements of its choice.

Definition 2.20 (Simulation soundness)**.** *Let $\Pi \in \mathcal{F}_{\mathsf{NI}}$ that additionally satisfies soundness and zero-knowledge. We say Π enjoys simulation soundness if, for every PPT adversary $\mathcal{A}$, the following holds:*

$$\Pr\left[\begin{array}{l}\mathcal{V}(pp, x, \pi) = 1 \wedge (x, \pi) \notin Q \wedge x \notin \mathcal{L} \\ \end{array}\;\middle|\; \begin{array}{l}(pp, \tau_s) \leftarrow\!\$\ \mathcal{S}_0(1^\lambda) \\ (x, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_1(pp, \tau_s, \cdot)}(pp)\end{array}\right] \leq \mathsf{negl}(\lambda),$$

where Q contains the replies of the oracle $\mathcal{S}_1$.

Simulation-sound extractability. Finally, it is possible to equip simulation soundness with a flavour of extractability. Intuitively, the extractor should succeed in extracting a witness for any accepting proof generated by the adversary that was not previously output by the oracle generating simulated proofs.

Definition 2.21 (Simulation-sound extractability)**.** *Let $\Pi \in \mathcal{F}_{\mathsf{NI}}$ that additionally satisfies soundness, zero-knowledge, and proof-of-knowledge. Let $\mathcal{SE} = (\mathcal{SE}_0, \mathcal{SE}_1)$ be an algorithm that, on input 1^λ, returns (pp, τ_s, τ_e) such that (pp, τ_s) is identically distributed to the output of $\mathcal{S}_0$, with τ being the simulation trapdoor and ξ being the extraction trapdoor. We say Π enjoys simulation-sound extractability if, for every PPT adversary $\mathcal{A}$, the following holds:*

$$\Pr\left[\begin{array}{l}\mathcal{V}(pp, x, \pi) = 1 \wedge (x, \pi) \notin Q \wedge (x, w) \in \mathscr{R}_{\mathcal{L}} \\ \end{array}\;\middle|\; \begin{array}{l}(pp, \tau, \xi) \leftarrow\!\$\ \mathcal{SE}_0(1^\lambda) \\ (x, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_1(pp, \tau, \cdot)}(pp, \xi) \\ w \leftarrow \mathcal{E}_1(pp, \xi, x, \pi)\end{array}\right] \geq 1 - \mathsf{negl}(\lambda),$$

where Q contains the replies of the oracle $\mathcal{S}_1$.

2.6.4 Commitment schemes

Informally, a commitment scheme consists of two algorithms: one to "commit" to messages, one to "open" commitments. Whenever a party commits to a message, it also computes an "opening" value, which is required for the opening phase to be successful. This way, a party can publish a commitment *com* of some message m, and later publish the respective opening which may be used to check that *com* indeed contained the message m.

Definition 2.22 (Non-interactive string commitment scheme)**.** *A* `NIC` *for message space* $\mathcal{M}$ *is a tuple of algorithms* $(\mathsf{setup}, \mathsf{Com}, \mathsf{Open})$ *defined as follows:*

- $\mathsf{setup}(1^\lambda)$*: Upon input a unary description of the security parameter* λ*, output public parameters* pp*.*
- $\mathsf{Com}(pp, m)$*: Upon input public parameters* pp *and a message* m*, output a commitment* com *and opening information* o*.*
- $\mathsf{Open}(pp, com, o)$*: Upon input public parameters* pp*, a commitment* com*, and an opening* o*, output a message* m*.*

We say `NIC` *satisfies correctness if the following holds:*

$$\Pr\left[\mathsf{Open}(pp, com, o) = m \mid pp \leftarrow\!\!\$\ \mathsf{setup}(1^\lambda); (com, o) \leftarrow\!\!\$\ \mathsf{Com}(pp, m)\right] \geq 1 - \mathsf{negl}(\lambda).$$

As for commonly-considered security properties, we expect that opening a commitment to two different values should be unfeasible. Moreover, the commitment should reveal no information about the committed message. The former is formalized with the notion of *binding*; the latter with the notion of *hiding*.

Definition 2.23 (Binding non-interactive commitment scheme)**.** *Let* `NIC` *be a non-interactive string commitment scheme satisfying correctness, as per Definition 2.22. We say* `NIC` *is computationally binding if, for all PPT adversaries* $\mathcal{A}$ *and all* $m_0 \neq m_1$*, with* $m_0, m_1 \in \mathcal{M}$*:*

$$\Pr\left[\mathsf{Open}(pp, com, o_0) = m_0 \ \wedge\ \mathsf{Open}(pp, com, o_1) = m_1 \;\middle|\; \begin{array}{l} pp \leftarrow\!\!\$\ \mathsf{setup}(1^\lambda) \\ (com, o_0, o_1) \leftarrow \mathcal{A}(pp) \end{array}\right] \leq \mathsf{negl}(\lambda).$$

Definition 2.24 (Hiding non-interactive commitment scheme)**.** *Let* `NIC` *be a non-interactive string commitment scheme satisfying correctness, as per Definition 2.22. We say* `NIC` *is computationally hiding if, for all* $m_0 \neq m_1$*, with* $m_0, m_1 \in \mathcal{M}$*, and for all* $pp \leftarrow\!\!\$\ \mathsf{setup}(1^\lambda)$*, the following holds:*

$$\{\mathsf{Com}(pp, m_0)\} \approx_c \{\mathsf{Com}(pp, m_1)\}.$$

Similarly to NIZK, if the above properties hold against unbounded adversaries, we refer to statistical binding and statistical hiding, respectively.

2.6.5 Threshold secret sharing

A secret sharing scheme allows a party to "split" a message into different shares that may be used to reconstruct the message itself. Concretely, these shares can be distributed among different parties, who may reconstruct the original message by cooperating. In this composition, we discuss only *threshold* secret sharing - namely, any subset of shares of size t can be used to reconstruct the message, whereas any subset of size $t' < t$ reveals no information about the message.

Definition 2.25 (t-out-of-n secret sharing scheme)**.** *A* t*-out-of-*n *secret sharing scheme* `SS` *is a pair of algorithms (*`share`*,* `reconstruct`*) defined as follows:*

- `share`(m)*: Upon input a secret* m*, output a set of* n *shares* $(s_1, \ldots, s_n)$*.*
- `reconstruct`$(s_1, \ldots, s_t)$*: Upon input* t *shares* $(s_1, \ldots, s_t)$*, output a message* m*.*

We say $\mathtt{SS}$ *satisfies correctness if,* $\forall m$, $\forall S = \{i_1, \ldots, i_t\} \subseteq \{1, \ldots, n\}$ *of size* t*, it holds that:*

$$\Pr[\mathtt{reconstruct}(s_{i_1}, \ldots, s_{i_t}) = m \mid (s_1, \ldots, s_n) \leftarrow\!\!\$\ \mathtt{share}(m)] = 1.$$

Definition 2.26 (Perfect security of a t-out-of-n secret sharing scheme)**.** *Let* $\mathtt{SS} = (\mathtt{share}, \mathtt{reconstruct})$ *be a* t*-out-of-*n *secret sharing scheme satisfying correctness. We say* $\mathtt{SS}$ *achieves perfect security if,* $\forall m, m'$*,* $\forall S \subseteq \{1, \ldots, n\}$*, where* $|S| < t$*, the following holds:*

$$\{(s_i : i \in S) \mid (s_1, \ldots, s_n) \leftarrow\!\!\$\ \mathtt{share}(m)\} \equiv \{(s'_i : i \in S) \mid (s'_1, \ldots, s'_n) \leftarrow\!\!\$\ \mathtt{share}(m')\}.$$

3 Secure Multi-Party Computation

In this section, we introduce the formal tools that enable the design of secure multi-party computation protocols in the presence of *authenticated* channels between parties[11]. Throughout the section, we focus on the two-party computation setting, given that the prime focus of this composition lies in functionalities of this kind. Nonetheless, the treatment can be generalized to n parties.

In Section 3.1, we introduce the real-ideal paradigm: a fundamental modelling artifact that enables the design of MPC protocols in the presence of a generic threat model. In Section 3.2, we cast the paradigm to the setting of eavesdroppers, exhibiting a simple MPC protocol that instantiates secure messaging transmission by leveraging a EUF-CMA secure encryption scheme. In Section 3.3, we cast the paradigm to the setting of malicious parties. In Section 3.4, we discuss how the real-ideal paradigm can be modified to ensure security even if a MPC protocol is composed arbitrarily with other protocols (so-called "universal composability"). As a concrete example, in Section 3.5 we formally introduce Oblivious Transfer, summarizing the formalism behind the universally composable instantiation of [PVW08].

3.1 The real-ideal paradigm

In the context of a MPC protocol Π computing a functionality f, the real-ideal paradigm prescribes the definition of a real-world experiment and an ideal-world experiment. Informally:

- The real-world experiment $\texttt{REAL}$ models a run of Π, in which parties interact and compute f in the presence of an adversary $\mathcal{A}$ with some pre-specified adversarial capabilities (*e.g.*, eavesdropping the communication channel, or corrupting either of the parties).
- The ideal-world experiment $\texttt{IDEAL}$, where parties supply their input to a trusted entity $\mathcal{F}$ (also known as an *"ideal functionality"*) that securely evaluates the functionality f, alongside an ideal-world adversary $\mathcal{S}$ (often referred to as the *"simulator"*) with the same adversarial capabilities of $\mathcal{A}$.

Given these two worlds, the real-ideal paradigm is defined as an indistinguishability-based game, in which an efficient distinguisher $\mathcal{D}$ is asked to distinguish whether it is observing the real-world experiment or the ideal-world experiment. In particular, $\mathcal{D}$ supplies inputs to the parties as well as a description d of the adversarial capabilities[12], and obtains the output of the honest parties as well as the output of the adversary (who may also control either of the parties). We provide a formal definition in what follows:

Definition 3.1 (Secure realization of f)**.** *Let Π be a two-party protocol correctly evaluating a functionality f when both parties are honest. We say Π securely realizes f if, for all PPT adversaries $\mathcal{A}$, there exists a PPT adversary $\mathcal{S}$ for which the following holds for all PPT distinguishers $\mathcal{D}$:*

$$\{\texttt{REAL}_{\Pi,\mathcal{A},\mathcal{D}}(1^\lambda, x_1, x_2, \mathsf{d})\} \approx_c \{\texttt{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{D}}(1^\lambda, x_1, x_2, \mathsf{d})\}.$$

In the next subsections, we provide motivating examples and pictorial depictions of the above paradigm for different adversarial models. As a primer, we consider security in the presence of eavesdroppers, providing a complete security proof for a protocol realizing secure message transmission. We then allow the adversary to actively participate into the protocol by fully corrupting either of the parties.

3.2 Secure MPC in the presence of eavesdroppers

In Figure 3.1, we depict the real-ideal paradigm in the presence of eavesdroppers.

[11] Indeed, this is a ubiquitous assumption that is implicit in the majority of works designing secure MPC protocols. Nonetheless, this modelling choice is too strong for the design of certain functionalities (*e.g.*, PAKE). We will discuss the implications of removing this assumption in the next section.

[12] Often times, definitions report an index i in place of d, representing the index of the party to be corrupted, stating that the indistinguishability should hold for all i. Here we provide a slightly less structured notation that aims at capturing also other adversarial capabilities (*e.g.*, eavesdroppers).

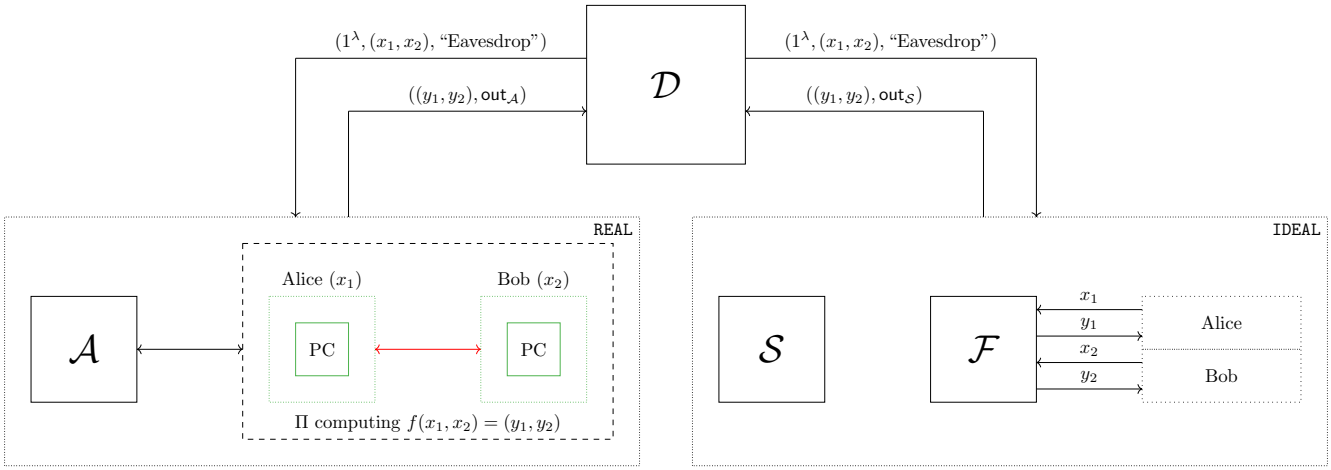


Figure 3.1: The real-ideal paradigm for a two-party protocol in the presence of eavesdroppers.

The real-world experiment. In the real-world experiment, Alice and Bob run the protocol Π evaluating f with the inputs (x_1, x_2) supplied by $\mathcal{D}$, returning output (y_1, y_2). $\mathcal{A}$ interacts with the protocol execution by leveraging its adversarial capabilities - in this case, eavesdropping the communication channel used by Alice and Bob throughout the protocol run. At any time, the adversary terminates and outputs $\mathsf{out}_\mathcal{A}$ to $\mathcal{D}$.

The ideal-world experiment. In the ideal-world experiment, Alice and Bob provide the inputs (x_1, x_2) received by $\mathcal{D}$ to $\mathcal{F}$, who securely evaluates $f(x_1, x_2)$ and returns (y_1, y_2) to the respective parties. The parties return (y_1, y_2) to $\mathcal{D}$. $\mathcal{S}$, in this case, receives no further information, as all parties are honest and there are no channels to be compromised. At any time, $\mathcal{S}$ terminates and outputs $\mathsf{out}_\mathcal{S}$ to $\mathcal{D}$.

Why does such a definition make sense? Recall that secure MPC protocols aim at correctly evaluating some functionality f while retaining the privacy of the parties' inputs. Crucially, the trusted functionality $\mathcal{F}$ evaluating f in the ideal world preserves both these properties at once. Hence, if no efficient distinguisher $\mathcal{D}$ can determine whether it is interacting with `REAL` or `IDEAL`, the real-world protocol run retains the same properties of the ideal world.

It is clear that (y_1, y_2) are identically distributed in both experiments. But how can the indistinguishability between `REAL` and `IDEAL` hold if the ideal world has no channel to be eavesdropped? The idea is to appropriately define the ideal-world adversary $\mathcal{S}$ to produce an output that is indistinguishable to that of $\mathcal{A}$. For that, $\mathcal{S}$ internally runs $\mathcal{A}$ by "faking" a real-world protocol run between two honest parties *without knowledge of their input*. If $\mathcal{S}$ manages to generate a transcript that is computationally indistinguishable from one generated in the real world, $\mathcal{A}$ will produce the same $\mathsf{out}_\mathcal{A}$ it would have output in the real world. Hence, $\mathcal{S}$ can forward $\mathsf{out}_\mathcal{S} \approx_c \mathsf{out}_\mathcal{A}$ to $\mathcal{D}$, who will therefore be unable to distinguish the two worlds apart (except with negligible probability).

Secure Message Transmission. As an example, let us consider the secure message transmission functionality $f_{\mathsf{SMT}}(m, \perp) = (\perp, m)$. With this functionality, Alice wants to send a sensitive message m to Bob.

From the previous section, we intuitively expect an IND-CPA secure encryption scheme to be sufficient in enabling Alice to securely send m to Bob in the presence of eavesdroppers by means of a protocol structured as Figure 3.2. Indeed, Bob can retrieve the message by decrypting an IND-CPA secure ciphertext which reveals no information to any other entity.

In what follows, we show that the protocol in Figure 3.2 is provably secure under the real-ideal paradigm. In particular, we formally argue that the protocol realizes the secure message transmission functionality

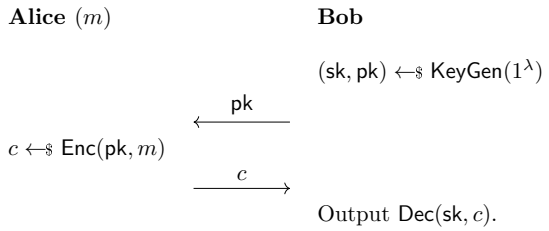


Figure 3.2: Alice transmitting an encrypted message to Bob using an IND-CPA public-key encryption scheme.

f_{SMT} in the presence of eavesdroppers. As a salient point, we will define a simulator $\mathcal{S}$ that replicates what the adversary sees by eavesdropping a protocol run between Alice and Bob *without knowing Alice's secret message*.

Theorem 3.1. *Let* $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be an IND-CPA secure public-key encryption scheme. Then, the protocol in Figure 3.2 realizes* f_{SMT} *in the presence of eavesdroppers.*

Proof. The first step consists of identifying what the external distinguisher $\mathcal{D}$ represented in Figure 3.1 can infer by interacting with either `REAL` or `IDEAL`. In particular, $\mathcal{D}$ learns: (i) the output of Alice and Bob, and (ii) the output of the adversary. Since we only consider eavesdroppers, item (i) is trivially identically distributed in both experiments: in the real world, Bob outputs m by correctness of the protocol; in the ideal world, Bob outputs m by correctness of the ideal functionality $\mathcal{F}$. It remains to argue that (ii) holds.

To do that, we focus on what $\mathcal{A}$ can observe (and control) in the context of a protocol run between Alice and Bob. Since $\mathcal{A}$ is restricted to eavesdropping, it may exclusively learn the *transcript* of a protocol run (*i.e.*, the set of messages exchanged by Alice and Bob). In particular, $\mathcal{A}$ observes a public key pk, flowing from Bob to Alice, and a ciphertext c, flowing from Alice to Bob.

Next, we build our simulator $\mathcal{S}$. The simulator's strategy consists of internally running the real-world adversary $\mathcal{A}$ while "faking" a protocol run by mimicking what Alice and Bob would have done in the real world. Recall that, as per Figure 3.1, Alice and Bob are exclusively interacting with the ideal functionality, whereas the adversary expects to observe a transcript generated from a protocol run. For that, $\mathcal{S}$:

- Simulates Bob by following exactly the protocol specification.
- Simulates Alice by encrypting the all-zero string instead of Alice's original input m. This makes the simulator *independent* from m.

We depict the simulator pictorially in Figure 3.3.

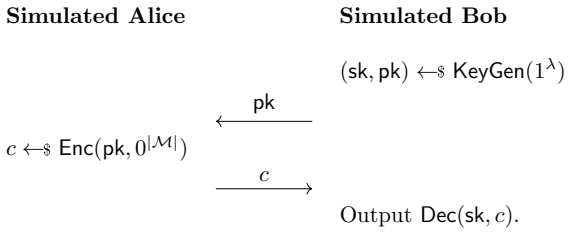


Figure 3.3: A simulator for the protocol in Figure 3.2 with honest Alice and honest Bob.

Let $\{\mathsf{pk}, c\}_{\mathtt{REAL}}$ be the distribution of transcripts output by real-world protocol runs and $\{\mathsf{pk}, c\}_{\mathtt{IDEAL}}$ be the distribution of transcripts output by simulated protocol runs. We conclude the proof by arguing that no efficient adversary can distinguish these two distributions. Indeed, by arguing that this is the case, the internal run of $\mathcal{A}$ of the simulator will behave as if it was in the real world (and, in particular, it will compute the same output); hence, $\mathcal{D}$ would be unable to have any non-negligible distinguishing advantage in telling the two experiments apart. Intuitively, this is the case, as:

- Both runs compute pk exactly the same way. Hence, $\{\mathsf{pk}\}_{\texttt{REAL}} \equiv \{\mathsf{pk}\}_{\texttt{IDEAL}}$.
- The real-world run computes c as an encryption of m, whereas the simulated run computes c as an encryption of $0^{|\mathcal{M}|}$. But the public-key encryption scheme is IND-CPA secure, so c does not reveal any information about the plaintext. Hence, $\{c\}_{\texttt{REAL}} \approx_c \{c\}_{\texttt{IDEAL}}$.

In Lemma 3.1 we formally prove that, as long as the public-key encryption scheme is IND-CPA secure, $\{\mathsf{pk}, c\}_{\texttt{REAL}} \approx_c \{\mathsf{pk}, c\}_{\texttt{IDEAL}}$. □

Lemma 3.1. *Transcripts obtained from the protocol in Figure 3.2 are computationally indistinguishable from transcripts obtained from the simulator in Figure 3.3.*

Proof. Assume, by contradiction, that $\{\mathsf{pk}, c\}_{\texttt{REAL}} \not\approx_c \{\mathsf{pk}, c\}_{\texttt{IDEAL}}$. This implies the existence of a PPT adversary $\mathcal{A}$ who can tell apart, with non-negligible probability, a real-world transcript produced by the protocol in Figure 3.2 from a simulated transcript produced by the simulator in Figure 3.3. Similarly to the proof of Theorem 2.2, we can use $\mathcal{A}$ to design a PPT reduction $\mathsf{Red}^{\texttt{IND-CPA}}$ to the IND-CPA security of Π.

Reduction against IND-CPA of Π $\mathsf{Red}^{\texttt{IND-CPA}}$		
1	$\mathsf{pk} \leftarrow \mathcal{C}^{\texttt{IND-CPA}}$	*Parse the public key obtained by $\mathcal{C}^{\texttt{IND-CPA}}$ into a key for the protocol Π.*
2	$(m, 0^{\lvert\mathcal{M}\rvert}) \leftarrow \mathcal{M}$	*Pick two messages: the secret message of Alice m and the all-zero string.*
3	$c^* \leftarrow \mathcal{C}^{\texttt{IND-CPA}}(m, 0^{\lvert\mathcal{M}\rvert})$	*Obtain a ciphertext of either message from $\mathcal{C}^{\texttt{IND-CPA}}$.*
4	$b' \leftarrow \mathcal{A}(\mathsf{pk}, c^*)$	*Provide the adversary with a protocol transcript.*
5	Output b'.	*Output whatever the adversary outputs.*

Figure 3.4: Description of the reduction $\mathsf{Red}^{\texttt{IND-CPA}}$, which acts simultaneously as the adversary for the $\texttt{IND-CPA}$ game and as the challenger for the distinguishing game described in Lemma 3.1.

Let us analyze the probability of $\mathsf{Red}^{\texttt{IND-CPA}}$ winning the $\texttt{IND-CPA}$ game. Again, we consider whether $\mathcal{C}^{\texttt{IND-CPA}}$ computes c^* as an encryption of m or $0^{|\mathcal{M}|}$ separately:

- $\mathcal{C}^{\texttt{IND-CPA}}$ produces an encryption of m. In this case, $\mathsf{Red}^{\texttt{IND-CPA}}$ is emulating a transcript for a real-world execution of the protocol.
- $\mathcal{C}^{\texttt{IND-CPA}}$ produces an encryption of $0^{|\mathcal{M}|}$. In this case, $\mathsf{Red}^{\texttt{IND-CPA}}$ is emulating a transcript for a simulated execution of the protocol.

Since, by contradiction, $\mathcal{A}$ is able to distinguish real-world transcripts from simulated-world transcripts with non-negligible advantage, $\mathsf{Red}^{\texttt{IND-CPA}}$ retains the same advantage in winning the $\texttt{IND-CPA}$ game. But this is a contradiction, as Π is assumed to be IND-CPA secure. □

3.3 Secure MPC in the presence of malicious parties

Until now we only considered adversaries *external* to the protocol run. The next natural adversarial capability consists of allowing either of the parties to be compromised. In particular, an adversary leveraging this capability would target the remaining honest party with the intent of undermining the guarantees of secure MPC - either by violating the privacy of the honest party's input, or by causing the honest party to evaluate the functionality incorrectly.

For this short introduction, we focus our attention on *malicious* corruptions[13], by which the adversary may run any arbitrary strategy on behalf of the compromised party. For simplicity of representation, we depict the resulting model in Figure 3.5 in the presence of a corrupted Bob. Of course, in order for an MPC protocol to be deemed secure in the presence of a generic malicious party, the indistinguishability must hold also if Alice is corrupted and Bob is honest.

[13]Often times, before moving to the malicious setting, introductions to secure MPC discuss *semi-honest* corruptions, an intermediate adversarial capability which restricts the adversary to learn the internal state of a party which follows the protocol specification. Under this assumption, one may, for instance, design more efficient protocols, or account for a larger amount of corrupted parties. For formal details, we refer to [Lin16, Chapter 4].

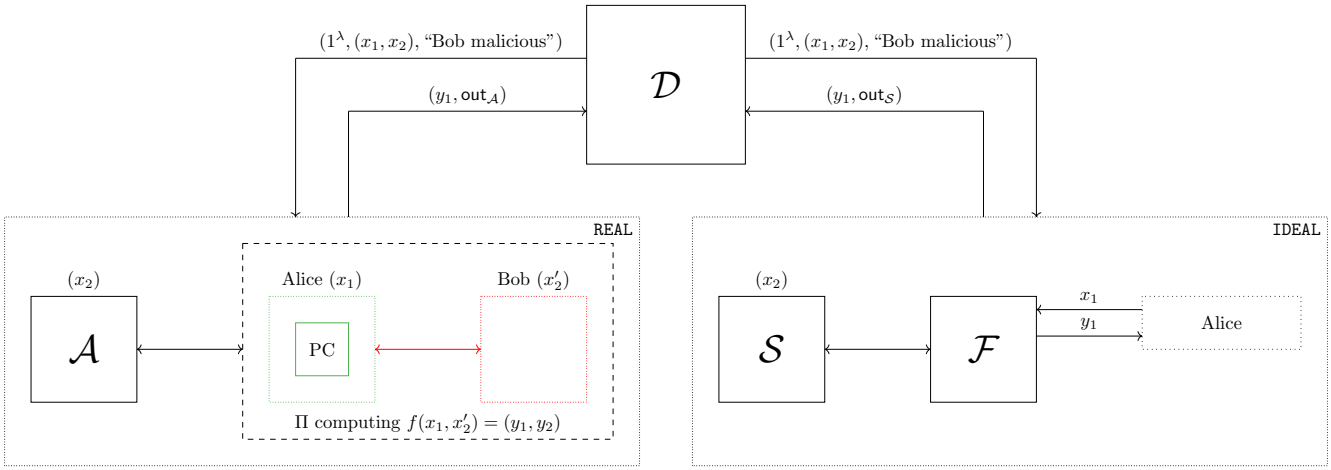


Figure 3.5: The real-ideal paradigm for a two-party protocol in the presence of a malicious Bob.

We first describe the two experiments by assuming that the real-world protocol preserves fairness and guaranteed output delivery (*i.e.*, the adversary cannot prevent honest Alice from getting the output of the computation). Then, we briefly touch on why this assumption is in general too strong in the context of two-party computation, and how the model can be adapted accordingly.

The real-world experiment. Since Bob is corrupted, $\mathcal{D}$ reveals its respective input x_2 to $\mathcal{A}$. However, $\mathcal{A}$ can make Bob follow any arbitrary strategy. In particular, there is no guarantee that Bob will use x_2 as input to the protocol, or that it will participate in a protocol run in its entirety. For instance, Bob could use any arbitrary x'_2 as input, or stop responding throughout the protocol run. Since we assume that both parties always obtain their output, Alice and Bob compute $f(x_1, x'_2)$ by means of Π. Alice outputs y_1 to $\mathcal{D}$, whereas the adversary outputs $\mathsf{out}_\mathcal{A}$ to $\mathcal{D}$. We do not show y_2 as a separate output of the experiment (as we did in Figure 3.1), given that $\mathcal{A}$ can embed the output of malicious Bob into $\mathsf{out}_\mathcal{A}$.

The ideal-world experiment. Since Bob is corrupted, $\mathcal{D}$ reveals its respective input x_2 to $\mathcal{S}$. Similarly to the case of eavesdroppers, $\mathcal{S}$ has to produce $\mathsf{out}_\mathcal{S} \approx_c \mathsf{out}_\mathcal{A}$. In particular, $\mathcal{S}$ has to simulate a protocol run on behalf of the honest parties (who instead interact directly with the functionality) to ensure the internally-run $\mathcal{A}$ can produce the same output. However, $\mathcal{S}$ also needs to be careful in determining which input to supply to $\mathcal{F}$ on behalf of the corrupted party. To do that, $\mathcal{S}$ "extracts" Bob's input x'_2 by observing the behaviour of $\mathcal{A}$ within the simulated protocol run (we briefly discuss how this can be done at the end of this subsection). $\mathcal{S}$ then forwards x'_2 to $\mathcal{F}$ on behalf of Bob. At this point, $\mathcal{F}$ evaluates $f(x_1, x'_2)$, and outputs y_1 to Alice and y_2 to $\mathcal{S}$. Alice outputs y_1 to $\mathcal{D}$, while $\mathcal{S}$ forwards y_2 to $\mathcal{A}$ and outputs $\mathsf{out}_\mathcal{S} \leftarrow \mathsf{out}_\mathcal{A}$.

Security with aborts. Unfortunately, the above model is too strong in general, as there are certain functionalities for which it is impossible to obtain guaranteed output delivery and fairness, unless at least the majority of parties is honest (*e.g.*, coin tossing [Cle86]). In the two-party setting, this means that the corrupted party might obtain the output of a functionality while preventing the honest party from obtaining its own (*e.g.*, by not running the protocol to completion). In light of these considerations, the ideal world of Figure 3.5 can be slightly altered to embed these weaknesses. In particular, the ideal functionality $\mathcal{F}$ can be modified to accept two special messages from $\mathcal{S}$: abort and $\mathsf{continue}$. $\mathcal{F}$ reacts to these messages as follows:

- $\mathcal{S}$ may supply abort as the input of the corrupted party. In this case, $\mathcal{F}$ returns $\perp$ to the honest party and terminates.

- $\mathcal{S}$ may supply some input x_2' as the input of the corrupted party. In this case, $\mathcal{F}$ correctly evaluates $f(x_1, x_2')$, and releases y_2 to $\mathcal{S}$. Now $\mathcal{S}$ can either allow $\mathcal{F}$ to reveal y_1 to the honest party by means of a continue message, or have $\mathcal{F}$ output $\perp$ to the honest party by means of an abort message.

Extracting inputs in the ideal world. We mentioned that $\mathcal{S}$ needs to efficiently extract some input x_2' by observing its internal run of $\mathcal{A}$ in order to ensure that $\mathcal{F}$ will evaluate an output that is consistent with a real-world protocol run. For that, the simulator is granted some additional power that is crucially inaccessible to real-world adversaries[14]. This is often times achieved by "rewinding" the real-world adversary or by introducing a setup assumption in the protocol. We discuss these strategies in what follows.

- **Rewinding.** $\mathcal{S}$ "rewinds" the internal run of $\mathcal{A}$ up to a polynomial number of times in the security parameter, and uses the joint information collected across all the runs to extract the adversary's input. Note that a real-world adversary would not be able to rewind an honest party, given that it is a distinct entity over which the adversary has no control. We refer to [Lin16] for further details, alongside complete protocols and detailed security proofs for functionalities of wide interest.
- **Introducing setup assumptions.** If the protocol is designed by assuming access to a trust assumption (*e.g.*, a random oracle or a common reference string), $\mathcal{S}$ is granted the power to *control* the trust assumption. For instance, it may have access to all the random oracle queries and program their answers accordingly, or it could generate the CRS alongside some trapdoor information. This makes the simulator *straight-lined* (*i.e.*, non-rewinding); a property that is particularly convenient for the definition of protocols with universal composability in mind (more in the next section). We will see examples of this strategy throughout the composition.

3.4 Universal Composability

With the tools described in the previous subsections, it is possible to design a protocol Π for some functionality f that retains correctness and security even if one of the two parties deviates arbitrarily from the intended execution of Π. But what happens if we consider a more complex system in which Π is invoked multiple times alongside other protocols? In other words: do the security guarantees obtained by the real-ideal paradigm hold even if we *compose* our protocol with others?

As a piece of good news, we obtain sequential composition for free as long as one runs some protocol Π realizing f *atomically* (*i.e.*, without any interleaving of messages with other protocol runs). Unfortunately, arguing parallel composition in general requires reproving the security of the entire composed protocol. As a failing example, if a protocol's proof of security defines a simulator $\mathcal{S}$ that rewinds the real-world adversary $\mathcal{A}$ twice, running the same protocol in parallel polynomially-many times in the security parameter would cause the simulator to rewind $2^{\mathsf{poly}(\lambda)}$ times. This would invalidate the simulator, given that it would require exponential time to run to completion.

This restriction greatly hinders the usability of MPC protocols in real-world deployments, where devices are extremely dynamic and complex environments that typically host multiple processes at once running in an interleaved fashion. To overcome this limitation, we shift our attention to the framework of Universal Composability [Can01], which enables the design of MPC protocols that retain their security guarantees even in the presence of arbitrary composition.

From a definitional standpoint, the most salient modification to the real-ideal paradigm consists of replacing $\mathcal{D}$ with an *interactive distinguisher* that is allowed to interact throughout the `REAL` and `IDEAL` experiments. This is often referred to as an "environment" (hereby denoted with $\mathcal{Z}$), which represents everything that might happen externally with respect to the protocol run. In particular, $\mathcal{Z}$ may freely communicate with the adversary of the respective experiment. The resulting model is shown in Figure 3.6.

Within this composition, we focus on recapping the most basic capabilities of the UC framework by assuming that protocols run with a fixed set of parties, and that $\mathcal{Z}$ may only perform *static* corruptions (*i.e.*, $\mathcal{Z}$ decides *before* the protocol run which parties to corrupt and how).

[14] Indeed, if such a strategy was accessible to real-world adversaries, it could be leveraged to extract an honest party's inputs in a real-world protocol run, trivially voiding any security guarantee.

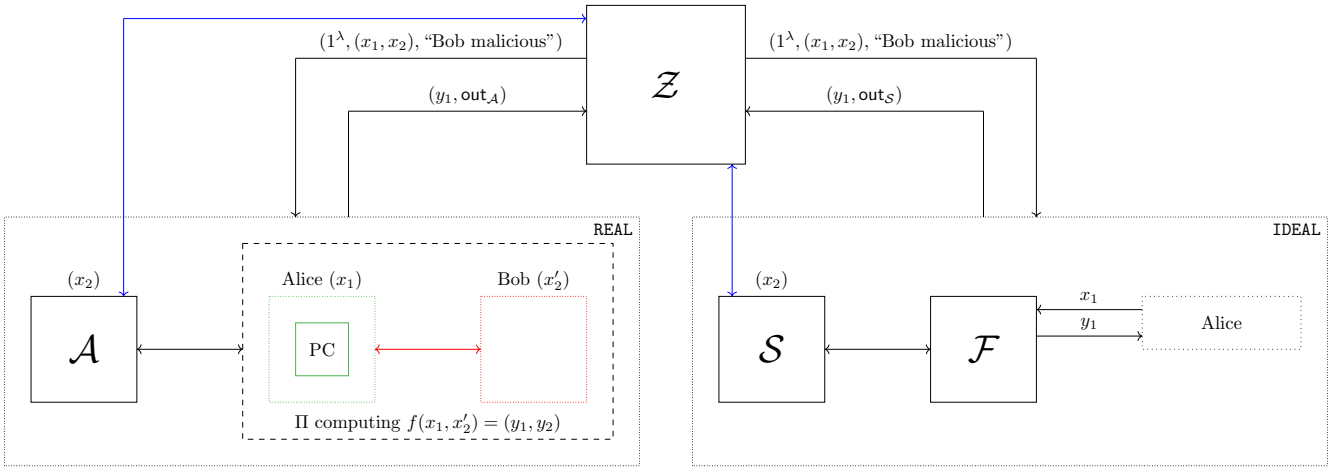


Figure 3.6: The real-ideal paradigm for a two-party protocol in the presence of a malicious Bob coordinated by an interactive distinguisher $\mathcal{Z}$.

Definition 3.2 (UC-emulation)**.** *Let* Π *be a two-party protocol correctly evaluating a functionality* f *when both parties are honest. We say* Π *UC-realizes* f *if, for all PPT adversaries* $\mathcal{A}$*, there exists a PPT adversary* $\mathcal{S}$ *for which the following holds for all PPT environments* $\mathcal{Z}$*:*

$$\{\mathtt{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\lambda, x_1, x_2, \mathsf{d})\} \approx_c \{\mathtt{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\lambda, x_1, x_2, \mathsf{d})\}.$$

As shown in [Can01], instead of explicitly quantifying the adversary in the definition above by considering arbitrary adversarial strategies, it suffices to consider so-called "dummy adversaries" - namely, adversaries that exclusively act as a relay for the environment.

The universal composition theorem. One of the most important results stemming from this model is the *universal composition theorem* [Can01], which greatly simplifies the design of security proofs in the context of composition. In particular, let us say that Π is a protocol that UC-realizes some ideal functionality $\mathcal{F}$, where Π invokes a sub-protocol Γ, and Γ is a protocol that UC-realizes some ideal functionality $\mathcal{G}$. The universal composition theorem guarantees that the security of Π is equivalent to that of $\Pi^{\Gamma\to\mathcal{G}}$, where $\Pi^{\Gamma\to\mathcal{G}}$ is the protocol Π for which the respective sub-protocol Γ is replaced by an evaluation of the ideal functionality $\mathcal{G}$. We depict the result in Figure 3.7.

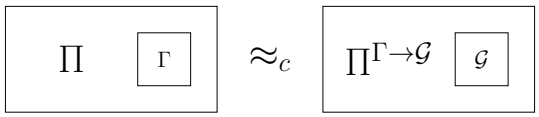


Figure 3.7: On the left, a protocol Π invoking, as a sub-routine, a protocol Γ UC-realizing an ideal functionality $\mathcal{G}$. On the right, a protocol Π invoking the ideal functionality $\mathcal{G}$ as sub-routine.

Such an approach greatly improves the modularity of Π: once the protocol is proved to be UC-secure, its security will hold irrespectively of how $\mathcal{G}$ is instantiated. This, *e.g.*, allows one to freely swap Γ with a different (possibly more efficient) Γ', as long as Γ' also UC-realizes $\mathcal{G}$. Whenever Π invokes an ideal functionality $\mathcal{G}$ as part of its execution, we say that Π runs in the $\mathcal{G}$-hybrid model.

On simulation by rewinding. Given that $\mathcal{Z}$ interacts with the adversary arbitrarily, rewinding is no longer a viable simulation strategy. This is the case, as $\mathcal{Z}$ would trivially detect any rewinding of $\mathcal{A}$ performed

by $\mathcal{S}$ in the ideal-world experiment. To overcome this limitation, a frequently-used strategy consists of designing a straight-lined simulator by leveraging a setup assumption (as discussed in the previous section). Indeed, a plethora of expressive two-party functionalities cannot be designed without assuming a trusted setup [CF01, CKL03].

3.5 An example: Universally Composable Oblivious Transfer

Oblivious transfer is a two-party functionality by which a sender S, who knows two inputs (x_0, x_1), interacts with a receiver R, who wants to learn the sender's input indexed by a choice bit σ. More formally, $f_{\mathsf{OT}}((x_0, x_1), \sigma) = (\perp, x_\sigma)$.

An alternative representation of f_{OT} consists of directly characterizing the behaviour of the ideal functionality in the ideal world, depicted in Figure 3.8. Looking ahead, such a representation will be particularly convenient while capturing more complex ideal functionalities (such as PAKE or authenticated communication).

Functionality $\mathcal{F}_{\mathsf{OT}}$

Upon receiving a query $(\text{SENDER}, \mathsf{sid}, (x_0, x_1))$ **from** S**:**

Record $(\text{SENDER}, \mathsf{sid}, (x_0, x_1))$. Ignore subsequent commands of the form $(\text{SENDER}, \mathsf{sid}, \cdot)$.

Upon receiving a query $(\text{RECEIVER}, \mathsf{sid}, \sigma)$ **from** R**:**

Check if a record $(\text{SENDER}, \mathsf{sid}, (x_0, x_1))$ exists. If so, output (sid, x_σ) to R, sid to the adversary $\mathcal{S}$, and halt. Otherwise, send nothing to R but continue running.

Figure 3.8: Description of the oblivious transfer functionality $\mathcal{F}_{\mathsf{OT}}$ [PVW08].

Party IDs and session IDs. Each party involved in the protocol run is assigned an identifying "party ID" (PID) as part of its input. In this case, the sender is identified by the symbol S, whereas the receiver by the symbol R. We will frequently use the symbol P_i to denote the PID of the i-th party participating in a protocol run. Additionally, each party receives a "session ID" (SID) as part of its input, which is used to identify different protocol runs and different calls to the same ideal functionality. Whenever a party calls an ideal functionality, it forwards both its PID and its SID. In this case, the sender may interact with $\mathcal{F}_{\mathsf{OT}}$ by specifying its PID S, its SID sid, alongside its inputs (x_0, x_1) in a SENDER query. Likewise, the receiver can forward its PID R, its SID sid, and its choice bit σ in a RECEIVER query. $\mathcal{F}_{\mathsf{OT}}$ only reacts to a query initiated by the correct PID (*i.e.*, the sender cannot run receiver queries and vice versa). Moreover, each session of $\mathcal{F}_{\mathsf{OT}}$ is evaluated separately.

3.5.1 Dual-Mode Encryption

As a stepping stone towards an instantiation of $\mathcal{F}_{\mathsf{OT}}$, we introduce the notion of *dual-mode* encryption of [PVW08]. These encryption schemes operate like traditional public-key encryption schemes, except for the following differences.

First, they introduce the notion of *encryption/decryption branches*. In particular, the party generating the key pair must additionally specify a *decryption* branch $\sigma \in \{0, 1\}$ as an additional input to the KeyGen algorithm. The party holding the public key can, instead, encrypt its message by picking an *encryption* branch $b \in \{0, 1\}$. The party holding the secret key is able to decrypt the ciphertext successfully only if $\sigma = b$.

Second, they rely on a common reference string that may be setup either in *messy* mode or *decryption* mode. These modes are computationally indistinguishable and induce different algorithms for the generation of a trapdoor, yielding different security guarantees. In messy mode, messages encrypted on branch $b \neq \sigma$ are information theoretically hidden, while the holder of the trapdoor can always identify a messy branch

for any (possibly malformed) pk (*i.e.*, a branch for which $b \neq \sigma$). In decryption mode, the holder of the trapdoor can generate a public key while being able to decrypt both branches.

We report the formal definition in what follows.

Definition 3.3 (Dual-Mode Encryption [PVW08])**.** *A dual-mode encryption scheme consists of a tuple of algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{FindMessy}, \mathsf{TrapKeyGen})$ *with the following properties:*

1. ***Completeness for decryptable branch:*** *For every* $\mu \in \{\mathsf{mes}, \mathsf{dec}\}$*, every* $(\mathsf{crs}, t) \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, \mu)$*, every* $\sigma \in \{0,1\}$*, every* $(pk, sk) \leftarrow\!\!\$\ \mathsf{KeyGen}(\sigma)$*, and every* $m \in \{0,1\}^k$*, decryption is correct on branch* σ*, i.e.,* $\mathsf{Dec}(sk, \mathsf{Enc}(pk, \sigma, m)) = m$*. It also suffices for decryption to be correct with overwhelming probability over the randomness of the entire experiment.*

2. ***Indistinguishability of modes:*** *Let* $\mathsf{SetupMessy}(1^\lambda) = \mathsf{Setup}(1^\lambda, \mathsf{mes})$ *and* $\mathsf{SetupDec}(1^\lambda) = \mathsf{Setup}(1^\lambda, \mathsf{dec})$*. Let* $\mathsf{SetupMessy}_1$ *and* $\mathsf{SetupDec}_1$ *denote the first outputs of* $\mathsf{SetupMessy}$ *and* $\mathsf{SetupDec}$*, respectively. Then,* $\mathsf{SetupMessy}_1(1^\lambda) \stackrel{c}{\approx} \mathsf{SetupDec}_1(1^\lambda)$*.*

3. ***(Messy mode) Trapdoor identification of a messy branch:*** *For every* $(\mathsf{crs}, t) \leftarrow\!\!\$\ \mathsf{SetupMessy}(1^\lambda)$ *and every (possibly malformed)* pk*,* $\mathsf{FindMessy}(t, pk)$ *outputs a branch value* $b \in \{0,1\}$ *such that* $\mathsf{Enc}(pk, b, \cdot)$ *is messy. Namely, for every* $m_0, m_1 \in \{0,1\}^k$*,* $\mathsf{Enc}(pk, b, m_0) \stackrel{s}{\approx} \mathsf{Enc}(pk, b, m_1)$*.*

4. ***(Decryption mode) Trapdoor generation of keys decryptable on both branches:*** *For every* $(\mathsf{crs}, t) \leftarrow\!\!\$\ \mathsf{SetupDec}(1^\lambda)$*,* $\mathsf{TrapKeyGen}(t)$ *outputs* (pk, sk_0, sk_1) *such that for every* $\sigma \in \{0,1\}$*,* $(pk, sk_\sigma) \stackrel{s}{\approx} \mathsf{KeyGen}(\sigma)$*.*

An instantiation from DDH. Next, we recap the instantiation of dual-mode encryption from DDH of [PVW08, Section 5].

Construction 3.1 (Dual-Mode Encryption from DDH [PVW08])**.** *The CRS is a tuple* (g_0, h_0, g_1, h_1)*, with different trapdoors according to the mode of operation.*

- $\mathsf{Setup}(1^\lambda, \mu)$*: Compute* $(\mathbb{G}, g, q) \leftarrow\!\!\$\ \mathsf{GroupGen}(1^\lambda)$*. All the following algorithms implicitly take* $(\mathbb{G}, g, q)$ *as input. If* $\mu = \mathsf{mes}$*, run* $\mathsf{SetupMessy}(1^\lambda)$*; if* $\mu = \mathsf{dec}$*, run* $\mathsf{SetupDec}(1^\lambda)$*.*
  - $\mathsf{SetupMessy}(1^\lambda)$*: Sample random generators* $g_0, g_1 \in \mathbb{G}$*. Choose distinct non-zero* $x_0, x_1 \leftarrow\!\!\$\ \mathbb{Z}_\mathsf{q} \setminus \{0\}$*. Let* $h_b \leftarrow g_b^{g_b}$ *for* $b \in \{0,1\}$*. Output a tuple* (crs, t)*, where* $\mathsf{crs} \leftarrow (g_0, h_0, g_1, h_1)$ *and* $t \leftarrow (x_0, x_1)$*.*
  - $\mathsf{SetupDec}(1^\lambda)$*: Sample a random generator* $g_0 \in \mathbb{G}$*, a random non-zero* $y \leftarrow\!\!\$\ \mathbb{Z}_\mathsf{q} \setminus \{0\}$*, and let* $g_1 = g_0^y$*. Choose a random non-zero* $x \leftarrow\!\!\$\ \mathbb{Z}_\mathsf{q} \setminus \{0\}$*. Let* $h_b \leftarrow g_b^x$ *for* $b \in \{0,1\}$*. Output a tuple* (crs, t)*, where* $\mathsf{crs} \leftarrow (g_0, h_0, g_1, h_1)$*, and* $t \leftarrow y$*.*

  All the following algorithms implicitly take crs *as input.*
- $\mathsf{KeyGen}(\sigma)$*: Choose* $r \leftarrow\!\!\$\ \mathbb{Z}_\mathsf{q} \setminus \{0\}$*. Let* $(g, h) \leftarrow (g_\sigma^r, h_\sigma^r)$*. Output a tuple* $(sk, pk) \leftarrow (r, (g, h))$*.*
- $\mathsf{Enc}(pk, m, b)$*: Parse* pk *as* (g, h)*. If* $g = 1 \in \mathbb{G}$*, output nothing. Else, sample* $s, r \leftarrow\!\!\$\ \mathbb{Z}_\mathsf{q}$*. Output a ciphertext* $c \leftarrow (g_b^s h_b^t, g^s h^t m) = (c_1, c_2)$*.*
- $\mathsf{Dec}(sk, c)$*: Parse* c *as* (c_1, c_2)*. Output* c_2 / c_1^{sk}*.*
- $\mathsf{FindMessy}(t, pk)$*: Parse the messy mode trapdoor* t *as* (x_0, x_1)*, and the public key* pk *as* (g, h)*. If* $g = 1 \in \mathbb{G}$*, output an arbitrary* $b \in \{0,1\}$ *as a messy branch. If* $h \neq g^{x_0}$*, output* $b = 0$ *as a messy branch. Otherwise, we have* $h = g^{x_0} \neq g^{x_1}$ *(as* $x_0 \neq x_1$ *and* g *is a generator), so output* $b = 1$ *as a messy branch.*
- $\mathsf{TrapKeyGen}(t)$*: Parse* t *as* y*. Choose* $r \leftarrow\!\!\$\ \mathbb{Z}_\mathsf{q} \setminus \{0\}$*. Compute* $pk \leftarrow (g_0^r, h_0^r)$*, and output* $(pk, r, r/y)$*.*

[PVW08, Theorem 5.2] shows that the above is a dual-mode encryption scheme. Completeness for the decryptable branch follows by inspection. We briefly discuss the remaining properties. Intuitively, if the system is set up in decryption mode, an entity controlling the decryption trapdoor $t = y$ would run $\mathsf{TrapKeyGen}$ to output a public key pk while having access to a secret key r that decrypts ciphertexts generated on branch 0, and a secret key r/y that decrypts ciphertexts generated on branch 1. If the system is set up in messy mode, an entity controlling the messy trapdoor $t = (x_0, x_1)$ would run $\mathsf{FindMessy}$ upon receiving a public key pk, and leverage x_0, x_1 to determine which of the two branches may not be decrypted by the respective secret key sk (if any is defined). The two modes of operation are computationally indistinguishable, as $\mathsf{SetupMessy}$ outputs a non-DH tuple, whereas $\mathsf{SetupDec}$ outputs a DH tuple.

3.5.2 Instantiating $\mathcal{F}_{\mathsf{OT}}$ from Dual-Mode Encryption

Dual-mode encryption allows one to realize oblivious transfer in a straight-forward manner. Intuitively, the receiver generates the key pair by specifying a decryption branch matching the choice bit σ, whereas the sender encrypts input x_i on encryption branch i, for $i \in \{0, 1\}$. This way, the receiver will be able to decrypt only one of the two inputs (and in particular, x_σ). This generic construction was presented in [PVW08], and is depicted pictorially in Figure 3.9.

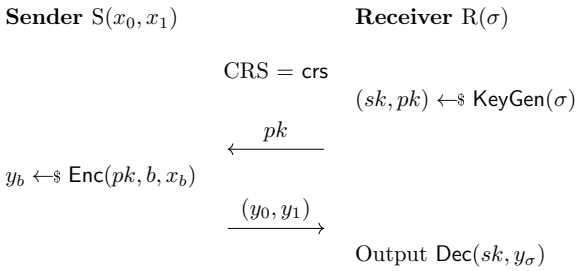


Figure 3.9: The generic framework for $\mathcal{F}_{\mathsf{OT}}$ of [PVW08].

Theorem 3.2. *The construction in Figure 3.9 UC-realizes $\mathcal{F}_{\mathsf{OT}}$ in the $\mathcal{F}_{\mathsf{crs}}$-hybrid model in the presence of a malicious party. Additionally, if the system is initialized in messy mode, the sender has statistical security and the receiver has computational security, whereas in decryption mode the security properties are mirrored.*

Proof (Sketch). We briefly outline how $\mathcal{S}$ extracts the inputs of each corruption scenario by leveraging the trapdoors obtained by generating the CRS. The full proof of this theorem may be found in [PVW08].

Malicious sender. In this case, $\mathcal{S}$ should simulate the behaviour of the honest receiver to the eyes of $\mathcal{A}$ without having access to the choice bit σ. Moreover, $\mathcal{S}$ should forward to $\mathcal{F}_{\mathsf{OT}}$ the correct inputs of the malicious sender (x'_0, x'_1) by observing the behaviour of $\mathcal{A}$ in the protocol run.

To do that, $\mathcal{S}$ sets up the CRS of the system in decryption mode. This means that $\mathcal{S}$ will be able to decrypt ciphertexts on *both* encryption branches. After that, it simulates the honest receiver by picking a random $\sigma \in \{0, 1\}$ for the KeyGen algorithm, and forwards pk to $\mathcal{A}$. Upon receiving two ciphertexts (y'_0, y'_1) from $\mathcal{A}$, $\mathcal{S}$ exploits the decryption mode of the CRS to decrypt *both* ciphertexts, retrieving inputs (x'_0, x'_1). Now $\mathcal{S}$ can simply forward the correct (x'_0, x'_1) to $\mathcal{F}_{\mathsf{OT}}$.

Malicious receiver. In this case, $\mathcal{S}$ should simulate the behaviour of the honest sender to the eyes of $\mathcal{A}$ without having access to its inputs (x_0, x_1). Moreover, $\mathcal{S}$ should forward to $\mathcal{F}_{\mathsf{OT}}$ the correct inputs of the malicious sender σ by observing the behaviour of $\mathcal{A}$ in the protocol run.

In this case, $\mathcal{S}$ sets up the CRS of the system in messy mode. This means that, $\mathcal{S}$ will *always* be able to extract a messy encryption branch from any (possibly malformed) public key pk (*i.e.*, a branch for which the owner of pk will be unable to decrypt any ciphertext).

Upon receiving some pk' from $\mathcal{A}$, $\mathcal{S}$ exploits the messy mode of the CRS to retrieve a messy branch b. With that, it forwards $1-b$ as input to $\mathcal{F}_{\mathsf{OT}}$, which returns an input x_{1-b} (recall that the sender is a dummy party who forwarded (x_0, x_1) to $\mathcal{F}_{\mathsf{OT}}$). Now, $\mathcal{S}$ simply encrypts x_{1-b} on branch $1-b$ (which can be decrypted by $\mathcal{A}$) and the all-zero string on branch b. □

3.5.3 The multi-session oblivious transfer ideal functionality $\hat{\mathcal{F}}_{\mathsf{OT}}$

An interesting property of the protocol of Figure 3.9 is that it allows one to reuse the same CRS across multiple protocol runs. In particular, instead of showing that the protocol realizes the $\mathcal{F}_{\mathsf{OT}}$ functionality, one can prove that it realizes the *multi-session* oblivious transfer functionality $\hat{\mathcal{F}}_{\mathsf{OT}}$, which we define in what follows.

Functionality $\hat{\mathcal{F}}_{\mathsf{OT}}$

Upon receiving a query $(\text{SENDER}, \mathsf{sid}, \mathsf{ssid}, (x_0, x_1))$ **from** S**:**

Record $(\text{SENDER}, \mathsf{sid}, \mathsf{ssid}, (x_0, x_1))$. Ignore subsequent commands of the form $(\text{SENDER}, \mathsf{sid}, \mathsf{ssid}, \cdot)$.

Upon receiving a query $(\text{RECEIVER}, \mathsf{sid}, \mathsf{ssid}, \sigma)$ **from** R**:**

Check if a record $(\text{SENDER}, \mathsf{sid}, \mathsf{ssid}, (x_0, x_1))$ exists. If so, output $(\mathsf{sid}, \mathsf{ssid}, x_\sigma)$ to R, $(\mathsf{sid}, \mathsf{ssid})$ to the adversary $\mathcal{S}$. Otherwise, send nothing to R but continue running.

Figure 3.10: Description of the multi-session oblivious transfer functionality $\hat{\mathcal{F}}_{\mathsf{OT}}$.

The main novelty of multi-session functionalities lies in the introduction of a sub-session id ssid as an additional parameter to each query. Intuitively, $\hat{\mathcal{F}}_{\mathsf{OT}}$ behaves as $\mathcal{F}_{\mathsf{OT}}$, but queries are separated by observing the overall SID $(\mathsf{sid}, \mathsf{ssid})$.

Part II

PAKE From Oblivious Transfer

Part Overview

In the previous sections, we summarized the formal tools required to capture basic cryptographic primitives, as well as the framework of secure multi-party computation. For the latter, we focussed our attention on handling eavesdroppers and malicious parties in the presence of authenticated communication channels. Unfortunately, these "traditional" models might still be unsatisfactory in capturing certain cryptographic primitives and real-world settings. For instance, it makes little sense to design an Authenticated Key Exchange protocol while already pre-assuming the existence of an authenticated channel between parties. Likewise, assuming that the real world has "honest parties" running the protocol exactly as per their theoretical specification might be unrealistic.

In this portion of the composition, we first summarize modelling extensions that weaken the aforementioned trust assumptions, as well as the background required to formally capture Password-Authenticated Key Exchange. Then, we shift our attention towards presenting our first universally composable subversion-resilient PAKE protocol, which provides formal security guarantees even if one of the parties is malicious while the remaining honest party's device is subject to some undetectable corruption.

Organization. In Section 4, we convey relevant background required for the design of our first srUC PAKE protocol. More specifically, we first explicitly characterize communication channels as ideal functionalities in the context of the UC framework, discussing the implications of carrying out secure MPC without authenticated channels by briefly recapping the notion of "split authentication" due to [BCL+05]. We then describe PAKE as an ideal functionality, presenting the main ideas behind the UC-secure PAKE-from-OT protocol due to [CDVW12]. Finally, we discuss the subversion-resilient UC (srUC) framework of [CMNV22], which extends the UC framework in order to capture undetectable subversion attacks by leveraging cryptographic reverse firewalls.

In Section 5, we provide a complete formal treatment of our first srUC PAKE protocol, which focuses on sanitizing the PAKE-from-OT protocol due to [CDVW12].

4 Background

In this section, we introduce essential background that will be used within the formalization of our first subversion-resilient PAKE protocol, such as the modelling of communication channels, PAKE, and undetectable subversion attacks.

In Section 4.1, we explicitly model communication channels within the UC framework by means of appropriate ideal functionalities, placing particular emphasis on the unauthenticated setting by summarizing the notion of split authentication due to [BCL+05]. In Section 4.2, we characterize PAKE as an ideal functionality, presenting the PAKE-from-OT protocol of [CDVW12]. In Section 4.3, we summarize the main technical novelties introduced by the srUC framework with respect to traditional UC.

4.1 Communication channels as functionalities

Until now, we treated channel assumptions as being external to our model. However, while moving to practical deployments, one has to ensure that these channels can actually be instantiated. One way to do so in the UC framework consists of modelling channels as ideal functionalities. This way, one can first construct a protocol that UC-realizes an ideal functionality capturing some channel assumption, and then design protocols relying on that specific channel assumption by invoking the respective functionality.

In this subsection, we first describe authenticated communication as an ideal functionality, highlighting that such a functionality cannot be instantiated in the bare model (*i.e.*, without relying on any setup assumption). We then present split authentication: the best possible security that can be obtained without relying on any setup assumptions (and may therefore be realized in the unauthenticated setting). Finally, we mention a generic transformation due to [BCL+05], which allows one to replace authenticated channels with split-authenticated channels in suitable protocols.

4.1.1 Authenticated channels

We start by considering the ideal functionality for authenticated message transmission $\mathcal{F}_{\mathsf{MAUTH}}$ of [BCL+05], reported in Figure 4.1. Intuitively, this functionality allows parties to exchange messages that are revealed to the adversary. Notably, the adversary may *exclusively* control the delivery timing of the messages without altering them or introducing any of its own.

Functionality $\mathcal{F}_{\mathsf{MAUTH}}$

Initialization

Upon receiving query $(\textsc{Init}, \mathsf{sid})$, **where** $\mathsf{sid} = (\mathcal{P}, \mathsf{sid}')$**:**

Record $\mathcal{P}$ and forward it to the adversary $\mathcal{S}$.

Message authentication

Upon receiving query $(\textsc{Send}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', m)$, **where** $\mathsf{P}, \mathsf{P}' \in \mathcal{P}$**:**

Send $(\mathsf{P}, \mathsf{P}', m)$ to $\mathcal{S}$ and add $(\mathsf{P}, \mathsf{P}', m)$ to an (initially empty) list $\mathcal{W}$ of waiting messages. The same triple can appear multiple times in the list.

Upon receiving query $(\textsc{Deliver}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', m)$ **from** $\mathcal{S}$**:**

- If $\mathsf{P} \in \mathcal{P}$ is uncorrupted, then: if there is a triple $(\mathsf{P}, \mathsf{P}', m) \in \mathcal{W}$, remove one appearance of the triple from $\mathcal{W}$ and output $(\textsc{Receive}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', m)$ to P'.
- Else, if $\mathsf{P} \in \mathcal{P}$ and P is corrupted, output $(\textsc{Receive}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', m)$ to P' regardless of $\mathcal{W}$.

In any other case, do nothing.

Figure 4.1: Description of the message-authentication functionality $\mathcal{F}_{\mathsf{MAUTH}}$ [BCL+05].

Description. The initialization step binds the set of parties $\mathcal{P}$ that will be able to exchange messages in an authenticated fashion. The adversary is notified of the set of parties that will be involved in the interaction. After that, any party in $\mathcal{P}$ can send messages to any other party in $\mathcal{P}$ by means of a SEND query. Each message is placed in a waiting list $\mathcal{W}$, and is ultimately delivered to an honest party whenever $\mathcal{S}$ allows the delivery by means of a DELIVER query. If a sender P is corrupted, $\mathcal{S}$ can deliver any message to its peer P' irrespectively of $\mathcal{W}$.

Instantiation attempt: using exclusively digital signatures. Let us now attempt to instantiate $\mathcal{F}_{\mathsf{MAUTH}}$ in the bare model (*i.e.*, without any setup assumption). For simplicity, let us discuss the two-party setting. In Section 2.5, we introduced digital signatures as a tool to allow a party to authenticate its messages. By following the blueprint used to handle eavesdropping (Figure 3.2), one might expect to obtain message authenticity in the presence of a fully insecure communication channel just by leveraging EUF-CMA security, obtaining a protocol as in Figure 4.2.

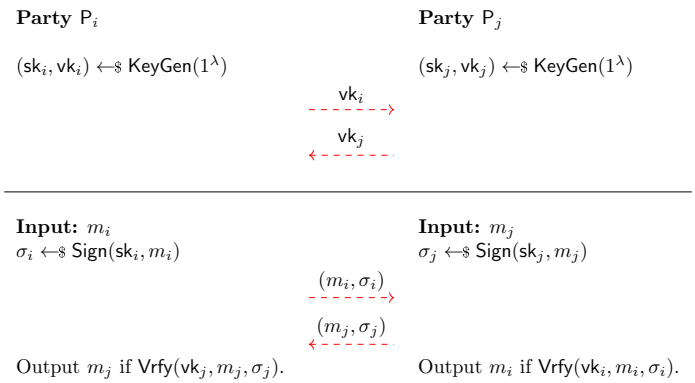


Figure 4.2: Two parties exchanging an authentic message via an insecure channel (attempt).

Unfortunately, this approach fails to achieve authenticity. To be concrete, we discuss how the adversary can impersonate P_i and attack P_j by making it output some message $\overline{m}_i \neq m_i$. Even though the protocol is not vulnerable to existential forgeries against P_i's original verification key vk_i, $\mathcal{A}$ could intercept vk_i flowing onto the channel and replace it with its own $\overline{\mathsf{vk}}_i$. After that, the adversary simply drops the tuple (m_i, σ_i) sent by P_i and forges an accepting signature for $\overline{m}_i$ (by using $\overline{\mathsf{sk}}_i$), forwarding $(\overline{m}_i, \overline{\sigma}_i)$ to P_j.

By using the above strategy, the adversary can impersonate each party's peer, effectively executing two protocol runs in parallel. We depict the result in Figure 4.3.

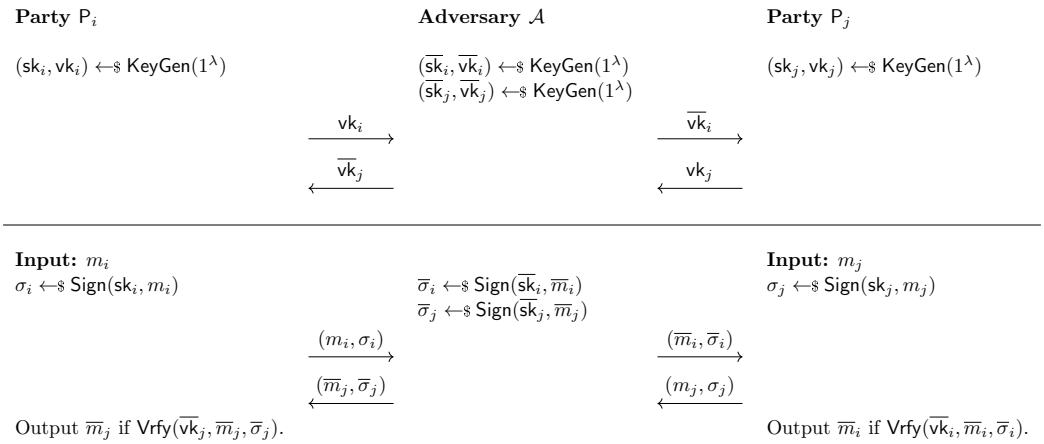


Figure 4.3: A man-in-the-middle attack to the protocol of Figure 4.2.

An impossibility in the bare model. As hinted above, designing protocols without any setup whatsoever exposes us to a new set of challenges, given that the adversary can always mount a man-in-the-middle attack. Indeed, [Can03] shows that it is impossible to UC-realize $\mathcal{F}_{\mathsf{MAUTH}}$ in the bare model, while exhibiting an instantiation that assumes the existence of a (trusted) PKI.

4.1.2 Split authentication

The implications of the previous subsection are dire: without any setup the adversary can always run impersonation attacks. In light of that, the next best possible security guarantee one can hope to obtain lies in the notion of "split authentication", as defined in [BCL+05]. In particular, the only additional attack an adversary may mount by exploiting the lack of setup assumptions consists of partitioning parties into different execution sets *before* they authenticate. For instance, in the two-party setting, the adversary either allows parties to mutually authenticate, or it impersonates each party's peer by performing a man-in-the-middle (as shown in Figure 4.4). This way, even though honest parties could be running the entire protocol with the adversary without even noticing it, they can rest assured that they will complete the protocol run while interacting with the same entity.

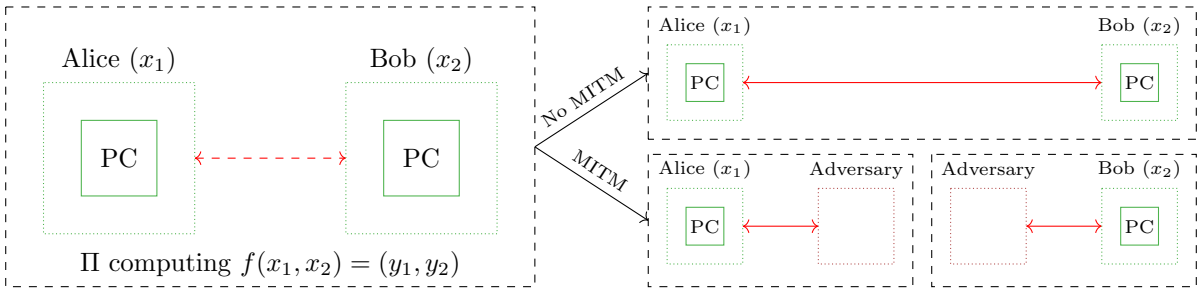


Figure 4.4: Adversarial capabilities in the unauthenticated setting in the presence of split authentication among two honest parties.

[BCL+05] formalizes this notion with the ideal functionality for *split-authenticated* communication $\mathcal{F}_{\mathsf{SA}}$, which we report in what follows.

Functionality $\mathcal{F}_{\mathsf{SA}}$

Initialization

Upon receiving query $(\textsc{Init}, \mathsf{sid})$, **where** $\mathsf{sid} = (\mathcal{P}, \mathsf{sid}')$**:**

Record $\mathcal{P}$ and forward it to $\mathcal{S}$.

Upon receiving query $(\textsc{Init}, \mathsf{sid}, \mathsf{P}, H, \mathsf{sid}_H)$ **from** $\mathcal{S}$**:**

Verify that party $\mathsf{P} \in \mathcal{P}$, that the list H of party identities includes P, and that for all recorded sets H', either $H \cap H'$ contains only corrupted parties and $\mathsf{sid}_H \neq \mathsf{sid}_{H'}$, or $H' = H$ and $\mathsf{sid}_H = sid_{H'}$. If so, output $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_H)$ to P and record (H, sid_H) if not yet recorded. Else, do nothing.

Message authentication

Upon receiving query $(\textsc{Send}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', m)$, **where** $\mathsf{P} \in \mathcal{P}$**:**

Send $(\mathsf{P}, \mathsf{P}', m)$ to $\mathcal{S}$ and add $(\mathsf{P}, \mathsf{P}', m)$ to an (initially empty) list $\mathcal{W}$ of waiting messages. The same triple can appear multiple times in the list.

Upon receiving query $(\textsc{Deliver}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', m)$ **from** $\mathcal{S}$**:**

- If P' did not previously receive an $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_H)$ output then do nothing.
- Else, if P is in the authentication set H of P', and P is uncorrupted, then: if there is a triple $(\mathsf{P}, \mathsf{P}', m) \in$

$\mathcal{W}$, remove one appearance of the triple from $\mathcal{W}$ and output $(\text{RECEIVE}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', m)$ to P'. Otherwise do nothing.

- Else (i.e., P' received $(\text{INIT}, \mathsf{sid}, \mathsf{sid}_H)$, and either P is corrupted or $\mathsf{P} \notin H$), output $(\text{RECEIVE}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', m)$ to P', regardless of $\mathcal{W}$.

Figure 4.5: The split message-authentication functionality $\mathcal{F}_{\mathsf{SA}}$ [BCL+05].

Description. The $\mathcal{F}_{\mathsf{SA}}$ functionality runs exactly as the $\mathcal{F}_{\mathsf{MAUTH}}$ functionality, except for the initialization of the channels. If the adversary decides to place two parties in the same authentication set H, then they will be able to exchange subsequent messages as if they had access to an authenticated channel. Otherwise, each party would be able to exchange authenticated messages with the adversary (*i.e.*, the adversary would impersonate each party's peer).

An instantiation of $\mathcal{F}_{\mathsf{SA}}$. [BCL+05] shows a protocol that UC-realizes $\mathcal{F}_{\mathsf{SA}}$, assuming $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ to be a EUF-CMA signature scheme. Given that we restrict our attention on PAKE, a 2-party functionality, we show in Figure 4.6 the protocol of [BCL+05] cast to the 2-party setting for a cleaner presentation. The n-party protocol runs the same, except that in the link initialization phase party P_i determines its session ID by collecting n verification keys and n signatures.

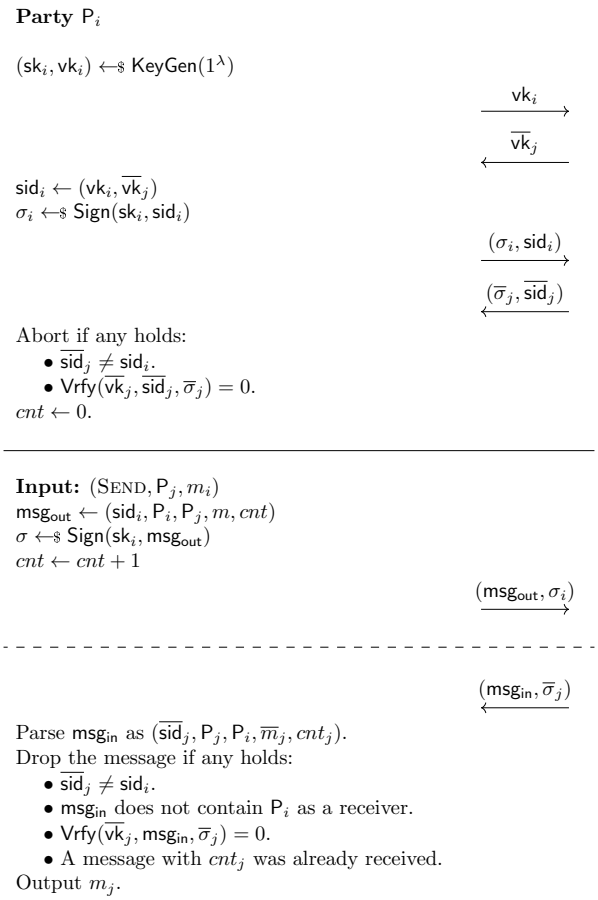


Figure 4.6: Session-authenticated channel of [BCL+05]. On the top half, the setup phase. On the bottom half, the message authentication phase, depicting sending a message, and receiving and verifying an inbound message.

Description. The intuition behind this protocol finds similarities to the attempt we made in Figure 4.2. In the initialization phase, party P_i locally generates a key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ and sends vk_i to its peer. Upon receiving some verification key $\overline{\mathsf{vk}}_j$ (either their peer's or the adversary's), P_i forwards to its peer a signature on $\mathsf{sid}_i = (\mathsf{vk}_i, \overline{\mathsf{vk}}_j)$. Intuitively, this signature represents an acknowledgement that party P_i will be exchanging messages with some entity identified by a verification key $\overline{\mathsf{vk}}_j$ (again, either P_j or the adversary). Upon receiving some signature $\overline{\sigma}_j$ on some SID $\overline{\mathsf{sid}}_j$, P_i checks whether the other entity has acknowledged the same set of parties (by verifying the signature). If so, the initialization phase is complete, and P_i will be able to exchange messages in an authenticated manner with the entity identified by $\overline{\mathsf{vk}}_j$.

Split functionalities. Traditional ideal functionalities are designed with authenticated channels in mind, and may therefore be hard (if not impossible) to realize without any setup. For instance, the notion of "honest majority" stops making any sense, alongside all related feasibility results leveraging this property. For that, [BCL+05] additionally coins the notion of a "split" ideal functionality $s\mathcal{F}$, which empowers the adversary to disconnect parties before they actually run the ideal functionality $\mathcal{F}$ (reported below).

Split functionality $s\mathcal{F}$

$s\mathcal{F}$ is a *split* ideal functionality of $\mathcal{F}$ that interacts with an adversary $\mathcal{S}$ and a set of parties, and behaves as follows.

Initialization

- Upon activation with input $(\text{INIT}, \mathsf{sid})$ from party P, parse $\mathsf{sid} = (\mathcal{P}, \mathsf{sid}')$ where $\mathcal{P}$ is a set of parties that includes P. Forward $(\text{INIT}, \mathsf{sid}, \mathsf{P})$ to the adversary $\mathcal{S}$.
- Upon receiving the message $(\text{INIT}, \mathsf{sid}, \mathsf{P}, H, \mathsf{sid}_H)$, from $\mathcal{S}$: Verify that party $H \subseteq \mathcal{P}$, that the list H of party identities includes P, and that for all recorded sets H', either (i) $H \cap H'$ contains only corrupted parties and $\mathsf{sid}_H \neq \mathsf{sid}_{H'}$, or (ii) $H' = H$ and $\mathsf{sid}_H = \mathsf{sid}_{H'}$. If any check fails, do nothing. Otherwise, record $(\text{INIT}, \mathsf{sid}, \mathsf{sid}_H)$, output the tuple to P, and initialize an instance of the ideal functionality $\mathcal{F}$ with SID sid_H, denoted as $\mathcal{F}_H$. The adversary plays the role of parties $\mathcal{P} - H$ in $\mathcal{F}_H$.

Computation

- Upon receiving an input $(\text{INPUT}, \mathsf{sid}, v)$ from party $\mathsf{P} \in \mathcal{P}$, find the set H such that $\mathsf{P} \in H$, and forward the message v from P to $\mathcal{F}_H$. If no such set is found, ignore the input.
- Upon receiving a message $(\text{INPUT}, \mathsf{sid}, H, \mathsf{P}, v)$ from $\mathcal{S}$, if $\mathcal{F}_H$ is initialized and $\mathsf{P} \in \mathcal{P} - H$, forward v to $\mathcal{F}_H$ as if coming from party P. Otherwise, ignore the message.
- When an instance $\mathcal{F}_H$ generates an output v for party $\mathsf{P} \in H$, output v to P. When the output is for a party $\mathsf{P} \in \mathcal{P} - H$ or for $\mathcal{S}$, send the output to $\mathcal{S}$.

Figure 4.7: The generic split functionality $s\mathcal{F}$ [BCL+05].

Description. Let us consider again the two-party setting for simplicity. Before the protocol run, parties are either placed in the same authentication set H (*i.e.*, they are allowed to mutually authenticate) or not (*i.e.*, the adversary impersonates each party's peer). In the former case, the parties run the functionality $\mathcal{F}$ on authenticated channels. In the latter case, each party participates in a separate run of $\mathcal{F}$ where the adversary plays the role of its peer.

From authenticated to unauthenticated channels. One central result of [BCL+05] shows a generic transformation for which any protocol UC n-realizing some n-party functionality $\mathcal{F}$ relying on authenticated channels ($\mathcal{F}_{\mathsf{MAUTH}}$) can be compiled into a similar protocol that UC-realizes the split functionality $s\mathcal{F}$, but now just relying on split-authenticated channels ($\mathcal{F}_{\mathsf{SA}}$). Note that this transformation only makes sense if the protocol relying on authenticated channels *multi-realizes* the functionality $\mathcal{F}$, *i.e.*, it retains security even if the same setup assumption is shared across multiple runs. Concretely, in a two-party protocol that relies on a CRS in the presence of unauthenticated channels, the same CRS will be available to both parties

irrespectively of the partition operated from the adversary. In particular, if the adversary partitions the parties, there will be two protocol runs in parallel relying on the same setup assumption.

4.2 Password-Authenticated Key Exchange

The essence of (balanced[15]) PAKE consists of allowing two peers to mutually authenticate and derive a high-entropy key from some (possibly low-entropy) password, as long as both parties supply the same password throughout the protocol run. This high-entropy key can later be used to establish an authenticated (or a secure) channel (*e.g.*, by using symmetric-key cryptography).

Given that passwords are considered to be low-entropy, the security definition of PAKE must take into account the fact that the adversary can guess the password with non-negligible probability. In particular, a malicious party may attempt to authenticate against an honest party without knowing its password. Thus, a protocol realizing PAKE is secure if no adversary is able to break it with probability better than guessing the password outright.

In this subsection, we formally characterize PAKE as an ideal functionality. In particular, we focus on *balanced* PAKE *with implicit mutual authentication*[16]. As a stepping stone, we first present the randomized equality ideal functionality $\mathcal{F}_{\mathsf{RE}}$, which captures PAKE in the presence of authenticated channels, and show how this primitive can be instantiated from oblivious transfer by summarizing the protocol due to [CDVW12]. Then, we exhibit the PAKE ideal functionality $\mathcal{F}_{\mathsf{PAKE}}$, and discuss how one can apply the generic transformation of split functionalities described in the previous subsection to transform the aforementioned protocol for randomized equality into a protocol for PAKE. Looking ahead, this strategy will be the core foundation for the design of our first srUC PAKE protocol.

4.2.1 The Randomized Equality functionality $\mathcal{F}_{\mathsf{RE}}$

PAKE with authenticated channels essentially boils down to performing equality checking between the inputs (passwords) supplied by two parties. In particular, these parties output the same high-entropy key only if they have supplied the same password; otherwise, they receive two unrelated and uniformly distributed keys. In what follows, we describe a variation of the randomized equality ideal functionality $\mathcal{F}_{\mathsf{RE}}$ of [CDVW12], with technical improvements from [DHP+18].

Functionality $\mathcal{F}_{\mathsf{RE}}$

The functionality $\mathcal{F}_{\mathsf{RE}}$ is parameterized by a security parameter λ. It interacts with an initiator I, a responder R, and the adversary $\mathcal{S}$ via the following messages:

Upon receiving a query $(\textsc{NewSession}, \mathsf{sid}, \mathrm{I}, \mathrm{R}, \mathsf{pw}^{\mathrm{I}})$, **from** I**:**

Record $(\mathrm{I}, \mathrm{R}, \mathsf{pw}^{\mathrm{I}})$ and send a message $(\mathsf{sid}, \mathrm{I}, \mathrm{R})$ to $\mathcal{S}$. Ignore all future messages from I.

Upon receiving a query $(\textsc{ok}, \mathsf{sid})$ **from** $\mathcal{S}$**:**

Send a message $(\textsc{wakeup}, \mathsf{sid}, \mathrm{I}, \mathrm{R})$ to R. Ignore all future $(\textsc{ok})$ messages.

Upon receiving a query $(\textsc{Respond}, \mathsf{sid}, \mathrm{I}, \mathrm{R}, \mathsf{pw}^{\mathrm{R}})$ **from** R**:**

- If $\mathsf{pw}^{\mathrm{R}} = \mathsf{pw}^{\mathrm{I}}$, choose $\mathsf{skey} \leftarrow\!\!\$\ \{0,1\}^{\lambda}$ and store $\mathsf{skey}_{\mathrm{I}} = \mathsf{skey}_{\mathrm{R}} = \mathsf{skey}$.
- If $\mathsf{pw}^{\mathrm{R}} \neq \mathsf{pw}^{\mathrm{I}}$, then set $\mathsf{skey}_{\mathrm{I}} \leftarrow\!\!\$\ \{0,1\}^{\lambda}$, $\mathsf{skey}_{\mathrm{R}} \leftarrow\!\!\$\ \{0,1\}^{\lambda}$.

In both cases, ignore subsequent inputs from R.

[15] In balanced PAKE, parties mutually authenticate by both supplying the same password. In augmented PAKE, one party (usually a client) has knowledge of a password, whereas its peer (usually a server) has only knowledge of some one-way function of the password (*e.g.*, a hash). While the latter approach also finds natural applications, we do only focus on balanced PAKE within this composition.

[16] In PAKE with implicit mutual authentication, parties always get an output key, but they cannot ascertain whether they got the same key. With explicit mutual authentication, parties either get the same key or a special symbol signalling the authentication failure. One may obtain the latter notion from the former by adding explicit key confirmation steps after the protocol's conclusion, or by using higher-level protocols (*e.g.*, successfully exchanging messages in secure message transmission).

Upon receiving a query $(\textsc{NewKey}, \mathsf{sid}, \mathsf{P}, K)$, $\mathsf{P} \in \{\mathrm{I}, \mathrm{R}\}$ **from** $\mathcal{S}$**:**

- If any of the following conditions hold, output (sid, K) to party P:
  - P is corrupted.
  - $\mathsf{pw}^{\mathrm{I}} = \mathsf{pw}^{\mathrm{R}}$, and the peer of P is corrupted.
- Otherwise, output $(\mathsf{sid}, \mathsf{skey}_{\mathsf{P}})$ to party P.

Ignore all subsequent $(\textsc{NewKey}, \mathsf{P})$ queries for the same party P.

Figure 4.8: The randomized equality functionality $\mathcal{F}_{\mathsf{RE}}$.

Description. $\mathcal{F}_{\mathsf{RE}}$ features an initiator I and a responder R, who obtain the same random high-entropy value skey only if both parties have input the same password. More specifically, the initiator declares its intention to run the functionality with the responder by means of a NewSession query with password pw^{I}. The adversary $\mathcal{S}$ is notified of this interaction, and controls the delivery timing of the initialization message destined to the receiver. Whenever this occurs, the responder inputs its own password pw^{R} with a Respond query, and the functionality prepares the output for both parties depending on whether $\mathsf{pw}^{\mathrm{R}} = \mathsf{pw}^{\mathrm{I}}$ or not. In the former case, the output will be the same random key skey for both parties; in the latter, two keys sampled independently at random. Finally, the adversary controls the delivery of the output of the functionality separately for each party by means of a NewKey query, in which it also specifies a key K. If party P is corrupted, or the adversary had input party P's password on behalf of its corrupted peer (*i.e.*, it has successfully guessed the password of honest P), P receives the adversarially-generated key K. In any other case, party P receives whatever has been stored in $\mathsf{skey}_{\mathsf{P}}$.

4.2.2 A protocol realizing $\mathcal{F}_{\mathsf{RE}}$ in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model

In what follows, we discuss the protocol UC-realizing $\mathcal{F}_{\mathsf{RE}}$ due to [CDVW12]. For ease of exposition, we show a protocol run using a 1-bit password without considering explicit mutual authentication.

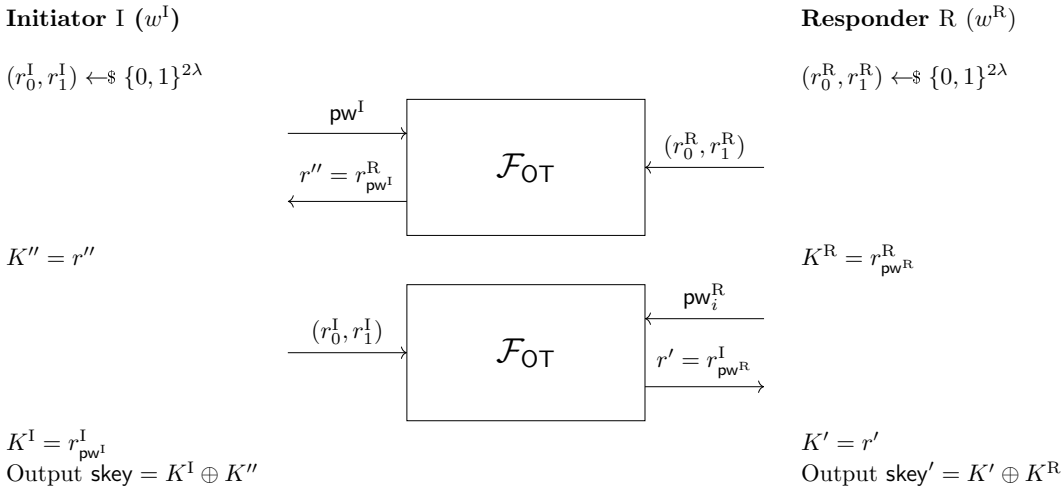


Figure 4.9: $\mathcal{F}_{\mathsf{RE}}$ from OT [CDVW12] with a 1-bit password.

Description. Let us focus on the first run of $\mathcal{F}_{\mathsf{OT}}$. Here, the responder inputs two random strings $(r_0^{\mathrm{R}}, r_1^{\mathrm{R}})$ whose length matches the security parameter, whereas the initiator inputs its 1-bit password pw^{I}. From the $\mathcal{F}_{\mathsf{OT}}$ run, the initiator obtains exclusively the random string $r^{\mathrm{R}}_{\mathsf{pw}^{\mathrm{I}}}$. This string is interpreted by the initiator

as the responder's contribution to the final session key. On the other hand, the responder selects the random string indexed by its password pw^{R} as its contribution to the final session key. Intuitively, in order for the parties to derive the same key, they need to select the same random value. But, by security of $\mathcal{F}_{\mathsf{OT}}$, this occurs only if $\mathsf{pw}^{\mathrm{I}} = \mathsf{pw}^{\mathrm{R}}$.

Unfortunately, a single OT run is not sufficient to securely instantiate $\mathcal{F}_{\mathsf{RE}}$, as a malicious responder would be able to fully determine the value of the final session key (even without knowing its peer's password!). This may be achieved, *e.g.*, by having a malicious responder setting $r_0^{\mathrm{R}} = r_1^{\mathrm{R}}$. For that, the parties engage in a second run of $\mathcal{F}_{\mathsf{OT}}$ with reversed roles by performing exactly the same procedure. Each party computes the final session key by XORing the two random values, foiling the above attack.

Extension to n-bit passwords. In order to obtain a randomized equality protocol for n-bit passwords, it suffices for parties to run the above procedure for every bit of the password. The final session key is derived by XORing the keys obtained from all the above runs.

4.2.3 The PAKE functionality $\mathcal{F}_{\mathsf{PAKE}}$

As remarked before, $\mathcal{F}_{\mathsf{RE}}$ captures PAKE in the presence of authenticated communication. By removing this assumption, we need to adapt the description of $\mathcal{F}_{\mathsf{RE}}$ to account for the lack of authentication between parties. Concretely, in a similar fashion to split authentication, each honest party may end up running the PAKE protocol with the adversary without realizing it.

From a formal standpoint, this additional adversarial capability is captured by providing the adversary with the possibility to perform one *online* password guess against each party. This capability stems from the possibility for the adversary to guess each party's password by impersonating its peer in a protocol run.

In what follows, we provide a formal description of the $\mathcal{F}_{\mathsf{PAKE}}$ functionality that follows the style of [CDVW12] while embedding minor variations to achieve consistency with the previously-described $\mathcal{F}_{\mathsf{RE}}$, as well as technical improvements from [DHP+18]. More specifically, the behaviour of $\mathcal{F}_{\mathsf{PAKE}}$ is conceptually close to that of $\mathcal{F}_{\mathsf{RE}}$, with the important difference that the adversary is now allowed to perform (online) password guesses in order to influence the keys output by the functionality.

Functionality $\mathcal{F}_{\mathsf{PAKE}}$

The functionality $\mathcal{F}_{\mathsf{PAKE}}$ is parameterized by a security parameter λ, and it interacts with an initiator I, a responder R, and the adversary $\mathcal{S}$ via the following queries:

Upon receiving a query $(\textsc{NewSession}, \mathsf{sid}, \mathrm{I}, \mathrm{R}, \mathsf{pw}^{\mathrm{I}})$ **from** I**:**

Record $(\mathrm{I}, \mathrm{R}, \mathsf{pw}^{\mathrm{I}})$, mark it as fresh, and leak $(\mathsf{sid}, \mathrm{I}, \mathrm{R})$ to $\mathcal{S}$. Ignore all future messages from I.

Upon receiving a query $(\textsc{ok}, \mathsf{sid})$ **from** $\mathcal{S}$**:**

Send a message $(\textsc{wakeup}, \mathsf{sid}, \mathrm{I}, \mathrm{R})$ to R. Ignore all future $(\textsc{ok})$ messages.

Upon receiving a query $(\textsc{Respond}, \mathsf{sid}, \mathrm{I}, \mathrm{R}, \mathsf{pw}^{\mathrm{R}})$ **from** R**:**

Record $(\mathrm{R}, \mathrm{I}, \mathsf{pw}^{\mathrm{R}})$ and mark it as fresh.

Upon receiving a query $(\textsc{TestPwd}, \mathsf{sid}, \mathsf{P}, \widetilde{\mathsf{pw}})$ **from the adversary** $\mathcal{S}$**:**

If $\mathsf{P} \in \{\mathrm{I}, \mathrm{R}\}$ and there exists a record of the form $(\mathsf{P}, \cdot, \mathsf{pw})$ which is fresh, then:

- If $\widetilde{\mathsf{pw}} = \mathsf{pw}$, mark the record as $\mathsf{compromised}$ and return "correct guess" to $\mathcal{S}$.
- If $\widetilde{\mathsf{pw}} \neq \mathsf{pw}$, mark the record as $\mathsf{interrupted}$ and return "wrong guess" to $\mathcal{S}$.

Upon receiving a query $(\textsc{NewKey}, \mathsf{sid}, \mathsf{P}_i, K)$ **from** $\mathcal{S}$**, where** $|K| = \lambda$**:**

If $\mathsf{P}_i \in \{\mathrm{I}, \mathrm{R}\}$ and there is a record of the form $(\mathsf{P}_i, \mathsf{P}_j, \mathsf{pw}_i)$ that is not marked as $\mathsf{completed}$, with P_j being the peer of P_i, then:

- If any of the following conditions hold, output (sid, K) to party P_i:
  - P_i is corrupted.

– This record is fresh, there exists a record $(\mathsf{P}_j, \mathsf{P}_i, \mathsf{pw}_j)$ with $\mathsf{pw}_i = \mathsf{pw}_j$, and P_j is corrupted.
– This record is compromised.

- If this record is fresh, both parties are honest, and there exists a record $(\mathsf{P}_j, \mathsf{P}_i, \mathsf{pw}_j)$ with $\mathsf{pw}_j = \mathsf{pw}_i$, choose $\mathsf{skey} \leftarrow\!\!\$\ \{0,1\}^\lambda$. Output skey to P_i, and append skey to the record $(\mathsf{P}_i, \mathsf{P}_j, \mathsf{pw}_i)$.
- If this record is fresh, both parties are honest, and there exists a record $(\mathsf{P}_j, \mathsf{P}_i, \mathsf{pw}_j, \mathsf{skey})$ with $\mathsf{pw}_j = \mathsf{pw}_i$, output skey to P_i.
- If none of the above rules apply, choose $\mathsf{skey}' \leftarrow\!\!\$\ \{0,1\}^\lambda$ and output it to party P_i.

In any case, mark the record $(\mathsf{P}_i, \cdot, \mathsf{pw}_i)$ as completed.

Figure 4.10: The PAKE functionality $\mathcal{F}_{\mathsf{PAKE}}$.

Description. Compared to $\mathcal{F}_{\mathsf{RE}}$, each record now has a status associated to it, which is initially set to fresh. If the adversary performs a password guess (by means of a TestPwd query) against party P's fresh record and fails, the target record is marked as interrupted. If instead the password guess succeeds, the target record is marked as compromised. Either way, the adversary is notified of the outcome of its guess. Finally, the adversary controls the delivery of the output of the functionality separately for each party by means of a NewKey query, in which it also specifies a key K. The same (random) key is output to both honest parties only if (i) the input passwords were the same, and (ii) both records were fresh when their respective key was determined (*i.e.*, no password guesses occurred). Otherwise, the functionality outputs the adversarially-generated key K to P if any of the following occurs: (i) P is corrupted, (ii) the adversary does not perform any TestPwd queries but runs the functionality as the peer of P with input the password of P, or (iii) the adversary guesses the password for P via a TestPwd query. In any other case, the functionality outputs a random key to P.

4.2.4 From $\mathcal{F}_{\mathsf{RE}}$ to $\mathcal{F}_{\mathsf{PAKE}}$

In Section 4.1.2, we highlighted that any protocol n-realizing an n-party functionality $\mathcal{F}$ by relying on $\mathcal{F}_{\mathsf{MAUTH}}$ can be transformed into a protocol realizing $s\mathcal{F}$ by relying on $\mathcal{F}_{\mathsf{SA}}$. As a first step towards obtaining a protocol for PAKE, we would like to invoke this transformation onto the $\mathcal{F}_{\mathsf{RE}}$-from-OT protocol described in Section 4.2.1 to replace authenticated channels with split authentication (which *can* be instantiated in the unauthenticated setting). However, in order to do so, one has to argue that the protocol in Figure 4.9 n-realizes $\mathcal{F}_{\mathsf{RE}}$ (*i.e.*, multiple runs of the protocol may reuse the same setup).

For that, [CDVW12, Theorem 2] points out that the protocol in Figure 4.9 also implements the multi-session version of $\mathcal{F}_{\mathsf{RE}}$ as long as it is instantiated with multi-session OT (instead of $\mathcal{F}_{\mathsf{OT}}$), where each new session of $\mathcal{F}_{\mathsf{RE}}$ uses a new invocation of the protocol for multi-session OT. In particular, we can use the protocol for $\hat{\mathcal{F}}_{\mathsf{OT}}$ presented in Section 3.5.2, which in turn may be realized in the $\mathcal{F}_{\mathsf{crs}}$-hybrid model. With that, we can invoke the transformation of [BCL+05], which allows us to obtain a protocol for $s\mathcal{F}_{\mathsf{RE}}$ in the $\mathcal{F}_{\mathsf{crs}}$-hybrid model[17].

All that remains to show is that $\mathcal{F}_{\mathsf{PAKE}}$ can be instantiated from $s\mathcal{F}_{\mathsf{RE}}$ in a trivial fashion. Intuitively this is the case, as the power of the adversary to disconnect parties in $s\mathcal{F}_{\mathsf{RE}}$ can be mapped to TestPwd queries in $\mathcal{F}_{\mathsf{PAKE}}$ by means of a trivial protocol (as shown in [DHP+18]). More specifically, the adversary can participate to the protocol realizing $s\mathcal{F}_{\mathsf{RE}}$ by first partitioning honest parties, and then impersonating their peer by using any arbitrary password. This way, the adversary obtains one password guess against each party, which matches the adversarial capabilities prescribed within $\mathcal{F}_{\mathsf{PAKE}}$. We will exhibit the description of this trivial protocol alongside a formal proof in Section 5.5.

[17] The presence of a setup assumption is expected, as PAKE cannot be UC-realized without setup (as shown in [CHK+05]).

4.3 Subversion-resilient UC with Cryptographic Reverse Firewalls

Next, we provide a brief overview of the subversion-resilient UC framework of [CMNV22], highlighting the main novelties with respect to the traditional UC framework.

4.3.1 Specious corruptions

The first important difference compared to traditional UC consists of allowing the adversary to carry out *subversion attacks* against parties' devices. [CMNV22] focuses on a specific class of subversions denoted as *specious corruptions*, which is characterized by the notion of *black-box testing*. In particular, an efficient distinguisher cannot tell whether a party is implementing the honest protocol or some subverted variant by interacting with it in a black-box manner. Intuitively, a specious corruption can be thought of as a subversion that remains undetectable.

More formally, let C be an entity implementing some protocol Π, and let $\tilde{\mathsf{C}}$ be an entity implementing a subverted implementation of Π that is sampled alongside some (adversarial) auxiliary information a from some efficiently sampleable distribution D. Let T be a PPT algorithm that has non-rewinding black-box access to either of the entities. We say that $\tilde{\mathsf{C}}$ is specious if no PPT test T can determine whether it is interacting with C or $\tilde{\mathsf{C}}$. This is formalized by the following game:

Game $\mathtt{Test}_{\mathsf{T},\mathsf{D}}$	
1	$(\tilde{\mathsf{C}}, a) \leftarrow \mathsf{D}$, where a is an auxiliary string.
2	Sample $b \leftarrow\!\!\$\ \{0,1\}$.
3	If $b = 0$, run T with black-box access to $\tilde{\mathsf{C}}$.
	If $b = 1$, run T with black-box access to C.
4	Let b' be the output of T. Output $b == b'$.

Game 6: Description of the testing game $\mathtt{Test}_{\mathsf{T},\mathsf{D}}$.

Definition 4.1 (Specious subversion [CMNV22])**.** *Let* D *be an efficiently sampleable distribution. We say* D *is computationally specious if, for all PPT tests* T*, the following holds:*

$$\Pr[\mathtt{Test}_{\mathsf{T},\mathsf{D}} = 1] \leq 1/2 + \mathsf{negl}(\lambda).$$

Remark. *In contrast with all the other security games presented within this composition, the security game is actually set up by the adversary against a (honest) test* T*! Indeed, the auxiliary string* a *is some additional piece of information that is exclusively accessible to the adversary whenever the subversion is initialized.*

A concrete example. [CMNV22, Lemma 1] considers a C implementing a protocol Π that outputs fresh uniform random strings. A specious $\tilde{\mathsf{C}}$ could be one that implements the following protocol: (i) hard-wire a key k for a pseudo-random permutation P_k; (ii) initialize a random value x, which acts as a counter; (iii) output $P_k(x)$ and add 1 to the counter. The respective specious distribution D would sample k at random and output $(\tilde{\mathsf{C}}, a = k)$, where a is revealed to the adversary. Without knowledge of k, the strings output by the specious core are indistinguishable from random. However, the adversary (who has knowledge of k) can determine that the implementation is subverted by querying $\tilde{\mathsf{C}}$ twice, retrieving (y_1, y_2), and checking whether $P_k^{-1}(y_2) - P_k^{-1}(y_1) = 1$.

Why is a specious distribution required? As stressed in [CMNV22], introducing a specious distribution D that is quantified *after* the test is necessary to avoid trivializing the notion of specious corruption, which is crucially defined over *all* possible tests. In the example above, one could define a test for each value of k which attempts to follow the adversarial strategy. Hence, a test successfully detecting the subversion will always exist. Instead, by asking that $\tilde{\mathsf{C}}$ is sampled from a distribution D, no such test can exist, as T is quantified *before* $\tilde{\mathsf{C}}$ and cannot predict the value k in advance (except with negligible probability).

Specious environments and specious adversaries. The srUC framework works in the static setting, meaning that any corruption of parties should be determined *before* the protocol run. In particular, an environment is said to be "specious" if its first instruction consists of sampling a specious corruption $(\tilde{\mathsf{C}}, a)$ from some specious distribution D for every party in the protocol. A (dummy) adversary is said to be "specious" if it provides as input $(\textsc{Specious}, \tilde{\mathsf{C}})$ to a party, where $\tilde{\mathsf{C}}$ was previously sampled by the environment. Finally, a party is "specious" if, upon receiving a $(\textsc{Specious}, \tilde{\mathsf{C}})$ instruction from the adversary as its first input, it runs $\tilde{\mathsf{C}}$ (instead of C). [CMNV22, Theorem 1] proves that this restriction of environments and adversaries does not affect the UC composition theorem.

4.3.2 Modelling parties

In order to handle specious corruptions, the srUC framework introduces reverse firewalls in the modelling of parties. Each party P is modelled as two independent components: a core C and a reverse firewall F. The former is in charge of running the functional component of the protocol; the latter accesses and sanitizes the traffic of the core it is protecting *without* having access to its respective core's inputs. These two components can be corrupted independently: cores can be Honest, Specious, or Malicious; firewalls can be Honest, SemiHonest, or Malicious. The joint behaviour of the two components can be mapped to that of a regular party. For example, if a core is Honest and its firewall is SemiHonest, the overall party should be treated as Honest, given that the firewall should not degrade the security of its honest core. We report all possible combinations of this mapping in Table 4.1.

Core C	Firewall F	Party P in $\mathcal{F}$
Honest	Honest	Honest
Honest	SemiHonest	Honest
Specious	Honest	Honest
Honest	Malicious	Isolate
Specious	SemiHonest	Malicious
Specious	Malicious	Malicious
Malicious	Honest	Malicious
Malicious	SemiHonest	Malicious
Malicious	Malicious	Malicious

Table 4.1: The corruption translation table of [CMNV22]. The highlighted rows represent the sufficient cases to be considered within security proofs.

As discussed in [CMNV22, Section 2.3], certain corruption combinations are redundant within security proofs. For instance, while simulating a Malicious party, it suffices to consider the case of Malicious core and Malicious firewall, as this combination provides the adversary with strictly more power than the other cases. For the Honest setting instead, one has to simulate the cases (i) Honest core and SemiHonest firewall, and (ii) Specious core and Honest firewall separately, as the adversarial capabilities of these two cases are incomparable. Finally, the Isolate case is weaker compared to a Malicious corruption, as it models the setting where a malicious firewall simply cuts the communication of an honest core with the outside world.

4.3.3 Ideal functionalities

Splitting parties into two components comes with additional modelling choices for ideal functionalities, which are separated in *sanitizable* and *regular* ideal functionalities.

Sanitizable ideal functionalities explicitly interact with cores and firewalls, exposing, for each party P, an input-output interface IO interacting with C, and a sanitation interface S interacting with F. As a

concrete example, the sanitizable string commitment ideal functionality $\mathcal{F}_{\mathsf{sCOM}}$ [CMNV22, Section 3] allows the committer's firewall to blind the input of the committer's core.

On the other hand, regular ideal functionalities match traditional UC ideal functionalities, exposing a single interface for each party. In particular, a regular ideal functionality is not aware of cores and firewalls. Note that it is still perfectly reasonable to design subversion-resilient protocols that realize regular ideal functionalities. For instance, [CMNV22, Section 4] shows an instantiation of the regular coin tossing ideal functionality $\mathcal{F}_{\mathsf{TOSS}}$, and throughout this composition we will instantiate the regular $\mathcal{F}_{\mathsf{PAKE}}$ functionality.

According to the type of functionality that is being realized, two separate notions of security are presented in [CMNV22]: subversion-resilient UC (srUC) security for *sanitizable* ideal functionalities, and *wrapped* subversion-resilient UC (wsrUC) security for *regular* ideal functionalities. The only difference introduced by the latter consists of adding a "wrapper" to $\mathcal{F}$ that translates corruptions following Table 4.1 and forwards messages between the parties' components and $\mathcal{F}$. This is necessary, as otherwise the environment would always have a trivial distinguishing advantage: the real-world protocol explicitly separates cores and firewalls, whereas $\mathcal{F}$ only interacts with parties. The resulting functionality is denoted as $\mathsf{Wrap}(\mathcal{F})$.

Definition 4.2 captures security of a *sanitizing* protocol Π that instantiates cores and firewalls interacting with the interfaces exposed by a sanitizable ideal functionality $\mathcal{G}$ in order to UC-realize a wrapped regular functionality $\mathsf{Wrap}(\mathcal{F})$ in the $\mathcal{G}$-hybrid model. This yields wsrUC security. Definition 4.3 is essentially the same, but captures security for protocols realizing a *sanitizable* ideal functionality, yielding srUC security.

Definition 4.2 (Wrapped subversion-resilient UC security [CMNV22])**.** *Let $\mathcal{F}$ be an ideal functionality for n parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$. Let Π be a sanitizing protocol with n cores $\mathsf{C}_1, \ldots, \mathsf{C}_n$ and n firewalls $\mathsf{F}_1, \ldots, \mathsf{F}_n$. Let $\mathcal{G}$ be a sanitizable ideal functionality which can be used by the sanitizing protocol Π. We say that Π wsrUC-realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model if Π UC-realizes $\mathsf{Wrap}(\mathcal{F})$ in the $\mathcal{G}$-hybrid model, with the restriction that we only quantify over specious environments and specious adversaries.*

Definition 4.3 (Subversion-resilient UC security [CMNV22])**.** *Let $\mathcal{F}$ be an ideal functionality for n parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$. Let Π be a sanitizing protocol with n cores $\mathsf{C}_1^{\mathcal{F}}, \ldots, \mathsf{C}_n^{\mathcal{F}}$ and n firewalls $\mathsf{F}_1^{\mathcal{F}}, \ldots, \mathsf{F}_n^{\mathcal{F}}$. Let $\mathcal{G}$ be a sanitizable ideal functionality which can be used by a protocol Π interacting on IO interfaces and sanitation interfaces. We say that Π srUC-realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model if $\mathcal{F}$ can be written as a well-formed sanitizable ideal functionality[18], and Π UC-realizes $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, with the restriction that we only quantify over specious environments and specious adversaries.*

4.3.4 Communication channels

From a formal standpoint, a core and a firewall constituting a party are separate entities that cannot communicate with each other. For that, [CMNV22] assumes as a building block the ideal functionality for sanitizable authenticated communication $\mathcal{F}_{\mathsf{SAT}}$, which fundamentally embeds three capabilities:

- It allows to distribute a setup (*e.g.*, a CRS) by means of a Setup algorithm.
- It provides *secure* channels between cores and their respective firewall.
- It provides *authenticated* channels between firewalls.

Following [CMMV24], we report a variant of the description of $\mathcal{F}_{\mathsf{SAT}}$ that does *not* include the first capability. This is a design choice that allows to better separate setup and communication: indeed, the former may be captured by a separate ideal functionality $\mathcal{F}_{\mathsf{crs}}$.

Functionality $\mathcal{F}_{\mathsf{SAT}}$

- On input $(\textsc{Send}, \mathsf{P}_i, \mathsf{P}_j, a)$ on IO_i, it forwards the tuple on S_i. As in the original description, we assume that a is sent at most once from honest parties.

[18] A well-formed sanitizable ideal functionality can be decomposed into a central part that does the computation and interacts with parties, and an outer layer exposing IO and sanitation interfaces, similarly to a wrapped ideal functionality.

- On input $(\text{SEND}, \mathsf{P}_i, \mathsf{P}_k, b)$ on S_i, it leaks the tuple to the adversary $\mathcal{S}$, and internally stores the tuple.
- On input $(\text{DELIVER}, (\text{SEND}, \mathsf{P}_i, \mathsf{P}_k, b))$ from the adversary, where the SEND tuple is stored, it outputs $(\text{RECEIVE}, \mathsf{P}_i, \mathsf{P}_k, b)$ on S_k and deletes the tuple.
- On input $(\text{RECEIVE}, \mathsf{P}_i, \mathsf{P}_m, c)$ on S_m, it outputs $(\text{RECEIVE}, \mathsf{P}_i, \mathsf{P}_m, c)$ on IO_m.

Figure 4.11: The sanitizable message-authentication functionality $\mathcal{F}_{\mathsf{SAT}}$.

Importantly, we remark that $\mathcal{F}_{\mathsf{SAT}}$ induces a core-to-core authenticated channel. While this is an acceptable backbone of communication for protocols designing functionalities relying on authenticated channels, it makes little sense to instantiate PAKE under this setting. As part of our contributions, in Section 5.4, we overcome this limitation by defining a weaker functionality for sanitizable split-authentication $s\mathcal{F}_{\mathsf{SAT}}$ that models the unauthenticated setting by allowing the adversary to partition parties, in line with the split authenticated framework of [BCL+05]. In particular, $s\mathcal{F}_{\mathsf{SAT}}$ only assumes secure between cores and their respective firewall. We compare $\mathcal{F}_{\mathsf{SAT}}$ to $s\mathcal{F}_{\mathsf{SAT}}$ pictorially in Figure 4.12.

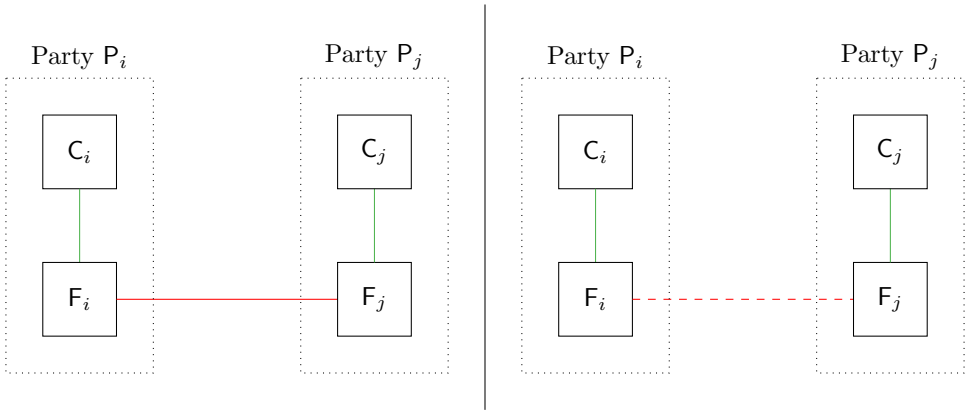


Figure 4.12: On the left, communication as in $\mathcal{F}_{\mathsf{SAT}}$. On the right, communication as in $s\mathcal{F}_{\mathsf{SAT}}$.

Note that the backbone of communication of $s\mathcal{F}_{\mathsf{SAT}}$ is indeed the minimal security assumption in the context of reverse firewalls, as otherwise the adversary could simply speciously corrupt a core C_i and analyze the traffic from C_i to F_i, nullifying any sanitation the firewall may have introduced. From a practical standpoint, this assumption is perfectly reasonable. In close-range environments, such as LANs in enterprises or home networks, a user may have its device connected to a firewall (which could very well be the router itself) by means of a physical cable, and it is reasonable to assume that no wiretapping will occur.

The framework would still hold even in the presence of remote firewalls: a core could establish a secure channel with its firewall via a maliciously-secure 2PC protocol (*e.g.*, by running a maliciously secure 2PC-PAKE) in a preliminary setup phase. Notice that this still makes sense in our threat model, as the adversary would have to mount an attack against an honest party, which is constituted of either a specious core and an honest firewall or an honest core and a semi-honest firewall. In either case, one of the two components remains honest throughout the setup of the secure channel.

4.3.5 Strong sanitation of firewalls

In order to evaluate the effectiveness of honest firewalls in protecting specious cores, [CMNV22] defines the convenient property of "strong sanitation". Informally, we say that a firewall is strongly sanitizing if no efficient adversary can distinguish an execution of the protocol with a specious core equipped with an honest firewall from an execution of the protocol with an honest core equipped with an honest firewall.

Definition 4.4 (Strong Sanitation [CMNV22])**.** *Let* $(\mathsf{C}_i, \mathsf{F}_i)$ *be a party in the* $\mathcal{G}$*-hybrid model. Let* $\hat{\mathsf{C}}_i$ *be the corresponding incorruptible core. Let* Π_i *be the protocol for the* $\mathcal{F}_{\mathsf{SAT}}$*-hybrid model, where party* i *is* $(\mathsf{C}_i, \mathsf{F}_i)$*, and all other parties are dummy parties. Let* Π'_i *be the protocol for the* $\mathcal{F}_{\mathsf{SAT}}$*-hybrid model where party* i *is* $(\hat{\mathsf{C}}_i, \mathsf{F}_i)$*, and all other parties are dummy parties. We say that* F_i *is strongly sanitizing if, for all poly-time environments* $\mathcal{Z}$ *which do not corrupt* F_i *but which are allowed a specious corruption of the core, it holds that* $\text{EXEC}_{\mathcal{Z},\Pi_i,\mathcal{A}} \approx \text{EXEC}_{\mathcal{Z},\Pi'_i,\mathcal{A}}$*, where* $\mathcal{A}$ *is the dummy adversary.*

Simplifying the security analysis. As shown in [CMNV22], some of the combinations highlighted in Table 4.1 can be omitted throughout the security analysis. In particular, the ISOLATE case is typically modelled as a MALICIOUS corruption in the authenticated setting (and as a MITM attack in the unauthenticated setting [CMMV24]), and can therefore be safely dropped from the analysis. Moreover, the case of SPECIOUS core and HONEST firewall can also be omitted, as long as the firewalls are strongly sanitizing. With these simplifications, one can prove security of a protocol by only simulating (i) HONEST core and SEMIHONEST firewall, and (ii) MALICIOUS core and MALICIOUS firewall.

4.3.6 On input-triggering attacks

As remarked in Section 4.3.1, specious corruptions are characterized in [CMNV22] by the notion of black-box testing. However, one important detail that is implicit in all the protocols and results of [CMNV22] is that all specious implementations should additionally be "functionality-maintaining", meaning that their behaviour should be indistinguishable to that of an honest implementation for *all* inputs and input messages. This follows the spirit of "signalling subversions" considered in [CMNV22, Definition 2], which consists of covertly exfiltrating information throughout a protocol run. Indeed, the example we reported in Section 4.3.1 is specious *and* functionality-maintaining. As a commonality with other definitions in the literature (*e.g.*, [CGPS21]), this characterization of subversion attacks rules out a specific class of input-triggering attacks that may cause the specious core to behave arbitrarily upon receiving a specific triggering message (*e.g.*, by crashing).

A concrete vulnerability to input-triggering attacks. Consider the ideal functionality for sanitizable string commitment $\mathcal{F}_{\mathsf{sCOM}}$ defined in [CMNV22, Section 3]. Here, the committer may commit to a value m, and its firewall may blind the committed value by using some blinding factor r with a blinding operation $\oplus$. Whenever the committer instructs the functionality to open the commitment, $\mathcal{F}_{\mathsf{sCOM}}$ outputs $m \oplus r$ to the core of the receiver.

An input-triggering attack against any protocol realizing $\mathcal{F}_{\mathsf{sCOM}}$ unfolds as follows. Consider a specious receiver core protected by an honest firewall that interacts with a malicious committer. First, the environment samples the specious receiver core $\tilde{\mathsf{C}}$ with auxiliary information $a = V$, where $\tilde{\mathsf{C}}$ is defined in such a way that it crashes upon receiving the value V from the committer (otherwise it behaves honestly).

Given that the adversary has knowledge of a (by definition of specious corruption), it can now instruct the (malicious) core of the committer to commit to a triggering value V, and the (malicious) firewall of the committer to use the identity id as a blinding factor (*e.g.*, 0 if the blinding operation is $\oplus$). This way, the receiver's core obtains $V \oplus id = V$ from $\mathcal{F}_{\mathsf{sCOM}}$. In the strong sanitation game, a specious receiver core will always crash without output, whereas an honest receiver core would output V. This yields a trivial distinguishing advantage to the environment.

A definitional gap. The problematic aspect that prevents one from capturing input-triggering attacks via the srUC framework lies in the characterization of strong sanitation (Definition 4.4). Indeed, this definition prescribes indistinguishability of simulation by taking all parties except for the one with a specious core to be dummy parties (*i.e.*, honest parties): entities that are inherently unable to mount input-triggering attacks.

Handling input-triggering attacks. The considerations reported above are posterior to [CMMV24]. Indeed, in [CMMV24], we followed the srUC framework by taking Definition 4.1 at face value - *i.e.*, without

considering the additional requirement of functionality maintenance of specious corruptions. In particular, all the protocols and ideal functionalities thereby defined consider the problem of sanitizing all inbound messages, a strategy which appears to be sufficient in thwarting input-triggering attacks. However, given that we do not provide a formal extension to the model to comprehensively account for input triggers, the only formal security guarantees one can hope to achieve matches the ones provided by the srUC framework - *i.e.*, no exfiltration is possible. For this reason, in the next section we propose a revised version of [CMMV24] that fully adheres to the srUC framework. We leave it as an interesting open problem to properly formalize the above considerations.

5 Subversion-Resilient PAKE from Oblivious Transfer

In [CMMV24], we feature the first construction of a srUC PAKE protocol with security in the presence of subversion attacks. In light of the observations raised in Section 4.3.6, in this section we present a revised version of [CMMV24], which retains exactly the same structure of the seminal work while simplifying some of the protocols and ideal functionalities to formally match the subversion-resilient UC framework [CMNV22]. In Appendix A.2, we discuss further input-triggering attacks with respect to the protocols and ideal functionalities presented in [CMMV24].

5.1 Summary

At its core, the design of our subversion-resilient PAKE protocol consists of sanitizing the strategy outlined in Section 4.2: we first design a subversion-resilient protocol for randomized equality, and then remove authenticated channels to obtain a subversion-resilient protocol for PAKE.

Randomized equality in the presence of subversion. As our first step, we realize subversion-resilient randomized equality (*i.e.*, PAKE with authenticated channels) assuming the existence of the sanitizable authenticated communication functionality $\mathcal{F}_{\mathsf{SAT}}$ of [CMNV22]. This is achieved by sanitizing the RE-from-OT protocol of [CDVW12] described in Section 4.2.1. Along the way, we coin the notion of sanitizable oblivious transfer alongside a sanitizable ideal functionality $\mathcal{F}_{\mathsf{sOT}}$, providing a matching instantiation by sanitizing the construction from dual-mode encryption of [PVW08] reported in Section 3.5.1.

Sanitizable split-authenticated communication. Following [BCL+05], we define the ideal functionality for *sanitizable* split authentication $s\mathcal{F}_{\mathsf{SAT}}$, which allows the adversary to partition parties, and prove that a modification of the transformation of [BCL+05] (summarized in Section 4.1.2) allows lifting any protocol that multi-realizes a functionality $\mathcal{F}$ assuming authenticated channels to one that realizes the corresponding split functionality $s\mathcal{F}$ without any assumption on channels, even in the presence of subversion. In the process, we realize $s\mathcal{F}_{\mathsf{SAT}}$ by sanitizing the protocol of [BCL+05, Section 4.2] described in Section 4.1.2, introducing a new notion of *key-sanitizable* signature schemes with a matching security property. This improves on an open problem from [CMNV22], where the authors were only able to realize $\mathcal{F}_{\mathsf{SAT}}$ assuming the presence of a PKI and by moving to a "three-tier model" variant of the framework, in which each party has an additional *operative* component that may only be honest or malicious[19]. On the contrary, we realize the backbone of communication among components in the two-tier model without assuming a PKI, although only for the unauthenticated setting (*i.e.*, $s\mathcal{F}_{\mathsf{SAT}}$).

Transforming randomized equality into PAKE. By applying the aforementioned transformation to our randomized equality protocol, we realize subversion-resilient PAKE by constructing a protocol with access to the split version of the randomized equality functionality.

5.1.1 Technical overview

Below, we provide an overview of the technical contributions, explaining the main ideas and tools behind our subversion-resilient PAKE protocol.

Sanitizing OT. The first step towards the definition of our PAKE protocol lies in characterizing oblivious transfer in the presence of specious subversion attacks. Indeed, the (non-sanitized) functionality would allow a (specious) sender to sample the inputs in a leaky manner and send them over to a corrupted party. For this reason, we must ensure that firewalls can intervene in the run of the ideal functionality by sanitizing

[19] Even if used exclusively throughout the setup phase, providing access to an operative component that is immune to subversion is a strong assumption that weakens any result achieved in the srUC framework. Indeed, the three-tier model provides a trivial solution to counteract specious corruptions of the core for *any* functionality, as the operative is in principle allowed to run any protocol on behalf of the core.

the sender's inputs accordingly. We formalize this requirement by designing a *sanitizable* OT functionality $\mathcal{F}_{\mathsf{sOT}}$ in a similar fashion to how [CMNV22] captured sanitizable commitments. In particular, $\mathcal{F}_{\mathsf{sOT}}$ behaves as shown in Figure 5.1. Note that, consistently with the sanitizable authenticated string commitment functionality of [CMNV22], the firewall of the sender returns its blinding factors to its core after they have been used. Intuitively, this ensures that the core has a view of the protocol that is consistent with that of its peer.

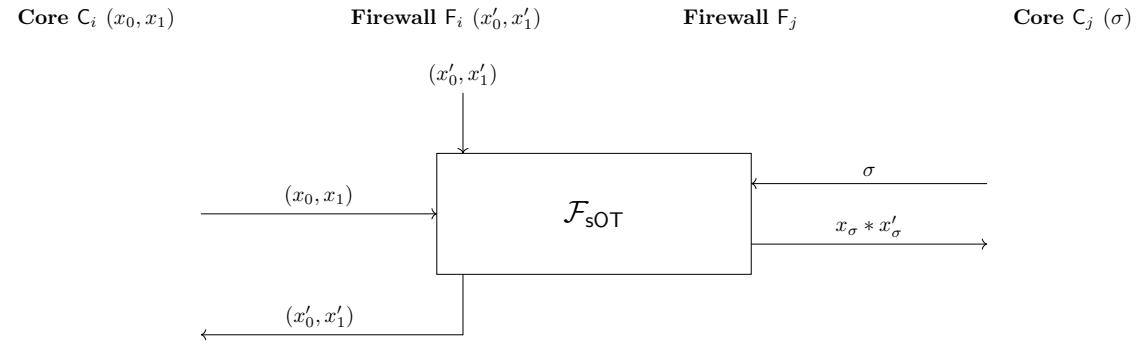


Figure 5.1: Our sanitizable OT functionality $\mathcal{F}_{\mathsf{sOT}}$, with $*$ being a blinding operation for the input domain.

In order to instantiate $\mathcal{F}_{\mathsf{sOT}}$, we start by considering dual-mode encryption as presented in [PVW08] (summarized in Section 3.5.1). As a brief recap, in these cryptosystems the party who wishes to decrypt messages generates a key pair by specifying a decryption branch σ, and the party holding the public key specifies an encryption branch b for each ciphertext. Decryption succeeds only for ciphertexts generated on the decryption branch.

Moving to the subversion setting, we introduce a new primitive that we call *sanitizable homomorphic* dual-mode encryption, extending dual-mode encryption by additionally providing: (1) a procedure to carry out homomorphic operations on ciphertexts (*e.g.*, $\mathsf{Enc}(m_1) * \mathsf{Enc}(m_2) = \mathsf{Enc}(m_1 * m_2)$), (2) a procedure to maul an encryption key pk to a different encryption key $\widetilde{pk}$, and (3) a procedure to maul a ciphertext under encryption key $\widetilde{pk}$ to a ciphertext of the same message under encryption key pk. Looking ahead, item (1) allows firewalls to sanitize the messages input to the OT, and items (2, 3) allow firewalls to first blind a public key, introducing a layer of sanitation, and align encryptions accordingly, stripping the layer of sanitation away to preserve correctness. The construction from DDH of [PVW08, Section 5] (summarized in Section 3.5.1) can be extended to verify our newly-introduced properties in a straight-forward manner.

With that, we instantiate $\mathcal{F}_{\mathsf{sOT}}$ by sanitizing the protocol of [PVW08, Section 4] (summarized in Section 3.5.2). The resulting sanitized protocol roughly unfolds as follows. The receiver produces a key pair that may only be used to decrypt values on the encryption branch matching the choice bit σ, and sends the public key towards the sender. This key is sanitized by the receiver's firewall, introducing a layer of sanitation. Upon receiving the (sanitized) key, the sender encrypts value x_b on encryption branch b, for $b \in \{0, 1\}$, and forwards these ciphertexts towards the receiver. The sender's firewall leverages the homomorphic property of our sanitizable homomorphic dual-mode encryption scheme to blind ciphertexts and their respective plaintexts. Finally, the receiver's firewall removes the layer of sanitation so that its core can successfully decrypt the sanitized plaintext on branch $b = \sigma$.

In the security proof, we first show that the construction is strongly sanitizing by using the aforementioned sanitation-friendly properties. After that, the simulation becomes extremely close to the one of the original protocol, as it leverages the two (computationally indistinguishable) modes of the CRS to map the behaviour of the adversary to consistent queries to $\mathcal{F}_{\mathsf{sOT}}$.

We conclude the section by remarking that, exactly as in the original protocol of [PVW08], it is possible to re-use the same CRS across multiple protocol runs. Hence, we obtain a protocol that multi-realizes $\mathcal{F}_{\mathsf{sOT}}$ (*i.e.*, a protocol that realizes the the multi-session sanitizable OT functionality $\hat{\mathcal{F}}_{\mathsf{sOT}}$).

Sanitizing Randomized Equality. Next, we sanitize the $\mathcal{F}_{\mathsf{RE}}$-from-$\mathcal{F}_{\mathsf{OT}}$ protocol of [CDVW12] (summarized in Section 4.2.1), which roughly unfolds as follows: for an n-bit password, each party runs $\mathcal{F}_{\mathsf{OT}}$ n-times as the sender, inputting two random strings for each OT run, and n-times as the receiver, inputting the i-th bit of its password in the i-th run. Intuitively, the sender of each batch of OTs is able to choose the same random strings that were selected by the receiver only if the passwords are the same, and all these strings can be combined to derive a common shared key.

After defining $\mathcal{F}_{\mathsf{sOT}}$, sanitizing the aforementioned protocol becomes immediate. In order to thwart information leakage originating from a biased sampling of the random strings, each firewall introduces uniformly-random strings in the batch of OTs where its core acts as a sender. In order to ensure correctness of the protocol, these blinding factors are returned to each firewall's respective core after they have been used (as per the description of $\mathcal{F}_{\mathsf{sOT}}$), so that each core can locally adjust its session key consistently with the sanitation.

Split functionalities in the srUC model. A PAKE protocol establishes (over an *unauthenticated* channel) a secret key among parties that share a common password. As per our considerations in Section 4.2, it makes little sense to build a PAKE protocol in a setting that already assumes the existence of authenticated channels.

Given that [CMNV22] exclusively refers to authenticated channels, which are formalized with the ideal functionality for "sanitizable authenticated transmission" $\mathcal{F}_{\mathsf{SAT}}$, in this work we extend the notion of split functionalities (summarized in Section 4.1.2) to the srUC model. More specifically, we show that the generic transformation of [BCL+05] that allows one to remove authenticated communication from suitable protocols carries over to our setting whenever the underlying *unauthenticated* channel is sanitizable. The latter notion is captured by the split version of $\mathcal{F}_{\mathsf{SAT}}$, that we call $s\mathcal{F}_{\mathsf{SAT}}$. This functionality allows the adversary to split parties in different authentication sets in a "link initialization" phase, before any message is exchanged. After that, the behaviour is exactly the same as $\mathcal{F}_{\mathsf{SAT}}$, except that the adversary may deliver arbitrary messages to parties within different authentication sets.

A crucial component of the transformation is the construction of a protocol realizing $s\mathcal{F}_{\mathsf{SAT}}$. For that, we introduce *key-sanitizable* signatures, a new primitive that provides: (1) a procedure to maul a verification key vk into $\widetilde{\mathsf{vk}}$, and (2) a procedure to maul an accepting signature under verification key vk into an accepting signature under verification key $\widetilde{\mathsf{vk}}$. We then show that one may obtain a key-sanitizable signature scheme by appropriately extending the BLS signature scheme [BLS01]. With that, we instantiate $s\mathcal{F}_{\mathsf{SAT}}$ by sanitizing the protocol realizing $\mathcal{F}_{\mathsf{SA}}$ (described in Section 4.1.2): parties exchange locally-generated keys, which are used to first "initialize the link" by determining a session ID, and then sign messages that are exchanged through the link. Firewalls sanitize these keys and re-align signatures accordingly to preserve correctness while preventing leakage of information.

Once a protocol for $s\mathcal{F}_{\mathsf{SAT}}$ is in place, one can simply white-box inspect the proofs of [BCL+05] and adapt them to the srUC setting. The core result is a lemma stating that any protocol n-realizing a n-party functionality $\mathcal{F}$ in the wsrUC model assuming $\mathcal{F}_{\mathsf{SAT}}$ can be compiled into a protocol realizing $s\mathcal{F}$ in the wsrUC model assuming $s\mathcal{F}_{\mathsf{SAT}}$. Given that any n-party functionality $\mathcal{F}$ can be n-realized in the wsrUC model by the subversion-resilient GMW compiler of [CMNV22], we also obtain a theorem stating that any n-party split functionality can be realized in the wsrUC model using only *unauthenticated* channels (in the $s\mathcal{F}_{\mathsf{SAT}}$-hybrid model), matching [BCL+05, Theorem 10]. As in traditional UC, a protocol poly-realizing a functionality roughly means that polynomially-many instances of that protocol may re-use the same setup.

The final PAKE protocol. At last, we combine all our ingredients together to realize our srUC PAKE protocol. First, we apply the split transformation to our protocol realizing $\mathcal{F}_{\mathsf{RE}}$ in the authenticated setting, obtaining a protocol that realizes $s\mathcal{F}_{\mathsf{RE}}$ in the unauthenticated setting. Then, with a similar argument to that of [DHP+18], we argue that $s\mathcal{F}_{\mathsf{RE}}$ is sufficient to instantiate $\mathcal{F}_{\mathsf{PAKE}}$. This can be shown by exhibiting a trivial protocol in the $s\mathcal{F}_{\mathsf{RE}}$-hybrid model that exclusively interacts with $s\mathcal{F}_{\mathsf{RE}}$, and by showing that the power of splitting parties in $s\mathcal{F}_{\mathsf{RE}}$ can be mapped to the power of performing password queries in $\mathcal{F}_{\mathsf{PAKE}}$.

Finally we observe that, as a corollary of the generic result of the previous section, one also gets a protocol realizing $s\mathcal{F}_{\mathsf{RE}}$ by relying on the srUC GMW compiler from [CMNV22], although with worse efficiency than our concrete construction from DDH. For that, we provide a hand-wavy comparison of the two constructions by considering communication and round complexity.

Further remarks. Importantly, in this work we consider a PAKE functionality that only provides implicit rather than explicit authentication. This means that, while parties can rest assured that any other party capable of deriving the same session key must be in possession of the password, there is no direct assurance that the counterpart has successfully computed the session key upon completion of the protocol. This decision was made for two primary reasons: (1) it streamlines our results, as explicit mutual authentication typically requires incorporating additional "key confirmation" steps at the protocol's conclusion, which would complicate our protocol with the need for further sanitation processes, and (2) in many practical scenarios, such as secure channels, explicit authentication is not a requirement, as it is inherently provided by any higher-level protocol that utilizes PAKE as a foundation. For instance, in applications involving secure messaging, the act of successfully exchanging messages serves as explicit confirmation that both parties share the same session key.

Moreover, as a technical remark stemming from the srUC model, the PAKE functionality we realize implicitly includes the wrapper of [CMNV22] that simply adds dummy firewall parties in order to prevent trivial distinguishing from the environment. This also holds for $\mathcal{F}_{\mathsf{RE}}$, but causes no differences in the behaviour of both functionalities. For a cleaner presentation and following the presentation style of [CMNV22], we omit the wrapper when using hybrid functionalities.

5.1.2 Related work

Next, we discuss related work on reverse firewalls, subversion-resilient cryptography in general, and PAKE.

Reverse firewalls and subversion. Reverse firewalls were introduced in [MS15], who showed how to construct reverse firewalls for oblivious transfer (OT) and two-party computation with semi-honest security. Follow up works showed how to construct reverse firewalls for many other cryptographic primitives and protocols including: secure message transmission and key agreement [DMS16, CMY+16], interactive proof systems [GMV20], and maliciously secure MPC for both the case of static [CDN20] and adaptive [CGPS21] corruptions. However, most of these constructions lack modularity, as the security of each firewall is proven in isolation and does not extend to larger protocols when combined with other firewalls.

More recently and concurrently to this work, an alternative framework for subversion-resilient UC was put forward in [ABMO23]. Compared to [CMNV22], this framework captures reverse firewalls in the plain UC model, but characterizes subversion by exclusively allowing an adversary to tamper with the function generating the randomness of a protocol.

[Rin23] explored the problem of achieving subversion-resilient AKE in a standalone fashion (*i.e.*, without reverse firewalls or watchdogs), providing intuition on why realizing this primitive appears to be hard in such an adversarial setting.

Additional work on subversion includes algorithm substitution attacks [BPR14, AMV15, DFP15], parameter subversion [BFS16, ABLZ17, Fuc18, ALSZ20], Cliptography [AFMV19, RTYZ17, RTYZ16], subliminal channels [Sim84, Sim86] to list a few. We refer to [MS15, CMNV22] for further related works, such as *watchdogs* and *self-guarding*.

PAKE. The seminal work [CHK+05] formalizes PAKE as an ideal functionality, and proposes an efficient protocol securely realizing this functionality in the setting of malicious corruptions and under *universal composability* [Can01]. The description was later extended to *explicit* mutual authentication in [GK10] and [CDVW12], in which parties are able to tell whether they effectively authenticated or not. Our work is the first to achieve subversion-resilient PAKE in the UC framework.

Shortcomings of PAKE functionalities. Recent works have raised technical concerns regarding the definition of PAKE functionalities widely used across the literature. Specifically, [AHH21] observed that several definitions, including the one of the seminal paper on UC PAKE [CHK+05], allow the adversary to set the key output by an honest party even without knowing the password. Similarly to [DHP+18], our definitions of $\mathcal{F}_{\mathsf{RE}}$ and $\mathcal{F}_{\mathsf{PAKE}}$ do not embed this shortcoming.

Additionally, [RX23] show an impossibility result proving that any 2-party $\mathcal{F}_{\mathsf{PAKE}}$ may be instantiated by an incorrect 0-round protocol. In order to overcome this limitation, they show that either (i) the underlying PAKE protocol is assumed to be correct; (ii) the simulator gets limited in power; or (iii) a third party responsible for routing messages is introduced in $\mathcal{F}_{\mathsf{PAKE}}$. For this work, we solve this shortcoming by considering approach (i), following the spirit of discarding "trivial protocols" in the context of UC (*e.g.*, the empty protocol), as discussed in [CKL03].

5.1.3 Organization

In Section 5.2, we define and instantiate sanitizable oblivious transfer. In Section 5.3, we instantiate a subversion-resilient protocol for the randomized equality ideal functionality. In Section 5.4, we define and instantiate the sanitizable split-authenticated functionality, and port to the srUC framework the transformation of [BCL+05] that allows one to remove authenticated channels from suitable protocols. In Section 5.5, we combine the results of previous sections to achieve subversion-resilient PAKE. In Figure 5.2, we show a visual summary of how our results are related to one another.

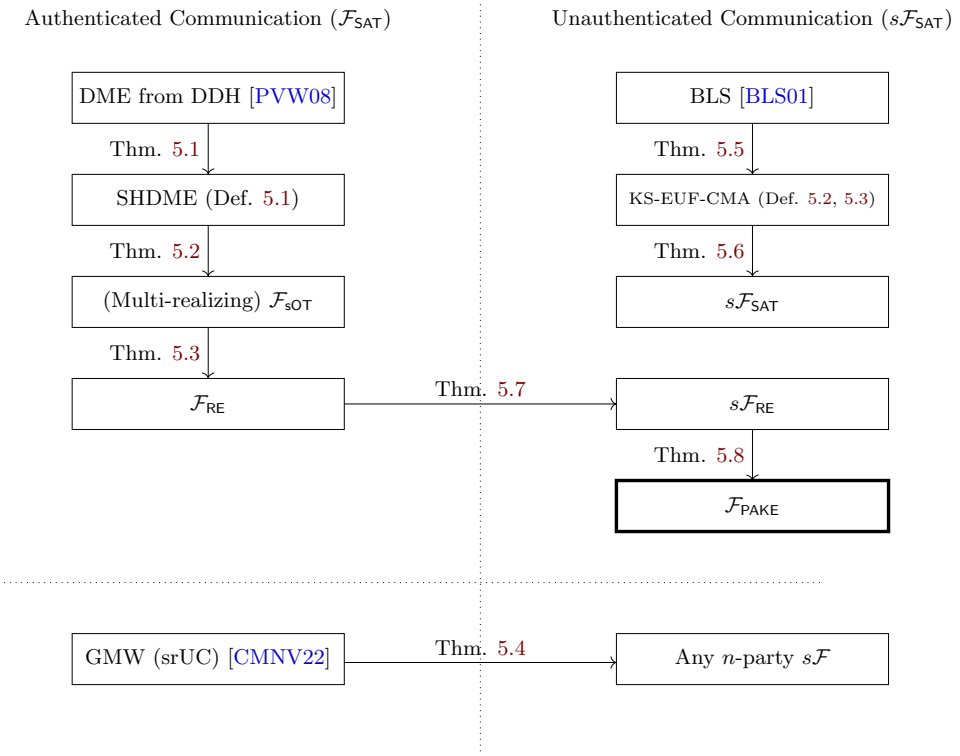


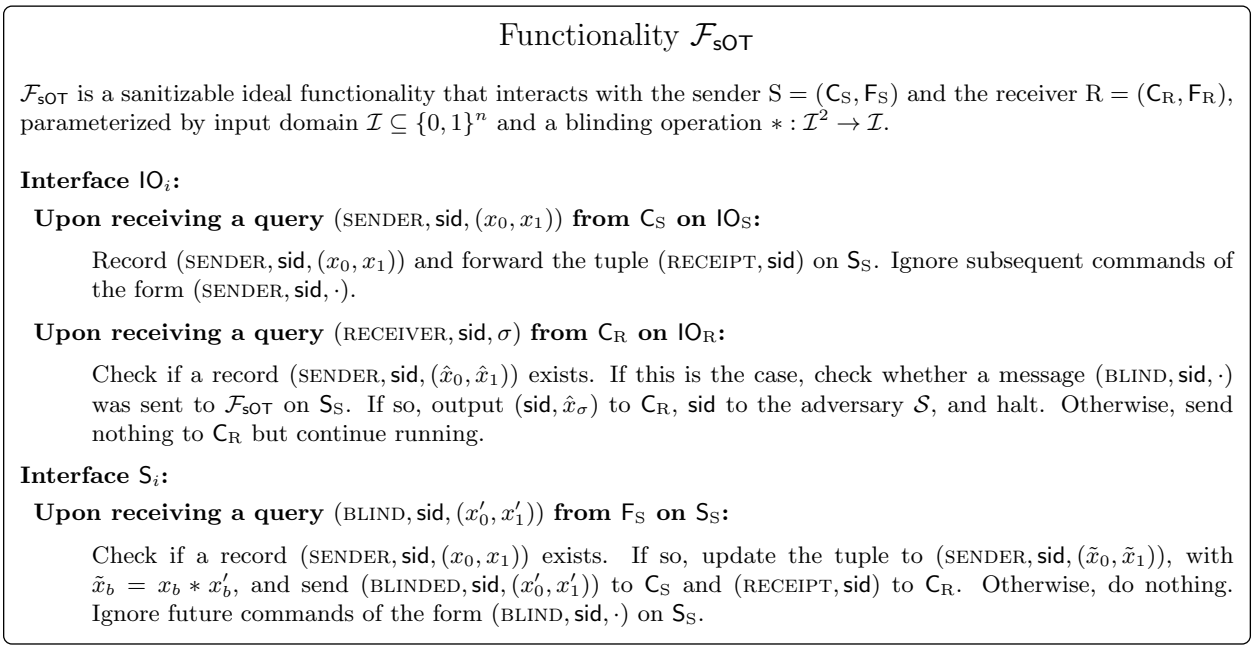
Figure 5.2: A visual summary of our contributions. All the functionalities are realized in the srUC framework of [CMNV22]. DME stands for Dual-Mode Encryption. SHDME stands for Sanitizable Homomorphic DME. KS-EUF-CMA stands for Key-Sanitizable EUF-CMA.

5.2 Sanitizing Oblivious Transfer

In this subsection, we design and instantiate a *sanitizable* ideal functionality for oblivious transfer. For that, we first define *sanitizable homomorphic* dual-mode encryption, and exhibit an instantiation from the DDH assumption. We then use this new primitive to sanitize the generic framework for OT of [PVW08], obtaining a protocol that realizes the *sanitizable* oblivious transfer functionality $\mathcal{F}_{\mathsf{sOT}}$. Finally, we argue that, in line with the instantiation of [PVW08], our protocol can reuse the same CRS across multiple runs, thus realizing the multi-session extension of $\mathcal{F}_{\mathsf{sOT}}$ (also denoted by $\hat{\mathcal{F}}_{\mathsf{sOT}}$).

5.2.1 The sanitizable oblivious transfer functionality $\mathcal{F}_{\mathsf{sOT}}$

Following the ideas presented in the technical overview, we describe the *sanitizable* ideal functionality for oblivious transfer $\mathcal{F}_{\mathsf{sOT}}$, in which the sender's firewall may intervene in the sanitation of its core's inputs.



Functionality $\mathcal{F}_{\mathsf{sOT}}$

$\mathcal{F}_{\mathsf{sOT}}$ is a sanitizable ideal functionality that interacts with the sender $\mathrm{S} = (\mathsf{C}_\mathrm{S}, \mathsf{F}_\mathrm{S})$ and the receiver $\mathrm{R} = (\mathsf{C}_\mathrm{R}, \mathsf{F}_\mathrm{R})$, parameterized by input domain $\mathcal{I} \subseteq \{0,1\}^n$ and a blinding operation $* : \mathcal{I}^2 \to \mathcal{I}$.

Interface IO_i:

Upon receiving a query $(\textsc{sender}, \mathsf{sid}, (x_0, x_1))$ **from** C_S **on** IO_S**:**

Record $(\textsc{sender}, \mathsf{sid}, (x_0, x_1))$ and forward the tuple $(\textsc{receipt}, \mathsf{sid})$ on S_S. Ignore subsequent commands of the form $(\textsc{sender}, \mathsf{sid}, \cdot)$.

Upon receiving a query $(\textsc{receiver}, \mathsf{sid}, \sigma)$ **from** C_R **on** IO_R**:**

Check if a record $(\textsc{sender}, \mathsf{sid}, (\hat{x}_0, \hat{x}_1))$ exists. If this is the case, check whether a message $(\textsc{blind}, \mathsf{sid}, \cdot)$ was sent to $\mathcal{F}_{\mathsf{sOT}}$ on S_S. If so, output $(\mathsf{sid}, \hat{x}_\sigma)$ to C_R, sid to the adversary $\mathcal{S}$, and halt. Otherwise, send nothing to C_R but continue running.

Interface S_i:

Upon receiving a query $(\textsc{blind}, \mathsf{sid}, (x'_0, x'_1))$ **from** F_S **on** S_S**:**

Check if a record $(\textsc{sender}, \mathsf{sid}, (x_0, x_1))$ exists. If so, update the tuple to $(\textsc{sender}, \mathsf{sid}, (\tilde{x}_0, \tilde{x}_1))$, with $\tilde{x}_b = x_b * x'_b$, and send $(\textsc{blinded}, \mathsf{sid}, (x'_0, x'_1))$ to C_S and $(\textsc{receipt}, \mathsf{sid})$ to C_R. Otherwise, do nothing. Ignore future commands of the form $(\textsc{blind}, \mathsf{sid}, \cdot)$ on S_S.

Figure 5.3: Description of the sanitizable oblivious transfer functionality $\mathcal{F}_{\mathsf{sOT}}$.

Description. Conceptually, this functionality has the same structure of the sanitizable string commitment functionality presented in [CMNV22]. In particular, whenever a sender's core picks its two inputs (x_0, x_1), the functionality notifies its respective firewall. The firewall, in turn, supplies two additional inputs (x'_0, x'_1) that are used within the functionality to blind the original inputs of the sender. The receiver is notified of the availability of the (blinded) inputs, and the blinding factors are returned to the sender's core. Finally, the receiver can query the functionality with its choice bit σ to retrieve the blinded input $\hat{x}_\sigma$. We explicitly parameterize the blinding operation $*$, which may be tailored to the input domain of choice (*e.g.*, for additive blinding, $x_0 * x'_0 = x_0 \oplus x'_0$; for multiplicative blinding, $x_0 * x'_0 = x_0 \cdot x'_0$).

5.2.2 Sanitizable Homomorphic Dual-Mode Encryption

Looking ahead, we need to augment dual-mode encryption to allow the sanitation of public keys, ciphertexts, and plaintexts related to ciphertexts. For that, we formally define *sanitizable homomorphic* dual-mode encryption in what follows.

Definition 5.1 (Sanitizable Homomorphic Dual-Mode Encryption)**.** *A sanitizable homomorphic dual-mode encryption scheme consists of a tuple of algorithms* (Setup, KeyGen, Enc, Dec, FindMessy, TrapKeyGen, HomOp, MaulPK, AlignEnc)*, with* $\mathcal{R}_{\mathsf{MaulPK}}$ *being the randomness space for the* MaulPK *and* AlignEnc *algorithms, with the following properties:*

1. ***Dual-mode encryption:*** *The tuple of algorithms* (Setup, KeyGen, Enc, Dec, FindMessy, TrapKeyGen) *constitutes a dual-mode encryption scheme, as per Definition 3.3.*
2. ***Homomorphic ciphertexts:*** *For every* $\sigma \in \{0,1\}$*, for every* $(pk, sk) \leftarrow\!\$\ \mathsf{KeyGen}(\sigma)$*, for every* $c_i \leftarrow\!\$\ \mathsf{Enc}(pk, \sigma, m_i)$*, with* $i \in \{0,1\}$ *and* $m_i \in \{0,1\}^n$*,* $\mathsf{HomOp}(m_0, m_1)$ *outputs a fresh ciphertext of message* $m_0 * m_1$*, i.e.,* $\mathsf{HomOp}(c_0, c_1) = \mathsf{Enc}(pk, \sigma, (m_0 * m_1))$*.*
3. ***Consistent key sanitation:*** *For every* $\sigma \in \{0,1\}$*, for every* $(pk, sk) \leftarrow\!\$\ \mathsf{KeyGen}(\sigma)$*, for every* $\rho \in \mathcal{R}_{\mathsf{MaulPK}}$*,* $\mathsf{MaulPK}(pk, \rho)$ *outputs a fresh encryption key* $\widetilde{pk}$ *with the following property. For every* $\tilde{c} \leftarrow\!\$\ \mathsf{Enc}(\widetilde{pk}, \sigma, m)$*, with* $i \in \{0,1\}$ *and* $m \in \{0,1\}^n$*,* $\mathsf{AlignEnc}(c, \rho)$ *produces a new ciphertext* c *under public key* pk*, i.e.,* $\mathsf{AlignEnc}(\tilde{c}, \rho) = c$*, where* $c \in \mathsf{Enc}(pk, \sigma, m)$*.*

Intuitively, MaulPK and AlignEnc are defined as a (symmetric) tuple of algorithms. Concretely, a firewall will first sanitize an outbound public key by running MaulPK with some randomness ρ. Then, upon receiving any ciphertext encrypted under the mauled public key, the firewall will "strip" the layer of sanitation by running AlignEnc with ρ to output a ciphertext containing the same message for the non-mauled public key pk.

Instantiation from DDH. The DDH cryptosystem presented in Construction 3.1 is compatible with all the additional interfaces we introduced in Definition 5.1. In what follows, we define algorithms that match the newly introduced properties in a straight-forward manner:

- $\mathsf{MaulPK}(pk, \rho)$: Output pk^ρ.
- $\mathsf{AlignEnc}(c, \rho)$: Parse c as (c_1, c_2). Output $\tilde{c} = (c_1^\rho, c_2)$.
- $\mathsf{HomOp}(c_0, c_1)$: Output $c_0 \cdot c_1$.

Theorem 5.1. *The DDH cryptosystem of [PVW08] with the additional algorithms specified above is a sanitizable homomorphic dual-mode cryptosystem, assuming that DDH is hard for* $\mathbb{G}$*.*

Proof (Theorem 5.1). Correctness and all the properties related to the original dual-mode encryption scheme are unchanged (as per [PVW08, Theorem 5.2]). It remains to show that the cryptosystem has homomorphic ciphertexts and consistent key sanitation.

- **Homomorphic ciphertexts.** The property follows by observing that the underlying DDH cryptosystem is multiplicative homomorphic. In the following, $\mathsf{crs} = (g_0, g_1, h_0, h_1)$, $(sk, pk) \leftarrow\!\$\ (r, (g_\sigma^r, h_\sigma^r))$, and s, s', t, t' are random coins sampled by the Enc algorithm.

$$\begin{aligned}\mathsf{Enc}(pk, b, m_0) \cdot \mathsf{Enc}(pk, b, m_1) &= (g_b^s h_b^t, g_\sigma^{rs} h_\sigma^{rt} m_0)(g_b^{s'} h_b^{t'}, g_\sigma^{rs'} h_\sigma^{rt'} m_1) \\ &= (g_b^{(s+s')} h_b^{(t+t')}, g_\sigma^{r(s+s')} h_\sigma^{r(t+t')} m_0 \cdot m_1) \\ &= \mathsf{Enc}(pk, b, m_0 \cdot m_1).\end{aligned}$$

- **Consistent key sanitation.** We prove this property by showing that an aligned ciphertext can be decrypted successfully. In the following, $\mathsf{crs} = (g_0, g_1, h_0, h_1)$, $(sk, pk) \leftarrow\!\$\ (r, (g_\sigma^r, h_\sigma^r))$, $\widetilde{pk} = (g_\sigma^{r\rho}, h_\sigma^{r\rho}) \leftarrow \mathsf{MaulPK}(pk, \rho)$, $(c_1, c_2) \leftarrow\!\$\ \mathsf{Enc}(\widetilde{pk}, \sigma, m)$, and s, t are random coins sampled by the Enc algorithm.

$$\begin{aligned}\mathsf{Dec}(sk, \mathsf{AlignEnc}(c, \rho)) &= \mathsf{Dec}(sk, (c_1^\rho, c_2)) \\ &= c_2 / (c_1^\rho)^r \\ &= g_\sigma^{r\rho s} h_\sigma^{r\rho t} m / g_\sigma^{s\rho r} h_\sigma^{t\rho r} \\ &= m.\end{aligned}$$

□

5.2.3 A generic instantiation of $\mathcal{F}_{\mathsf{sOT}}$

As discussed in Section 4.2.1, having access to dual-mode encryption allows one to instantiate $\mathcal{F}_{\mathsf{OT}}$ in a natural manner: the receiver computes a key pair by using its choice bit σ as the selected decryption branch, and the sender encrypts each of its inputs x_b on a separate encryption branch $b \in \{0,1\}$. The receiver will only be able to decrypt the ciphertext on branch $\sigma = b$.

In Figure 5.4, we propose a sanitation of the aforementioned protocol. In particular, we leverage homomorphic ciphertexts to blind the sender's inputs while also refreshing the randomness of the respective ciphertexts. Moreover, consistent key sanitation ensures that the receiver's outbound encryption key is sanitized while inbound ciphertexts can be realigned in a consistent fashion for decryption purposes. Note that, after the sanitation is operated, the sender's firewall returns its (blinding) inputs (x'_0, x'_1) to its core. This is to ensure consistency with the description of $\mathcal{F}_{\mathsf{sOT}}$,

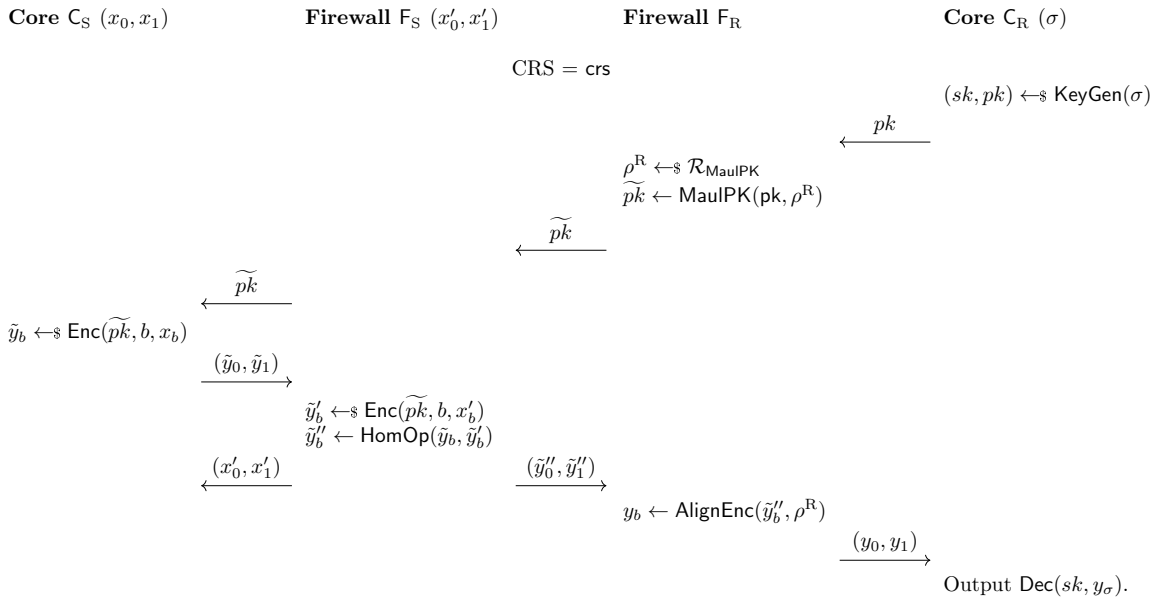


Figure 5.4: A sanitation of the generic framework of [PVW08] realizing $\mathcal{F}_{\mathsf{sOT}}$. Any computation having b as a subscript is run for all $b \in \{0,1\}$.

Theorem 5.2. *The protocol in Figure 5.4, parameterized by* $\mathsf{mode} \in \{\mathsf{mes}, \mathsf{dec}\}$*, realizes the sanitizable functionality* $\mathcal{F}_{\mathsf{sOT}}$ *in the* $(\mathcal{F}_{\mathsf{SAT}}, \mathcal{F}_{\mathsf{crs}})$*-hybrid model under static corruptions. For* $\mathsf{mode} = \mathsf{mes}$*, the sender's security is statistical and the receiver's security is computational; for* $\mathsf{mode} = \mathsf{dec}$*, the security properties are reversed.*

Intuitively, we first show that firewalls are strongly sanitizing by leveraging sanitizable homomorphic dual-mode encryption. Then, we simulate similarly to the original proof of [PVW08]. We defer the formal details to Appendix A.1.1.

Multi-session $\mathcal{F}_{\mathsf{sOT}}$. As discussed in Section 3.5.3, the generic framework of [PVW08] realizes the multi-session version of $\mathcal{F}_{\mathsf{OT}}$ (also denoted as $\hat{\mathcal{F}}_{\mathsf{OT}}$). Given that our protocol in Figure 5.4 has the same structure as the protocol of [PVW08], we observe that we can also reuse the same CRS across multiple runs, each with a distinct sub-session ID. The presence of subverted cores does not affect this property, as the sanitation operated from firewalls uses independently-sampled random strings for each sub-protocol run.

5.3 Sanitizing Randomized Equality

Next, we present our sanitized protocol for the (regular) randomized equality ideal functionality $\mathcal{F}_{\mathsf{RE}}$ that relies on authenticated channels (*i.e.*, $\mathcal{F}_{\mathsf{SAT}}$). In particular, we sanitize the RE-from-OT protocol of [CDVW12, Section 2.2] (summarized in Section 4.2.1) by using $\mathcal{F}_{\mathsf{sOT}}$, restricting to implicit mutual authentication as per the considerations in the technical overview. For ease of exposition, we depict the protocol in Figure 5.5 assuming 1-bit passwords. The n-bit password case runs exactly in the same way except that (i) it uses n OTs within the multi-session sanitizable OT functionality $\hat{\mathcal{F}}_{\mathsf{sOT}}$, and (ii) it computes keys using operator $*$ with n random strings rather than only one.

Compared to the non-sanitized protocol, whenever a core acts as a sender in a run of $\mathcal{F}_{\mathsf{OT}}$ by sampling two (potentially leaky) random strings, its respective firewall provides two uniformly-random strings to prevent exfiltration. We parameterize the input domain $\mathcal{I}$ and the respective blinding operation $*$, in line with the description of $\mathcal{F}_{\mathsf{sOT}}$.

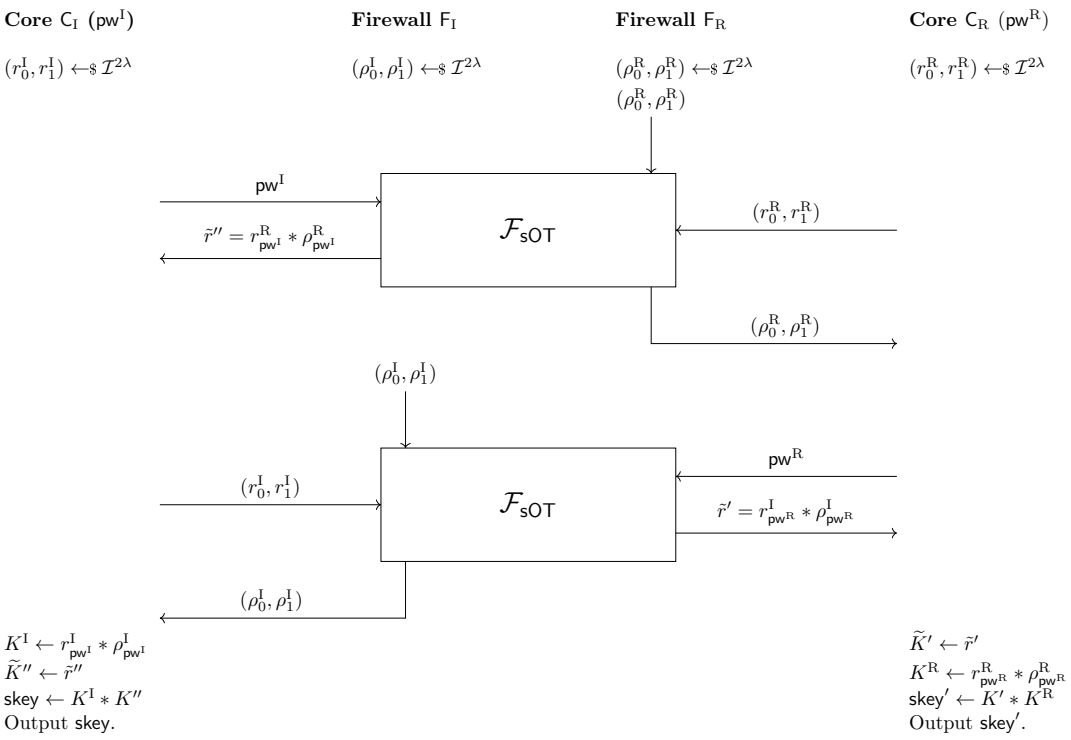


Figure 5.5: A sanitizing protocol for $\mathcal{F}_{\mathsf{RE}}$ from sanitizable OT with a 1-bit password.

Theorem 5.3. *The protocol in Figure 5.5 wsrUC-realizes the $\mathcal{F}_{\mathsf{RE}}$ ideal functionality in the $(\mathcal{F}_{\mathsf{sOT}}, \mathcal{F}_{\mathsf{SAT}})$-hybrid model under static corruptions.*

Within the proof, we first show strong sanitation of firewalls (which immediately follows by the structure of $\mathcal{F}_{\mathsf{sOT}}$), and then simulate similarly to [CDVW12]. We defer the formal proof and an explicit analysis of correctness to Appendix A.1.2.

5.4 Subversion-resilient Split Functionalities

In this subsection, we extend the characterization of split functionalities of [BCL+05] to the srUC framework. Informally, we want to show that, for *any* well-formed[20] regular n-party ideal functionality $\mathcal{F}$, there exists a

[20]The "well-formed" property is to rule out unrealistic functionalities as explained in [CLOS02, BCL+05].

protocol that realizes the n-party $s\mathcal{F}$ functionality with wsrUC-security in the CRS model. More formally, the goal of this section consists of proving an adaptation of [BCL+05, Theorem 10] to our setting, *i.e.*:

Theorem 5.4. *Let $\mathcal{F}$ be a (regular) n-party UC functionality. Then, assuming key-sanitizable signatures, there exists a protocol that securely realizes the n-party split functionality $s\mathcal{F}$ in the wsrUC model.*

Towards that, we follow the same strategy as [BCL+05] and proceed in three stages:

- *Link initialization*: The first step consists of building the *sanitizable* split-authenticated functionality $s\mathcal{F}_{\mathsf{SAT}}$ that parties will use to communicate on. The $s\mathcal{F}_{\mathsf{SAT}}$ functionality can be seen as the split version of the $\mathcal{F}_{\mathsf{SAT}}$ functionality.
- *Multi-session security*: As the second step, we show that when authenticated channels are available, any functionality can be "poly-realized" in the wsrUC model. Here, poly-realizing a functionality informally means that the security of a protocol implementing some functionality still holds even when multiple (*i.e.*, poly-many) instances of the protocol share the *same setup*. For that, we show that the subversion-resilient GMW protocol from [CMNV22] poly-realizes any functionality in the wsrUC model.
- *Unauthenticated channels*: Finally, we adapt the generic transformation of [BCL+05] that transforms any protocol π that n-realizes a n-party functionality $\mathcal{F}$ given authenticated channels (*i.e.*, $\mathcal{F}_{\mathsf{SAT}}$) in the wsrUC model into a protocol that realizes $s\mathcal{F}$ given access to $s\mathcal{F}_{\mathsf{SAT}}$ in the wsrUC model.

Next, we look at each of these stages individually towards proving Theorem 5.4.

5.4.1 Building link initialization

In what follows, we formally define $s\mathcal{F}_{\mathsf{SAT}}$ (*i.e.*, the split version of the functionality $\mathcal{F}_{\mathsf{SAT}}$ of [CMNV22] for sanitizable authenticated communication) and build a protocol that realizes it in the srUC model. For that, we introduce the notion of *key-sanitizable* signatures and show a matching instantiation by appropriately extending the BLS signature scheme [BLS01].

Description of $s\mathcal{F}_{\mathsf{SAT}}$. The $s\mathcal{F}_{\mathsf{SAT}}$ functionality has a similar structure to $\mathcal{F}_{\mathsf{SAT}}$, with the addition of having a link initialization phase. In contrast to $\mathcal{F}_{\mathsf{SAT}}$, the only guarantee provided by the functionality is that each party will be interacting with the same entity throughout the entire protocol run, but that entity could either be the expected party or the adversary itself. We describe $s\mathcal{F}_{\mathsf{SAT}}$ next.

Functionality $s\mathcal{F}_{\mathsf{SAT}}$

$s\mathcal{F}_{\mathsf{SAT}}$ is a sanitizable ideal functionality that interacts with an adversary $\mathcal{S}$ and a set of parties, each composed of a core C and a firewall F. The functionality consists of the following communication interfaces.

Initialization

- Upon activation with input $(\textsc{Init}, \mathsf{sid})$ from party P: Parse $\mathsf{sid} = (\mathcal{P}, \mathsf{sid}')$ where $\mathcal{P}$ is a set of parties that includes P. Forward $(\textsc{Init}, \mathsf{sid}, \mathsf{P})$ to the adversary $\mathcal{S}$.
- Upon receiving the message $(\textsc{Init}, \mathsf{sid}, \mathsf{P}, H, \mathsf{sid}_H)$, from $\mathcal{S}$: Verify that $H \subseteq \mathcal{P}$, that the list H of party identities includes $\mathsf{P} = (\mathsf{C}, \mathsf{F})$, and that for all recorded sets H' either (i) $H \cap H'$ contains only corrupted parties (as per the standard corruption transition table in Table 4.1) and $\mathsf{sid}_H \neq \mathsf{sid}_{H'}$, or (ii) $H' = H$ and $\mathsf{sid}_H = \mathsf{sid}_{H'}$. If any of the check fails, do nothing. Otherwise, output $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_H)$ to P and record (H, sid_H) if not yet recorded.

Message Authentication

- Upon receiving the message $(\textsc{Send}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, m)$ on IO_i where $\mathsf{P}_j \in \mathcal{P}$: Output the tuple on S_i.
- Upon receiving the message $(\textsc{Send}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, \widetilde{m})$ on S_i: Add the tuple to an (initially empty) list $\mathcal{W}$ of waiting messages. The same tuple can appear multiple times in the list. Then, leak the tuple to $\mathcal{S}$.
- Upon receiving the message $(\textsc{Deliver}, (\textsc{Send}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, \widetilde{m}))$ from $\mathcal{S}$:
  - If P_j did not previously receive an $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_H)$ output, do nothing.

– Else, if P_i is in the authentication set H of P_j, and P_i is uncorrupted, then: if there is a tuple $(\textsc{Send}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, \widetilde{m}) \in \mathcal{W}$, remove one appearance of the tuple from $\mathcal{W}$ and output $(\textsc{Receive}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, \widetilde{m})$ on S_j. Otherwise, do nothing.

– Else (*i.e.*, P_j received $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_H)$, and either P_i is corrupted or $\mathsf{P}_i \notin H$), output $(\textsc{Receive}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, \widetilde{m})$ on S_j, regardless of $\mathcal{W}$.

- Upon receiving the message $(\textsc{Receive}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, \hat{m})$ on S_j, output the tuple on IO_j.

Figure 5.6: Description of the sanitizable split-authenticated communication functionality $s\mathcal{F}_{\mathsf{SAT}}$.

Description. The functionality consists of a preliminary initialization phase and the actual message authentication phase. In the initialization phase, the adversary controls how parties will be partitioned in the respective authentication sets. Intuitively, parties within the same authentication set will be able to communicate as if there was an authenticated channel between them. It is however possible for the adversary to participate in different authentication sets on behalf of all corrupted parties and any party outside of that authentication set. In the message authentication phase, honest parties will transmit messages in an authenticated fashion within the same authentication set. However, they may very well receive messages out of the blue from the adversary on behalf of any party that is corrupted or outside the authentication set.

With respect to sanitation, whenever a core sends a message m with destination P_j on IO_i, the message is output on S_i. This means that m is output to a firewall that will decide if (and how) to sanitize m to $\widetilde{m}$ in any arbitrary way, without involving the functionality in the sanitation process. Once the firewall determines the message $\widetilde{m}$ to send to P_j, $\widetilde{m}$ is leaked to the adversary. According to the partition of parties performed in the link authentication phase, the adversary has different capabilities:

- If the recipient party is within the same authentication set, the message is added to a message queue, and the adversary can exclusively control its delivery time. This behaviour is indeed equivalent to $\mathcal{F}_{\mathsf{SAT}}$, in which the message is stored and then output to the recipient party whenever the adversary decides to do so.
- If P_i is corrupted or the parties are in different authentication sets, the adversary may deliver arbitrary messages to P_j, disregarding the message queue.

Whenever the adversary allows the delivery of a message, that message is output to the firewall F_j. Similarly to the sending phase, F_j may now modify the message arbitrarily without involving the functionality. Once a (potentially different) message $\hat{m}$ is determined by F_j, it is delivered by the functionality to C_j.

We stress that, exactly as in $\mathcal{F}_{\mathsf{SAT}}$, cores and their respective firewall are allowed to freely communicate through secure channels. This is achieved by means of Send messages (from a core to its firewall), and Receive messages (from a firewall to its core). In principle, a firewall may send back any message to its core, even if it was not related to any Deliver message.

Key-sanitizable signature schemes. In the construction of $\mathcal{F}_{\mathsf{SA}}$ of [BCL+05, Section 4.2] (reported in Section 4.1.2), parties exchange locally-generated keys and sign their messages in order to preserve the split-authenticated security of the communication channel. However, outbound verification keys have to be sanitized by firewalls: an operation that breaks correctness in the verification of the respective signatures. In order to overcome this limitation, we introduce a new notion that we call *key-sanitizable* signature schemes.

Informally, a *key-sanitizable* signature scheme allows mauling a verification key vk to $\widetilde{\mathsf{vk}}$ by means of an algorithm MaulVK that takes as input randomness ρ. The same randomness may be re-used by an algorithm $\mathsf{AlignSig}$ to align an (accepting) signature σ produced under secret key sk, producing a signature $\widetilde{\sigma}$ that verifies with mauled key $\widetilde{\mathsf{vk}}$. The latter operation should also be invertible, meaning that the signature σ may be recomputed from $\widetilde{\sigma}$ and ρ. We formally define this notion as a natural extension of traditional signatures in Definition 5.2, introducing a matching security notion in Definition 5.3 that extends EUF-CMA

security to account for the newly-introduced algorithms. This new security notion is implied in a black-box manner by any EUF-CMA scheme supporting the aforementioned algorithms.

Definition 5.2 (Key-sanitizable signature scheme)**.** *A key-sanitizable signature scheme consists of a tuple of algorithms* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{MaulVK}, \mathsf{AlignSig}, \mathsf{UnAlignSig})$*, with* $\mathcal{R}_{\mathsf{MaulVK}}$ *being the randomness space of the* MaulVK*,* AlignSig*, and* UnAlignSig *algorithms, with the following properties:*

1. ***Correctness:*** $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *constitutes a correct digital signature scheme, as per Definition 2.12.*
2. ***Consistent key sanitation:*** *For every* $(\mathsf{sk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda)$*, for every* $\rho \in \mathcal{R}_{\mathsf{MaulVK}}$*,* $\mathsf{MaulVK}(\mathsf{vk}, \rho)$ *outputs a new verification key* $\widetilde{\mathsf{vk}}$ *with the following property. For every* $\sigma \leftarrow\!\!\$\ \mathsf{Sign}(\mathsf{sk}, m)$ *with* $m \in \{0,1\}^n$*,* $\mathsf{AlignSig}((\mathsf{vk}, \sigma, m), \rho)$ *produces an accepting signature* $\widetilde{\sigma}$ *for message* m *verifiable by verification key* $\widetilde{\mathsf{vk}}$*, i.e.,* $\mathsf{Vrfy}(\widetilde{\mathsf{vk}}, \mathsf{AlignSig}((\mathsf{vk}, \sigma, m), \rho)) = 1$*, where* $\widetilde{\mathsf{vk}} = \mathsf{MaulVK}(\mathsf{vk}, \rho)$ *and* $\sigma = \mathsf{Sign}(\mathsf{sk}, m)$*.*
3. ***Alignment invertibility:*** *For every* $(\mathsf{sk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda)$*, for every* $\sigma \leftarrow\!\!\$\ \mathsf{Sign}(\mathsf{sk}, m)$ *with* $m \in \{0,1\}^n$*, for every* $\rho \in \mathcal{R}_{\mathsf{MaulVK}}$*, for every* $\widetilde{\mathsf{vk}} = \mathsf{MaulVK}(\mathsf{vk}, \rho)$*, for every* $\widetilde{\sigma} = \mathsf{AlignSig}((\mathsf{vk}, \sigma, m), \rho)$*, the algorithm* UnAlignSig *returns the original signature* σ*, i.e.,* $\mathsf{UnAlignSig}((\widetilde{\mathsf{vk}}, \widetilde{\sigma}, m), \rho) = \sigma$*.*

Game $\texttt{KS-EUF-CMA}_{\mathcal{A}}$		
1	$(\mathsf{sk}, \mathsf{vk}) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda)$	*Sample a keypair at random.*
2	$\rho \leftarrow\!\!\$\ \mathcal{R}_{\mathsf{MaulVK}}$	*Sample a random blinding factor.*
3	$\widetilde{\mathsf{vk}} \leftarrow \mathsf{MaulVK}(\mathsf{vk}, \rho)$	*Align* vk *to the blinding factor* ρ*.*
4	$M \leftarrow \emptyset$.	*Initialize the set of queried messages* M *with an empty set.*
5	$(m^*, \widetilde{\sigma}^*) \leftarrow \mathcal{A}(\widetilde{\mathsf{vk}}, \rho)^{\mathsf{Sign}_{\mathsf{sk}}(\cdot)}$	*Give* $\widetilde{\mathsf{vk}}$ *to* $\mathcal{A}$*, who outputs a signature* $\widetilde{\sigma}^*$ *for a message* m^**.* $\mathcal{A}$ *has oracle access to* $\mathsf{Sign}_{\mathsf{sk}}$*.*
4	Output $\mathsf{Vrfy}(\widetilde{\mathsf{vk}}, m^*, \widetilde{\sigma}^*) == 1 \wedge m \notin M$.	*Output 1 if a (forged) signature on a new message is valid.*
Oracle $\mathsf{Sign}_{\mathsf{sk}}(m)$		
1	Add m to the set of messages M.	
2	$\sigma \leftarrow\!\!\$\ \mathsf{Sign}(\mathsf{sk}, m)$	*Compute* σ *using the original* sk*.*
3	Output $\widetilde{\sigma} = \mathsf{AlignSig}((\mathsf{vk}, \sigma, m), \rho)$.	*Align* σ *to verify under* $\widetilde{\mathsf{vk}}$*.*

Game 7: Description of the KS-EUF-CMA game $\texttt{KS-EUF-CMA}_{\mathcal{A}}$.

Definition 5.3 (Key-sanitizable EUF-CMA security)**.** *Let* Π *be a key-sanitizable signature scheme, as per Definition 5.2. We say* Π *enjoys key-sanitizable existential unforgeability against chosen-message attacks if, for all PPT adversaries* $\mathcal{A}$*, the following holds:*

$$\Pr[\texttt{KS-EUF-CMA}_{\mathcal{A}} = 1] \leq \mathsf{negl}(\lambda).$$

Discussion. In the context of a KS-EUF-CMA scheme, the adversary may query the Sign oracle arbitrarily to obtain accepting signatures under the mauled verification key $\widetilde{\mathsf{vk}}$. Even by observing these signatures, the adversary cannot forge any signature that verifies under $\widetilde{\mathsf{vk}}$ for a non-queried message. This security property holds even if the blinding factor is known to the adversary (as it would be the case, *e.g.*, if a semi-honest firewall operates the sanitation).

Lemma 5.1. *Any EUF-CMA signature scheme that supports algorithms* MaulVK*,* AlignSig*, and* UnAlignSig*, as defined in Definition 5.2, is also KS-EUF-CMA.*

The proof consists of a black-box reduction to EUF-CMA, and is deferred to Appendix A.1.3.

Instantiation from BLS. In what follows, we augment the BLS signature scheme (reported in Definition 2.2) with the additional algorithms required by a key-sanitizable signature scheme:

- $\mathsf{MaulVK}(\mathsf{vk}, \rho) = \mathsf{vk}^{\rho}$.
- $\mathsf{AlignSig}((\mathsf{vk}, \sigma, m), \rho) = \sigma^{\rho}$.
- $\mathsf{UnAlignSig}((\mathsf{vk}, \widetilde{\sigma}, m), \rho) = \widetilde{\sigma}^{\rho^{-1}}$.

Theorem 5.5. *The BLS signature scheme [BLS01] with the additional algorithms specified above is a key-sanitizable signature scheme with KS-EUF-CMA security, assuming the CDH assumption holds in $\mathbb{G}$ and that H acts as a random oracle.*

Proof (Theorem 5.5). First, we inspect the MaulVK, AlignSig, and UnAlignSig algorithms we introduced, showing consistent key sanitation and alignment invertibility. Then, by observing that the BLS signature scheme is EUF-CMA (as shown in [BLS01]), we invoke Lemma 5.1 to prove that our scheme is also KS-EUF-CMA.

- **Consistent key sanitation.** Denoting $(\mathsf{sk}, \mathsf{vk}) = (x, g^x) \leftarrow\!\!\$\ \mathsf{KeyGen}(1^\lambda)$, $\sigma = H(m)^x \leftarrow \mathsf{Sign}(\mathsf{sk}, m)$ with $m \in \{0,1\}^n$, $\rho \leftarrow\!\!\$\ \mathcal{R}_{\mathsf{MaulVK}}$, $\widetilde{\mathsf{vk}} \leftarrow \mathsf{MaulVK}(\mathsf{vk}, \rho) = g^{x\rho}$, $\widetilde{\sigma} \leftarrow \mathsf{AlignSig}((\mathsf{vk}, \sigma, m), \rho) = (H(m)^x)^\rho$, the verification of signature $\widetilde{\sigma}$ for message m under $\widetilde{\mathsf{vk}}$ (*i.e.*, $\mathsf{Vrfy}(\widetilde{\mathsf{vk}}, (\widetilde{\sigma}, m)) : \hat{e}(\widetilde{\sigma}, g) == \hat{e}(H(m), \widetilde{\mathsf{vk}})$) succeeds:

$$\begin{aligned}\hat{e}(\widetilde{\sigma}, g) &= \hat{e}(H(m)^{x\rho}, g)\\ &= \hat{e}(H(m), g^{x\rho})\\ &= \hat{e}(H(m), \widetilde{\mathsf{vk}}).\end{aligned}$$

- **Alignment invertibility.** With the same notation used within the above proof, we have:

$$\mathsf{UnAlignSig}((\widetilde{\mathsf{vk}}, \widetilde{\sigma}, m), \rho) = (H(m)^{x\rho})^{\rho^{-1}} = H(m)^x = \mathsf{Sign}(\mathsf{sk}, m).$$

□

Realizing $s\mathcal{F}_{\mathsf{SAT}}$. We now describe a protocol that realizes $s\mathcal{F}_{\mathsf{SAT}}$, which follows a similar structure to that of [BCL+05, Section 4.2] (reported in Section 4.1.2). Again, for ease of exposition, we discuss the protocol in the 2-party setting. The n-party protocol runs the same, except that in the link initialization phase core C_i determines its session ID by collecting n verification keys and n signatures.

Link initialization phase. In the link initialization phase (depicted in Figure 5.7), the firewall blinds the outbound verification key vk_i to $\widetilde{\mathsf{vk}}_i$ using blinding factor ρ_i, and forwards it (supposedly) to party P_j. The blinded verification key $\widetilde{\mathsf{vk}}_i$ acts as an identifier for party P_i, and is therefore returned to the core. The core then uses sk_i to sign its SID $\widetilde{\mathsf{sid}}_i$, which contains the concatenation of $\widetilde{\mathsf{vk}}_i$ and vk_j (*i.e.*, the set of identifiers gathered by P_i for the initialization phase). Then, the firewall aligns the signature so that it verifies under verification key $\widetilde{\mathsf{vk}}_i$ and outputs it (supposedly) to party P_j, along with $\widetilde{\mathsf{sid}}_i$. Whenever the firewall receives a message $(\mathsf{sid}_j, \sigma_j)$, it first checks whether sid_j is consistent with $\widetilde{\mathsf{sid}}_i$, and whether the signature verifies. If not, the firewall aborts. Otherwise, the firewall initializes a counter *cnt*, which is forwarded back to the core, and the message authentication phase may be carried out.

Message authentication phase. In the message authentication phase (depicted in Figure 5.8), we distinguish the four interfaces in different interactions:

1. Whenever the core of the calling protocol inputs a message $(\textsc{Send}, \mathsf{P}_j, m)$ on IO_i, C_i outputs a respective Send message on S_i (*i.e.*, to the firewall of the calling protocol).

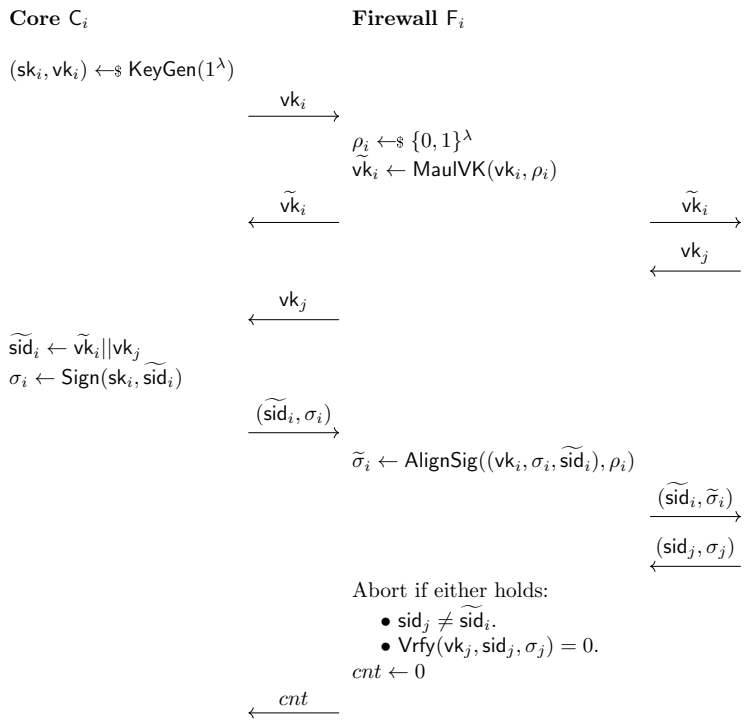


Figure 5.7: A sanitizing protocol implementing the link initialization phase of $s\mathcal{F}_{\mathsf{SAT}}$.

2. Whenever the firewall of the calling protocol inputs a message $(\textsc{Send}, \mathsf{P}_j, \widetilde{m})$ on S_i, F_i sends the message back to its core C_i, who produces a signature σ_i using secret key sk_i and forwards σ_i to F_i. F_i then aligns the signature to $\widetilde{\sigma}_i$ using the same ρ_i of the link initialization phase, updates the counter, and sends the message and the signature (supposedly) to party P_j.

3. Whenever F_i receives a message $\widetilde{m}$ (supposedly) coming from party P_j, it first checks whether the SID matches the one of the core, whether the destination party matches the local party, whether the signature verifies, and whether a message with the same counter was not already received. If any check fails, the message is dropped. Otherwise, the message is output on S_i (*i.e.*, to the firewall of the calling protocol).

4. Whenever the firewall of the calling protocol inputs a message $(\textsc{Receive}, \mathsf{P}_j, \hat{m})$ on S_i, C_i outputs a respective Receive message on IO_i (*i.e.*, to the core of the calling protocol).

Theorem 5.6. *The protocol depicted in Figures 5.7, 5.8 realizes the $s\mathcal{F}_{\mathsf{SAT}}$ functionality, assuming a deterministic KS-EUF-CMA signature scheme and the presence of secure channels between cores and their respective firewall.*

Intuitively, the proof runs as the one for the non-sanitized protocol of [BCL+05], except that the blinding operations of firewalls thwart subversion attacks, and consistency between keys is obtained by using each party's sanitized key. We defer the formal proof to Appendix A.1.4.

5.4.2 Multi-realizing any ideal functionality in the wsrUC model

Next, we focus on showing that any ideal functionality may be multi-realized in the wsrUC model. Formally, we prove the following lemma:

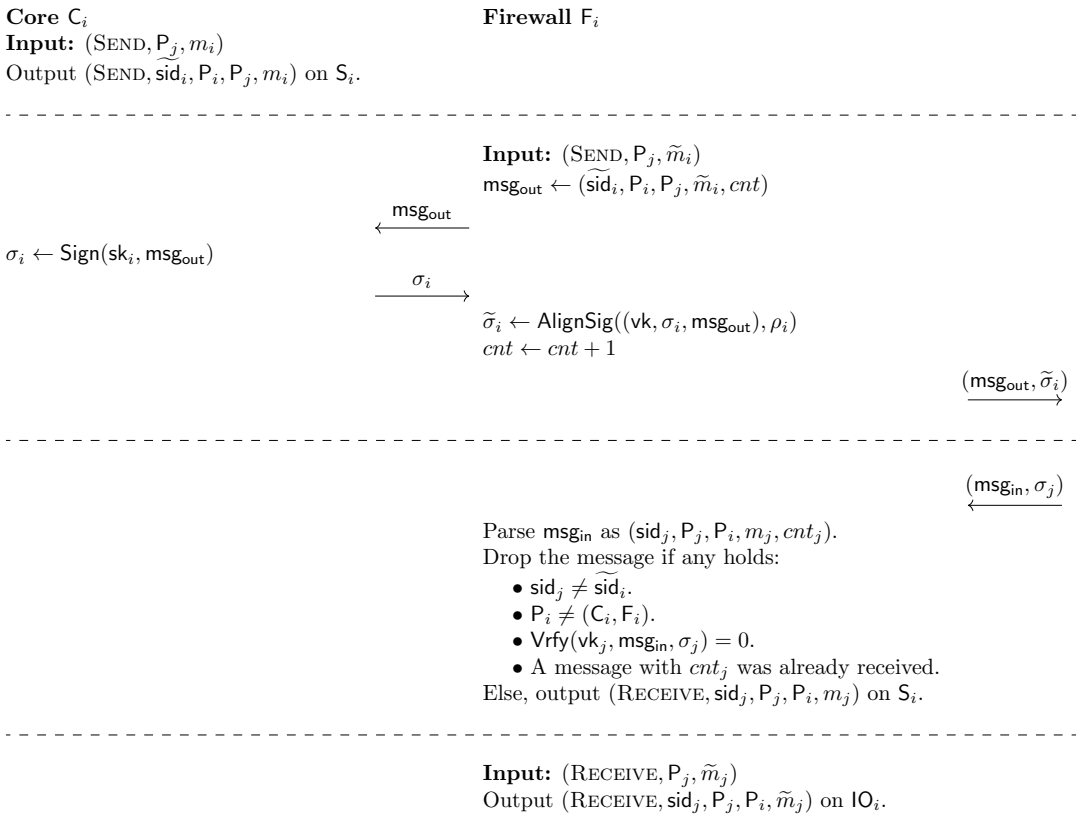


Figure 5.8: A sanitizing protocol implementing the message authentication phase of $s\mathcal{F}_{\mathsf{SAT}}$, split in each of the interfaces.

Lemma 5.2. *For any regular (well-formed) ideal functionality $\mathcal{F}$ there exists a protocol π that n-realizes $\mathcal{F}$ in the wsrUC model assuming authenticated channels in the presence of static and malicious adversaries for $n \in poly(\lambda)$. Moreover, the protocol π is such that all instances of π use a single instance of $\mathcal{F}_{\mathsf{crs}}$.*

Informally, such a protocol can be obtained from the adaptation of the GMW compiler to the srUC framework shown in [CMNV22]. The formal proof of the lemma is essentially [BCL+05, Theorem 13] verbatim, except that we replace results for the UC framework with their counterparts in the srUC framework, shown in [CMNV22] (*e.g.*, the UC composition theorem and the GMW compiler). We provide details in Appendix A.1.5.

5.4.3 Realizing generic split functionalitites

We finally show that any protocol π that wsrUC-n-realizes an n-party functionality $\mathcal{F}$ in the $\mathcal{F}_{\mathsf{SAT}}$-hybrid model (*i.e.*, using authenticated channels) can be compiled into a protocol Π that wsrUC-realizes the split n-party functionality $s\mathcal{F}$ in the $s\mathcal{F}_{\mathsf{SAT}}$-hybrid model (*i.e.*, using unauthenticated channels). The $s\mathcal{F}$ functionality is exactly the same as in Figure 4.7. Indeed, since we wsrUC-realize a *regular* ideal functionality $\mathcal{F}$ assuming $\mathcal{F}_{\mathsf{SAT}}$, our end goal is to wsrUC-realize the split counterpart of $\mathcal{F}$ assuming $s\mathcal{F}_{\mathsf{SAT}}$, which is also a *regular* ideal functionality.

Lemma 5.3. *Let $\mathcal{G}$ be a setup functionality, let $\mathcal{F}$ be a n-party ideal functionality, and let $\pi_{\mathcal{F}}$ be a protocol that securely n-realizes $\mathcal{F}$ in the wsrUC model with authenticated communication (i.e., $\mathcal{F}_{\mathsf{SAT}}$) and a single instance of $\mathcal{G}$. Then, there exists a protocol $\Pi_{\mathcal{F}}$ wsrUC-realizing the split functionality $s\mathcal{F}$ using a single instance of $s\mathcal{F}_{\mathsf{SAT}}$ and a single instance of $\mathcal{G}$.*

To prove this theorem, we adapt the proof of [BCL+05, Lemma 4.1] to the wsrUC model. First, we describe the protocol $\Pi_{\mathcal{F}}$, which is obtained by adapting the compiler presented in [BCL+05]. In particular, the compiler of [BCL+05] transforms a protocol $\pi_{\mathcal{F}}$ realizing functionality $\mathcal{F}$ in the UC $\mathcal{F}_{\mathsf{MAUTH}}$-hybrid model into a protocol $\Pi_{\mathcal{F}}$ realizing functionality $s\mathcal{F}$ in the UC $\mathcal{F}_{\mathsf{SA}}$-hybrid model. This result can be mapped to our setting by replacing $\mathcal{F}_{\mathsf{MAUTH}}$ with $\mathcal{F}_{\mathsf{SAT}}$, and $\mathcal{F}_{\mathsf{SA}}$ with $s\mathcal{F}_{\mathsf{SAT}}$, with the crucial detail that messages coming from $s\mathcal{F}_{\mathsf{SAT}}$ are forwarded to the instance of the protocol $\pi_{\mathcal{F}}$ on the appropriate interface (*i.e.*, IO or S), rather than having a single interface for each party. Then, we simply follow the proof of [BCL+05, Lemma 4.1] accounting for the additional communication between cores and firewalls and for the presence of specious cores, as per the srUC framework. We defer the description of $\Pi_{\mathcal{F}}$ and the formal proof to Appendix A.1.6.

5.4.4 Putting it all together

We showed that the characterization of split functionalities of [BCL+05] can be cast in the subversion-resilient UC model in the same way as in standard UC. Namely, one can build a protocol n-realizing a regular functionality for the authenticated channel setting and simply invoke Lemma 5.3 to obtain security of the split version of the protocol in the unauthenticated channel setting. Since there exists a protocol n-realizing *any* regular ideal functionality in the authenticated setting (by using the srUC GMW compiler of [CMNV22], as per Lemma 5.2), there also exists a matching n-party protocol in the unauthenticated setting realizing the split version of the same functionality, yielding Theorem 5.4.

5.5 Sanitizing PAKE

So far we only referred to the $\mathcal{F}_{\mathsf{RE}}$ functionality, in which the adversary is unable to perform any (online) password guesses. To overcome this limitation, we follow the blueprint presented in Section 4.2.4 to obtain a protocol for $\mathcal{F}_{\mathsf{PAKE}}$. First, we argue that our protocol in Section 5.3 can be compiled in a protocol that realizes $s\mathcal{F}_{\mathsf{RE}}$. This is achieved by invoking Lemma 5.3. Then, we show that $s\mathcal{F}_{\mathsf{RE}}$ is sufficient to realize $\mathcal{F}_{\mathsf{PAKE}}$. We conclude the section by highlighting that it is also possible to obtain a protocol for $s\mathcal{F}_{\mathsf{RE}}$ by using the general-purpose result given by Theorem 5.4 (which internally relies on the srUC GMW compiler). In that regard, we provide a hand-wavy performance comparison of such a protocol with our instantiation from DDH.

5.5.1 From $\mathcal{F}_{\mathsf{RE}}$ to $\mathcal{F}_{\mathsf{PAKE}}$

The protocol we presented in Section 5.3 realizes $\mathcal{F}_{\mathsf{RE}}$ in the presence of subversion attacks in the authenticated setting. Proceeding as [CDVW12], we convert it to a protocol for $s\mathcal{F}_{\mathsf{RE}}$, obtaining the following theorem:

Theorem 5.7. *There exists a protocol that wsrUC-realizes the $s\mathcal{F}_{\mathsf{RE}}$ ideal functionality in the $(\mathcal{F}_{\mathsf{crs}}, s\mathcal{F}_{\mathsf{SAT}})$-hybrid model under static corruptions. The protocol is based on the DDH assumption, runs in a constant number of rounds, and has a communication complexity of $O(n)$ group elements per session key.*

Proof (Theorem 5.7). The proof of this theorem is the proof of [CDVW12, Theorem 2] verbatim. First, we observe that the multi-session version of $\mathcal{F}_{\mathsf{RE}}$ can be implemented by having access to the multi-session version of $\mathcal{F}_{\mathsf{sOT}}$ (each new session of $\mathcal{F}_{\mathsf{RE}}$ uses a new invocation of the protocol for $\mathcal{F}_{\mathsf{sOT}}$). Then, we observe that our protocol in Section 5.2 implements the multi-session version of $\mathcal{F}_{\mathsf{sOT}}$ in the $\mathcal{F}_{\mathsf{crs}}$-hybrid model. Hence, we can invoke Lemma 5.3, which allows us to replace $\mathcal{F}_{\mathsf{SAT}}$ with $s\mathcal{F}_{\mathsf{SAT}}$, yielding a protocol for the split version of randomized equality (*i.e.*, $s\mathcal{F}_{\mathsf{RE}}$). □

All that remains to show is that $\mathcal{F}_{\mathsf{PAKE}}$ can be instantiated from $s\mathcal{F}_{\mathsf{RE}}$ in the presence of subversion attacks. More formally:

Theorem 5.8. *There exists a protocol in the $s\mathcal{F}_{\mathsf{RE}}$-hybrid model that instantiates $\mathcal{F}_{\mathsf{PAKE}}$ in the presence of subversion attacks.*

[DHP+18] exhibits a trivial protocol in the $s\mathcal{F}_{\mathsf{RE}}$-hybrid model that realizes $\mathcal{F}_{\mathsf{PAKE}}$. In particular, their protocol exclusively interacts with $s\mathcal{F}_{\mathsf{RE}}$. This fact allows to port their protocol and its related proof to our setting in a straight-forward manner, as intuitively such a protocol inherits the structure and the security properties of $s\mathcal{F}_{\mathsf{RE}}$. We report the formal proof in Appendix A.1.7.

5.5.2 A hand-wavy performance comparison

An alternative route to realize $\mathcal{F}_{\mathsf{PAKE}}$ consists of invoking Theorem 5.4 to obtain a protocol wsrUC-realizing $s\mathcal{F}_{\mathsf{RE}}$, and then applying the transformation of Theorem 5.8. In particular, as per Lemma 5.2, this protocol relies on the srUC GMW compiler of [CMNV22]. In order to establish an informal comparison with our instantiation from DDH (given by Theorem 5.7), we first observe that both these protocols rely on Lemma 5.3 to move from the authenticated setting to the unauthenticated setting. Hence, it suffices to compare the protocols in the authenticated setting. For our hand-wavy comparison, we compare round complexity and communication complexity.

Our instantiation from DDH, as per Figure 5.5, essentially relies on n runs of $\mathcal{F}_{\mathsf{sOT}}$ that share the same CRS, where n is the bit-length of the password. By our specific instantiation of $\mathcal{F}_{\mathsf{sOT}}$, each party sends 1 public key and 2 SHDME encryptions (= 4 group elements) for each bit of the password. Hence, our protocol runs in 2 rounds (by batching messages for sOTs) with a communication complexity of $O(n)$ group elements.

On the other hand, the instantiation from the srUC GMW compiler requires each party to (i) generate its random tape jointly with its peer; (ii) commit to its input; (iii) prove in zero-knowledge that each step of a semi-honest protocol realizing $\mathcal{F}_{\mathsf{RE}}$ was executed correctly. (i) requires 3 rounds: 1 for committing to some locally-generated randomness and 2 for the coin tossing functionality. (ii) requires 1 round. (iii) requires at least the same number of rounds of a semi-honest execution of an r-round protocol realizing $\mathcal{F}_{\mathsf{RE}}$. Hence, we end up with at least $4 + r$ rounds. We then observe that the coin tossing functionality of [CMNV22, Section 4] relies on the sanitizable commitment functionality (presented in [CMNV22, Section 3]), which is realized by computing and forwarding bit-wise commitments (each containing 2 group elements) under the DDH assumption. Given that the input to the semi-honest instantiation of $\mathcal{F}_{\mathsf{RE}}$ is an n-bit password, and that the random strings used to generate the random tape have size λ, the communication complexity of the first two steps is already $O(n + \lambda)$.

We conclude that our instantiation from DDH has a better round and communication complexity even prior to the run of the compiled semi-honest instantiation of $\mathcal{F}_{\mathsf{RE}}$ of the protocol from GMW. We further remark that, in step (iii), the protocol from GMW requires the generation of rerandomizable NIZK arguments for each message of the protocol, increasing the complexity further.

Part III

PAKE From Smooth-Projective Hashing

Part Overview

In the previous section, we introduced our first srUC PAKE protocol by first instantiating randomized equality from subversion-resilient Oblivious Transfer, and then replacing authenticated channels with split authentication by appropriately sanitizing the generic transformation due to [BCL+05]. In spite of its high modularity, this construction is still relatively inefficient, both in terms of round complexity and communication complexity.

In this portion of the composition, we aim at improving the aforementioned performance metrics by designing a different srUC PAKE protocol geared towards efficiency. Towards this goal, we first present further technical background pertaining smooth-projective hashing that complements the modelling techniques described in Section 4.

Organization. In Section 6, we introduce further technical tools that will be of interest for the design of our more efficient srUC PAKE protocol. In particular, we focus on giving a long-winded introduction to smooth-projective hashing and related extensions, such as *trapdoor* smooth-projective hashing and *malleable* smooth-projective hashing, summarizing how this primitive can be used to instantiate the efficient 1-round UC PAKE protocol of [BP13].

In Section 7, we comprehensively discuss our second srUC PAKE protocol, which is obtained from sanitizing the aforementioned 1-round PAKE protocol.

6 Background

This section complements the modelling techniques presented in Section 4 by introducing relevant background required for the design of our (more efficient) subversion-resilient PAKE from smooth-projective hashing. As a salient modelling difference, this protocol will be designed in the bare model (*i.e.*, *without* constructing split authentication first).

In Section 6.1, we summarize further background related to PAKE and NIZK. In Section 6.2, we introduce smooth-projective hashing. In Section 6.3, we discuss *trapdoor* smooth-projective hashing, as defined in [BP13], highlighting how this primitive can be used to instantiate a 1-round UC-secure PAKE protocol in Section 6.4. In Section 6.5, we summarize *malleable* smooth-projective hashing, as defined in [CMY+16]. As a build-up for the constructions that will be part of our srUC PAKE protocol, we enrich this section by reporting relevant instantiations for all these different properties alongside their related security proofs.

6.1 Preliminaries

In what follows, we introduce an alternative description of $\mathcal{F}_{\mathsf{PAKE}}$, as well as more advanced properties pertaining NIZK.

6.1.1 An alternative description of $\mathcal{F}_{\mathsf{PAKE}}$

For notational convenience, throughout this portion of the composition we consider an alternative (but equivalent) description of $\mathcal{F}_{\mathsf{PAKE}}$ compared to that of Section 4.2.3, which does not explicitly distinguish initiators and responders.

Functionality $\mathcal{F}_{\mathsf{PAKE}}$

The functionality $\mathcal{F}_{\mathsf{PAKE}}$ is parameterized by a security parameter λ, and it interacts with parties P, P', and the adversary $\mathcal{S}$ via the following queries:

Upon receiving a query $(\textsc{NewSession}, \mathsf{sid}, \mathsf{P}, \mathsf{P}', \mathsf{pw})$ **from** P**:**

Record $(\mathsf{P}, \mathsf{P}', \mathsf{pw})$, mark it as fresh, and leak $(\mathsf{sid}, \mathsf{P}, \mathsf{P}')$ to $\mathcal{S}$. Ignore all future messages from P.

Upon receiving a query $(\textsc{TestPwd}, \mathsf{sid}, \mathsf{P}, \widetilde{\mathsf{pw}})$ **from the adversary** $\mathcal{S}$**:**

If there exists a record of the form $(\mathsf{P}, \cdot, \mathsf{pw})$ which is fresh, then:

- If $\widetilde{\mathsf{pw}} = \mathsf{pw}$, mark the record as $\mathsf{compromised}$ and return "correct guess" to $\mathcal{S}$.
- If $\widetilde{\mathsf{pw}} \neq \mathsf{pw}$, mark the record as $\mathsf{interrupted}$ and return "wrong guess" to $\mathcal{S}$.

Upon receiving a query $(\textsc{NewKey}, \mathsf{sid}, \mathsf{P}, K)$ **from** $\mathcal{S}$**, where** $|K| = \lambda$**:**

If there is a record of the form $(\mathsf{P}, \mathsf{P}', \mathsf{pw})$ that is not marked as $\mathsf{completed}$, then:

- If any of the following conditions hold, output (sid, K) to party P:
  - P is corrupted.
  - This record is fresh, there exists a record $(\mathsf{P}', \mathsf{P}, \mathsf{pw}')$ with $\mathsf{pw}' = \mathsf{pw}$, and P' is corrupted.
  - This record is $\mathsf{compromised}$.
- If this record is fresh, both parties are honest, and there exists a record $(\mathsf{P}', \mathsf{P}, \mathsf{pw}')$ with $\mathsf{pw}' = \mathsf{pw}$, choose $\mathsf{skey} \leftarrow\!\!\$\ \{0,1\}^\lambda$. Output skey to P, and append skey to the record $(\mathsf{P}, \mathsf{P}', \mathsf{pw})$.
- If this record is fresh, both parties are honest, and there exists a record $(\mathsf{P}, \mathsf{P}', \mathsf{pw}', \mathsf{skey})$ with $\mathsf{pw}' = \mathsf{pw}$, output skey to P.
- If none of the above rules apply, choose $\mathsf{skey}' \leftarrow\!\!\$\ \{0,1\}^\lambda$ and output it to party P.

In any case, mark the record $(\mathsf{P}, \cdot, \mathsf{pw})$ as $\mathsf{completed}$.

Figure 6.1: An alternative description of the PAKE functionality $\mathcal{F}_{\mathsf{PAKE}}$.

6.1.2 Controlled-malleable NIZKs

Looking ahead, our subversion-resilient PAKE protocol will have cores generating NIZK proofs for statements that get sanitized by firewalls. Of course, we need to ensure that these firewalls can sanitize proofs consistently with the sanitation operated on the statements. For that, we leverage *controlled malleable* NIZKs, as introduced in [CKLM12]. We first briefly recap the notion of "malleable" proof systems, and later discuss how simulation extractability (Definition 2.21) can be adapted to account for the newly-introduced malleability.

Malleable proof systems. Let $\mathscr{R}$ be a relation for an NP-language $\mathcal{L}$. Informally, a malleable proof system allows transforming an accepting proof π for a statement x to an accepting proof π' for a (potentially different) statement x', where x' is obtained from x by applying some transformation T. This transformation should be *admissible*, meaning that any $(x, w) \in \mathscr{R}$ should be mapped to a $(x', w') \in \mathscr{R}$.

Definition 6.1 (Admissible transformation [CKLM12])**.** *Let $\mathscr{R}$ be a binary relation. Let $T = (T_x, T_w)$ be a tuple of efficiently computable functions, where $T_x, T_w : \{0,1\}^* \to \{0,1\}^*$. We say T is admissible for $\mathscr{R}$ if, for any $(x, w) \in \mathscr{R}$, $(T_x(x), T_w(w)) \in \mathscr{R}$.*

As a concrete example, let us consider the language of ElGamal tuples $\mathcal{L} = \{(g, h, u, v) \mid \exists r \text{ s.t. } u = g^r \wedge v = h^r\}$. An admissible transformation $T = (T_x, T_w)$ for $\mathscr{R}_\mathcal{L}$ for some r' is such that $T_x(g, h, u, v) = (g, h, u \cdot g^{r'}, v \cdot h^{r'})$ and $T_w(r') = r + r'$. Hence, for a statement $(x, w) \in \mathscr{R}_\mathcal{L}$, $(T_x(x), T_w(w)) \in \mathscr{R}_\mathcal{L}$.

Definition 6.2 (Allowable set of transformations [CKLM12])**.** *Let $\mathcal{T}$ be a set of transformations. We say $\mathcal{T}$ is allowable for $\mathscr{R}$ if, for every $T \in \mathcal{T}$, T is admissible for $\mathscr{R}$.*

With respect to the previous example, each transformation T relates to a specific r'. A set of allowable transformations $\mathcal{T}$ contains, *e.g.*, the transformations defined for every r'.

Definition 6.3 (Malleable non-interactive proof system)**.** *Let $\Pi \in \mathcal{F}_{\mathsf{NI}}$, and let $\mathcal{T}$ be a set of allowable transformations over $\mathcal{L}$. We say Π is a malleable non-interactive proof system with respect to $\mathcal{T}$ if there exists an algorithm ZKEval such that, for all crs, x, π such that $\mathcal{V}(\mathsf{crs}, x, \pi) = 1$, the following holds:*

$$\Pr\left[\ T \in \mathcal{T} \wedge\ \mathcal{V}(\mathsf{crs}, T_x(x), \widetilde{\pi}) = 1 \ \middle|\ \widetilde{\pi} \leftarrow\!\$\ \mathsf{ZKEval}(\mathsf{crs}, T, \{x, \pi\})\ \right] = 1.$$

Strong derivation privacy. In proving security of certain protocols, mauled proofs should be indistinguishable from freshly-simulated proofs. In particular, the output of ZKEval mauling an accepting proof π on statement x with transformation T should be indistinguishable from a fresh simulated proof computed on statement $T_x(x)$. This requirement is captured with the notion of strong derivation privacy, which we report in what follows.

Definition 6.4 (Strong derivation privacy [CKLM12])**.** *Let Π be a malleable proof system, as per Definition 6.3, that additionally satisfies soundness and zero-knowledge. We say Π enjoys strong derivation privacy if, for every PPT adversary $\mathcal{A}$, the following holds:*

$$\Pr\left[\begin{array}{c}\mathcal{V}(\mathsf{crs}, x, \pi) = 1 \wedge T \in \mathcal{T} \\ \wedge\ b = b'\end{array} \middle|\ \begin{array}{l}(\mathsf{crs}, \tau_s) \leftarrow \mathcal{S}_0(1^\lambda) \\ (x, w, \pi, T) \leftarrow \mathcal{A}(\mathsf{crs}, \tau_s) \\ b \leftarrow\!\$\ \{0,1\} \\ \widetilde{\pi} \leftarrow\!\$ \begin{cases}\mathsf{ZKEval}(\mathsf{crs}, T, \{x, \pi\}) & \text{if } b = 0 \\ \mathcal{S}_1(\mathsf{crs}, T_x(x), \tau_s) & \text{if } b = 1\end{cases} \\ b' \leftarrow \mathcal{A}(\widetilde{\pi})\end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Looking ahead, this property will be particularly convenient to ensure that, even though an accepting (subverted) π might leak information, a firewall running ZKEval outputs a proof on the mauled statement that is indistinguishable from a freshly-simulated proof (which, in turn, does no longer leak any information).

Controlled-malleable simulation-sound extractability. In the context of malleable proof systems, the notion of simulation-sound extractability is no longer meaningful. Indeed, if $\mathcal{L}$ is an NP-language, the adversary can query the simulator to obtain an accepting proof π' on $x' \notin \mathcal{L}$, and then maul a proof with the allowable transformation to obtain an accepting proof for which the extractor will fail to extract (as no witness exists for such a statement).

For that, [CKLM12] coined the weaker notion of *controlled-malleable* SSE, which essentially adapts SSE to account for the malleability of proof systems. In particular, it is acceptable for the extractor to either output a witness, or a previously-queried statement alongside an allowable transformation $T = (T_x, T_w) \in \mathcal{T}$. This way, if the adversary attempts the aforementioned attack, the extractor will successfully extract a tuple (x, T), where x is a previously-queried statement such that $x' = T_x(x)$ (*i.e.*, the adversary queried the simulator on x and mauled the respective proof and statement with allowable transformation $T \in \mathcal{T}$). We provide a formal definition in what follows.

Definition 6.5 (Controlled-malleable simulation-sound extractability [CKLM12])**.** *Let* Π *be a malleable proof system, as per Definition 6.3, that additionally satisfies soundness, zero-knowledge, and proof-of-knowledge. Let* $\mathcal{SE} = (\mathcal{SE}_0, \mathcal{SE}_1)$ *be an algorithm that, on input* 1^λ*, returns* $(\mathsf{crs}, \tau, \xi)$ *such that* (crs, τ) *is identically distributed to the output of* $\mathcal{S}_0$*. We say* Π *enjoys controlled-malleable simulation-sound extractability if, for every PPT adversary, the following holds:*

$$\Pr\left[\begin{array}{c} \mathcal{V}(\mathsf{crs}, x', \pi') = 1 \wedge (x', \pi') \notin Q \\ \wedge \\ (w' \neq \bot \wedge (x', w') \notin \mathscr{R}_{\mathcal{L}} \\ \vee \\ (x, T) \neq (\bot, \bot) \wedge (x \notin Q_x \vee x' \neq T_x(x) \vee T \notin \mathcal{T}) \\ \vee \\ (w', x, T) = (\bot, \bot, \bot)) \end{array} \middle| \begin{array}{l} (\mathsf{crs}, \tau, \xi) \leftarrow \mathcal{SE}_0(1^\lambda) \\ (x', \pi') \leftarrow \mathcal{A}^{\mathcal{S}_1(\mathsf{crs}, \tau, \cdot)}(\mathsf{crs}, \xi) \\ (w', x, T) \leftarrow \mathcal{E}_1(\mathsf{crs}, \xi, x, \pi) \end{array} \right] \leq \mathsf{negl}(\lambda),$$

where Q *contains the replies of the oracle* $\mathcal{S}_1$*, and* Q_x *contains the queried statements.*

6.2 Smooth-projective hash functions

A family of smooth-projective hash functions is a family of hash functions defined over some domain Set alongside a hard-membership NP-language L. In particular, the elements of Set are partitioned according to their membership with respect to L. The salient feature of SPHFs lies in allowing the evaluation of hashes in two different ways by means of asymmetric keypairs. Hashing keys (the "private" keys) may be used to evaluate the hash for any element in the domain (*i.e.*, even for statements not in the language), whereas projection keys (the "public" keys) allow the evaluation of the hash only for statements in the language. The latter procedure may be invoked only by having access to the witness of the respective statement. Smoothness ensures that, if the projection key is used with a statement not in the language, it will lead to the evaluation a uniformly-distributed hash (rather than the same hash evaluated by using the respective hashing key).

Sets of NP languages. Several definitions of SPHFs (including the ones within this composition) consider a *set* of hard-membership NP languages $\mathtt{L} = \{L_{\mathsf{lpp},\mathsf{aux}}\}$ (rather than a single language L), whose spaces of statements share the same domain Set. lpp represents some public setup parameter (*e.g.*, a CRS) and aux represents private information (*e.g.*, the plaintext of an ElGamal ciphertext). We denote with $\mathtt{R} = \{R_{\mathsf{lpp},\mathsf{aux}}\}$ the set of respective NP-relations for language membership. In what follows, we report two sets of languages of interest for the remainder of this section:

- The set of languages of DH tuples $\mathtt{L}^{\mathsf{DH}}$ contains languages $L_{\mathsf{lpp},\mathsf{aux}} = \{(u, v) \mid \exists w \text{ s.t. } u = g^w \wedge v = h^w\}$, where $\mathsf{Set} = \mathbb{G}^{1\times 2}$, $\mathsf{lpp} = (g, h)$ and $\mathsf{aux} = \bot$. The respective set of NP-relations $\mathtt{R} = \{R_{\mathsf{lpp},\mathsf{aux}}\}$ contains relations $R_{\mathsf{lpp},\mathsf{aux}}((u, v), w)$ that output 1 if $u = g^w \wedge v = h^w$.

- The set of languages of ElGamal ciphertexts $\mathtt{L}^{\mathsf{ElGamal}}$ contains languages $L_{\mathsf{lpp},\mathsf{aux}} = \{(c_1, c_2) \mid \exists w \text{ s.t. } c_1 = g^w \wedge c_2 = m \cdot h^w\}$, where $\mathsf{Set} = \mathbb{G}^{1\times 2}$, $\mathsf{lpp} = (g, h)$ and $\mathsf{aux} = m$. The respective set of NP-relations $\mathtt{R} = \{R_{\mathsf{lpp},\mathsf{aux}}\}$ contains relations $R_{\mathsf{lpp},\mathsf{aux}}((c_1, c_2), w)$ that output 1 if $c_1 = g^w \wedge c_2 = m \cdot h^w$.

Definition 6.6 (Projective hash function (PHF))**.** *Let* $\mathtt{L} = \{L_{\mathsf{lpp},\mathsf{aux}}\}$ *be a set of NP languages defined over a domain* Set*, and let* $\mathtt{R} = \{R_{\mathsf{lpp},\mathsf{aux}}\}$ *be the respective NP relations for language membership. Let* $\Pi_{\mathsf{SPHF}} = (\mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash})$ *be a tuple of algorithms defined as follows:*

- $\mathsf{Setup}(1^\lambda)$*: Upon input a unary description of the security parameter* λ*, output public parameters* pp*.* pp *includes* lpp*, which identifies a set of languages* $L_{\mathsf{lpp},\mathsf{aux}}$*.*
- $\mathsf{HashKG}(\mathsf{pp}, \mathsf{aux})$*: Upon input public parameters* pp *and auxiliary information* aux*, output an hashing key* hk*.*
- $\mathsf{ProjKG}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk})$*: Upon input public parameters* pp*, auxiliary information* aux*, and a hashing key* hk*, output a projection key* hp*.*[21]
- $\mathsf{Hash}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk}, x)$*: Upon input public parameters* pp*, auxiliary information* aux*, a hashing key* hk*, and a statement* x*, output a hash value* $\mathsf{hv} \in \mathcal{Y}$*.*
- $\mathsf{ProjHash}(\mathsf{pp}, \mathsf{aux}, \mathsf{hp}, x, w)$*: Upon input public parameters* pp*, auxiliary information* aux*, a projection key* hp*, a statement* x*, and a witness* w*, output a hash value* $\mathsf{hv} \in \mathcal{Y}$*.*

We say Π_{SPHF} *is a correct projective hash function if, for all* aux *and for all* $(x, w) \in R_{\mathsf{lpp},\mathsf{aux}}$*, the following holds:*

$$\Pr\left[\mathsf{Hash}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk}, x) = \mathsf{ProjHash}(\mathsf{pp}, \mathsf{aux}, \mathsf{hp}, x, w) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{hk} \leftarrow \mathsf{HashKG}(\mathsf{pp}, \mathsf{aux}) \\ \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk}) \end{array}\right] = 1$$

Definition 6.7 (Smooth-projective hash function (SPHF))**.** *Let* Π_{PHF} *be a correct projective hash function, as per Definition 6.6. We say* Π_{PHF} *is a SPHF with perfect smoothness if, for all* $x \notin L_{\mathsf{lpp},\mathsf{aux}}$*, the following holds:*

$$\left[(\mathsf{hv}, \mathsf{hp}) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda) \\ \mathsf{hk} \leftarrow\!\!\$\ \mathsf{HashKG}(\mathsf{pp}, \mathsf{aux}) \\ \mathsf{hv} \leftarrow \mathsf{Hash}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk}, x) \\ \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk}) \end{array}\right] \equiv \left[(\mathsf{hv}, \mathsf{hp}) \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda) \\ \mathsf{hk} \leftarrow\!\!\$\ \mathsf{HashKG}(\mathsf{pp}, \mathsf{aux}) \\ \mathsf{hv} \leftarrow\!\!\$\ \mathcal{Y} \\ \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk}) \end{array}\right]$$

6.2.1 SPHFs for DH tuples and ElGamal ciphertexts

As a warm-up, we provide simple SPHFs for $\mathtt{L}^{\mathsf{DH}}$ and $\mathtt{L}^{\mathsf{ElGamal}}$. We start by considering $\mathtt{L}^{\mathsf{DH}}$:

Construction 6.1 (A SPHF for $\mathtt{L}^{\mathsf{DH}}$)**.**

- $\mathsf{Setup}(1^\lambda)$*: Compute* $(\mathbb{G}, g, q) \leftarrow\!\!\$\ \mathsf{GroupGen}(1^\lambda)$*. Compute* $s \leftarrow\!\!\$\ \mathbb{Z}_q$*, and evaluate* $h = g^s$*. Output* $\mathsf{pp} \leftarrow (1^\lambda, \mathbb{G}, g, q, h)$*.*
- $\mathsf{HashKG}(\mathsf{pp}, \mathsf{aux})$*: Output* $\mathsf{hk} \leftarrow (\alpha_1, \alpha_2) \leftarrow\!\!\$\ \mathbb{Z}_q^2$*.*
- $\mathsf{ProjKG}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk})$*: Parse* hk *as* (α_1, α_2)*. Output* $\mathsf{hp} \leftarrow g^{\alpha_1} \cdot h^{\alpha_2}$*.*
- $\mathsf{Hash}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk}, x)$*: Parse* hk *as* (α_1, α_2)*, and* x *as* (u, v)*. Output* $\mathsf{hv} \leftarrow u^{\alpha_1} \cdot v^{\alpha_2}$*.*
- $\mathsf{ProjHash}(\mathsf{pp}, \mathsf{aux}, \mathsf{hp}, x, w)$*: Output* $\mathsf{hv} \leftarrow \mathsf{hp}^w$*.*

Theorem 6.1. *Construction 6.1 is a smooth-projective hash function for the set of languages* $\mathtt{L}^{\mathsf{DH}}$*.*

[21] There is a variant of this notion that lets the projection key depend on the statement x. This variant is called GL-type [GL03]. Here, we consider the so-called KV-type instead [KV11].

Proof. We follow the proof style of [GL03, Section 7.1]. First, we argue correctness.

$$\begin{aligned}\mathsf{Hash}(\mathsf{pp},\mathsf{aux},\mathsf{hk},x) &= u^{\alpha_1}\cdot v^{\alpha_2}\\ &= (g^w)^{\alpha_1}\cdot(h^w)^{\alpha_2}\\ &= (g^{\alpha_1})^w\cdot(h^{\alpha_2})^w\\ &= (g^{\alpha_1}\cdot h^{\alpha_2})^w\\ &= \mathsf{hp}^w = \mathsf{ProjHash}(\mathsf{pp},\mathsf{aux},\mathsf{hp},x,w).\end{aligned}$$

Next, we argue smoothness by showing that the output of Hash is uniformly distributed over $\mathbb{G}$, even with knowledge of hp. To do that, we inspect the output of Hash when $x \notin L_{\mathsf{lpp},\mathsf{aux}}$, *i.e.*, $x = (u,v) = (g^w, h^{w'})$ for some $w' \neq w$:

$$\begin{aligned}\mathsf{Hash}(\mathsf{pp},\mathsf{aux},\mathsf{hk},x) &= u^{\alpha_1}\cdot v^{\alpha_2}\\ &= (g^w)^{\alpha_1}\cdot(h^{w'})^{\alpha_2}\\ &= g^{w\alpha_1}\cdot h^{w'\alpha_2}\\ &= g^{w\alpha_1}\cdot(g^s)^{w'\alpha_2}\\ &= g^{w\alpha_1+sw'\alpha_2}.\end{aligned}$$

Recalling that $\mathsf{hp} = g^{\alpha_1}\cdot h^{\alpha_2} = g^{\alpha_1+s\alpha_2}$, we observe that hp and hv can be represented as a system of equations, as follows:

$$log_g(\mathsf{hv}) = w\alpha_1 + sw'\alpha_2$$
$$log_g(\mathsf{hp}) = \alpha_1 + s\alpha_2$$

Since $w' \neq w$, these equations are linearly independent, *i.e.*, for every choice of hp and hv there exists a pair (α_1,α_2) satisfying this equation. Hence, hp provides no information on hv, meaning that hv is uniformly distributed over $\mathbb{G}$ even with knowledge of hp. □

Next, we provide a SPHF for $\mathsf{L}^{\mathsf{ElGamal}}$. The construction is essentially the same as the one reported above, with the difference that the Hash algorithm leverages aux (*i.e.*, the encrypted message) to evaluate the hash consistently with $\mathsf{ProjHash}$.

Construction 6.2 (A SPHF for $\mathsf{L}^{\mathsf{ElGamal}}$)**.**

- $\mathsf{Setup}(1^\lambda)$*: Compute* $(\mathbb{G},g,q) \leftarrow\$ \mathsf{GroupGen}(1^\lambda)$*. Compute* $s \leftarrow\$ \mathbb{Z}_\mathsf{q}$*, and evaluate* $h = g^s$*. Output* $\mathsf{pp} \leftarrow (1^\lambda,\mathbb{G},g,q,h)$*.*
- $\mathsf{HashKG}(\mathsf{pp},\mathsf{aux})$*: Output* $\mathsf{hk} \leftarrow (\alpha_1,\alpha_2) \leftarrow\$ \mathbb{Z}_\mathsf{q}^2$*.*
- $\mathsf{ProjKG}(\mathsf{pp},\mathsf{aux},\mathsf{hk})$*: Parse* hk *as* (α_1,α_2)*. Output* $\mathsf{hp} \leftarrow g^{\alpha_1}\cdot h^{\alpha_2}$*.*
- $\mathsf{Hash}(\mathsf{pp},\mathsf{aux},\mathsf{hk},x)$*: Parse* hk *as* (α_1,α_2)*, and* x *as* (c_1,c_2)*. Output* $\mathsf{hv} \leftarrow c_1^{\alpha_1}\cdot(c_2/\mathsf{aux})^{\alpha_2}$*.*
- $\mathsf{ProjHash}(\mathsf{pp},\mathsf{aux},\mathsf{hp},x,w)$*: Output* $\mathsf{hv} \leftarrow \mathsf{hp}^w$*.*

6.2.2 A SPHF for linearly representable languages

The constructions presented in the previous subsection can be seen as instantiations of the generic framework of [BBC+13], which supports so-called "linearly representable" languages. Very informally, these are languages whose statements can be represented as a linear combination of group elements. Crucially, this linear combination may be efficiently computed only by knowing the witness of the respective statement.

We proceed more formally in what follows. From a notational standpoint, while referring to a multiplicative cyclic group $\mathbb{G}$ with operator $\cdot$, we denote $u \odot r = r \odot u = u^r$ for $u \in \mathbb{G}$ and $r \in \mathbb{Z}_\mathsf{q}$, and $u \oplus v = u \cdot v$, for $u,v \in \mathbb{G}$. $\oplus$ and $\odot$ represent, respectively, addition and multiplication in the exponent in $\mathbb{G}$. If no symbol is present between two group elements, we implicitly assume the operator to be $\cdot$. Symbols in bold represent vectors (*e.g.*, $\boldsymbol{\alpha}$).

Definition 6.8 (Linearly representable language)**.** *Let* $\mathtt{L} = \{L_{\mathsf{lpp},\mathsf{aux}}\}$ *be a family of NP-languages parameterized by* $(\mathsf{lpp}, \mathsf{aux})$ *with witness relations* $\mathtt{R} = \{R_{\mathsf{lpp},\mathsf{aux}}\}$*, and assume that* Set *is a domain such that, for every* $(\mathsf{lpp}, \mathsf{aux})$*,* $L_{\mathsf{lpp},\mathsf{aux}} \subset \mathsf{Set}$*. We call* $\mathtt{L}$ *a family of linearly representable languages if there exist positive integers* m, n*, group* $\mathbb{G}$ *of prime order* q*, efficiently computable function* $\Gamma : \mathsf{Set} \to \mathbb{G}^{m\times n}$ *and family of functions* $\{\theta_{\mathsf{lpp},\mathsf{aux}} : \mathsf{Set} \to \mathbb{G}^{1\times n}\}$ *which are efficiently computable given* $(\mathsf{lpp}, \mathsf{aux})$*, such that for every* $x \in \mathsf{Set}$*,* $C \in L_{\mathsf{lpp},\mathsf{aux}} \iff \exists \boldsymbol{\lambda} \in \mathbb{Z}_{\mathsf{q}}^{1\times m} : \theta_{\mathsf{lpp},\mathsf{aux}}(x) = \boldsymbol{\lambda} \odot \Gamma(x)$*. We also require that for every* $(x, w) \in R_{\mathsf{lpp},\mathsf{aux}}$*,* $\boldsymbol{\lambda}$ *is efficiently computable given* x *and* w*.*

Intuitively, a statement x in Set is in the language $L_{\mathsf{lpp},\mathsf{aux}}$ if $\theta_{\mathsf{lpp},\mathsf{aux}}(x)$ can be represented as a linear combination of $\Gamma(x)$ and $\boldsymbol{\lambda}$, where $\boldsymbol{\lambda}$ can be efficiently computed only by knowing the witness. To be more concrete, we show that $\mathtt{L}^{\mathsf{DH}}$ and $\mathtt{L}^{\mathsf{ElGamal}}$ are linearly representable languages:

- For $\mathtt{L}^{\mathsf{DH}}$, $\mathsf{Set} = \mathbb{G}^{1\times 2}$, $m = 1$, $n = 2$, and:

$$\Gamma(x) = \begin{pmatrix} g & h \end{pmatrix} \in \mathbb{G}^{1\times 2} \quad \theta_{\mathsf{lpp},\mathsf{aux}}(x) = \begin{pmatrix} u & v \end{pmatrix} \in \mathbb{G}^{1\times 2} \quad \boldsymbol{\lambda}(w) = w \in \mathbb{Z}_{\mathsf{q}}$$

  If $x \in L_{\mathsf{lpp},\mathsf{aux}}$, $x = (u, v) = (g^w, h^w)$. Then, $\theta_{\mathsf{aux},\mathsf{lpp}}(x) = (g^w, h^w) = w \odot (g, h) = \boldsymbol{\lambda}(w) \odot \Gamma(x)$, verifying Definition 6.8.

- For $\mathtt{L}^{\mathsf{ElGamal}}$, $\mathsf{Set} = \mathbb{G}^{1\times 2}$, $m = 1$, $n = 2$, and:

$$\Gamma(x) = \begin{pmatrix} g & h \end{pmatrix} \in \mathbb{G}^{1\times 2} \quad \theta_{\mathsf{lpp},\mathsf{aux}}(x) = \begin{pmatrix} c_1 & c_2/\mathsf{aux} \end{pmatrix} \in \mathbb{G}^{1\times 2} \quad \boldsymbol{\lambda}(w) = w \in \mathbb{Z}_{\mathsf{q}}$$

  If $x \in L_{\mathsf{lpp},\mathsf{aux}}$, $x = (c_1, c_2) = (g^w, \mathsf{aux} \cdot h^w)$. Then, $\theta_{\mathsf{aux},\mathsf{lpp}}(x) = (g^w, (\mathsf{aux} \cdot h^w)/\mathsf{aux}) = w \odot (g, h) = \boldsymbol{\lambda}(w) \odot \Gamma(x)$, verifying Definition 6.8.

As a few important observations, $\Gamma(x)$ is a constant function in both the above examples (*i.e.*, independent from the input statement). Throughout the composition, we exclusively refer to languages for which Γ satisfies this property; hence, we will drop the additional input x. Moreover, we observe that $\theta_{\mathsf{lpp},\mathsf{aux}}(x)$ is the identity function for $\mathtt{L}^{\mathsf{DH}}$, while this is not the case for $\mathtt{L}^{\mathsf{ElGamal}}$. Looking ahead, this slight detail will have implications in constructing SPHFs that support additional malleability properties.

A generic SPHF for linearly representable languages. From the above, one may obtain the generic framework[22] for SPHFs due to [BBC+13]:

Construction 6.3 (A SPHF for linearly representable languages)**.**

- $\mathsf{Setup}(1^\lambda)$*: Sample* lpp*, and output* $\mathsf{pp} \leftarrow (1^\lambda, \mathsf{lpp})$*.*
- $\mathsf{HashKG}(\mathsf{pp}, \mathsf{aux})$*: Output* $\mathsf{hk} \leftarrow \boldsymbol{\alpha} \leftarrow (\alpha_1, \ldots, \alpha_n)^\mathsf{T} \leftarrow\!\!\$\ \mathbb{Z}_{\mathsf{q}}^n$*.*
- $\mathsf{ProjKG}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk} = \boldsymbol{\alpha})$*: Output* $\mathsf{hp} \leftarrow \gamma \leftarrow \Gamma \odot \boldsymbol{\alpha}$*.*
- $\mathsf{Hash}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk} = \boldsymbol{\alpha}, x)$*: Output* $\mathsf{hv} \leftarrow \theta_{\mathsf{aux},\mathsf{lpp}}(x) \odot \boldsymbol{\alpha}$*.*
- $\mathsf{ProjHash}(\mathsf{pp}, \mathsf{aux}, \mathsf{hp} = \gamma, x, w)$*: Output* $\mathsf{hv} \leftarrow \boldsymbol{\lambda}(w) \odot \gamma$*.*

Correctness is straight-forward to prove: if $x \in L_{\mathsf{lpp},\mathsf{aux}}$, $\theta_{\mathsf{aux},\mathsf{lpp}}(x) \odot \boldsymbol{\alpha} = \boldsymbol{\lambda}(w) \odot \Gamma \odot \boldsymbol{\alpha} = \boldsymbol{\lambda}(w) \odot \gamma$. The remaining formal proof of security is discussed in [BBC+13].

6.3 Trapdoor SPHFs

Trapdoor SPHFs (**T**SPHFs), as defined in [BP13], extend SPHFs with two additional properties. First, they introduce a trapdoor setup that enables the evaluation of hashes of any statement consistently with the output of Hash by knowing *exclusively* the respective projection key alongside the trapdoor (instead of the hashing key or the witness). This is formalized with an additional setup algorithm TSetup, which generates some CRS crs' alongside a trapdoor τ'. Second, they provide an algorithm VerHP that may be used to

[22] The framework still holds even if Γ is not a constant function. In that case, one obtains a GL-SPHF.

ascertain whether a projection key is generated consistently with the CRS crs' output by TSetup. In what follows, it is implicitly assumed that Setup also generates a CRS crs (with associated trapdoor τ). We stress that this CRS is *different* from the CRS output by TSetup.

Definition 6.9 (Trapdoor projective hash function (TPHF))**.** *A trapdoor PHF* Π_{TPHF} *is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{TSetup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash}, \mathsf{THash}, \mathsf{VerHP})$ *with the following syntax:*

- Setup *is defined exactly as in Definition 6.6.*
- $\mathsf{TSetup}(\mathsf{pp})$*: Upon input public parameters, output a CRS* crs' *alongside a trapdoor* τ'*.*
- $\mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash}$ *are defined as in Definition 6.6, while additionally taking* crs' *as input.*
- $\mathsf{THash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp}, x, \tau')$*: Upon input public parameters* $(\mathsf{pp}, \mathsf{crs}')$*, auxiliary input* aux*, projection key* hp*, statement* x*, and trapdoor* τ'*, output a hash value* hv*.*
- $\mathsf{VerHP}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp})$*: Upon input public parameters* $(\mathsf{pp}, \mathsf{crs}')$*, auxilary input* aux*, and a projection key* hp*, output* 1 *if* hp *is a valid projection key,* 0 *otherwise.*

We say Π_{TPHF} *is a correct TPHF if* $(\mathsf{Setup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash})$ *form a correct PHF (as per Definition 6.6) and, for every* $x \in \mathsf{Set}$ *and for every* aux*, the following holds:*

$$\Pr\left[\begin{array}{c|l} \mathsf{Hash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hk}, x) = \mathsf{THash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp}, x, \tau') & \mathsf{pp} \leftarrow\!\$\ \mathsf{Setup}(1^\lambda) \\ \wedge & (\mathsf{crs}', \tau') \leftarrow\!\$\ \mathsf{TSetup}(\mathsf{pp}) \\ \mathsf{VerHP}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp}) = 1 & \mathsf{hk} \leftarrow \mathsf{HashKG}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}) \\ & \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hk}) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

This latter property is referred to as ***trapdoor correctness****.*

Definition 6.10 (Trapdoor smooth-projective hash function (TSPHF))**.** *Let* Π_{TPHF} *be a correct TPHF, as per Definition 6.11. We say* Π_{TPHF} *is a TSPHF if the following properties hold:*

- ***Computational smoothness.*** *For every PPT adversary* $\mathcal{A}$ *the following holds:*

$$\Pr\left[\begin{array}{c|l} b = b' & (\mathsf{pp}, \tau) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda) \\ & (x, \mathsf{aux}) \leftarrow \mathcal{A}(\mathsf{pp}, \tau) \\ & (\mathsf{crs}', \tau') \leftarrow\!\$\ \mathsf{TSetup}(1^\lambda) \\ & \mathsf{hk} \leftarrow \mathsf{HashKG}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}) \\ & \mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hk}) \\ & b \leftarrow \{0,1\} \\ & \textit{if } x \in L_{\mathsf{lpp},\mathsf{aux}} \textit{ or } b = 0 \\ & \quad H \leftarrow \mathsf{Hash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hk}, x) \\ & \textit{else} \\ & \quad H \leftarrow\!\$\ \mathcal{Y} \\ & b' \leftarrow \mathcal{A}(\mathsf{hp}, H) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

- ***Soundness.*** *For every adversary* $\mathcal{A}$ *and for every* $(x, w) \in R_{\mathsf{lpp},\mathsf{aux}}$*:*

$$\Pr\left[\begin{array}{c|l} \mathsf{VerHP}(\mathsf{pp}, \mathsf{aux}, \mathsf{crs}', \mathsf{hp}) = 1 \wedge & (\mathsf{pp}, \tau) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda) \\ \mathsf{THash}(\mathsf{pp}, \mathsf{aux}, \mathsf{crs}', \tau', \mathsf{hp}, x) \neq & (\mathsf{crs}', \tau') \leftarrow\!\$\ \mathsf{TSetup}(\mathsf{pp}) \\ \mathsf{ProjHash}(\mathsf{pp}, \mathsf{aux}, \mathsf{crs}', \mathsf{hp}, x, w) & (\mathsf{hp}, \mathsf{aux}, x, w) \leftarrow \mathcal{A}(\mathsf{pp}, \tau, \mathsf{crs}') \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Discussion. TSPHFs typically consider computational smoothness (in place of statistical smoothness) due to the introduction of the TSetup algorithm. Aside from a different syntax compared to Definition 6.7, the rationale remains unchanged: no efficient adversary should be able to distinguish the output of Hash from random for any element not in the language. This holds even if the adversary is given access to the trapdoor τ of the Setup algorithm[23]. As an important remark, the challenger's runtime depends on whether it can efficiently check $x \in L_{\mathsf{lpp},\mathsf{aux}}$[24]. On the other hand, soundness ensures that no adversary can craft an accepting hp while causing the output of THash to be inconsistent with respect to $\mathsf{ProjHash}$ for any element in the language.

[23] Intuitively, this property will be used throughout security proofs to ensure that reductions of intermediate hybrids will be able to use τ independently from the trapdoor τ' introduced by TSetup.

[24] Looking ahead, the construction for linearly representable languages of [BP13] assumes that language membership can be efficiently checked given access to the trapdoor τ of Setup, meaning that the challenger will also be efficient.

6.3.1 A TSPHF for linearly representable languages

In what follows, we describe the generic construction of [BP13] for linearly representable languages. Here, the operators $\odot, \oplus$ reported in the previous subsection are extended as follows: $g_0 \odot g_1 = g_1 \odot g_0 = \hat{e}(g_0, g_1)$, for $g_0 \in \mathbb{G}_0$ and $g_1 \in \mathbb{G}_1$.

Construction 6.4 (A TSPHF for linearly representable languages)**.**

- $\mathsf{Setup}(1^\lambda)$*: Sample* lpp *and output* $\mathsf{pp} = (1^\lambda, \mathsf{lpp})$.
- $\mathsf{TSetup}(\mathsf{pp})$*: Compute* $\tau' \leftarrow\!\!\$\ \mathbb{Z}_\mathsf{q}$*. Compute* $\mathsf{crs}' \leftarrow \zeta = g_1^{\tau'}$*. Output* (crs', τ').
- $\mathsf{HashKG}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux})$*: Output* $\mathsf{hk} \leftarrow \boldsymbol{\alpha} \leftarrow\!\!\$\ \mathbb{Z}_\mathsf{q}^n$.
- $\mathsf{ProjKG}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux} = \zeta, \mathsf{hk} = \boldsymbol{\alpha})$*: Compute* $\gamma \leftarrow \Gamma \odot \boldsymbol{\alpha}$ *and* $\chi \leftarrow \zeta \odot \boldsymbol{\alpha}$*. Output* $\mathsf{hp} \leftarrow (\gamma, \chi)$.
- $\mathsf{Hash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hk} = \boldsymbol{\alpha}, x)$*: Compute* $\theta_{\mathsf{lpp},\mathsf{aux}}(x) \odot \boldsymbol{\alpha} \odot g_1$.
- $\mathsf{ProjHash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp}, x, w)$*: Using* x *and* w*, compute* $\boldsymbol{\lambda}$ *according to Definition 6.8 and output* $\boldsymbol{\lambda} \odot \gamma \odot g_1$.
- $\mathsf{THash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \tau', \mathsf{hp}, x)$*: Parse* hp *as* (γ, χ)*, and output* $\tau'^{-1} \odot \theta_{\mathsf{lpp},\mathsf{aux}}(x) \odot \chi$.
- $\mathsf{VerHP}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux} = \zeta, \mathsf{hp})$*: Parse* hp *as* (γ, χ)*, and verify that* $\chi \in \mathbb{G}_1^n$ *and* $\zeta \odot \gamma = \Gamma \odot \chi$.

Discussion. In essence, this construction is obtained by equipping Construction 6.3 with a bilinear pairing $\hat{e} : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$. In particular, the SPHF is evaluated as before in $\mathbb{G}_0$, whereas $\mathbb{G}_1$ is used to encode the trapdoor information required to verify the newly-introduced properties. In particular, γ is the functional component of projection keys, whereas χ may be used to run THash (provided one has knowledge of τ') and VerHP. Note that both γ and χ encode the hashing key $\boldsymbol{\alpha}$.

6.3.2 TSPHFs for DH tuples and ElGamal ciphertexts

To be concrete, we instantiate the above framework with L^{DH} and $\mathsf{L}^{\mathtt{ElGamal}}$.

Construction 6.5 (A TSPHF for L^{DH})**.** *Let* $\mathsf{BilGroupGen}$ *be an algorithm that outputs* $(\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T, g_0, g_1, q)$*, where* $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T$ *are groups of prime order* q *supporting the bilinear pairing operation* $\hat{e} : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$*, with* g_0 *(resp.* g_1*) being a generator of* $\mathbb{G}_0$ *(resp.* $\mathbb{G}_1$*).*

- $\mathsf{Setup}(1^\lambda)$*: Compute* $(\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T, g_0, g_1, q) \leftarrow\!\!\$\ \mathsf{BilGroupGen}(1^\lambda)$*. Compute* $s \leftarrow\!\!\$\ \mathbb{Z}_\mathsf{q}$*, and evaluate* $h_0 \leftarrow g_0^s$*. Compute* $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{CRSSetup}(1^\lambda)$*. Output* $\mathsf{pp} \leftarrow (1^\lambda, \mathsf{crs}, \mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T, g_0, g_1, q, h_0)$.
- $\mathsf{TSetup}(\mathsf{pp})$*: Compute* $\tau' \leftarrow\!\!\$\ \mathbb{Z}_\mathsf{q}$*. Compute* $\mathsf{crs}' \leftarrow \zeta \leftarrow g_1^{\tau'}$*. Output* (crs', τ').
- $\mathsf{HashKG}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux})$*: Output* $\mathsf{hk} \leftarrow (\alpha_1, \alpha_2) \leftarrow\!\!\$\ \mathbb{Z}_\mathsf{q}^2$.
- $\mathsf{ProjKG}(\mathsf{pp}, \mathsf{crs}' = \zeta, \mathsf{aux}, \mathsf{hk})$*: Parse* hk *as* (α_1, α_2)*. Compute* $\gamma \leftarrow g_0^{\alpha_1} \cdot h_0^{\alpha_2}$*. Compute* $\chi \leftarrow (\zeta^{\alpha_1}, \zeta^{\alpha_2})$*. Output* $\mathsf{hp} \leftarrow (\gamma, \chi)$.
- $\mathsf{Hash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hk}, x)$*: Parse* hk *as* (α_1, α_2)*, and* x *as* (u, v)*. Output* $\mathsf{hv} \leftarrow \hat{e}(u^{\alpha_1} \cdot v^{\alpha_2}, g_1)$.
- $\mathsf{ProjHash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp}, x, w)$*: Parse* hp *as* (γ, χ)*. Output* $\mathsf{hv} \leftarrow \hat{e}(\gamma^w, g_1)$.
- $\mathsf{THash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp}, x = (u, v), \tau')$*: Parse* hp *as* (γ, χ)*, and parse* χ *as* (χ_1, χ_2)*. Output* $\mathsf{hv} \leftarrow \hat{e}(u^{\tau'^{-1}}, \chi_1) \cdot \hat{e}(v^{\tau'^{-1}}, \chi_2)$.
- $\mathsf{VerHP}(\mathsf{pp}, \mathsf{crs}' = \zeta, \mathsf{aux}, \mathsf{hp})$*: Parse* hp *as* (γ, χ)*, and parse* χ *as* (χ_1, χ_2)*. Output* 1 *if* $\chi \in \mathbb{G}_1^2$ *and* $\hat{e}(\gamma, \zeta) = \hat{e}(g_0, \chi_1) \cdot \hat{e}(h_0, \chi_2)$.

Theorem 6.2. *Construction 6.5 is a TSPHF for* L^{DH}.

Proof (Sketch). As discussed in [BP13, Section 4.3], computational smoothness follows by assuming that (i) the DDH assumption holds in $\mathbb{G}_1$, and that (ii) τ can be used to efficiently compute the discrete logarithms of Γ. In what follows, we explicitly expand the remaining properties for further clarity.

- **Correctness.** Correctness essentially consists in carrying out the same steps as Construction 6.1 in $\mathbb{G}_0$:

$$\begin{aligned}\mathsf{Hash}(\mathsf{pp},\mathsf{crs}',\mathsf{aux},\mathsf{hk},x) &= \hat{e}(u^{\alpha_1}\cdot v^{\alpha_2}, g_1)\\ &= \hat{e}((g_0^w)^{\alpha_1}\cdot(h_0^w)^{\alpha_2}, g_1)\\ &= \hat{e}((g_0^{\alpha_1})^w\cdot(h_0^{\alpha_2})^w, g_1)\\ &= \hat{e}((g_0^{\alpha_1}\cdot h_0^{\alpha_2})^w, g_1)\\ &= \hat{e}(\gamma^w, g_1) = \mathsf{ProjHash}(\mathsf{pp},\mathsf{crs}',\mathsf{aux},\mathsf{hp},x,w).\end{aligned}$$

- **Trapdoor correctness.** In order to argue trapdoor correctness, we first show that the output of THash matches Hash for all $x \in \mathsf{Set}$:

$$\begin{aligned}\mathsf{THash}(\mathsf{pp},\mathsf{crs}',\mathsf{aux},\mathsf{hp}=(\gamma,(\chi_1,\chi_2)),x=(u,v),\tau') &= \hat{e}(u^{\tau'^{-1}},\chi_1)\cdot\hat{e}(v^{\tau'^{-1}},\chi_2)\\ &= \hat{e}(g_0^{w\tau'^{-1}}, g_1^{\tau'\alpha_1})\cdot\hat{e}(h_0^{w\tau'^{-1}}, g_1^{\tau'\alpha_2})\\ &= \hat{e}(g_0, g_1^{\alpha_1})^{w\tau'\tau'^{-1}}\cdot\hat{e}(g_0^s, g_1^{\alpha_2})^{w\tau'\tau'^{-1}}\\ &= \hat{e}(g_0,g_1)^{w\alpha_1}\cdot\hat{e}(g_0,g_1)^{ws\alpha_2}\\ &= \hat{e}(g_0,g_1)^{w(\alpha_1+s\alpha_2)}\\ &= \hat{e}(g_0^{w(\alpha_1+s\alpha_2)}, g_1)\\ &= \hat{e}(u^{\alpha_1}\cdot v^{\alpha_2}, g_1) = \mathsf{Hash}(\mathsf{pp},\mathsf{crs}',\mathsf{aux},\mathsf{hp},x).\end{aligned}$$

  Then, we show that honestly-generated keys are such that VerHP outputs 1 - *i.e.*, that $\hat{e}(\gamma,\mathsf{crs}') = \hat{e}(g_0,\chi_1)\cdot\hat{e}(h_0,\chi_2)$.

$$\begin{aligned}\hat{e}(\gamma,\mathsf{crs}') &= \hat{e}(g_0^{\alpha_1}\cdot h_0^{\alpha_2}, g_1^{\tau'})\\ &= \hat{e}(g_0^{\alpha_1+s\alpha_2}, g_1^{\tau'})\\ &= \hat{e}(g_0,g_1)^{\tau'(\alpha_1+s\alpha_2)}\\ &= \hat{e}(g_0,g_1)^{\tau'\alpha_1}\cdot\hat{e}(g_0,g_1)^{\tau's\alpha_2}\\ &= \hat{e}(g_0, g_1^{\tau'\alpha_1})\cdot\hat{e}(g_0^s, g_1^{\tau'\alpha_2})\\ &= \hat{e}(g_0,\chi_1)\cdot\hat{e}(h_0,\chi_2).\end{aligned}$$

- **Perfect soundness.** By definition of perfect soundness, the adversary outputs a projection key hp that causes VerHP to output 1. By construction, this means that hp is distributed exactly as a honestly-generated key. By trapdoor correctness, the output of THash matches the output of Hash for any $x \in \mathsf{Set}$. By correctness, the output of Hash matches the output of $\mathsf{ProjHash}$ for any $x \in L_{\mathsf{lpp},\mathsf{aux}}$. Putting everything together, the output of THash always matches the output of $\mathsf{ProjHash}$ for any $x \in L_{\mathsf{lpp},\mathsf{aux}}$.

□

The construction for $\mathsf{L}^{\mathsf{ElGamal}}$ is essentially the same, except for algorithms Hash and THash evaluating hashes by additionally leveraging aux (similarly to the SPHF for ElGamal described in Construction 6.2).

Construction 6.6 (A TSPHF for $\mathsf{L}^{\mathsf{ElGamal}}$)**.** *Let* $\mathsf{BilGroupGen}$ *be an algorithm that outputs* $(\mathbb{G}_0,\mathbb{G}_1,\mathbb{G}_T,g_0,g_1,q)$*, where* $\mathbb{G}_0,\mathbb{G}_1,\mathbb{G}_T$ *are groups of prime order* q *supporting the bilinear pairing operation* $\hat{e}:\mathbb{G}_0\times\mathbb{G}_1\to\mathbb{G}_T$*, with* g_0 *(resp.* g_1*) being a generator of* $\mathbb{G}_0$ *(resp.* $\mathbb{G}_1$*).*

- $\mathsf{Setup}(1^\lambda)$*: Compute* $(\mathbb{G}_0,\mathbb{G}_1,\mathbb{G}_T,g_0,g_1,q) \leftarrow\$ \mathsf{BilGroupGen}(1^\lambda)$*. Compute* $s \leftarrow\$ \mathbb{Z}_\mathsf{q}$*, and evaluate* $h_0 \leftarrow g_0^s$*. Compute* $\mathsf{crs} \leftarrow\$ \mathsf{CRSSetup}(1^\lambda)$*. Output* $\mathsf{pp} \leftarrow (1^\lambda,\mathsf{crs},\mathbb{G}_0,\mathbb{G}_1,\mathbb{G}_T,g_0,g_1,q,h_0)$*.*
- $\mathsf{TSetup}(\mathsf{pp})$*: Compute* $\tau' \leftarrow\$ \mathbb{Z}_\mathsf{q}$*. Compute* $\mathsf{crs}' \leftarrow \zeta \leftarrow g_1^{\tau'}$*. Output* (crs',τ')*.*

- $\mathsf{HashKG}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux})$: *Output* $\mathsf{hk} \leftarrow (\alpha_1, \alpha_2) \leftarrow\!\!\$\ \mathbb{Z}_{\mathsf{q}}^2$.
- $\mathsf{ProjKG}(\mathsf{pp}, \mathsf{crs}' = \zeta, \mathsf{aux}, \mathsf{hk})$: *Parse* hk *as* (α_1, α_2). *Compute* $\gamma \leftarrow g_0^{\alpha_1} \cdot h_0^{\alpha_2}$. *Compute* $\chi \leftarrow (\zeta^{\alpha_1}, \zeta^{\alpha_2})$. *Output* $\mathsf{hp} \leftarrow (\gamma, \chi)$.
- $\mathsf{Hash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hk}, x)$: *Parse* hk *as* (α_1, α_2), *and* x *as* (c_1, c_2). *Output* $\mathsf{hv} \leftarrow \hat{e}(c_1^{\alpha_1} \cdot (c_2/\mathsf{aux})^{\alpha_2}, g_1)$.
- $\mathsf{ProjHash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp}, x, w)$: *Parse* hp *as* (γ, χ). *Output* $\mathsf{hv} \leftarrow \hat{e}(\gamma^w, g_1)$.
- $\mathsf{THash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp}, x = (c_1, c_2), \tau')$: *Parse* hp *as* (γ, χ), *and parse* χ *as* (χ_1, χ_2). *Output* $\mathsf{hv} \leftarrow \hat{e}(c_1^{\tau'^{-1}}, \chi_1) \cdot \hat{e}((c_2/\mathsf{aux})^{\tau'^{-1}}, \chi_2)$.
- $\mathsf{VerHP}(\mathsf{pp}, \mathsf{crs}' = \zeta, \mathsf{aux}, \mathsf{hp})$: *Parse* hp *as* (γ, χ), *and parse* χ *as* (χ_1, χ_2). *Output* 1 *if* $\chi \in \mathbb{G}_1^2$ *and* $\hat{e}(\gamma, \zeta) = \hat{e}(g_0, \chi_1) \cdot \hat{e}(h_0, \chi_2)$.

6.4 Universally composable PAKE from TSPHFs

[BP13] shows a one-round PAKE protocol that relies on a labeled CCA-secure encryption scheme by instantiating the aforementioned generic TSPHF on the language of Cramer-Shoup ciphertexts (which is a linearly representable language). Intuitively, both parties produce a key pair for the TSPHF, as well as a labeled CCA-secure encryption of their password. The label contains the PIDs of the parties involved in the protocol run, as well as the locally-computed projection key. After having exchanged the projection key and the ciphertext, each party locally checks whether the projection key they received is consistent with crs'. If so, they proceed to the evaluation of the session key.

In particular, party P uses its hk to evaluate an hash of c' means of Hash (as P does not know the witness that was used to produce c'). P' produces a matching hash by running $\mathsf{ProjHash}$ on c' using the hp associated with hk (leveraging the witness w). Then, parties evaluate a second hash by running the aforementioned strategy on reversed roles (*i.e.*, by using $\mathsf{hk}', \mathsf{hp}', c, w$). The session key is given by combining the two hashes by means of $\oplus$. Intuitively, the parties will evaluate the same session key by correctness of the TSPHF, as long as they have supplied the same password and no man-in-the-middle occurred. We exhibit the protocol pictorially in Figure 6.2.

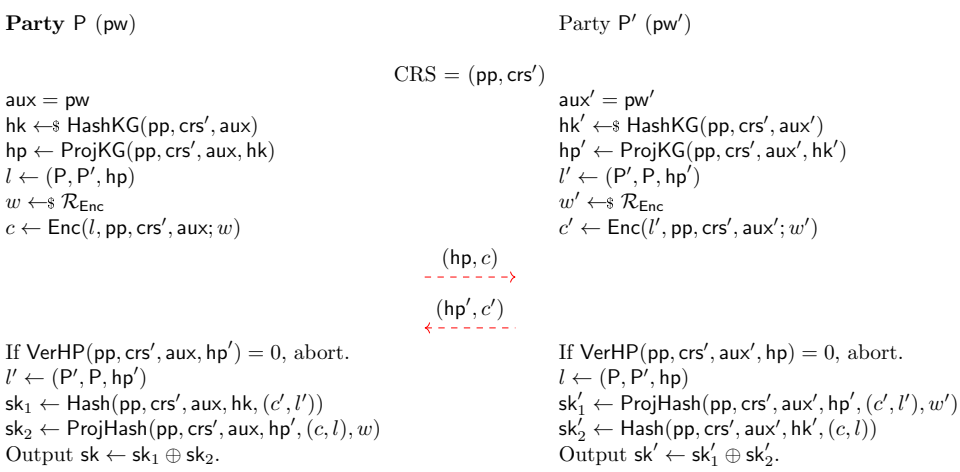


Figure 6.2: The PAKE protocol from TSPHFs due to [BP13].

Theorem 6.3. *The protocol in Figure 6.2 UC-realizes* $\mathcal{F}_{\mathsf{PAKE}}$ *in the* $\mathcal{F}_{\mathsf{crs}}$*-hybrid model in the presence of malicious static corruptions.*

Proof (Sketch). In what follows, we provide an overview of the proof strategy. Looking ahead, we will delve into further details while discussing our sanitized variant of the aforementioned protocol in Section 7.3. A complete proof for this setting may be found in [BP13, Appendix D.1].

Let us consider the simulation of an honest party P within the protocol run. A simulator should run the protocol on behalf of honest parties by computing a ciphertext of some dummy password (in place of the real input passwords). Upon receiving a tuple on behalf of the peer P' of P, the simulator first checks whether this tuple was "oracle-generated", where oracle-generated roughly means that the tuple was generated by the simulator itself on behalf of a honest P'. If this is the case, it means the adversary did not actively tamper with the protocol run; therefore, the simulator simply instructs $\mathcal{F}_{\mathsf{PAKE}}$ to output the session keys to the respective dummy parties. Otherwise, the adversary had interfered with the protocol run, and the simulator should ascertain whether the adversary was able to guess the password of party P or not. To do so, it decrypts the non-oracle generated ciphertext by using the trapdoor for the CCA-secure scheme to retrieve the adversary's guess, which is forwarded to $\mathcal{F}_{\mathsf{PAKE}}$ by means of a TestPwd query. If the guess was correct, the simulator has to "adjust" the key generated by P to match the one computed by the adversary. This is achieved by using the trapdoor of the TSPHF (recall that P originally provided an encryption of a dummy password), and then send a NewKey query to $\mathcal{F}_{\mathsf{PAKE}}$ containing the adversarially-generated key. Otherwise (*i.e.*, the adversary's guess was incorrect), the simulator should just forward a NewKey query to the ideal functionality (which will return random and independent keys to the parties).

In order to move from the real world to the ideal world, we run through a series of hybrids in which we first replace session keys for honest parties that receive oracle-generated messages with (possibly related) random keys (*i.e.*, the first case of the aforementioned simulator), and then do the same for non-oracle generated messages (unless the adversary guesses the password; in which case, the session key will be adversarially-controlled). □

6.5 Malleable SPHFs

Finally, we consider malleable SPHFs (**M**SPHFs): an extension of SPHFs due to [CMY+16] that augments traditional SPHFs to allow the rerandomization of keys and statements while preserving the correctness of the SPHF. More specifically, MSPHFs introduce:

1. Algorithms that maul (or rerandomize) the projection keys (or statements).
2. Algorithms that update the (unsanitized) hash value to match the new projection key (or statement).

In the original treatment of [CMY+16], these objects were defined with reverse firewalls in mind. Concretely, a RF could blind a projection key (resp. an element) sent by a core with some randomness (using the first class of algorithms), and later reuse the same randomness to forward a quantity that may be used to update the (unsanitized) hash value in a consistent manner.

This setting imposes some limitations to what inputs the aforementioned classes of algorithms may use. In particular, all these operations should be carried out *without* knowledge of the witness or the hashing key, as these values are stored on the cores that are being protected (and hence, are inaccessible to RFs). Moreover, in all the applications of [CMY+16], RFs act in a *transparent* fashion: namely, both classes of algorithms are *exclusively* run by firewalls. For that, the syntax of these algorithms is defined without having access to the auxiliary input of the SPHF, which is implicitly assumed to be private to the core running the functional component of the protocol (*e.g.*, a password of a PAKE protocol). We provide a definition in what follows.

Definition 6.11 (Malleable projective hash function (MPHF))**.** *Let* Π_{PHF} *be a correct PHF, as per Definition 6.6. A malleable PHF* Π_{MPHF} *is a tuple of algorithms* $(\Pi_{\mathsf{PHF}}, \mathsf{SampR}, \mathsf{MaulK}, \mathsf{MaulH}, \mathsf{MaulHK}, \mathsf{SampW}, \mathsf{ReranE}, \mathsf{ReranH})$ *with the following syntax:*

- $\mathsf{SampR}(\mathsf{pp})$*: Upon input public parameters* pp*, output a randomness* $\widetilde{r}$*.*
- $\mathsf{MaulK}(\mathsf{pp}, \mathsf{hp}, \widetilde{r})$*: Upon input public parameters* pp*, a projection key* hp *and a randomness* $\widetilde{r}$*, output a new projection key* $\widetilde{\mathsf{hp}}$*.*
- $\mathsf{MaulH}(\mathsf{pp}, \mathsf{hp}, x, \widetilde{r})$*: Upon input public parameters* pp*, a projection key* hp*, a statement* x*, and a randomness* $\widetilde{r}$*, outputs a hash value* $\widetilde{\mathsf{hv}}$*.*

- $\mathsf{MaulHK}(\mathsf{pp}, \mathsf{hk}, \widetilde{r})$: *Upon input public parameters* pp*, a hashing key* hk *and a randomness* $\widetilde{r}$*, output a new hashing key* $\widetilde{\mathsf{hk}}$.
- $\mathsf{SampW}(\mathsf{pp})$: *Upon input public parameters* pp*, output a randomness* $\widetilde{w}$.
- $\mathsf{ReranE}(\mathsf{pp}, x, \widetilde{w})$: *Upon input public parameters* pp*, a statement* x*, and a randomness* $\widetilde{w}$*, output a new statement* $\widetilde{x}$.
- $\mathsf{ReranH}(\mathsf{pp}, \mathsf{hp}, x, \widetilde{w})$: *Upon input public parameters* pp*, a projection key* hp*, a statement* x*, and a randomness* $\widetilde{w}$*, output a hash value* $\widetilde{\mathsf{hv}}$.

We say Π_{MPHF} *is a correct malleable PHF if the following properties hold:*

- ***Projection Consistency****. For every* $x \in \mathsf{Set}$ *and for every* aux*:*

$$\Pr\left[\mathsf{Hash}(\mathsf{pp}, \mathsf{aux}, \widetilde{\mathsf{hk}}, x) = \mathsf{Hash}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk}, x) \oplus \widetilde{\mathsf{hv}} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow\!\$\ \mathsf{Setup}(1^\lambda) \\ \mathsf{hk} \leftarrow\!\$\ \mathsf{HashKG}(\mathsf{pp}, \mathsf{aux}) \\ \mathsf{hp} \leftarrow\!\$\ \mathsf{ProjKG}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk}) \\ \widetilde{r} \leftarrow\!\$\ \mathsf{SampR}(\mathsf{pp}) \\ \widetilde{\mathsf{hp}} \leftarrow \mathsf{MaulK}(\mathsf{pp}, \mathsf{hp}, \widetilde{r}) \\ \widetilde{\mathsf{hv}} \leftarrow \mathsf{MaulH}(\mathsf{pp}, \mathsf{hp}, x, \widetilde{r}) \\ \widetilde{\mathsf{hk}} \leftarrow \mathsf{MaulHK}(\mathsf{pp}, \mathsf{hk}, \widetilde{r}) \end{array}\right] = 1.$$

- ***Rerandomization Consistency****. For every* $x \in \mathsf{Set}$ *and for every* aux*:*

$$\Pr\left[\mathsf{Hash}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk}, \widetilde{x}) = \mathsf{Hash}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk}, x) \oplus \widetilde{\mathsf{hv}} \;\middle|\; \begin{array}{l} \mathsf{pp} \leftarrow\!\$\ \mathsf{Setup}(1^\lambda) \\ \mathsf{hk} \leftarrow\!\$\ \mathsf{HashKG}(\mathsf{pp}, \mathsf{aux}) \\ \mathsf{hp} \leftarrow\!\$\ \mathsf{ProjKG}(\mathsf{pp}, \mathsf{aux}, \mathsf{hk}) \\ \widetilde{w} \leftarrow \mathsf{SampW}(\mathsf{pp}) \\ \widetilde{x} \leftarrow \mathsf{ReranE}(\mathsf{pp}, x, \widetilde{w}) \\ \widetilde{\mathsf{hv}} \leftarrow \mathsf{ReranH}(\mathsf{pp}, \mathsf{hp}, x, \widetilde{w}) \end{array}\right] = 1.$$

- ***Membership Preservation.*** *For every* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*, every* $x \in \mathsf{Set}$*, every* aux *and every* $\widetilde{r}$ *output by* $\mathsf{SampR}(\mathsf{pp})$*, we have* $x \in L_{\mathsf{lpp},\mathsf{aux}} \iff \mathsf{ReranE}(\mathsf{pp}, x, \widetilde{r}) \in L_{\mathsf{lpp},\mathsf{aux}}$.

Definition 6.12 (Malleable smooth-projective hash function (MSPHF))**.** *Let* Π_{MPHF} *be a correct MPHF, as per Definition 6.11. We say* Π_{MPHF} *is a MSPHF if the following properties hold:*

- Π_{PHF} *is a SPHF, as per Definition 6.7.*
- ***Key Indistinguishability****. For every PPT adversary* $\mathcal{A}$*:*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{ll} \mathsf{pp} \leftarrow\!\$\ \mathsf{Setup}(1^\lambda) & b \leftarrow\!\$\ \{0,1\} \\ (\mathsf{hp}_1, \mathsf{hp}_2) \leftarrow \mathcal{A}(\mathsf{pp}) & \widetilde{\mathsf{hp}} \leftarrow \mathsf{MaulK}(\mathsf{pp}, \mathsf{hp}_b, \widetilde{r}) \\ \widetilde{r} \leftarrow\!\$\ \mathsf{SampR}(\mathsf{pp}) & b' \leftarrow \mathcal{A}(\widetilde{\mathsf{hp}}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

- ***Element Indistinguishability****. For every PPT adversary* $\mathcal{A}$*:*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{ll} \mathsf{pp} \leftarrow\!\$\ \mathsf{Setup}(1^\lambda) & b \leftarrow\!\$\ \{0,1\} \\ (x_1, x_2) \leftarrow \mathcal{A}(\mathsf{pp}) & \widetilde{x} \leftarrow \mathsf{ReranE}(\mathsf{pp}, x_b, \widetilde{w}) \\ \widetilde{w} \leftarrow\!\$\ \mathsf{SampW}(\mathsf{pp}) & b' \leftarrow \mathcal{A}(\widetilde{x}) \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Discussion. Intuitively, $\widetilde{r}$ acts as a blinding factor for projection keys, and it may be used to update the hash consistently with the blinded projection key. In particular, upon blinding a projection key with $\widetilde{r}$, one may recompute the difference introduced by the blinding operation in the evaluation of the hash of some element x by *exclusively* having access to $\widetilde{r}$ (and not hk or w). In the context of RFs, a firewall may blind an outbound projection key by means of MaulK and compute the respective hash update by running MaulH. Similar considerations apply to the blinding of statements by means of $\widetilde{w}$.

Remark. *For certain applications, preventing* MaulH *and* ReranH *from accessing* aux *is unnecessary. Concretely, in the srUC framework firewalls are allowed to return their randomness to the cores they are protecting; hence, it makes little sense to forbid cores (who are in possession of* aux*) from running the hash update algorithms to adjust their hash value locally without leveraging this additional information. As we will see in the next section, lifting this syntactical restriction allows for the design of more efficient MSPHFs for suitable languages.*

6.5.1 A MSPHF for a restricted class of linearly representable languages

In the previous sections, we described generic constructions for SPHFs and TSPHFs for *any* linearly representable language. Unfortunately, it is impossible to obtain a matching result for MSPHFs. Indeed, the language of Cramer-Shoup ciphertexts is a linearly representable language that can *never* admit a MSPHF. In particular, it is impossible to achieve element rerandomizability, as Cramer-Shoup is a CCA-secure (hence, non-malleable) encryption scheme.

The above considerations are in line with the results of [CMY+16], in which the authors show a generic MSPHF that works for a *restricted* class of linearly representable languages. In particular, this construction requires $\theta_{\mathsf{lpp},\mathsf{aux}}(x)$ to be the identity function, and is described in what follows.

Construction 6.7 (A MSPHF for linearly representable languages)**.**

- $\mathsf{Setup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash}$ *are the same as Construction 6.3.*
- $\mathsf{SampR}(\mathsf{pp})$*: Output* $\widetilde{r} \leftarrow \widetilde{\boldsymbol{\alpha}} \leftarrow\!\!\$\ \mathbb{Z}_{\mathsf{q}}^n$.
- $\mathsf{MaulK}(\mathsf{pp}, \mathsf{hp}, \widetilde{r} = \widetilde{\boldsymbol{\alpha}})$*: Output* $\widetilde{\mathsf{hp}} \leftarrow \mathsf{hp} \oplus \Gamma \odot \widetilde{\boldsymbol{\alpha}}$.
- $\mathsf{MaulH}(\mathsf{pp}, \mathsf{hp}, x, \widetilde{r} = \widetilde{\boldsymbol{\alpha}})$*: Output* $\widetilde{\mathsf{hv}} \leftarrow \theta(x) \odot \widetilde{\boldsymbol{\alpha}}$.
- $\mathsf{MaulHK}(\mathsf{pp}, \mathsf{hk} = \boldsymbol{\alpha}, \widetilde{r} = \widetilde{\boldsymbol{\alpha}})$*: Output* $\widetilde{\mathsf{hk}} \leftarrow \boldsymbol{\alpha} \oplus \widetilde{\boldsymbol{\alpha}}$.
- $\mathsf{SampW}(\mathsf{pp})$*: Output* $\widetilde{w} \leftarrow\!\!\$\ \mathbb{Z}_{\mathsf{q}}^n$.
- $\mathsf{ReranE}(\mathsf{pp}, x, \widetilde{w})$*: Derive* $\widetilde{\boldsymbol{\lambda}}$ *from* $\widetilde{w}$*. Output* $\widetilde{c} \leftarrow \theta(x) \oplus \widetilde{\boldsymbol{\lambda}} \odot \Gamma$.
- $\mathsf{ReranH}(\mathsf{pp}, \mathsf{hp}, x, \widetilde{w})$*: Derive* $\widetilde{\boldsymbol{\lambda}}$ *from* $\widetilde{w}$*. Output* $\widetilde{\mathsf{hv}} \leftarrow \widetilde{\boldsymbol{\lambda}} \odot \mathsf{hp}$.

6.5.2 A MSPHF for DH tuples

As a concrete example, we instantiate the above framework on the language of DH tuples - a language for which $\theta_{\mathsf{lpp},\mathsf{aux}}$ is indeed constant.

Construction 6.8 (A MSPHF for L^{DH})**.**

- $\mathsf{Setup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash}$ *are the same as Construction 6.1.*
- $\mathsf{SampR}(\mathsf{pp})$*: Output* $\widetilde{r} \leftarrow (\widetilde{\alpha}_1, \widetilde{\alpha}_2) \leftarrow\!\!\$\ \mathbb{Z}_{\mathsf{q}}^2$.
- $\mathsf{MaulK}(\mathsf{pp}, \mathsf{hp}, \widetilde{r} = (\widetilde{\alpha}_1, \widetilde{\alpha}_2))$*: Output* $\widetilde{\mathsf{hp}} \leftarrow \mathsf{hp} \cdot g^{\widetilde{\alpha}_1} \cdot h^{\widetilde{\alpha}_2}$.
- $\mathsf{MaulH}(\mathsf{pp}, \mathsf{hp}, x = (u, v), \widetilde{r} = (\widetilde{\alpha}_1, \widetilde{\alpha}_2))$*: Output* $\widetilde{\mathsf{hv}} \leftarrow u^{\widetilde{\alpha}_1} \cdot v^{\widetilde{\alpha}_2}$.
- $\mathsf{MaulHK}(\mathsf{pp}, \mathsf{hk} = (\alpha_1, \alpha_2), \widetilde{r} = (\widetilde{\alpha}_1, \widetilde{\alpha}_2))$*: Output* $\widetilde{\mathsf{hk}} \leftarrow (\alpha_1 + \widetilde{\alpha}_1, \alpha_2 + \widetilde{\alpha}_2)$.
- $\mathsf{SampW}(\mathsf{pp})$*: Output* $\widetilde{w} \leftarrow\!\!\$\ \mathbb{Z}_{\mathsf{q}}$.
- $\mathsf{ReranE}(\mathsf{pp}, x = (u, v), \widetilde{w})$*: Output* $\widetilde{c} \leftarrow (u \cdot g^{\widetilde{w}}, v \cdot h^{\widetilde{w}})$.
- $\mathsf{ReranH}(\mathsf{pp}, \mathsf{hp}, x, \widetilde{w})$*: Output* $\widetilde{\mathsf{hv}} \leftarrow \mathsf{hp}^{\widetilde{w}}$.

Theorem 6.4. *Construction 6.8 is a malleable smooth-projective hash function for* L^{DH}*, assuming the DDH assumption holds.*

Proof. By Theorem 6.1, Construction 6.8 is a SPHF. Hence, it suffices to prove the remaining correctness and security properties.

- **Projection consistency.** Let us consider a keypair constituted of $\mathsf{hk} = (\alpha_1, \alpha_2)$ and $\mathsf{hp} = g^{\alpha_1} \cdot h^{\alpha_2}$. First, we blind the keypair with blinding factor $\widetilde{r} = (\widetilde{\alpha}_1, \widetilde{\alpha}_2)$, obtaining $\widetilde{\mathsf{hp}} = \mathsf{hp} \cdot g^{\widetilde{\alpha}_1} \cdot h^{\widetilde{\alpha}_2}$ and $\widetilde{\mathsf{hk}} = (\alpha_1 + \widetilde{\alpha}_1, \alpha_2 + \widetilde{\alpha}_2)$. Then:

$$\begin{aligned}\mathsf{Hash}(\mathsf{pp}, \widetilde{\mathsf{hk}}, \mathsf{aux}, x = (u, v)) &= u^{\alpha_1 + \widetilde{\alpha}_1} \cdot v^{\alpha_2 + \widetilde{\alpha}_2} \\ &= (u^{\alpha_1} \cdot v^{\alpha_2}) \cdot (u^{\widetilde{\alpha}_1} \cdot v^{\widetilde{\alpha}_2}) \\ &= \mathsf{Hash}(\mathsf{pp}, \mathsf{hk}, \mathsf{aux}, x) \cdot \mathsf{MaulH}(\mathsf{pp}, \mathsf{hp}, x, \widetilde{r}).\end{aligned}$$

- **Element consistency.** Let us consider a statement $x = (u, v)$. We blind the statement with blinding factor $\widetilde{w}$, obtaining $\widetilde{x} = (u \cdot g^{\widetilde{w}}, v \cdot h^{\widetilde{w}})$. Then:

$$\begin{aligned}\mathsf{Hash}(\mathsf{pp}, \mathsf{hk} = (\alpha_1, \alpha_2), \mathsf{aux}, \widetilde{x}) &= (u \cdot g^{\widetilde{w}})^{\alpha_1} \cdot (v \cdot h^{\widetilde{w}})^{\alpha_2} \\ &= u^{\alpha_1} \cdot g^{\alpha_1 \widetilde{w}} \cdot v^{\alpha_2} \cdot h^{\alpha_2 \widetilde{w}} \\ &= (u^{\alpha_1} \cdot v^{\alpha_2}) \cdot (g^{\alpha_1} \cdot h^{\alpha_2})^{\widetilde{w}} \\ &= \mathsf{Hash}(\mathsf{pp}, \mathsf{hk}, \mathsf{aux}, x) \cdot (\mathsf{hp})^{\widetilde{w}} \\ &= \mathsf{Hash}(\mathsf{pp}, \mathsf{hk}, \mathsf{aux}, x) \cdot \mathsf{ReranH}(\mathsf{pp}, \mathsf{hp}, x, \widetilde{w}).\end{aligned}$$

- **Membership preservation.** Let us consider again a statement $x = (u, v)$. We blind the statement with blinding factor $\widetilde{w}$, obtaining $\widetilde{x} = (u \cdot g^{\widetilde{w}}, v \cdot h^{\widetilde{w}})$. If $x \in L_{\mathsf{lpp},\mathsf{aux}}$, x can be parsed as $(u, v) = (g^w, h^w)$ for some w. On the other hand, $\widetilde{x}$ can be parsed as $(u, v) = (g^{w+\widetilde{w}}, h^{w+\widetilde{w}}) \in L_{\mathsf{lpp},\mathsf{aux}}$.
- **Key indistinguishability** follows from the fact that $g^{\widetilde{\alpha}_1} \cdot h^{\widetilde{\alpha}_2} \cdot \mathsf{hp}$ is distributed uniformly over $\mathbb{G}$ for any $\mathsf{hp} \in \mathbb{G}$.
- **Element indistinguishability** follows from the DDH assumption.

□

What about $\mathsf{L}^{\mathsf{ElGamal}}$? Unfortunately, the limitation on $\theta_{\mathsf{lpp},\mathsf{aux}}$ imposed by the generic framework of Construction 6.7 makes it unsuitable to capture the language of ElGamal ciphertexts. In the next section, we will thoroughly discuss how this limitation can be overcome.

7 Subversion-resilient PAKE from Smooth-Projective Hashing

In this section, we present the salient results of a currently unpublished manuscript which focuses on the design of a more efficient srUC PAKE protocol by relying on smooth-projective hashing.

7.1 Summary

For the design of this srUC PAKE protocol, we sanitize the protocol described in Section 6.4.

Malleable SPHFs for ElGamal. In Section 6.5, we highlighted that the generic construction of [CMY+16] does not yield a MSPHF for the language of ElGamal ciphertexts - a building block that is essential for the design of our srUC PAKE protocol. We address this shortcoming by providing two substantially different MSPHFs for ElGamal. The first construction requires generalizing the syntax of [CMY+16], and is discussed in this section. These modifications are well-motivated from an application standpoint, as protocols in the srUC framework fully support our revisited syntax. The second construction retains the original syntax of [CMY+16] and is deferred to Appendix B.2, being an independent contribution that might find applicability in neighbouring research fields (*e.g.*, for constructions and frameworks that focus on the design of transparent RFs). Note that, even though the two instantiations have comparable efficiency, the former will lead to the design of a significantly more performing MTSPHF.

Malleable TSPHFs. Next, we provide a formal definition for malleable TSPHFs, which extends the malleability properties provided by MSPHFs to TSPHFs. In particular, this primitive allows firewalls to rerandomize both projection keys as well as elements of the underlying TSPHF. With that, we instantiate an efficient MTSPHF for the language of ElGamal ciphertexts by adapting the malleability portion of our newly-introduced MSPHFs for ElGamal to the TSPHFs of [BP13]. Again, we discuss the more efficient construction in this section, and defer the instantiation adhering to the original malleability syntax to Appendix B.2.

The final PAKE protocol. Sanitizing the PAKE protocol due to [BP13] (Section 6.4) requires non-trivial modifications to its structure, given its dependency on a CCA-secure encryption scheme - an inherently non-malleable primitive. We achieve this goal by replacing the CCA-secure encryption scheme with a rerandomizable CPA-secure one plus a CM-NIZK. The controlled malleability allows firewalls to sanitize ciphertexts and their related proofs while ensuring that adversaries cannot compromise the security of the protocol. We discuss the implications of these modifications further in the technical overview.

7.1.1 Technical overview

Below, we provide an overview of the technical ideas behind our results.

A MSPHF for the language of ElGamal ciphertexts. In Section 6.5, we established that the generic MSPHF of [CMY+16] (Construction 6.7) does not support the language of ElGamal ciphertexts. A first natural attempt to overcome this limitation would consist in augmenting the SPHF for ElGamal in Construction 6.2 by making it malleable in a similar manner to Construction 6.8. However, even though it would be possible to satisfy element consistency, projection consistency would fail. We provide further details in what follows.

Given projection key $\gamma = g^{\alpha_1} h^{\alpha_2}$, MaulK samples random elements $(\widetilde{\alpha}_1, \widetilde{\alpha}_2)$ and rerandomizes the projection key into $g^{\widetilde{\alpha}_1} h^{\widetilde{\alpha}_2} \gamma = g^{\alpha_1 + \widetilde{\alpha}_1} h^{\alpha_2 + \widetilde{\alpha}_2}$. In order to update the hash value $hv = c_1^{\alpha_1} (c_2 / \mathsf{aux})^{\alpha_2} = \gamma^w$ to match the rerandomization operated on the projection key, one could simply multiply hv by $\widetilde{hv} = c_1^{\widetilde{\alpha}_1} (c_2 / \mathsf{aux})^{\widetilde{\alpha}_2}$. However, this approach would require MaulH accessing aux, which makes it incompatible with the syntax of MSPHFs of [CMY+16]. Note that the alternative evaluation of $\widetilde{hv}$ as $(g^{\widetilde{\alpha}_1} h^{\widetilde{\alpha}_2})^w$ is also unsatisfactory, given that MaulH cannot access the witness w.

To solve this issue, we propose two approaches. In this section, we alter the syntax of [CMY+16] by providing the hash update algorithms with access to aux. This allows us to use the (efficient) MSPHF for ElGamal we sketched above. Looking ahead, this solution is compatible with our srUC PAKE, given that the srUC framework allows firewalls to return their randomness to their cores (who are in possession of aux). In Appendix B.2, we propose a different instantiation with comparable efficiency that preserves the original syntax of [CMY+16].

Malleable trapdoor smooth projective hash functions. Next, we introduce a new primitive that achieves the best of both worlds, *i.e.*, a SPHF that is both malleable and trapdoor. In particular, we instantiate a MTSPHF for the language of ElGamal ciphertexts, which will be essential for the design of our srUC PAKE.

In order to construct our MTSPHF, we augment the TSPHF for ElGamal described in Construction 6.6 with rerandomization algorithms, which are defined in a similar fashion to the ones described for the MSPHF of the previous paragraph. In particular, a projection key $(\gamma, (\chi_1, \chi_2)) = (g_0^{\alpha_1} h_0^{\alpha_2}, (\zeta^{\alpha_1}, \zeta^{\alpha_2}))$ is rerandomized to $(\gamma g_0^{\widetilde{\alpha}_1} h_0^{\widetilde{\alpha}_2}, (\chi_1 \zeta^{\widetilde{\alpha}_1}, \chi_2 \zeta^{\widetilde{\alpha}_2}))$, where $\zeta = g_1^{\tau'}$. Because we lifted the restriction of providing aux to the hash update algorithms, the hash value $hv = \hat{e}(c_1^{\alpha_1}, (c_2/\mathsf{aux})^{\alpha_2}, g_1)$ can be updated by multiplying it with $\hat{e}(c_1^{\widetilde{\alpha}_1}(c_2/\mathsf{aux})^{\widetilde{\alpha}_2}, g_1)$. In Section 7.2, we cover this construction in further details. In Appendix B.2, we show an alternative (less efficient) construction that supports the original malleability syntax.

Our PAKE protocol. In Section 7.3, we describe our new PAKE protocol in the srUC framework of [CMNV22]. As a starting point, we consider the PAKE protocol due to [BP13], which lies in the CRS-hybrid model while requiring a CCA-secure PKE scheme and a TSPHF for the language of ciphertexts. As remarked before, this TSPHF allows one to compute the same hash value either by using (i) the hashing key, the ciphertext, and the plaintext that was used to generate that ciphertext; (ii) the projection key, the ciphertext, and the randomness that was used to generate the ciphertext; or (iii) the projection key, the ciphertext, and the setup trapdoor.

The protocol of [BP13] (reported in Section 6.4) roughly unfolds as follows. Each party computes a key pair for the TSPHF, as well as a CCA-secure ciphertext of the password it holds, and forwards the projection key along with the ciphertext to its peer. Upon receiving a tuple (supposedly) from their peer, each party locally computes a session key by using (i) the local hashing key, the ciphertext of its peer, and the local password, and (ii) the projection key received from its peer, the local ciphertext, and the local randomness. The properties of the TSPHF are such that the evaluation of (i) of each party matches with the evaluation of (ii) of its peer, as long as both parties used the same password, and the adversary did not actively interfere with the protocol run by performing a man-in-the-middle attack.

In order to move to the sanitized setting, we first need to upgrade the primitives used by the original protocol so that RFs can successfully sanitize messages. To this extent, we replace the TSPHF with our newly-introduced MTSPHF, enabling firewalls to blind outbound projection keys. However, as we see next, sanitizing CCA-secure ciphertexts is less immediate, due to the intrinsic non-malleability of this primitive.

As a first attempt, one could replace the CCA-secure scheme with an RCCA-secure variant. However, it turns out that this causes the reductions of the original proof of [BP13] that require CCA security (*e.g.*, in Game 9) to fail. More concretely: in these games, we are attempting to replace the real password pw of an honest party with a dummy password $\overline{\mathsf{pw}}$, so that we obtain a new game that is no longer dependent on the respective party's input. Upon receiving a non-oracle generated ciphertext (*i.e.*, an adversarially-generated message), the reduction of the original proof queries the decryption oracle controlled by the challenger of CCA-security to retrieve the password that was used from the adversary to generate that specific message. In particular, the reduction should make the target honest party output a key computed as per the real protocol whenever the adversary correctly guesses the password. Otherwise, that party should output a random key.

If we replace this challenger with one for RCCA security, the adversary can simply rerandomize the challenge ciphertext to always cause the decryption query to output the special message $\mathtt{TEST}$. The crucial observation is that the same behaviour occurs even if the adversary supplies a non-oracle generated message

without rerandomizing the challenge ciphertext (and, in particular, by correctly guessing the target party's password). Note that, in the latter case, the reduction should complete the protocol run by outputting a session key that is generated as per the protocol run, whereas in the former case it should output a random session key (as the adversary was indeed unable to guess the password on its own, and it should be unable to determine the key computed from that party). Since it is no longer possible to tell the two settings apart, the reduction is unable to properly simulate the two hybrids.

We remedy this shortcoming by replacing the CCA-secure scheme with a rerandomizable CPA-secure scheme plus a controlled-malleable NIZK (CM-NIZK). This way, one can first extract the witness and the transformation associated with the ciphertext to determine whether it was rerandomized or not. Then, by applying CPA security, it is possible to hybrid away the password from the simulation whenever the message was rerandomized. Note that CPA-security now suffices, as the CM-NIZK allows reductions to determine whether messages are oracle-generated or not (in place of the decryption queries used for the CCA-secure encryption scheme of the original protocol). With this variation, firewalls can now successfully rerandomize outbound ciphertexts and maul outbound proofs so that they verify under the new statement (*i.e.*, the sanitized ciphertext). Hence, we obtain a PAKE protocol that fully supports sanitation.

We conclude the section by discussing our performance improvements against the subversion-resilient PAKE-from-OT protocol described in Section 5.

7.2 Malleable trapdoor smooth-projective hash functions

In this subsection, we first generalize the syntax of malleable SPHF due to [CMY+16], showing that our modifications are sufficient to provide a natural instantiation of a MSPHF for the language of ElGamal ciphertexts. Then, we formalize the notion of *malleable trapdoor SPHF* (MTSPHF) and instantiate a MT-SPHF for ElGamal. Looking ahead, this will be a core building block for our subversion-resilient PAKE protocol.

7.2.1 A MSPHF for ElGamal ciphertexts

Following the considerations outlined in the technical overview, the original syntax of [CMY+16] (reported in Definition 6.12) may be generalized by providing the MaulH and ReranH algorithms with access to aux. All the algorithmic behaviours, as well as security and correctness properties remain unchanged to Definition 6.12; hence, we do not provide an explicit separate definition. In what follows, we show that this modification enables us to design a MSPHF for the language of ElGamal ciphertexts in a similar fashion to the MSPHF for DH tuples described in Construction 6.8.

Construction 7.1 (A MSPHF for $\mathtt{L}^{\mathsf{ElGamal}}$)**.**

- $\mathsf{Setup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash}$ *are the same as Construction 6.2.*
- $\mathsf{SampR}(\mathsf{pp})$*: Output* $\widetilde{r} \leftarrow (\widetilde{\alpha}_1, \widetilde{\alpha}_2) \leftarrow\!\!\$\ \mathbb{Z}_{\mathsf{q}}^2$.
- $\mathsf{MaulK}(\mathsf{pp}, \mathsf{hp}, \widetilde{r} = (\widetilde{\alpha}_1, \widetilde{\alpha}_2))$*: Output* $\widetilde{\mathsf{hp}} \leftarrow \mathsf{hp} \cdot g^{\widetilde{\alpha}_1} \cdot h^{\widetilde{\alpha}_2}$.
- $\mathsf{MaulH}(\mathsf{pp}, \mathsf{aux}, \mathsf{hp}, x = (c_1, c_2), \widetilde{r} = (\widetilde{\alpha}_1, \widetilde{\alpha}_2))$*: Output* $\widetilde{\mathsf{hv}} \leftarrow c_1^{\widetilde{\alpha}_1} \cdot (c_2/\mathsf{aux})^{\widetilde{\alpha}_2}$.
- $\mathsf{MaulHK}(\mathsf{pp}, \mathsf{hk} = (\alpha_1, \alpha_2), \widetilde{r} = (\widetilde{\alpha}_1, \widetilde{\alpha}_2))$*: Output* $\widetilde{\mathsf{hk}} \leftarrow (\alpha_1 + \widetilde{\alpha}_1, \alpha_2 + \widetilde{\alpha}_2)$.
- $\mathsf{SampW}(\mathsf{pp})$*: Output* $\widetilde{w} \leftarrow\!\!\$\ \mathbb{Z}_{\mathsf{q}}$.
- $\mathsf{ReranE}(\mathsf{pp}, x = (c_1, c_2), \widetilde{w})$*: Output* $\widetilde{c} \leftarrow (c_1 \cdot g^{\widetilde{w}}, c_2 \cdot h^{\widetilde{w}})$.
- $\mathsf{ReranH}(\mathsf{pp}, \mathsf{aux}, \mathsf{hp}, x, \widetilde{w})$*: Output* $\widetilde{\mathsf{hv}} \leftarrow \mathsf{hp}^{\widetilde{w}}$.

Theorem 7.1. *Construction 6.8 is a malleable smooth-projective hash function for* $\mathtt{L}^{\mathsf{ElGamal}}$*, assuming the DDH assumption holds.*

The proof is essentially the same as Construction 6.8, and is therefore omitted.

7.2.2 Defining malleable trapdoor SPHFs

In order to define MTSPHFs, we equip Definition 6.10 with the malleability properties of Definition 6.12, with slight modifications to ensure that the algorithms for malleability are compatible with their trapdoor counterparts. For convenience, we report the complete syntax in a single definition.

Definition 7.1 (Malleable trapdoor projective hash function (MTPHF))**.** *Let* $\mathtt{L} = \{L_{\mathsf{lpp},\mathsf{aux}}\}$ *be a set of NP languages defined over a domain* Set*, and let* $\mathtt{R} = \{R_{\mathsf{lpp},\mathsf{aux}}\}$ *be the respective NP relations. A malleable trapdoor projective hash function* Π_{MTPHF} *is characterized by two tuples of algorithms: a trapdoor component, containing algorithms* (Setup, TSetup, HashKG, ProjKG, Hash, ProjHash, THash, VerHP)*, and a malleability component, containing algorithms* (SampR, MaulK, MaulH, MaulHK, SampW, ReranE, ReranH)*. More specifically:*

Trapdoor component.

- $\mathsf{Setup}(1^\lambda)$*: Upon input a unary description of the security parameter* λ*, output public parameters* pp*.* pp *includes* lpp*, which identifies a set of languages* $L_{\mathsf{lpp},\mathsf{aux}}$*.*
- $\mathsf{TSetup}(\mathsf{pp})$*: Upon input public parameters* pp*, output a CRS* crs' *alongside a trapdoor* τ'*.*
- $\mathsf{HashKG}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux})$*: Upon input public parameters* $(\mathsf{pp}, \mathsf{crs}')$ *and auxiliary information* aux*, output an hashing key* hk*.*
- $\mathsf{ProjKG}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hk})$*: Upon input public parameters* $(\mathsf{pp}, \mathsf{crs}')$*, auxiliary information* aux*, and a hashing key* hk*, output a projection key* hp*.*
- $\mathsf{Hash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hk}, x)$*: Upon input public parameters* $(\mathsf{pp}, \mathsf{crs}')$*, auxiliary information* aux*, a hashing key* hk*, and a statement* x*, output a hash value* $\mathsf{hv} \in \mathcal{Y}$*.*
- $\mathsf{ProjHash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp}, x, w)$*: Upon input public parameters* $(\mathsf{pp}, \mathsf{crs}')$*, auxiliary information* aux*, a projection key* hp*, a statement* x*, and a witness* w*, output a hash value* $\mathsf{hv} \in \mathcal{Y}$*.*
- $\mathsf{THash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp}, x, \tau')$*: Upon input public parameters* $(\mathsf{pp}, \mathsf{crs}')$*, auxiliary input* aux*, projection key* hp*, statement* x*, and trapdoor* τ'*, output a hash value* hv*.*
- $\mathsf{VerHP}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp})$*: Upon input public parameters* $(\mathsf{pp}, \mathsf{crs}')$*, auxilary input* aux*, and a projection key* hp*, output* 1 *if* hp *is a valid projection key,* 0 *otherwise.*

Malleability component.

- $\mathsf{SampR}(\mathsf{pp}, \mathsf{crs}')$*: Upon input public parameters* $(\mathsf{pp}, \mathsf{crs}')$*, output a randomness* $\widetilde{r}$*.*
- $\mathsf{MaulK}(\mathsf{pp}, \mathsf{crs}', \mathsf{hp}, \widetilde{r})$*: Upon input public parameters* $(\mathsf{pp}, \mathsf{crs}')$*, a projection key* hp *and a randomness* $\widetilde{r}$*, output a new projection key* $\widetilde{\mathsf{hp}}$*.*
- $\mathsf{MaulH}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp}, x, \widetilde{r})$*: Upon input public parameters* $(\mathsf{pp}, \mathsf{crs}')$*, auxiliary information* aux*, a projection key* hp*, a statement* x*, and a randomness* $\widetilde{r}$*, outputs a hash value* $\widetilde{\mathsf{hv}}$*.*
- $\mathsf{MaulHK}(\mathsf{pp}, \mathsf{crs}', \mathsf{hk}, \widetilde{r})$*: Upon input public parameters* $(\mathsf{pp}, \mathsf{crs}')$*, a hashing key* hk *and a randomness* $\widetilde{r}$*, output a new hashing key* $\widetilde{\mathsf{hk}}$*.*
- $\mathsf{SampW}(\mathsf{pp}, \mathsf{crs}')$*: Upon input public parameters* $(\mathsf{pp}, \mathsf{crs}')$*, output a randomness* $\widetilde{w}$*.*
- $\mathsf{ReranE}(\mathsf{pp}, \mathsf{crs}', x, \widetilde{w})$*: Upon input public parameters* $(\mathsf{pp}, \mathsf{crs}')$*, a statement* x*, and a randomness* $\widetilde{w}$*, output a new statement* $\widetilde{x}$*.*
- $\mathsf{ReranH}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp}, x, \widetilde{w})$*: Upon input public parameters* $(\mathsf{pp}, \mathsf{crs}')$*, auxiliary information* aux*, a projection key* hp*, a statement* x*, and a randomness* $\widetilde{w}$*, output a hash value* $\widetilde{\mathsf{hv}}$*.*

We say Π_{MTPHF} *is a correct MTPHF if it is both a correct TPHF (as per Definition 6.9) and a correct MPHF (as per Definition 6.11).*

Definition 7.2 (Malleable trapdoor smooth-projective hash function (MTSPHF))**.** *Let* Π_{MTPHF} *be a correct MTPHF, as per Definition 7.1. We say* Π_{MTPHF} *is a MTSPHF if is both a trapdoor smooth-projective hash function (as per Definition 6.10) and a malleable smooth-projective hash function (as per Definition 6.12).*

Moreover, key indistinguishability additionally requires the output of VerHP *to be* 1 *for both the adversarially-supplied projection keys.*

Discussion. In essence, in order to show that a MTPHF is correct, one has to prove correctness, trapdoor correctness, projection consistency, rerandomization consistency, and membership preservation. In order to show that a MTPHF is a MTSPHF, one has to prove smoothness, soundness, key indistinguishability, and element indistinguishability. The additional requirement imposed on key indistinguishability is necessary, as otherwise it would always be trivially possible to design an adversary that always wins the key indistinguishability game. In particular, such an adversary would supply projection keys $(\mathsf{hp}_1, \mathsf{hp}_2)$ such that one key verifies and the other does not to obtain a trivial distinguishing advantage.

7.2.3 A MTSPHF for ElGamal

Next, we show a MTSPHF for the language of ElGamal ciphertexts that adheres to our general syntax. This construction is obtained by adapting the malleability properties of our novel MSPHF for ElGamal (Construction 7.1) to the TSPHF for ElGamal described in Construction 6.6.

Construction 7.2 (A MTSPHF for $\mathtt{L}^{\mathsf{ElGamal}}$)**.**

- ***Trapdoor component.*** $\mathsf{Setup}, \mathsf{HashKG}, \mathsf{TSetup}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash}, \mathsf{THash}, \mathsf{VerHP}$ *are defined as in Construction 6.6.*
- ***Malleability component.***
  - $\mathsf{SampR}(\mathsf{pp}, \mathsf{crs}')$*: Output* $\widetilde{r} \leftarrow (\widetilde{\alpha}_1, \widetilde{\alpha}_2) \leftarrow\!\!\$\ \mathbb{Z}_{\mathsf{q}}^2$.
  - $\mathsf{MaulK}(\mathsf{pp}, \mathsf{crs}' = \zeta, \mathsf{hp} = (\gamma, \chi), \widetilde{r} = (\widetilde{\alpha}_1, \widetilde{\alpha}_2))$*: Parse* χ *as* (χ_1, χ_2)*. Output* $(\gamma \cdot g_0^{\widetilde{\alpha}_1} h_0^{\widetilde{\alpha}_2}, (\chi_1 \cdot \zeta^{\widetilde{\alpha}_1}, \chi_2 \cdot \zeta^{\widetilde{\alpha}_2}))$.
  - $\mathsf{MaulH}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp}, x = (c_1, c_2), \widetilde{r} = (\widetilde{\alpha}_1, \widetilde{\alpha}_2))$*: Output* $\hat{e}(c_1^{\widetilde{\alpha}_1} \cdot (c_2/\mathsf{aux})^{\widetilde{\alpha}_2}, g_1)$.
  - $\mathsf{MaulHK}(\mathsf{pp}, \mathsf{crs}', \mathsf{hk} = (\alpha_1, \alpha_2), \widetilde{r} = (\widetilde{\alpha}_1, \widetilde{\alpha}_2))$*: Output* $\widetilde{\mathsf{hk}} \leftarrow (\alpha_1 + \widetilde{\alpha}_1, \alpha_2 + \widetilde{\alpha}_2)$.
  - $\mathsf{SampW}(\mathsf{pp}, \mathsf{crs}')$*: Output* $\widetilde{w} \leftarrow\!\!\$\ \mathbb{Z}_{\mathsf{q}}$.
  - $\mathsf{ReranE}(\mathsf{pp}, \mathsf{crs}', x = (c_1, c_2), \widetilde{w})$*: Output* $(c_1 \cdot g_0^{\widetilde{w}}, c_2 \cdot h_0^{\widetilde{w}})$.
  - $\mathsf{ReranH}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp} = (\gamma, \chi), x, \widetilde{w})$*: Output* $\hat{e}(\gamma^{\widetilde{w}}, g_1)$.

Theorem 7.2. *Construction 7.2 is a malleable trapdoor smooth-projective hash function for* $\mathtt{L}^{\mathsf{ElGamal}}$*, assuming the DDH assumption holds.*

Proof. Construction 6.6 is a TSPHF for $\mathtt{L}^{\mathsf{ElGamal}}$. Hence, it suffices to prove the remaining malleability properties.

- **Projection consistency.** Let us consider a keypair constituted of $\mathsf{hk} = (\alpha_1, \alpha_2)$ and $\mathsf{hp} = (\gamma, \chi)$, where $\gamma = g_0^{\alpha_1} h_0^{\alpha_2}$. First, we blind the keypair with blinding factor $\widetilde{r} = (\widetilde{\alpha}_1, \widetilde{\alpha}_2)$, obtaining $\widetilde{\mathsf{hp}} = (\gamma \cdot g_0^{\widetilde{\alpha}_1} h_0^{\widetilde{\alpha}_2}, (\chi_1 \cdot \zeta^{\widetilde{\alpha}_1}, \chi_2 \cdot \zeta^{\widetilde{\alpha}_2}))$ and $\widetilde{\mathsf{hk}} = (\alpha_1 + \widetilde{\alpha}_1, \alpha_2 + \widetilde{\alpha}_2)$. Then:

$$\begin{aligned}
\mathsf{Hash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \widetilde{\mathsf{hk}}, x = (c_1, c_2)) &= \hat{e}(c_1^{\alpha_1 + \widetilde{\alpha}_1} \cdot (c_2/\mathsf{aux})^{\alpha_2 + \widetilde{\alpha}_2}, g_1) \\
&= \hat{e}(c_1^{\alpha_1} \cdot c_1^{\widetilde{\alpha}_1} \cdot (c_2/\mathsf{aux})^{\alpha_2} \cdot (c_2/\mathsf{aux})^{\widetilde{\alpha}_2}, g_1) \\
&= \hat{e}(c_1^{\alpha_1} \cdot (c_2/\mathsf{aux})^{\alpha_2}, g_1) \cdot \hat{e}(c_1^{\widetilde{\alpha}_1} \cdot (c_2/\mathsf{aux})^{\widetilde{\alpha}_2}, g_1) \\
&= \mathsf{Hash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hk}, x) \cdot \mathsf{MaulH}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp}, x, \widetilde{r}).
\end{aligned}$$

- **Element consistency.** Let us consider a statement $x = (c_1, c_2)$. We blind the statement with blinding

factor $\widetilde{w}$, obtaining $\widetilde{x} = (c_1 \cdot g_0^{\widetilde{w}}, c_2 \cdot h_0^{\widetilde{w}})$. Then:

$$\begin{aligned}
\mathsf{Hash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hk} = (\alpha_1, \alpha_2), \widetilde{x}) &= \hat{e}((c_1 \cdot g_0^{\widetilde{w}})^{\alpha_1} \cdot (c_2 \cdot h_0^{\widetilde{w}}/\mathsf{aux})^{\alpha_2}, g_1) \\
&= \hat{e}(c_1^{\alpha_1} \cdot (g_0^{\widetilde{w}})^{\alpha_1} \cdot (c_2/\mathsf{aux})^{\alpha_2} \cdot (h_0^{\widetilde{w}})^{\alpha_2}, g_1) \\
&= \hat{e}(c_1^{\alpha_1} \cdot (c_2/\mathsf{aux})^{\alpha_2}, g_1) \cdot \hat{e}((g_0^{\widetilde{w}})^{\alpha_1} \cdot (h_0^{\widetilde{w}})^{\alpha_2}, g_1) \\
&= \mathsf{Hash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hk}, x) \cdot \hat{e}((g_0^{\alpha_1} \cdot h_0^{\alpha_2})^{\widetilde{w}}, g_1) \\
&= \mathsf{Hash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hk}, x) \cdot \hat{e}(\gamma^{\widetilde{w}}, g_1) \\
&= \mathsf{Hash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hk}, x) \cdot \mathsf{ReranH}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp}, x, \widetilde{w}).
\end{aligned}$$

- **Membership preservation.** Let us consider again a statement $x = (c_1, c_2)$. We blind the statement with blinding factor $\widetilde{w}$, obtaining $\widetilde{x} = (c_1 \cdot g_0^{\widetilde{w}}, c_2 \cdot h_0^{\widetilde{w}})$. If $x \in L_{\mathsf{lpp},\mathsf{aux}}$, x can be parsed as $(c_1, c_2) = (g_0^w, \mathsf{aux} \cdot h_0^w)$ for some w. On the other hand, $\widetilde{x}$ can be parsed as $(c_1, c_2) = (g_0^{w+\widetilde{w}}, \mathsf{aux} \cdot h_0^{w+\widetilde{w}}) \in L_{\mathsf{lpp},\mathsf{aux}}$.
- **Key indistinguishability** follows from the fact that $(\zeta^{\widetilde{\alpha}_1}, \zeta^{\widetilde{\alpha}_2})$ is uniformly distributed over $\mathbb{G}_1^2$, and that $\mathsf{hp} \cdot g_0^{\widetilde{\alpha}_1} h_0^{\widetilde{\alpha}_2}$ is the unique solution for which $\widetilde{\mathsf{hp}}$ is an accepting projection key.
- **Element indistinguishability** follows from the DDH assumption.

□

7.3 Subversion-resilient PAKE from MTSPHFs

In Figure 7.1, we show our subversion-resilient PAKE protocol from MTSPHF. In what follows, Enc is the encryption algorithm of a rerandomizable CPA-secure encryption scheme; $\mathcal{P}, \mathcal{V}$, and ZKEval are algorithms of a CM-NIZK for the language $\mathcal{L}_{\mathsf{lpp}} = \{c \mid \exists(\mathsf{aux}, w) \text{ s.t. } \mathsf{Enc}(\mathsf{lpp}, \mathsf{aux}; w) = c\}$ with a set of allowable transformations $\mathcal{T} = \{T_{\widetilde{w}}\}$ mapping a ciphertext c encrypting a message aux with randomness w into a new ciphertext $\widetilde{c}$ obtained from rerandomizing c with randomness $\widetilde{w}$. Finally, the remaining algorithms are part of a MTSPHF for $L_{\mathsf{lpp},\mathsf{aux}} = \{c \mid \exists w \text{ s.t. } \mathsf{Enc}(\mathsf{lpp}, \mathsf{aux}; w) = c\}$. We implicitly assume that the public parameters pp include the linear public parameters lpp required from the encryption scheme and the MTSPHF, as well as the public parameters required for the CM-NIZK.

Theorem 7.3. *The protocol depicted in Figure 7.1 wsrUC-realizes the $\mathcal{F}_{\mathsf{PAKE}}$ ideal functionality in the $\mathcal{F}_{\mathsf{crs}}$-hybrid model, assuming secure channels between cores and their respective firewall.*

Proof (Sketch). In order to prove our theorem, we proceed by showing (i) correctness of the protocol, (ii) strong sanitation of firewalls, and (iii) simulation of the protocol by assuming honest parties being composed of an Honest core and a SemiHonest RF. (i) trivially follows from the consistency properties we introduced with our definition of MTSPHF. In particular, this is possible given that cores have access to the randomness that was used from their respective firewall to sanitize messages. (ii) follows by showing that firewalls can sanitize outbound messages by leveraging the malleability properties of the respective primitives. Finally, the high-level idea of item (iii) for the simulation of the protocol consists in following the structure of the original proof, which we reported in Section 6.4. The most salient variations we introduce arise from replacing the CCA-secure ciphertexts with rerandomizable CPA-secure ones plus CM-NIZKs (particularly, in the characterization of "oracle-generated" tuples, which are now classified according to the output of the extraction from the proofs), as well as the evaluation of portions of session keys due to the malleability properties introduced within the protocol. The latter is mostly a syntactic change. The details of the proof are deferred to Appendix B.1. □

A hand-wavy performance comparison. To complete our discussion, we provide a hand-wavy comparison between our new protocol and to the subversion-resilient PAKE protocol from OT presented in [CMMV24]. In particular, we focus on analyzing round complexity and element complexity.

It is clear, by Figure 7.1, that our protocol is a 1-round PAKE. For what concerns element complexity, we require (i) a rerandomizable CPA-secure encryption scheme, (ii) a MTSPHF, and (iii) a CM-NIZK for the

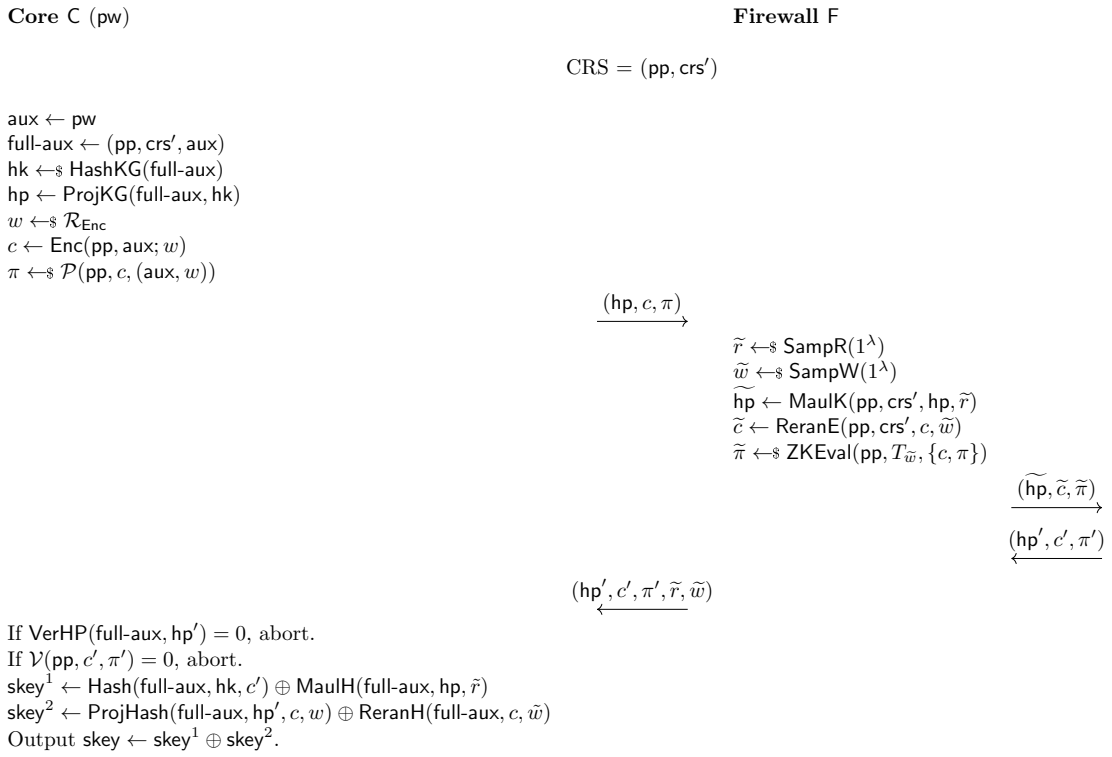


Figure 7.1: Our subversion-resilient PAKE protocol from MTSPHFs.

language of rerandomizable CPA-secure encryption schemes. Item (i) can be efficiently instantiated with the ElGamal encryption scheme by using two group elements embedding elements of size n. Item (ii) can also be efficiently instantiated with our construction provided in Section 7.2.3. In particular, $\mathsf{hp} = (\gamma, (\chi_1, \chi_2))$, where γ is a single group element and (χ_1, χ_2) are two group elements embedding inputs of size n. Finally, (iii) can be efficiently instantiated by using Groth-Sahai proofs, requiring a constant number of group elements embedding elements of size n.

Let us now consider the protocol of [CMMV24]. Very roughly, this protocol requires two subversion-resilient OTs (one for each party) *for each bit* of the password, as well as establishing a split-authenticated channel between parties. The only known instantiations of these building blocks are formalized in [CMMV24] and require, respectively, a constant number of group elements embedding elements of size λ for each subversion-resilient OT, and a BLS signature for each outbound message (alongside an initialization phase to setup parties' verification keys). From a round-complexity standpoint, the subversion-resilient OTs require two rounds (by batching together all the OT runs), and the split-authenticated channel introduces two additional rounds in its initialization phase, as well as an additional internal round (*i.e.*, a round between a core and its firewall) for each outbound message.

Putting everything together, we have that our subversion-resilient PAKE protocol runs in a single round (as opposed to the 4 rounds plus internal rounds of [CMMV24]) by using $O(n)$ bits per party (as opposed $O(n\lambda)$ bits, for an n-bit password, plus some overhead deriving from BLS signatures, for each party).

Part IV

Robust NIZK Combiners

Part Overview

In this portion of the composition, we depart from the subversion-resilient UC framework and consider the world of robust combiners as an alternative avenue to provide fault tolerance against (potentially) compromised devices used by honest parties. In particular, we introduce the first complete characterization of robust Non-Interactive Zero-Knowledge (NIZK) combiners, featuring formal definitions, impossibility results, and incomparable constructions.

Organization. This portion of the composition is structured in a single section (Section 8), as the results hereby presented are self-contained.

8 Robust Non-Interactive Zero-Knowledge Combiners

In this section, we introduce the first comprehensive characterization of robust combiners for Non-Interactive Zero-Knowledge: a result that also appeared in [CMVX25]. For convenience, we briefly summarize the notion of combiners in what follows, and how robustness has been formally characterized in prior literature.

Robust combiners. Informally, a combiner C for a primitive $\mathcal{P}$ is a cryptographic construction that, given access to a set of candidate instatiations of $\mathcal{P}$, itself instantiates the primitive $\mathcal{P}$. In particular, a t-out-of-n combiner C is *secure* assuming at least $t < n$ of the candidates $\Pi_1, \ldots, \Pi_n$ are *secure*.

In spite of the simple intuition, capturing combiners by means of a formal definition is a non-trivial task. Indeed, one cannot just say that a combiner C is secure assuming at least t of the n candidates $\Pi_1, \ldots, \Pi_n$ are secure. This is because C could always ignore all of the n candidates and construct the primitive $\mathcal{P}$ from scratch. To address this shortcoming, [HKN+05] introduced the notion of *robust* combiners, which makes the reductions to the underlying candidates explicit by asking that, for every adversary $\mathcal{A}$ breaking security of C, there exist at least $n-t+1$ reductions $\{\mathsf{Red}_i\}_{i\in[n-t+1]}$ (with running time at most a polynomial factor slower than the running time of $\mathcal{A}$) that break the security of distinct candidates. This way, we have a formal guarantee that C is secure assuming at least t of the candidates are secure (rather than some external computational assumption), and moreover C cannot construct $\mathcal{P}$ from scratch (as ensured by the existence of the above reductions). Concretely, the 1-out-of-n combiner for collision-resistant hashing C described in Section 1.4.1 is a *robust* combiner as, for every adversary $\mathcal{A}$ breaking CRH of C, there should exist at least $n-t+1 = n$ reductions breaking CRH of distinct candidates (*i.e.*, the existence of a collision for C implies the existence of collisions for *all* n candidates).

8.1 Summary

In this work, we provide the first comprehensive characterization of robust NIZK combiners, including formal definitions, impossibility results, and different constructions.

Defining robust NIZK combiners. At first, we provide a formal definition for t-out-of-n robust NIZK combiners. Our flavour of robustness follows the same rationale of [HKN+05], with non-trivial adaptations due to NIZK being a multi-property primitive that admits computational and statistical variants (more in the technical overview). In particular, we focus on NIZKs satisfying (knowledge) soundness and zero-knowledge, with our definition capturing (i) whether the combiner accesses its candidates in a black-box manner; (ii) whether the zero-knowledge simulator accesses the adversary as a black-box; (iii) whether the knowledge extractor (in the case of knowledge soundness) accesses the adversary as a black-box. For simplicity, in the rest of this subsection, we only focus on dimension (i) and always assume black-box extraction and simulation.

Negative results. Next, we argue that no t-out-of-n (even non-black-box) robust NIZK combiner exists for $t \leq \lfloor n/2 \rfloor$, unless the polynomial hierarchy collapses. This impossibility is unconditional and is sufficient to rule out robust NIZK combiners. Indeed, we show that if half of the input schemes are zero-knowledge (and not sound) and the second half are sound (and not zero-knowledge), then no combiner can be secure with such input candidates.

One may wonder: what happens if we assume that half of the input candidates are both zero-knowledge and sound, whereas the remaining half has no security at all? We show that no combiner can work in this setting either. However, this latter impossibility is conditioned on the existence of (i) NIZK with statistical soundness (and computational zero-knowledge), and (ii) NIZK with statistical zero-knowledge (and computational soundness).

Finally, we additionally rule out combiners that may take as input exclusively computationally-secure candidates, albeit only for the black-box case. This latter impossibility is conditioned on the existence of

(i) NIZK with statistical soundness (and computational zero-knowledge), (ii) NIZK with statistical zero-knowledge (and computational soundness), and (iii) dual-mode commitments.[25]

Positive results. To complement the above negative results, we provide constructions of robust NIZK combiners. In particular, we obtain three different (and incomparable) combiners:

- Our first construction (Section 8.4) yields a succinctness-preserving t-out-of-n black-box robust NIZK combiner assuming $t > \lfloor n/2 \rfloor$, where $n, t \in \mathsf{poly}(\lambda)$. This construction supports both soundness as well as knowledge soundness, and works for so-called *homomorphic* languages [Cra96], as defined by Maurer [Mau09]. These languages are such that (i) the witness w belongs to a group $(\mathbb{G}, \star)$, (ii) the statement x belongs to a group $(\mathbb{H}, \otimes)$, and (iii) there is a function $f : \mathbb{G} \to \mathbb{H}$ such that $f(x) \otimes f(y) = f(x \star y)$ and f is one-way function (*i.e.*, it is infeasible to compute x from $f(x)$ for a randomly chosen x). Concrete examples of such languages include the language for proving the knowledge of a discrete logarithm, that a tuple is Diffie-Hellman, the equality of embedded values, and more.
- Our second construction (Section 8.5) yields a succinctness-preserving t-out-of-n non-black-box robust NIZK combiner assuming $t > \lfloor n/2 \rfloor$, where $n, t \in O(1)$. This construction works for arbitrary NP languages.
- Our third construction (Section 8.6) yields a t-out-of-n non-black-box robust NIZK combiner assuming $t > \lfloor 2n/3 \rfloor$, where $n, t \in \mathsf{poly}(\lambda)$. This construction also works for arbitrary NP languages.

The aforementioned constructions are summarized in Table 8.1. Due to our impossibility results, our first and second constructions are essentially tight in terms of achievable parameters. We leave it as an interesting open problem to construct a black-box robust NIZK combiner supporting all of NP for the parameters regime $t > \lfloor n/2 \rfloor$ with $n, t \in \mathsf{poly}(\lambda)$.

Combiner	Threshold	Type	Input/Output Language	Proof Size
Section 8.4	$t > \lfloor n/2 \rfloor$	Black-box	Homomorphic	$n \cdot \lvert\pi\rvert$
Section 8.5	$t > \lfloor n/2 \rfloor$, $n, t \in O(1)$	Non-black-box	NP	$\mathsf{poly}(\lvert\pi\rvert)$
Section 8.6	$t > \lfloor 2n/3 \rfloor$	Non-black-box	NP	$n \cdot \lvert\pi\rvert + \mathsf{poly}(\lvert x\rvert, \lvert w\rvert, \lambda)$

Table 8.1: An overview of our positive results in terms of threshold parameters, type of black-boxness, supported language, and proof size. In particular, by *Input/Output Language*, we refer to both the language that must be supported by the input candidates and the language supported by the NIZK returned from the combiner. From the table, it can be seen that all our combiners are *language preserving*. We assume that all the input protocols issue proofs of size at most $|\pi|$ (where $|\pi|$ is at least polynomial in the security parameter λ). We note that the first and the second constructions preserve the succinctness of the input candidates.

8.1.1 Technical overview

Below, we give an overview of the main technical ideas behind our results.

Defining robust NIZK combiners. A NIZK scheme should enjoy at least two properties: zero-knowledge and soundness, which can be either computational or statistical. We discuss how these characteristics are embedded in our definitions, and how we formally characterize robustness for this primitive.

[25] A dual-mode commitment scheme is a commitment scheme defined in the CRS model, where the CRS can be sampled in two different modes, so that the commitment scheme is either statistically binding or statistically hiding according to the chosen mode. This primitive can be instantiated from a variety of assumptions, including DDH [Lin15].

Capturing the multi-property nature of NIZK. The first design choice stems from the multi-property nature of NIZK: in order for the combiner to be secure, should t candidates satisfy all the security properties at once, or does it suffice for t candidates to be secure with respect to each *individual* property? In this work, we follow the latter (more general) approach. Indeed, for our combiners, it suffices that t candidates are (knowledge) sound and t candidates are zero-knowledge, but these two subsets may not be identical. Hence, ours are *multi-property combiners* as defined in [FL08, FLP08, FLP14]. On the other hand, all of our constructions require that all of the candidates have a negligible completeness error: a common assumption in the literature on combiners.[26]

Characterizing robustness. In order to establish robustness of a combiner C, for every adversary $\mathcal{A}$ breaking the security of C, one has to design at least $n-t+1$ reductions breaking the security of distinct candidates. The same rationale applies also in the context of a multi-property primitive: if an adversary successfully attacks *any* of the security properties of the combiner, there should exist $n-t+1$ reductions to one of the security properties of the candidates. Indeed, even if a single security property is broken, the combiner (resp. the candidate) would no longer be securely instantiating the desired primitive. We capture this requirement by having two distinct definitions: one for soundness, and one for zero-knowledge. Roughly, we say a robust NIZK combiner is sound if, for every adversary $\mathcal{A}$ breaking the soundness of C, there exist at least $n-t+1$ reductions that break either the zero-knowledge of at least $n-t+1$ distinct candidates *or* the soundness of at least $n-t+1$ distinct candidates. The definition for zero-knowledge follows similarly. We stress that this definition allows, for example, to reduce the soundness of the combiner to the zero-knowledge of the input primitives. Despite this looking like an artificial way to make our definition more general, we show that this is instead instrumental to the design of certain combiners. More details on this later in this section.

Capturing computational and statistical properties. Our characterization of robustness is general enough to capture both computational and statistical variants of our security properties of interest. Indeed, whenever an adversary attacks the combiner in time τ, there must exist reductions attacking sufficiently-many candidates in time $\mathsf{poly}(\tau)$. Hence, our combiners may accept as input candidates that have computational or statistical properties while outputting a NIZK that has computational or statistical properties.

A relation with standard security properties. As a sanity check, throughout the formal discussion, we also argue that any robust NIZK combiner satisfying our definition, when instantiated with at least t secure NIZK candidates, yields a NIZK satisfying the standard definitions of (knowledge) soundness and zero-knowledge, and thus it can be employed directly in any application requiring these properties.

Why is our characterization of robustness the right one? Our definitions capture exactly what one should expect from a *robust* combiner - namely, that the security of the combiner only relies on the input candidates. In particular, in our definition, we state that the only way to attack the security of the combiner is to attack the security of the input candidates, meaning that the combiner cannot *create* security. This implies that the combiner cannot rely on additional computational assumptions (that are not directly implied by the input candidates), but also that no information-theoretic security should be created. For example, a combiner that, on input only NIZK candidates that enjoy computational zero-knowledge and computational soundness, returns a NIZK scheme that is statistically zero-knowledge would not satisfy our security definition. Indeed, in such a case, an (unbounded) adversary attacking the zero-knowledge of the combiner can never be meaningfully turned into an adversary attacking any of the properties of the input schemes, as their security guarantees are expected to hold only against polynomial-time adversaries. We believe that this feature captures well the fact that a combiner should be *simple* and should not create security without relying on the input schemes. This is, for example, in contrast with [HKNY23] where the authors construct

[26] But note that in the setting of NIZK proof systems, one can always amplify non-negligible completeness to overwhelming completeness (at the price of a larger running time for the prover). Alternatively, one can apply solutions à la [HKN+05] by having the combiner pre-emptively "testing" the correctness of each candidate.

a combiner for commitment schemes which always returns a statistical binding commitment (regardless of the security of the input schemes). We refer to Appendix C.1 for a more detailed comparison.

Impossibility results for $t \leq \lfloor n/2 \rfloor$. First, we explain how to prove that no robust NIZK combiner exists without assuming the majority of the candidates being secure. Let $n = 2$ and $t = 1$, and assume that there exists a 1-out-of-2 robust NIZK combiner C. Consider the following dummy NIZK candidates: (i) the prover in Π_1, upon input (x, w), always outputs 0 and the verifier only accepts 0 as a proof; (ii) the prover in Π_2, upon input (x, w), always outputs w and the verifier accepts if and only if w is a valid witness for x. Clearly, Π_1 is unconditionally zero-knowledge but not sound, whereas Π_2 is unconditionally sound but not zero-knowledge.

Yet, the definition of robust combiner implies that C must work for Π_1 and Π_2. In particular, given any unbounded adversary that breaks either the zero-knowledge or the soundness of C, there should exist $n - t + 1 = 2$ reductions, with at most a polynomial slowdown, that break either the zero-knowledge or the soundness of both Π_1 and Π_2; but this is impossible, as Π_1 is unconditionally zero-knowledge (resp. Π_2 is unconditionally sound). It follows that no (even unbounded) adversary can break C, and in particular that C must be both unconditionally sound and unconditionally zero-knowledge. But such a combiner cannot exist, unless the polynomial hierarchy collapses. Note that, for the above argument, it is completely irrelevant whether the combiner is black-box or not.

Ruling out more types of combiners: Step 1. We note that the proof we have just sketched crucially relies on the fact that C is a multi-property combiner. This, informally, requires that the combiner works even if one instance is zero-knowledge (but not sound) and the other instance is sound (but not zero-knowledge). In particular, our impossibility proof does not rule out the existence of a combiner that retains its security when receiving as input one scheme that is both zero-knowledge and sound (while the other scheme has no security at all). With our second impossibility result, we show that even in this case no robust NIZK combiner can exist.

Assume there exists a NIZK protocol Π_{SSND} that is statistically sound and computationally zero-knowledge, and a protocol Π_{SZK} that is computationally sound and statistically zero-knowledge. Suppose now by contradiction that a 1-out-of-2 combiner exists. Such a combiner, in particular, must retain its security when executed with the input candidates Π_{SSND} and Π_{SZK}.

The security of the combiner guarantees that if there exists an adversary that breaks the soundness of the combiner C, it is possible to construct reductions that either break the soundness or the zero-knowledge of both input candidates. However, because we assumed Π_{SSND} to be statistically sound, no (even unbounded) reductions can break the soundness of both Π_{SSND} and Π_{SZK}. Similarly, because we assumed that Π_{SZK} is statistical zero-knowledge, no (even unbounded) reductions can break the zero-knowledge of both Π_{SSND} and Π_{SZK}. This means that no (unbounded) adversary is able to attack the soundness of C (*i.e.*, the combined scheme is statistically sound).

Using a similar argument, we can claim that the zero-knowledge of C is statistical. But as recalled above, no NIZK can be at the same time statistically sound and statistically zero-knowledge, unless the polynomial hierarchy collapses. Note that our impossibility is based solely on the existence of Π_{SSND} and Π_{SZK}. Hence, it relies on the assumption that it is possible to design statistical zero-knowledge (computationally sound) and statistically sound (computationally zero-knowledge) NIZK protocols, but we do not need to rely on any specific instantiation of Π_{SSND} and Π_{SZK}.

Ruling out more types of combiners: Step 2. The impossibility we sketched above does not rule out the existence of a combiner that always produces a NIZK scheme that is only computationally zero-knowledge and computationally sound, as long as one of the input candidates is both computationally sound and computationally zero-knowledge. Hence, one may wonder whether such a combiner exists. We show that this is again not the case, albeit only for black-box combiners.

Suppose by contradiction that such a combiner C does exist. By assumption, the combiner must also work when it is run on input two schemes Π_1 and Π_2, both computationally zero-knowledge and computa-

tionally sound. The idea of our proof is to consider special instantiations of Π_1 and Π_2, which we gradually modify into two new protocols Π_{SZK} and Π_{SSND}, such that (as before) Π_{SZK} is statistical zero-knowledge (but computational sound) and Π_{SSND} is statistical sound (but computationally zero-knowledge). If the behaviour of (Π_1, Π_2) is computationally indistinguishable from the behaviour of $(\Pi_{\mathsf{SSND}}, \Pi_{\mathsf{SZK}})$, then C keeps working when it is executed on Π_{SZK} and Π_{SSND}. But now, following the arguments of our previous impossibility, we can again use C to construct a NIZK scheme that is statistical zero-knowledge and statistical sound (which cannot exist), thus reaching a contradiction. We refer to Section 8.3 for the precise descriptions of (Π_1, Π_2) and $(\Pi_{\mathsf{SSND}}, \Pi_{\mathsf{SZK}})$, and for all the formal details of our three impossibility proofs.

Combiner for homomorphic languages. Next, we discuss concrete instantiations of combiners that match our definitions. We start with the combiner for homomorphic languages. For simplicity, we describe the combiner for proving knowledge of the discrete logarithm $w \in \mathbb{Z}_{\mathsf{q}}$ of a group element $x = g^w \in \mathbb{G}$ (where g is the generator of the cyclic group $\mathbb{G}$ with order q). In the formal discussion, we give a more general construction that works for a large class of homomorphic languages, as defined by Maurer [Mau09]. Concrete examples of such languages include Diffie-Hellman, Guillou-Quisquater, proofs for equality of embedded values, and more.

Assume we have n NIZK candidates $\Pi_1, \ldots, \Pi_n$, each for proving knowledge of the discrete logarithm of a group element. We aim at constructing a combiner that returns a NIZK Π for the language of discrete logarithms which guarantees security (*i.e.*, zero-knowledge and knowledge soundness) assuming that t out of the n input schemes are secure.[27] The prover takes a random polynomial p of degree $t-1$ such that $p(0) = w$, and computes the shares $(i, p(i))$ for each $i \in [n]$, thus obtaining a t-out-of-n Shamir's secret sharing of the witness w. Then, the prover derives n *sub-statements* $x_i \leftarrow g^{p(i)}$ and runs the prover algorithm of the scheme Π_i on input the sub-statement x_i and its witness $p(i)$ for each $i \in [n]$, obtaining a proof π_i. The final proof π consists of all the generated proofs $\{\pi_i\}_{i\in[n]}$ and all the sub-statements $\{x_i\}_{i\in[n]}$. The verifier first checks that all the proofs are accepting (with respect to each individual scheme), and then runs the reconstruction of Shamir's secret sharing "in the exponent" in order to check that $g^{p(0)}$ equals x and that $g^{p(i)}$ equals x_i for all $i \in [n]$. The latter is possible because Shamir's reconstruction is based on Lagrange interpolation, which relies on summing t polynomial evaluations (multiplied by some constant). This operation can be done "in the exponent" by exploiting the homomorphic properties of the function $f(\alpha) = g^\alpha$ between the groups $\mathbb{G}$ and $\mathbb{H}$.

Zero-knowledge is guaranteed because the witnesses w_i of at least t of the sub-statements x_i are protected (thanks to the assumption that t candidates are zero-knowledge and that $t > \lfloor n/2 \rfloor$), hence the privacy property of Shamir's secret sharing implies that the witness w is protected. Knowledge soundness is guaranteed by the fact that, for at least t of the sub-statements x_i, the proof will be generated using a knowledge sound candidate, hence, it will be possible to extract t shares (polynomial evaluations), which we can use to reconstruct a candidate witness w'. It remains to argue that w' corresponds to w. But this must be the case, given that $g^{p(0)} = g^{w'}$.

Combiner from recursive proofs. The combiner we have just described works for a limited class of languages. Next, we propose a NIZK combiner that supports all of NP, assuming the underlying candidates also support all of NP. To simplify the description of our scheme, let us consider here the case where $n = 3$ and $t = 2$. The prover performs a recursive proof, using all possible combinations of the given candidates. More specifically, the prover computes a proof that $x \in \mathcal{L}$ using the i-th candidate, thus obtaining a proof π_i; then, using the scheme j (with $j > i$) it proves the following: "I know a proof π_i for the statement $x \in \mathcal{L}$ which the verifier of the scheme Π_i would accept". We denote this latter proof with $\pi_{i,j}$. The prover proceeds as above for each $i, j \in \{1, 2, 3\}$ with $j > i$, and finally it sends to the verifier $\pi_{1,2}, \pi_{1,3}, \pi_{2,3}$. The verifier accepts if all three proofs are accepting.

[27] Here, for simplicity, when we say that a scheme is secure, we mean that it enjoys both zero-knowledge and (knowledge) soundness. However, we remark that our construction also works in case the subsets of candidates that are either zero-knowledge or (knowledge) sound are not identical.

The above scheme is zero-knowledge, because every proof $\pi_{i,j}$ hides the witness: either the i-th candidate is zero-knowledge (and hence w is protected), or the j-th candidate is zero-knowledge (and hence the proof π_i, that could have leaked something about the witness, remains hidden to the adversary). Note that this holds given that $t > \lfloor n/2 \rfloor$. Knowledge soundness holds, because there exists at least one recursive proof $\pi_{i,j}$, generated using the i-th and the j-th candidates which are both knowledge sound (this again follows from the fact that $t > \lfloor n/2 \rfloor$). The construction can be extended to the case where n and t are arbitrary, but to keep the running time of the prover polynomial, we must require that n and t are constant in the security parameter.

Combiner from MPC-in-the-head. With our third construction we want to remove the limitation on n and t, while still supporting all of NP. The construction is inspired by the MPC-in-the-head approach proposed in [IKOS07], and a similar approach has been used in [GJS19] to construct amplifiers for zero-knowledge from sub-exponentially secure public-key encryption.

Our combiner is based on an n-party information-theoretic multi-party computation (MPC) protocol for the following function g: *given as input n shares of a t-out-of-n secret sharing scheme, perform the reconstruction thus obtaining a value w, and return* 1 *if and only if w is a valid witness for x.* The MPC protocol guarantees security as long as k parties are corrupted (with $k < n/2$).

To simplify the description of the combiner, let us first assume that the combiner has access to a non-interactive commitment scheme; we will explain later how to remove this assumption. Let $\mathsf{P}_1, \ldots, \mathsf{P}_n$ be the n parties of the MPC protocol. The prover of our combiner computes a t-out-of-n secret sharing of the witness w, thus obtaining the shares $w_1, \ldots, w_n$, and executes the MPC protocol for the above function g in its head simulating the role of each party P_i with input w_i. This process generates, for each party P_i, a set of ingoing and outgoing messages that we denote with $\mathsf{in\text{-}out}_i$ for $i \in [n]$. The prover then, for each $i \in [n]$, commits to these messages and uses the candidate Π_i to prove that the messages in $\mathsf{in\text{-}out}_i$ correspond to the view generated by running party P_i of the above MPC protocol, and that the committed values match the contents of $\mathsf{in\text{-}out}_i$. The prover finally sends all the generated proofs and all the commitments to the values $\mathsf{in\text{-}out}_i$. The verifier first checks whether there is a match between the commitments corresponding to the ingoing messages of a party P_i and the outgoing messages of all the other parties, and accepts the proof if all the proofs are valid.

The zero-knowledge property holds, given that there can be at most $n-t$ broken NIZK candidates which reveal the views of at most $n-t$ parties. In particular, these parties can be considered to be semi-honestly corrupted in the run of the MPC protocol. However, since the MPC protocol is secure against up to $k < n/2$ corrupted parties, the inputs (*i.e.*, the shares of the original witness) of the *honest parties* are still protected. Moreover, the transcript of the protocol reveals no information about these shares due to the hiding property of the commitment scheme. Given that at most $n-t$ shares can be leaked, the witness itself is protected due to the privacy of the secret sharing scheme.

To argue (knowledge) soundness, ideally we would like to rely on the correctness of the MPC protocol, similarly to what is done in [IKOS07]. Unfortunately, we cannot do that, given that a malicious prover can compute some of the messages in a malicious way. In particular, even if the (malicious) prover shows that it has generated correctly the messages of at least t parties, these parties may have been run using some bad randomness, which causes an incorrect output (*i.e.*, the correct output should be 0 because $x \notin \mathcal{L}$, but the MPC protocol returns 1). To solve this problem we require the MPC protocol to be *robust* - a property that was also used in [IKOS07] to prove some of their results. The robustness property requires that even if the randomness used by the honest parties is chosen adversarially, and $n-t$ parties are fully controlled by the adversary, the output of the honest parties is always correct. One possible instantiation of such an MPC protocol is [BGW88], which tolerates a corruption threshold of $k < \lfloor \frac{n-1}{3} \rfloor$, and, as claimed in [GMO16], is also robust.

It remains to explain how to remove the assumption that the combiner is given access to a secure non-interactive commitment scheme. We do that by having the combiner construct a non-interactive commitment scheme using the given NIZK candidates themselves. More precisely, we rely on [HN24, LMP24], in which the authors prove that one-way functions exist if: (i) every language in NP has a zero-knowledge proof

argument (*i.e.*, $\mathsf{NP} \subseteq \mathsf{ZKA}$), and (ii) ZKA contains non-trivial languages (*i.e.*, $\mathsf{ZKA} \not\subseteq \mathsf{ioP/poly}$). Hence, our combiner can first obtain a one-way function candidate f_i using the NIZK candidate Π_i, combine the one-way function candidates $\{f_i\}_{i\in[n]}$ into a secure one-way function construction f[28], and finally use f to obtain[29] a two-round commitment using Naor's construction [Nao91], that we can make non-interactive by plugging the first round of the commitment in the CRS. This makes our combiner unconditional under assumptions (i) and (ii); however, note that if assumption (i) does not hold there are languages in NP that do not admit zero-knowledge arguments (and hence, the problem of designing NIZK combiners becomes less interesting). Moreover, if (ii) does not hold, we have that $\mathsf{NP} \subseteq \mathsf{ioP/poly}$ (as $\mathsf{NP} \subseteq \mathsf{ZKA}$ is assumed to hold for our purposes, and $\mathsf{ZKA} \subseteq \mathsf{ioP/poly}$).

We stress that the above transformation fundamentally relies on both soundness *and* zero-knowledge of the candidates. In particular, while binding is unconditional, hiding relies on the security of the OWF, which in turn relies on the assumption that at least one candidate is simultaneously sound and zero-knowledge. This occurs if at least $n/2+1$ candidates are sound and $n/2+1$ candidates are zero-knowledge.

This fact is instrumental in proving the security of the overall combiner. In particular, note that the zero-knowledge property of the combiner relies on the hiding of the commitment, which, as just mentioned, relies on both the soundness as well as the zero-knowledge of the input candidates. Hence, our combiner represents a concrete example where the zero-knowledge property surprisingly relies also on the soundness of the input candidates. More precisely, when formally proving the zero-knowledge property of the combiner, we will show that if an adversary breaks the zero-knowledge of the combined scheme, then this adversary could be used to break either the *soundness* of at least $n-t+1$ input candidates or the zero-knowledge of at least $n-t+1$ input schemes.

We finally note that, since the prover runs the NIZK candidates to prove a statement involving the commitments (which, in turn, are obtained using the NIZK candidates themselves), the combiner is non-black-box in the use of the candidates.

8.1.2 Related work

Next, we discuss related work on combiners, as well as alternative approaches relating to NIZK amplification.

Combiners. Several previous works have considered and applied the concept of combiners, either implicitly or explicitly. The first formal treatment of robust combiners is due to [HKN+05], in the setting of oblivious transfer and key agreement protocols. In particular, this was the first work pointing out the necessity of making the reduction explicit in the definition, in order to rule out trivial solutions that ignore the underlying candidates.

Most relevant to our work is [Som06] that studies robust combiners for interactive proof systems. This work rules out very specific forms of robust combiners (*e.g.*, those that are deterministic or that work by splitting the statement into multiple parts).

A long line of research studies robust combiners for hash functions [FL08, FLP08, Mit13, MP14]. In this context, it is particularly challenging to obtain *short* combiners, in which the output length of the combiner is significantly shorter than the concatenation of the candidates' outputs. While there are lower bounds for short robust hash functions combiners [BB06, CRS+07, Pie07, Pie08, Mit12], recent work has shown that such lower bounds can be overcome in the random oracle model [DFG+23]. Robust combiners have also been considered for other cryptographic primitives, including public-key encryption [HKN+05, DK05], oblivious transfer [HKN+05, Som06, MPW07, CDFR17, FR24], commitment schemes [Som06], private information retrieval [MP06], key encapsulation mechanisms [GHP18], authenticated encryption with associated data [PR20], obfuscators [AJS17, FHNS16], and functional encryption [AJN+16, ABJ+19, JMS20].

NIZK combiners. In prior literature, NIZK was captured in the multi-string setting [GO07], *i.e.*, NIZKs that have access to multiple CRSs retaining their security guarantees as long as a subset of the CRSs is

[28] Robust combiners for one-way functions are well-known [Her05], for instance the classical concatenation combiner with input splitting would work.

[29] Naor's construction requires a pseudorandom generator, which can be constructed from one-way functions [HILL99].

secure. This technique can be regarded as a restricted class of combiners in which all candidates run the same (uncompromised) prover and verifier algorithms.

More general combiners were also discussed in the context of NIZK amplification [GJS19, BKP+24, BG24], and in a concurrent and independent work [AK25]. However, none of these works explicitly focus on formally defining combiners. Moreover, the constructions therein contained assume some external computational assumptions (one-way functions or polynomially-secure PKEs), making them incompatible with our notion of robustness. In what follows, we discuss how these results compare to ours.

Amplifiers. Intuitively, a NIZK amplifier takes as input an $(\varepsilon_s, \varepsilon_z)$-weak NIZK (*i.e.*, a NIZK for which the soundness and zero-knowledge errors are, respectively, $\varepsilon_s, \varepsilon_z$ for $\varepsilon_s + \varepsilon_z < 1$) and turns it into a NIZK (for the same language) with negligible soundness and zero-knowledge errors. NIZK amplifiers imply NIZK combiners: by selecting at random one of the NIZK proof systems given to the combiner, we get a weak NIZK that we can then amplify. Following this approach, one obtains [BG24]:

- A combiner for NIZK arguments assuming polynomially-secure public-key encryption.
- A combiner for NIZK proofs assuming polynomially-secure one-way functions.

In contrast, our combiners are unconditional in the sense that they do not require any additional assumptions besides the fact that t out of n of the given NIZK proofs systems are secure. Another limitation is that the approach based on amplifiers currently does not seem to preserve knowledge soundness of the input protocols, while all of our combiners do. We also remark that, by the above results for amplifiers, it is possible to design a combiner with $t > \lfloor n/2 \rfloor$ that works when the input candidates are *proof* systems (with unconditional soundness) *without* considering knowledge soundness. On input n NIZK candidates, the combiner constructs a OWF using the technique described in Section 8.1.1 (for our combiner from MPC-in-the-head), picks a random NIZK input candidate and applies the amplifier of [BG24] (we can use this amplifier given that it only requires the existence of OWFs and a weak NIZK as input).

Concurrent and independent work. In a concurrent and independent work [AK25], assuming one-way functions, the authors show how to obtain a t-out-of-n NIZK combiner for all of NP, with optimal parameters $t > \lfloor n/2 \rfloor$ and $n, t \in \mathsf{poly}(\lambda)$. This combiner, compared to our first two constructions, does not preserve the succinctness of the input schemes, and moreover relies on one-way functions (however, this assumption can be avoided using our approach sketched above, at the cost of making the combiner non-black-box). The authors also provide a combiner that preserves the succinctness of the input schemes, assuming collision-resistant hash functions. In comparison, the first two constructions we propose do not rely on any additional assumptions, but instead support a restricted class of languages or work under the assumption that n, t are constant.

8.2 Defining NIZK combiners

Our combiner definitions follow the spirit of the notion for hash-function combiners proposed in [Pie08]. At a high level, [Pie08] defines black-box combiners for collision resistant hash functions as a tuple of algorithms $(\mathsf{C}, \mathcal{A})$, where C is the algorithm combining n *candidates* (the hash function protocols in the case of [Pie08]). Intuitively, a t-out-of-n combiner requires that if an adversary breaks the security of the combiner, then it is possible to define reductions that break the security of at least $n - t + 1$ candidates. This captures the fact that the security of the combiner relies only on the security of at least t input candidates, and no additional cryptographic assumption is used. This, in turn, prevents the combiner from just ignoring the input candidate and creating from scratch a hash function.

In this work, we follow a similar blueprint and extend it to the case of non-interactive zero-knowledge. Intuitively, a robust NIZK combiner takes n input candidates, where t of these schemes are sound and t are zero-knowledge (the subset of secure schemes for each property may differ), to itself achieve soundness and zero-knowledge. In proving security for each of these properties, the combiner could very well use *all* the security properties provided by the candidates. More concretely, a crafty combiner could use the soundness

of the candidates as part of its proof for zero-knowledge (we cover this case in proving zero-knowledge of our construction in Section 8.6).

The definitions we put forward are general enough to capture whether the combiner has *black-box* or *non-black-box* access to the underlying primitives, and whether it considers black-box or non-black-box zero-knowledge. Moreover, whenever a combiner is secure according to our definitions, it is also secure according to the *standard* notions of zero-knowledge and soundness. Throughout this section, we provide formal definitions and the formal arguments of the above claims.

We conclude the section by also introducing a variation of the above definitions that allows us to capture combiners that consider knowledge soundness in place of soundness.

8.2.1 Formalizing correctness

From a correctness standpoint, our goal consists of combining non-interactive complete protocols, as per Definition 2.16. Roughly, whenever one defines a combiner C that combines $\mathcal{F}_{\mathsf{NI}}$ candidates, C itself has to be a $\mathcal{F}_{\mathsf{NI}}$.

Definition 8.1 (Correctness of an n-candidate $\mathcal{F}_{\mathsf{NI}}$-combiner)**.** *Let C be a PPT algorithm. C is a correct n-candidate $\mathcal{F}_{\mathsf{NI}}$-combiner if the following holds:*

$$\forall PPT\ f_1, \ldots, f_n \in \mathcal{F}_{\mathsf{NI}}, \mathsf{C}^{f_1,\ldots,f_n} \in \mathcal{F}_{\mathsf{NI}}.$$

Note that, by our definition of $\mathcal{F}_{\mathsf{NI}}$, it is acceptable to design combiners for some language $\mathcal{L}$ while using candidates for different languages $\mathcal{L}_1, \ldots, \mathcal{L}_n$. This approach makes our definition more general and strengthens our impossibility results.

8.2.2 Security breaks for soundness and ZK

Following the introduction of this section, our goal consisting in designing *robust* NIZK combiners, meaning that security should *exclusively* derive from the security of the candidates. Intuitively, in order to state that a security property (*i.e.*, soundness or ZK) of the combiner is satisfied, it suffices to show that any adversary breaking the soundness of the combiner *must* have broken soundness or ZK for at least $n - t + 1$ of the candidates. Before moving to the formal definitions of security for our combiners, we clearly formalize what it means to "break" our security properties of interest (*i.e.*, soundness, ZK), and how these breaks relate to standard definitions of such properties.

Breaking soundness. In order to capture soundness, we find it convenient to define the relation $\mathcal{R}_{\mathsf{Sound}}$. This relation contains all the tuples *(protocol, adversary)* such that the adversary outputs an accepting proof for a false statement. More formally:

Definition 8.2 ($\mathcal{R}_{\mathsf{Sound}}$)**.** *Let $\Pi \in \mathcal{F}_{\mathsf{NI}}$, and let $\mathcal{A}$ (i.e., the adversary) be an algorithm. We say that $(\Pi, \mathcal{A}) \in \mathcal{R}_{\mathsf{Sound}}$ if the following holds:*

$$\Pr\left[\mathcal{V}(pp, x, \pi) = 1 \wedge x \notin \mathcal{L} \mid pp \leftarrow\!\$\ \mathsf{setup}(1^{|x|}, 1^{\lambda}); (x, \pi) \leftarrow\!\$\ \mathcal{A}(pp)\right] \geq \mathsf{nonnegl}(\lambda).$$

Notice that $\mathcal{R}_{\mathsf{Sound}}$ is, by itself, sufficient to capture a security break for soundness in both its computational and statistical flavours. For example, by the standard definition of computational soundness (reported in Definition 2.17), a protocol Π is secure if there exists no efficient adversary $\mathcal{A}$ that can craft an accepting proof for a false statement with better non-negligible probability (*i.e.*, $\forall PPT\mathcal{A}, (\Pi, \mathcal{A}) \notin \mathcal{R}_{\mathsf{Sound}}$). Hence, as long as $\exists PPT\mathcal{A}$ *s.t.* $(\Pi, \mathcal{A}) \in \mathcal{R}_{\mathsf{Sound}}$, the computational soundness of Π is broken. By lifting the PPT requirement for $\mathcal{A}$, one obtains statistical soundness.

Breaking zero-knowledge. We now move to zero-knowledge. First, we define the relation $\mathcal{R}_{\mathsf{ZK}}$. A tuple *(protocol, (simulator, adversary))* belongs to $\mathcal{R}_{\mathsf{ZK}}$ if the adversary distinguishes between proofs generated using the honest prover of the protocol and proofs generated from the simulator. More formally:

Definition 8.3 ($\mathcal{R}_{\mathsf{ZK}}$)**.** *Let $\Pi \in \mathcal{F}_{\mathsf{NI}}$, let $\mathcal{A}$ be an algorithm, and let $\mathcal{S} = (\mathcal{S}_0, \mathcal{S}_1)$ be a PPT algorithm. Let* REAL *and* IDEAL *be two experiments defined as follows:*

- $\texttt{REAL} = \{(pp, x, w, \pi) \mid pp \leftarrow\!\$\ \mathsf{setup}(1^{|x|}, 1^\lambda);\ (x, w) \leftarrow\!\$\ \mathcal{A}(pp);\ \pi \leftarrow\!\$\ \mathcal{P}(pp, x, w)\}_{\lambda \in \mathbb{N}}$
- $\texttt{IDEAL}_{\mathcal{S}} = \{(pp, x, w, \pi) \mid (pp, \tau) \leftarrow\!\$\ \mathcal{S}^0(1^{|x|}, 1^\lambda);\ (x, w) \leftarrow\!\$\ \mathcal{A}(pp);\ \pi \leftarrow\!\$\ \mathcal{S}^1(pp, \tau, x)\}_{\lambda \in \mathbb{N}}$.

We say that $(\Pi, (\mathcal{A}, \mathcal{S})) \in \mathcal{R}_{\mathsf{ZK}}$ if the following holds:

$$\Pr\left[b = b' \;\middle|\; \begin{array}{l} b \leftarrow\!\$\ \{0,1\} \\ b' \leftarrow \mathcal{A}(pp, x, w, \pi) \end{array}; (pp, x, w, \pi) \leftarrow\!\$ \begin{cases} \texttt{REAL} & \text{if } b = 0 \\ \texttt{IDEAL}_{\mathcal{S}} & \text{if } b = 1 \end{cases}\right] \geq \frac{1}{2} + \mathsf{nonnegl}(\lambda).$$

Differently from soundness, $\mathcal{R}_{\mathsf{ZK}}$ does not capture a security break for zero-knowledge by itself, as it only refers to a specific adversary breaking a specific simulator. In order to obtain a break for either black-box or non-black-box zero-knowledge, we need to introduce quantifiers appropriately. We discuss black-box zero-knowledge in what follows, and defer non-black-box zero-knowledge to Appendix C.3.

Breaking black-box zero-knowledge. The standard definition of computational black-box zero-knowledge (reported in Definition 2.18) states that a protocol Π is secure if there exists an efficient (universal) simulator $\mathcal{S}$ that is secure against every possible efficient adversary $\mathcal{A}$ (*i.e.*, $\exists \mathcal{S}$ *s.t.* $\forall PPT \mathcal{A}, (\Pi, (\mathcal{A}, \mathcal{S})) \notin \mathcal{R}_{\mathsf{ZK}}$). Breaking black-box zero-knowledge therefore means that such a simulator cannot exist (which, in turn, means that *every* possible simulator can be attacked). This is formalized as follows:

$$\forall \mathcal{S} \exists PPT \mathcal{A} \text{ s.t. } (\Pi, (\mathcal{A}, \mathcal{S})) \in \mathcal{R}_{\mathsf{ZK}}.$$

Similarly to the case of soundness, it is possible to obtain statistical zero-knowledge by considering unbounded adversaries.

8.2.3 Combiners for soundness and ZK

In what follows, we provide a definition of combiners for soundness and zero-knowledge, with a label $\alpha\beta \in \{\mathsf{B}, \mathsf{N}\}^2$ that specifies whether the combiner has black-box access ($\alpha = \mathsf{B}$) or non-black-box access ($\alpha = \mathsf{N}$) to its candidates, and whether the combiner relies on black-box zero-knowledge ($\beta = \mathsf{B}$) or non-black-box zero-knowledge ($\beta = \mathsf{N}$) of the candidates. For instance, a NB (ZK, Sound)-combiner has non-black-box access to its candidates, and considers candidates satisfying black-box zero-knowledge (*i.e.*, for the secure candidates there exists a universal simulator). For convenience, we discuss soundness and zero-knowledge separately.

Soundness of (ZK, Sound)-combiners. Let C be a *correct* combiner that works having as input n candidates $f_1, \ldots, f_n \in \mathcal{F}_{\mathsf{NI}}$. Informally, we say that a (ZK, Sound)-combiner C is sound if the following holds: if there exists $\mathcal{A}$ running in time τ such that $(\mathsf{C}, \mathcal{A}) \in \mathcal{R}_{\mathsf{Sound}}$ (*i.e.*, C is not sound), then there must exist $n - t + 1$ adversaries (reductions) running in time $\mathsf{poly}(\tau)$ to either soundness or ZK of distinct candidates. More specifically, if the combiner is not sound, then one of the below must hold:

1. There exists a sequence of adversaries $\mathsf{Red}_{i_1}, \ldots, \mathsf{Red}_{i_{n-t+1}}$ such that $(f_{i_j}, \mathsf{Red}_{i_j}) \in \mathcal{R}_{\mathsf{Sound}}$ for all $j \in [n-t+1]$, with $\{i_1, \ldots, i_{n-t+1}\} \subseteq [n]$ (*i.e.*, there are at least $n-t+1$ candidates for which soundness is broken, as per Section 8.2.2).
2. For all possible choices of simulators $\mathcal{S}_{i_1}, \ldots, \mathcal{S}_{i_{n-t+1}}$, $\exists \mathsf{Red}_{i_1}, \ldots, \mathsf{Red}_{i_{n-t+1}}$ such that $(f_{i_j}, (\mathcal{S}_{i_j}, \mathsf{Red}_{i_j})) \in \mathcal{R}_{\mathsf{ZK}}$ for all $j \in [n-t+1]$ (*i.e.*, there are at least $n-t+1$ candidates for which black-box zero-knowledge is broken, as per Section 8.2.2).

Below, we provide a formal definition:

Definition 8.4 (Soundness of t-out-of-n $\mathsf{type} \in \{\mathsf{B},\mathsf{N}\}^2$ (ZK, Sound)-combiners)**.** *Let* C *be a correct* n*-candidate* $\mathcal{F}_{\mathsf{NI}}$*-combiner, and let* $\mathcal{A}$ *be an algorithm running in time* τ*. We say that a* t*-out-of-*n $\mathsf{type} \in \{\mathsf{B},\mathsf{N}\}^2$ *(ZK, Sound)-combiner* C *is sound if, after ordering the quantifiers according to* type *in Tables 8.2,8.3, the following holds:*

$$(\mathsf{C}^{f_1,\ldots,f_n}, \mathcal{A}^{f_1,\ldots,f_n}) \in \mathcal{R}_{\mathsf{Sound}} \implies \textit{any of the following:}$$

- $\mathbf{n-t+1}$ *reductions running in time* $\mathsf{poly}(\tau)$ *to the soundness of distinct candidates, i.e.:*

$$\exists I = \{i_1,\ldots,i_{n-t+1}\} \subseteq [n] \ s.t., \forall i_j \in I, \exists \mathsf{Red}_{i_j} \ s.t. \ (f_{i_j}, \mathsf{Red}_{i_j}) \in \mathcal{R}_{\mathsf{Sound}}.$$

- $\mathbf{n-t+1}$ *reductions running in time* $\mathsf{poly}(\tau)$ *to the black-box (resp. non-black-box) zero-knowledge of distinct candidates if* $\mathsf{type} = *\mathsf{B}$ *(resp.* $\mathsf{type} = *\mathsf{N}$*), i.e.:*

$$\exists I = \{i_1,\ldots,i_{n-t+1}\} \subseteq [n] \ s.t. \ \textit{the following holds:}$$
$$\forall i_j \in I, \mathsf{candZKQuant} \ s.t. \ (f_{i_j}, (\mathsf{Red}_{i_j}, \mathcal{S}_{i_j})) \in \mathcal{R}_{\mathsf{ZK}}.$$

type	*Structural quantifiers*	
B∗	$\exists \mathsf{C}$	$\forall f_1,\ldots,f_n \in \mathcal{F}$
N∗	$\forall f_1,\ldots,f_n \in \mathcal{F}$	$\exists \mathsf{C}$

Table 8.2: Classification of soundness for (ZK, Sound)-combiners.

type	candZKQuant
∗B	$\forall \mathcal{S}_{i_j} \ \exists \mathsf{Red}_{i_j}$
∗N	$\exists \mathsf{Red}_{i_j} \forall \mathcal{S}_{i_j}$

Table 8.3: Quantifiers for ZK of the candidates of (ZK, Sound)-Combiners. $\mathcal{S}_{i_j}$ is a PPT algorithm. Red_{i_j} runs in time $\mathsf{poly}(\tau)$.

Connections with the *standard* notion of soundness. The above definition implies that, if the combiner runs with sufficiently-many candidates that are secure (*i.e.*, at least t are sound and t are zero-knowledge), then it must be that C satisfies the standard notion of soundness. This holds because, by the above definition, the existence of an adversary $\mathcal{A}$ running in time τ that breaks the soundness of the combiner C (*i.e.*, $\exists \mathcal{A}$ *s.t.* $(\mathsf{C},\mathcal{A}) \in \mathcal{R}_{\mathsf{Sound}}$) implies the existence of $n-t+1$ reductions running in time $\mathsf{poly}(\tau)$ to one of the security properties of the candidates. However, if t of the candidates are secure (according to each security property), it is impossible to craft successful reductions for these schemes, which means that at most $n-t$ successful reductions can exist to each individual security property. Hence, no such $\mathcal{A}$ can exist (*i.e.*, $\forall \mathcal{A}, (\mathsf{C},\mathcal{A}) \notin \mathcal{R}_{\mathsf{Sound}}$). As discussed in Section 8.2.2, this means that the combiner is secure according to the standard definition of soundness.

Zero-knowledge of (ZK, Sound)-combiners. Following a similar approach, we characterize zero-knowledge. For example, in the case of $*\mathsf{B}$ (ZK, sound)-combiners (*i.e.*, combiners considering *black-box* zero-knowledge), we say that C running on input n candidates $f_1,\ldots,f_n$ is (black-box) zero-knowledge if the following happens: if for all simulators $\mathcal{S}$ there exists an adversary $\mathcal{A}$ running in time τ such that $(\mathsf{C},(\mathcal{S},\mathcal{A}))$ belongs to $\mathcal{R}_{\mathsf{ZK}}$ (*i.e.*, C is not black-box zero-knowledge), then it is possible to design at least $n-t+1$ reductions running in time $\mathsf{poly}(\tau)$ to one of the security properties of distinct candidates. In other words, if the combiner is not zero-knowledge, then it must be that at least $n-t+1$ input candidates are not sound or zero-knowledge. The formal definition follows.

Definition 8.5 (Non-Interactive Zero-Knowledge of t-out-of-n $\mathsf{type} \in \{\mathsf{B},\mathsf{N}\}^2$ (ZK, Sound)-combiners)**.** *Let* C *be a correct* n*-candidate* $\mathcal{F}_{\mathsf{NI}}$*-combiner. We say a* t*-out-of-*n $\mathsf{type} \in \{\mathsf{B},\mathsf{N}\}^2$ *(ZK, Sound)-combiner* C *is zero-knowledge if, after ordering the quantifiers according to* type *in Tables 8.3, 8.4, the following holds (*$\mathsf{combZKQuant}$ *is also defined in Table 8.4):*

$$\mathsf{combZKQuant}\ s.t.\ (\mathsf{C}^{f_1,\ldots,f_n},(\mathcal{A}^{f_1,\ldots,f_n},\mathcal{S})) \in \mathcal{R}_{\mathsf{ZK}} \implies \textit{any of the following:}$$

- $\mathbf{n-t+1}$ *reductions running in time* $\mathsf{poly}(\tau)$ *to soundness of the candidates, as in Definition 8.4.*
- $\mathbf{n-t+1}$ *reductions running in time* $\mathsf{poly}(\tau)$ *to black-box (resp. non-black-box) zero-knowledge of the candidates if* $\mathsf{type} = *\mathsf{B}$ *(resp.* $\mathsf{type} = *\mathsf{N}$*), as in Definition 8.4.*

type	*Structural quantifiers*	combZKQuant
BB	$\exists\mathsf{C}\forall f_1,\ldots,f_n \in \mathcal{F}$	$\forall\mathcal{S}\exists\mathcal{A}$
BN	$\exists\mathsf{C}\forall f_1,\ldots,f_n \in \mathcal{F}$	$\exists\mathcal{A}\forall\mathcal{S}$
NB	$\forall f_1,\ldots,f_n \in \mathcal{F}\exists\mathsf{C}$	$\forall\mathcal{S}\exists\mathcal{A}$
NN	$\forall f_1,\ldots,f_n \in \mathcal{F}\exists\mathsf{C}$	$\exists\mathcal{A}\forall\mathcal{S}$

Table 8.4: Classification of zero-knowledge for (ZK, Sound)-combiners.

Connections with the *standard* ZK notion. Let us focus on the black-box zero-knowledge case (a similar argument holds for the non-black-box case). The above definition implies that, if the combiner runs with sufficiently-many candidates that are secure, then it must be that C satisfies the standard notion of black-box zero-knowledge. This holds because, by the above definition, if the black-box zero-knowledge of the combiner C is broken (*i.e.*, $\forall\mathcal{S}\exists\mathcal{A}$ *s.t.* $(\mathsf{C},(\mathcal{A},\mathcal{S})) \in \mathcal{R}_{\mathsf{ZK}}$), it means that there exist $n-t+1$ reductions to one of the security properties of the candidates. However, if t of the candidates are secure (according to each security property), it is impossible to craft successful reductions for these schemes, which means that at most $n-t$ successful reductions can exist. Hence, it is impossible to break zero-knowledge for the combiner (*i.e.*, $\exists\mathcal{S}$ *s.t.* $\forall\mathcal{A},(\mathsf{C},(\mathcal{A},\mathcal{S})) \notin \mathcal{R}_{\mathsf{ZK}}$). As discussed in Section 8.2.2, this means that the combiner is secure according to the standard definition of black-box zero-knowledge, and in particular that it admits a universal ZK simulator. Therefore, our combiner can be used in any application in which a black-box zero-knowledge protocol is required.

8.2.4 Combiners for knowledge soundness and ZK

We observe that it is possible to adapt the definition provided in the previous subsection to a different set of security properties in a straight-forward manner. Indeed, throughout this composition we also consider $\mathsf{type} \in \{\mathsf{B},\mathsf{N}\}^3$ (ZK, AoK)-combiners, *i.e.*, combiners that achieve ZK and knowledge soundness by relying on the same properties for their candidates. Note that, in this case, type has a third literal, highlighting whether the combiner considers black-box or non-black-box knowledge soundness. The definition follows the same strategy we discussed for (ZK, Sound)-combiners: first, we formalize what it means to break knowledge soundness, and then provide security definitions for each property. In particular, breaking knowledge soundness or ZK of the combiner implies breaking either knowledge soundness or ZK of $n-t+1$ of the candidates. We defer the complete formal treatment to Appendix C.2.

8.3 Impossibility Results for $t \leq \lfloor \frac{n}{2} \rfloor$

In this subsection, we show that no t-out-of-n (ZK, Sound)-combiner exists for NP for $t \leq \lfloor \frac{n}{2} \rfloor$, according to our Definitions 8.4, 8.5, unless the polynomial hierarchy collapses.

Theorem 8.1. *Let* $\{\Pi_1,\ldots,\Pi_n\}$ *be a set of non-interactive complete candidates (i.e.,* $\forall i \in [n],\Pi_i \in \mathcal{F}_{\mathsf{NI}}$*). If there exists a* t*-out-of-*n type *(ZK, Sound)-combiner* $\mathsf{C}^{\Pi_1,\ldots,\Pi_n}$ *for the language* $\mathcal{L}$*, with* $\mathsf{type} \in \{\mathsf{B},\mathsf{N}\}^2$ *(according to Definitions 8.4, 8.5) (i.e., the combiner returns a protocol that is both sound and zero-knowledge), then either* $t > \lfloor \frac{n}{2} \rfloor$ *or* $\mathcal{L} \subseteq \mathsf{coAM} \cap \mathsf{AM}$*.*

Proof. In what follows, we exhibit a proof for $(n, t) = (2, 1)$, which can be trivially extended to the general case of (n, t) with $t \leq \lfloor \frac{n}{2} \rfloor$.

Let C be a 1-out-of-2 (ZK, Sound)-combiner for some NP-language $\mathcal{L}$. This combiner works assuming that the input protocols are for two languages $\mathcal{L}_{\mathsf{zk}}$ and $\mathcal{L}_{\mathsf{Sound}}$. Let us consider the following two NIZK candidates: $\Pi_{\mathsf{zk}} = (\mathsf{setup}_{\mathsf{zk}}, \mathcal{P}_{\mathsf{zk}}, \mathcal{V}_{\mathsf{zk}})$ and $\Pi_{\mathsf{Sound}} = (\mathsf{setup}_{\mathsf{Sound}}, \mathcal{P}_{\mathsf{Sound}}, \mathcal{V}_{\mathsf{Sound}})$, respectively for the languages $\mathcal{L}_{\mathsf{zk}}$ and $\mathcal{L}_{\mathsf{Sound}}$. The setup algorithms of both Π_{zk} and Π_{Sound} return nothing. The prover $\mathcal{P}_{\mathsf{zk}}$, on input a statement and a witness, returns 0. The verifier accepts a proof π_{zk} for a statement x only if $\pi_{\mathsf{zk}} = 0$. The prover $\mathcal{P}_{\mathsf{Sound}}$, on input a statement x and a witness w, returns the witness w. The verifier accepts a proof π only if w is a valid witness for x. In summary, we have that Π_{zk} is unconditional zero-knowledge (but not sound), and Π_{Sound} is unconditional sound (but not zero-knowledge).

By assumption, C is a valid combiner that on input Π_{zk} and Π_{Sound} returns a new scheme Π. By the definition of secure combiner, we have that if there exists an adversary, running in time τ, that attacks successfully the zero-knowledge (the soundness) of Π, then we can design $n - t + 1 = 2$ reductions to either the zero-knowledge or the soundness properties of both schemes, each (the reductions) running in time $\mathsf{poly}(\tau)$. We are going to prove that it is impossible to construct these two reductions. Proving this implies that the scheme produced by C is unconditional sound and unconditional zero-knowledge, which implies that $\mathcal{L} \subseteq \mathsf{coAM} \cap \mathsf{AM}$ due to [Ps05].

Assume that an unbounded adversary breaks either the zero-knowledge or the soundness of C. By the definition of t-out-of-n (ZK, Sound)-combiners, this must imply the existence of two (unbounded) reductions to zero-knowledge or two (unbounded) reductions to soundness of the input schemes. We first consider zero-knowledge. Having two reductions to zero-knowledge of the candidates implies having a reduction against zero-knowledge of Π_{Sound}, and a reduction against zero-knowledge of Π_{zk}. Let us focus on the reduction to zero-knowledge for Π_{zk}. A successful reduction $\mathsf{Red}_{\mathsf{zk}}$ needs to tell apart if a proof of Π_{zk} is simulated or not. However, no such $\mathsf{Red}_{\mathsf{zk}}$ can exist, as the zero-knowledge property of Π_{zk} is unconditional. Hence, it is impossible to have two (even unbounded) reductions to the zero-knowledge of the candidates. Let us now turn our attention to soundness. Similarly, in this case, we have a reduction $\mathsf{Red}_{\mathsf{Sound}}$ to the soundness of Π_{Sound} and a reduction to the soundness of Π_{zk}. We observe that the soundness of Π_{Sound} is unconditional, hence, no such reduction $\mathsf{Red}_{\mathsf{Sound}}$ can exist.

Given that we can never obtain $n - t + 1 = 2$ (even unbounded) reductions to either soundness or zero-knowledge of the candidates, it follows that the scheme obtained by running C on inputs Π_{zk} and Π_{Sound} is both unconditional sound and unconditional zero-knowledge (as no even unbounded adversary can attack either property of the combiner). But due to [Ps05], such a scheme exists only for languages in $\mathsf{coAM} \cap \mathsf{AM}$. This concludes the proof. □

Lifting the multi-property requirement. The previous impossibility result is sufficient to rule out robust combiners that satisfy our definitions. However, our proof crucially relies on the fact that one scheme is unconditionally sound (but not ZK), while the other is unconditionally ZK (but not sound). It is a natural question to investigate whether robust t-out-of-n NIZK combiners could exist if we restrict our attention to schemes that are simultaneously sound *and* zero-knowledge, with $t \leq \lfloor \frac{n}{2} \rfloor$.

In what follows, we show that this is indeed impossible, assuming the existence of any statistically sound and computationally zero-knowledge NIZK scheme, and the existence of any computationally sound and statistically zero-knowledge NIZK scheme. The strong evidence towards the existence of such protocols [BFM88, FLS90, FLS99, CHK03, GOS06, GOS12, SW14, LPWW20] helps us rule out the existence of these stronger combiners.

Theorem 8.2. *Let $T = \{\Pi_1, \ldots, \Pi_n\}$ with $\Pi_i \in \mathcal{F}_{\mathsf{NI}}$ for each $i \in [n]$, and let $K \subseteq T$ with $|K| = t$ be such that each $\Pi \in K$ is both sound and zero-knowledge. Assume the existence of statistically sound and computationally zero-knowledge NIZK schemes, and computationally sound and statistically zero-knowledge NIZK schemes. If there exists a t-out-of-n type (ZK, Sound)-combiner $\mathsf{C}^{\Pi_1,\ldots,\Pi_n}$ for the language $\mathcal{L}$ (according to Definitions 8.4, 8.5), with $\mathsf{type} \in \{\mathsf{B}, \mathsf{N}\}^2$ (i.e., the combiner returns a protocol that is both sound and zero-knowledge), then either $t > \lfloor \frac{n}{2} \rfloor$ or $\mathcal{L} \subseteq \mathsf{coAM} \cap \mathsf{AM}$.*

Proof. Let C be a 1-out-of-2 (ZK, Sound)-combiner for some NP-language $\mathcal{L}$. This combiner works assuming that the input protocols are for two languages $\mathcal{L}_{\mathsf{zk}}$ and $\mathcal{L}_{\mathsf{Sound}}$. Let us consider a computationally sound and statistically zero-knowledge scheme $\Pi_{\mathsf{zk}} = (\mathsf{setup}_{\mathsf{zk}}, \mathcal{P}_{\mathsf{zk}}, \mathcal{V}_{\mathsf{zk}})$, and a statistically sound and computationally zero-knowledge scheme $\Pi_{\mathsf{Sound}} = (\mathsf{setup}_{\mathsf{Sound}}, \mathcal{P}_{\mathsf{Sound}}, \mathcal{V}_{\mathsf{Sound}})$, respectively for the languages $\mathcal{L}_{\mathsf{zk}}$ and $\mathcal{L}_{\mathsf{Sound}}$. These schemes exist by assumption.

Similarly to Theorem 8.1, we argue that, by running C on input Π_{zk} and Π_{Sound}, we obtain a new scheme Π for which we cannot design $n - t + 1 = 2$ (even unbounded) reductions to either the zero-knowledge or the soundness of the candidates. By our definition of (ZK, Sound)-combiners, this implies that there exists no (even unbounded) adversary who successfully attacks the combiner. Hence, the scheme produced by C is both statistically sound and statistically zero-knowledge, implying that $\mathcal{L} \subseteq \mathsf{coAM} \cap \mathsf{AM}$ due to [Ps05].

Assume that an unbounded adversary breaks either the zero-knowledge or the soundness of C. Having two reductions to zero-knowledge of the candidates implies having a reduction against zero-knowledge of Π_{Sound}, and a reduction $\mathsf{Red}_{\mathsf{zk}}$ against zero-knowledge of Π_{zk}. However, no such $\mathsf{Red}_{\mathsf{zk}}$ can exist, as the zero-knowledge property of Π_{zk} is statistical (by assumption). Hence, it is impossible to have two (even unbounded) reductions to the zero-knowledge of the candidates. Similarly, having two reductions to soundness of the candidates implies having a reduction $\mathsf{Red}_{\mathsf{Sound}}$ to the soundness of Π_{Sound} and a reduction to the soundness of Π_{zk}. We observe that the soundness of Π_{Sound} is statistical (by assumption); hence, no such reduction $\mathsf{Red}_{\mathsf{Sound}}$ can exist.

□

Computationally-secure input candidates. The aforementioned impossibility result rules out a restricted class of robust t-out-of-n NIZK combiners, with $t \leq \lfloor n/2 \rfloor$, where the input candidates are simultaneously sound and zero-knowledge. However, our result crucially relies on a counterexample depending on candidates with statistical properties. Hence, robust t-out-of-n NIZK combiners could still exist if we restrict the input candidates to be *exclusively* computationally sound and computationally zero-knowledge, with $t \leq \lfloor n/2 \rfloor$. With the next impossibility result, we further rule out this restricted class of combiners under standard cryptographic assumptions, albeit only for black-box combiners.

Theorem 8.3. *Let $T = \{\Pi_1, \ldots, \Pi_n\}$ with $\Pi_i \in \mathcal{F}_{\mathsf{NI}}$ for each $i \in [n]$, and let $K \subseteq T$ with $|K| = t$ be such that each $\Pi \in K$ is both computationally sound and computationally zero-knowledge. Assume the existence of dual-mode commitments, one-way functions, statistically sound and computationally zero-knowledge NIZK schemes, and computationally sound and statistically zero-knowledge NIZK schemes. If there exists a t-out-of-n type (ZK, Sound)-combiner $\mathsf{C}^{\Pi_1,\ldots,\Pi_n}$ for the language $\mathcal{L}$ (according to Definitions 8.4, 8.5), with $\mathsf{type} = \mathsf{B}*$ (i.e., the combiner returns a protocol that is both sound and zero-knowledge while having black-box access to its candidates), then either $t > \lfloor \frac{n}{2} \rfloor$ or $\mathcal{L} \subseteq \mathsf{coAM} \cap \mathsf{AM}$.*

Proof. We first list the ingredients that will be required for the proof:

- $\Pi_{\mathsf{czkp}} = (\mathsf{setup}_{\mathsf{czkp}}, \mathcal{P}_{\mathsf{czkp}}, \mathcal{V}_{\mathsf{czkp}})$ is a computational zero-knowledge proof system.
- $\Pi_{\mathsf{szka}} = (\mathsf{setup}_{\mathsf{szka}}, \mathcal{P}_{\mathsf{szka}}, \mathcal{V}_{\mathsf{szka}})$ is a statistical zero-knowledge argument system.
- $\mathsf{dmcom}^{\mathsf{mode}} = (\mathsf{setup}^{\mathsf{mode}}_{\mathsf{dmcom}}, \mathsf{Com}^{\mathsf{mode}}_{\mathsf{dmcom}}, \mathsf{Open}^{\mathsf{mode}}_{\mathsf{dmcom}})$ is a dual-mode commitment scheme. If $\mathsf{mode} = \mathsf{hiding}$, the commitment is statistically hiding and computationally binding. If $\mathsf{mode} = \mathsf{binding}$, the commitment scheme is computationally hiding and statistically binding. The two modes are computationally indistinguishable. We remark that this primitive admits instantiations from standard cryptographic assumptions, including DDH [Lin15].
- $\mathtt{PRG}$ is a pseudo-random generator.

Next, we construct two schemes $\Pi_{\mathsf{Sound}} = (\mathsf{setup}_{\mathsf{Sound}}, \mathcal{P}_{\mathsf{Sound}}, \mathcal{V}_{\mathsf{Sound}})$ and $\Pi_{\mathsf{zk}} = (\mathsf{setup}_{\mathsf{zk}}, \mathcal{P}_{\mathsf{zk}}, \mathcal{V}_{\mathsf{zk}})$, working respectively for languages $\mathcal{L}_{\mathsf{Sound}}$ and $\mathcal{L}_{\mathsf{zk}}$, by using the ingredients above.

- $\mathsf{setup}_{\mathsf{Sound}}$ outputs the concatenation of $\mathsf{setup}_{\mathsf{Sound}}$ and a uniformly random string. $\mathcal{P}_{\mathsf{Sound}}$, $\mathcal{V}_{\mathsf{Sound}}$ are defined as $\mathcal{P}_{\mathsf{czkp}}, \mathcal{V}_{\mathsf{czkp}}$ with respect to the following language: $\mathcal{L}_{\mathsf{czkp}} = \{(\mathsf{crs}, x) \mid \exists w \ s.t. \ (x, w) \in \mathscr{R}_{\mathcal{L}_{\mathsf{Sound}}} \vee \exists r \ s.t. \ \mathsf{crs} = \mathtt{PRG}(r)\}$.

- $\mathsf{setup}_{\mathsf{zk}}$ outputs the concatenation of $\mathsf{setup}_{\mathsf{szka}}$ and $\mathsf{setup}^{\mathsf{hiding}}_{\mathsf{dmcom}}$. $\mathcal{P}_{\mathsf{zk}}$ computes a commitment of the witness com with opening o and a proof π by running $\mathcal{P}_{\mathsf{szka}}$ for the following language: $\mathcal{L}_{\mathsf{szka}} = \{(com, x) \mid \exists(w, o)\ s.t.\ (x, w) \in \mathscr{R}_{\mathcal{L}_{\mathsf{zk}}} \wedge w = \mathsf{Open}^{\mathsf{mode}}_{\mathsf{dmcom}}(com, o)\}$. It then outputs (com, π). $\mathcal{V}_{\mathsf{zk}}$ runs $\mathcal{V}_{\mathsf{szka}}$.

It is easy to see that Π_{Sound} is still a CZKP. Indeed, Π_{czkp} is statistically sound, and the CRS is generated as a uniformly random string. Moreover, Π_{zk} is still a SZKA: the output of the prover is (com, π), with com being a statistically hiding commitment and π being a statistically zero-knowledge argument.

Finally, we construct two CZKA schemes Π_1, Π_2 by "downgrading" the statistical properties of Π_{Sound}, Π_{zk} to computational. We do that as follows:

- Π_1 is the same as Π_{Sound}, except that the setup algorithm evaluates the common random string by sampling a random r and evaluating $\mathtt{PRG}(r)$.
- Π_2 is the same as Π_{zk}, except that the dual-mode commitment scheme is initialized in $\mathsf{binding}$ mode.

We are now ready for the proof. By contradiction, let us assume that C is a 1-out-of-2 (ZK, Sound)-combiner for some NP-language $\mathcal{L}$. This combiner works assuming that the input protocols are for two languages $\mathcal{L}_{\mathsf{zk}}$ and $\mathcal{L}_{\mathsf{Sound}}$.

We first argue that the combiner, upon input Π_1 and Π_2, outputs a CZKA. Since Π_1 and Π_2 are both secure CZKA, we can never have $n-t+1 = 2$ *efficient* reductions to either the zero-knowledge or the soundness of the candidates. Since C is a robust combiner, there exists no efficient adversary who successfully attacks the zero-knowledge or the soundness of the combiner. This, in turn, means that the scheme output by the combiner is a CZKA for the NP-language $\mathcal{L}$.

Next, we exhibit a hybrid chain by which we replace Π_1 and Π_2 with Π_{Sound} and Π_{zk}, respectively. Intuitively, we argue that the combiner still outputs a CZKA for $\mathcal{L}$ even if we switch computational properties of the candidates with statistical properties. Moreover, the CZKA output by the combiner upon input Π_1 and Π_2 is such that the joint distribution of its setup, prover, and verifier algorithms are computationally indistinguishable from the CZKA output by the combiner upon input Π_{Sound} and Π_{zk}. Essentially, this ensures that the combiner cannot detect the introduction of candidates with statistical properties.

- $\mathcal{H}_0$: The combiner runs with input candidates Π_1 and Π_2.
- $\mathcal{H}_1$: We replace Π_1 with Π_{Sound}. We observe that the combiner still outputs a CZKA, given that it is impossible to find $n - t + 1 = 2$ reductions to computational zero-knowledge or soundness of the candidates (as Π_{Sound} is a secure CZKP and Π_2 is a secure CZKA). Moreover, the distribution of the algorithms of the CZKA output by the combiner is computationally indistinguishable from $\mathcal{H}_0$. In particular, if we have a distinguisher telling apart $\mathcal{H}_0$ from $\mathcal{H}_1$, we can design a reduction to the security of the PRG. Since the PRG is secure by assumption, there can never exist such a distinguisher.
- $\mathcal{H}_2$: We replace Π_2 with Π_{zk}. With a similar argument as in the previous hybrid, the combiner still outputs a CZKA. Moreover, the distribution of the algorithms of the CZKA output by the combiner is computationally indistinguishable from $\mathcal{H}_1$, as otherwise it would be possible to design a reduction to the indistinguishability of modes of the dual-mode commitment scheme.

In $\mathcal{H}_2$, we have that C is a 1-out-of-2 combiner that outputs a CZKA for an NP-language $\mathcal{L}$ even when running with Π_{Sound} and Π_{zk} as inputs (which are, respectively, a CZKA and a SZKP). But by Theorem 8.2, such a combiner cannot exist for any NP-language (unless the polynomial hierarchy collapses). Hence, we have a contradiction. □

8.4 Combiner for Homomorphic Languages

In this subsection, we define a t-out-of-n BB (ZK, Sound)-combiner, with $t > \lfloor \frac{n}{2} \rfloor$, that may be used to prove the class of languages described in [Mau09] which are denoted as *homomorphic languages* [Cra96]. In particular, a homomorphic language $\mathcal{L}_{\mathsf{HL}}$ is defined with respect to the groups $(\mathbb{G}, \star), (\mathbb{H}, \otimes)$ and a homomorphism $f : \mathbb{G} \to \mathbb{H}$ (*i.e.*, $f(x) \otimes f(y) = f(x \star y)$), where f is a one-way function, and is represented by the set $\{x \mid \exists w \text{ s.t. } f(w) = x\}$. We conclude the section by showing that our combiner is also a t-out-of-n BBB (ZK, AoK)-combiner.

8.4.1 Preliminaries

Notationally, within this scheme we use hyperoperations as in [Goo47]. Let w be an element in group $\mathbb{G}$. We denote $w\{2\star\}v$ as the result of $\underbrace{w \star w \star \ldots \star w}_{v \text{ copies of } w}$, with $v \in \mathbb{Z}_{\mathsf{q}}$. We compactly represent $a_0 \star a_1 \star \ldots \star a_n$ with the symbol $\bigstar_{i=0}^{n} a_i$, and $a_0 \otimes a_1 \otimes \ldots \otimes a_n$ with the symbol $\bigotimes_{i=0}^{n} a_i$.
Finally, if f is a group homomorphism as defined above, the following facts hold:

$$f(w)\{2\otimes\}v = f(w\{2\star\}v) \tag{1}$$

$$\bigotimes_{j\in[n]} f(w_j) = f(\bigstar_{j\in[n]} w_j) \tag{2}$$

Fact 1 holds, as:

$$f(w)\{2\otimes\}v = \underbrace{f(w) \otimes f(w) \otimes \ldots \otimes f(w)}_{v \text{ copies of } f(w)} = f\underbrace{(w \star w \star \ldots \star w)}_{v \text{ copies of } w} = f(w\{2\star\}v).$$

Fact 2 holds, as:

$$\bigotimes_{j\in[n]} f(w_j) = f(w_1) \otimes \ldots \otimes f(w_n) = f(w_1 \star \ldots \star w_n) = f(\bigstar_{j\in[n]} w_j).$$

Shamir's secret sharing scheme. At its core, Shamir's secret sharing is a t-out-of-n secret sharing scheme that allows performing the standard sharing and reconstruction operations (as described in Definitions 2.25, 2.26) in a generic group $(\mathbb{G}, \star)$.

The sharing algorithm $\mathtt{share}_{\mathbb{G}}$ takes as input w, where w is an element of group $\mathbb{G}$, and samples $t-1$ random group elements $a_1, \ldots, a_{t-1}$ by using the generation algorithm $\mathtt{Gen}_{\mathbb{G}}(1^{\lambda})$ (this algorithm just samples uniformly at random elements from $\mathbb{G}$). A polynomial p of degree $t-1$ is defined by using coefficient a_i, and by setting $a_0 = w$. The shares are computed by evaluating $p(i)$, for $i \in [n]$:

$$\mathtt{share}_{\mathbb{G}}(w)\text{: } a_1, \ldots, a_{t-1} \leftarrow\!\$\ \mathtt{Gen}_{\mathbb{G}}(1^{\lambda}). \text{ Output } \{w_i \leftarrow w \star \bigstar_{j=1}^{t-1}(a_j\{2\star\}i^j)\}_{i\in[n]}.$$

The $\mathtt{reconstruct}_{\mathbb{G}}$ algorithm takes as input a subset $S = \{s_i\}_{i\in I}$ of t shares, where $I = i_1, \ldots, i_t \in \mathbb{Z}_{\mathsf{q}}$ is a set of evaluation points of the polynomial, and uses Lagrange interpolation to recompute $p(0)$ (*i.e.*, the secret):

$$\mathtt{reconstruct}_{\mathbb{G}}(\{s_i\}_{i\in I})\text{: Output } w \leftarrow \bigstar_{j\in I}(s_i\{2\star\}(\prod_{k\in I, k\neq j} \frac{k}{k-j})).$$

Since Shamir's secret sharing scheme relies on Lagrange interpolation, it is possible to reconstruct points in the polynomial that are different from the secret (*i.e.*, points evaluated in $l \neq 0$). To this extent, we denote the modified version of $\mathtt{reconstruct}_{\mathbb{G}}$ as $\overline{\mathtt{reconstruct}}_{\mathbb{G}}$. $\overline{\mathtt{reconstruct}}_{\mathbb{G}}$ takes as input a subset $S = \{s_i\}_{i\in I}$ of t shares and an index $l \notin I$ of the point to be reconstructed, and works as follows:

$$\overline{\mathtt{reconstruct}}_{\mathbb{G}}(\{s_i\}_{i\in I}, l)\text{: Output } w_l \leftarrow \bigstar_{j\in I}(s_i\{2\star\}(\prod_{k\in I, k\neq j} \frac{l-k}{j-k})).$$

We formalize the above with the following definitions.

Definition 8.6 (Shamir's t-out-of-n secret sharing scheme)**.** *Let* $\mathtt{SSS}$ *be a tuple of algorithms* $(\mathtt{share}_{\mathbb{G}}, \mathtt{reconstruct}_{\mathbb{G}}, \overline{\mathtt{reconstruct}}_{\mathbb{G}})$ *as defined above. We say* $\mathtt{SSS}$ *satisfies correctness if,* $\forall m$, $\forall S = \{i_1, \ldots, i_t\} \subseteq \{1, \ldots, n\}$ *of size* t*, it holds that:*

$$\Pr[\mathtt{reconstruct}_{\mathbb{G}}(s_{i_1}, \ldots, s_{i_t}) = m \mid (s_1, \ldots, s_n) \leftarrow\!\$\ \mathtt{share}_{\mathbb{G}}(m)] = 1.$$

Additionally, we require that for any set $S = \{i_1, \ldots, i_t\} \subseteq [n]$, the following holds: $\forall k \in [n] \setminus S$,

$$\Pr[\overline{\texttt{reconstruct}}_{\mathbb{G}}((s_{i_1}, \ldots, s_{i_t}), k) = s_k \mid (s_1, \ldots, s_n) \leftarrow\!\$\ \texttt{share}_{\mathbb{G}}(m)] = 1.$$

Definition 8.7 (Security of Shamir's t-out-of-n secret sharing scheme)**.** *Let* SSS *be a Shamir's t-out-of-n secret sharing scheme satisfying correctness. We say* SSS *is perfectly secure if, $\forall m, m'$, $\forall S \subseteq \{1, \ldots, n\}$, where $|S| < t$, the following holds:*

$$\{(s_i : i \in S) \mid (s_1, \ldots, s_n) \leftarrow \texttt{share}_{\mathbb{G}}(m)\} \equiv \{(s'_i : i \in S) \mid (s'_1, \ldots, s'_n) \leftarrow\!\$\ \texttt{share}_{\mathbb{G}}(m')\}.$$

Homomorphic interpolation. Next, we augment the aforementioned definitions with a special reconstruction algorithm that we denote with $\overline{\texttt{reconstruct}}_{\mathbb{H}}$, which allows performing the reconstruction *homomorphically*, exploiting the properties of the homomorphic language. More details follow.

If f is a group homomorphism as defined above, having access to a subset S of size t of functions of the shares (*i.e.*, $\{f(s_i)\}_{i \in I}$) allows the reconstruction of a function of the point in the polynomial (*i.e.*, $x_l = f(s_l)$ for $l \notin I$) as follows:

$$x_l = \bigotimes_{j \in I} (f(s_i)\{2\otimes\}(\prod_{k \in I, k \neq j} \frac{l-k}{j-k})).$$

We denote the reconstruction operation above as $\overline{\texttt{reconstruct}}_{\mathbb{H}}(\{x_i\}_{i \in I}, l)$. The correctness of this operation follows by applying Fact 1 and Fact 2:

$$\begin{aligned}
x_l &= \bigotimes_{j \in I} (f(s_i)\{2\otimes\}(\prod_{k \in I, k \neq j} \frac{l-k}{j-k})) \\
&= \bigotimes_{j \in I} (f(s_i\{2\star\}(\prod_{k \in I, k \neq j} \frac{l-k}{j-k}))) \\
&= f(\bigstar_{j \in I} (s_i\{2\star\}(\prod_{k \in I, k \neq j} \frac{l-k}{j-k}))) \\
&= f(\overline{\texttt{reconstruct}}_{\mathbb{G}}(\{s_i\}_{j \in I}, l) = f(w_l).
\end{aligned}$$

In the rest of this composition, whenever we talk about a homomorphic language $\mathcal{L}_{\mathsf{HL}}$ with respect to groups $(\mathbb{G}, \star)$, $(\mathbb{H}, \otimes)$ and a homomorphism $f : \mathbb{G} \to \mathbb{H}$, we implicitly require that the algorithms ($\texttt{share}_{\mathbb{G}}$, $\texttt{reconstruct}_{\mathbb{G}}$), as defined here, satisfy correctness and perfect security of Shamir's secret sharing scheme (respectively, Definition 8.6 and Definition 8.7). Looking ahead, the correctness property will be used within the proof of Theorem C.1, and the perfect security property will be used within the proof of Theorem C.3. In Appendix C.4 we give a few concrete examples of such homomorphic languages and associated sharing and reconstruction operations.

8.4.2 Combiner description

Let $\{\Pi_k = (\mathsf{setup}_k, \mathcal{P}_k, \mathcal{V}_k)\}_{k \in [n]}$ be non-interactive protocols for the homomorphic language $\mathcal{L}_{\mathsf{HL}}$ defined above. These protocols represent the input primitives of the combiner. Let x be the theorem the verifier wants to verify. At a high level, in our scheme the prover secret shares the witness w in $\mathbb{G}$ by running $\texttt{share}_{\mathbb{G}}$, and computes n *sub-statements* by evaluating f on each obtained share. Then the prover proves that each of these sub-statements belongs to the homomorphic language by using the input candidates (the k-th input candidate Π_k is used to prove the k-th sub-statement). The verifier picks the first t sub-statements and homomorphically reconstructs the polynomial in $\mathbb{H}$ by running $\overline{\texttt{reconstruct}}_{\mathbb{H}}$ (as defined in Section 8.4.1). The verifier then accepts if all the proofs are accepting, and the homomorphic reconstruction matches with the theorem x and with all of the sub-statements. We formally describe our combiner $\mathsf{C}_{\mathsf{HL}} = (\mathsf{C}_{\mathsf{setup}}, \mathsf{C}_{\mathsf{prove}}, \mathsf{C}_{\mathsf{vrfy}})$ in Figure 8.1:

Combiner C_{HL}

- $\mathsf{C}_{\mathsf{setup}}(1^l, 1^\lambda)$: Compute $pp_k \leftarrow\!\$\ \mathsf{setup}_k(1^l, 1^\lambda)$, for $k \in [n]$. Output $pp \leftarrow \{pp_k\}_{k\in[n]}$.
- $\mathsf{C}_{\mathsf{prove}}(pp, x, w)$:
  - Compute $(w_1, \ldots, w_n) \leftarrow\!\$\ \mathtt{share}_{\mathbb{G}}(w)$.
  - Compute $x_k \leftarrow f(w_k)$, for $k \in [n]$.
  - Compute $\pi_k \leftarrow \mathcal{P}_k(pp_k, x_k, w_k)$, for $k \in [n]$.
  - Output $\pi \leftarrow (\{\pi_k\}_{k\in[n]}, \{x_k\}_{k\in[n]})$.
- $\mathsf{C}_{\mathsf{vrfy}}(pp, x, \pi)$:
  - Parse (pp, π) as $(\{pp_k\}_{k\in[n]}, \{\pi_k\}_{k\in[n]}, \{x_k\}_{k\in[n]})$. Set $x_0 \leftarrow x$.
  - If $\mathcal{V}_k(pp_k, x_k, \pi_k)$ outputs 0 for some $k \in [n]$, output 0. Otherwise, for each $k \in [n] \setminus [t] \cup \{0\}$, behave as follows:
    - Compute $x'_k \leftarrow \overline{\mathtt{reconstruct}}_{\mathbb{H}}(\{x_j\}_{j\in[t]}, k)$.
    - Check whether $x'_k = x_k$. If the check fails then return 0.
  - Return 1.

Figure 8.1: Description of C_{HL}.

8.4.3 Proof sketch for the (ZK, Sound) case

Next, we provide a high-level overview of how our proofs for soundness and zero-knowledge work, deferring formal theorems and complete proofs to Appendix C.5.1, including the analysis of correctness.

Soundness. To show that C_{HL} is sound (as per Definition 8.4), we argue that the existence of an adversary that produces an accepting proof for a statement $x \notin \mathcal{L}_{\mathsf{HL}}$ implies having sufficiently-many reductions (*i.e.*, $n - t + 1$) to soundness of the underlying candidates. Suppose by contradiction that a prover can make a verifier accept a proof for a false statement (*i.e.*, x is s.t. $f(w) \neq x$ for all w) and that only $n - t$ reductions to the soundness are possible. This means that there are at least t sub-statements that are valid (*i.e.*, they belong to $\mathcal{L}_{\mathsf{HL}}$). Moreover, if the verifier of our combiner has accepted the proof, this means that the witnesses for all the sub-statements represent points that belong to the same polynomial of degree $t - 1$. In particular, this polynomial is univocally determined by the witnesses of the t valid sub-statements (that exist by contradiction). Given that this polynomial evaluated on 0 is equal to x (this holds because the verifier has accepted the proof) then the correctness of our secret sharing scheme and the homomorphic property of the language guarantees that x belongs to the language.

Zero-knowledge. Proving the zero-knowledge of this (and of the other schemes we propose) requires some particular care. The traditional way to prove the zero-knowledge of our protocol would be to design a zero-knowledge simulator that executes the simulators of t of the candidates. To prove that the output of this simulator is indistinguishable from the output of an adversarial verifier when interacting with an honest prover, we design a chain of indistinguishable hybrid experiments: the first corresponding to the experiment where the proof is generated via an honest prover, and the last corresponding to the simulated experiment. If by contradiction our simulator is not good (*i.e.*, zero-knowledge does not hold) this would imply that a pair of adjacent hybrids is not indistinguishable, hence, a reduction to a cryptographic primitive can be performed.

In the proof we have just sketched, we have that if by contradiction zero-knowledge does not hold, then we have *one* reduction. However, our security definition states that if zero-knowledge does not hold, then it must be possible to create $n - t + 1$ reductions. Clearly, the proof approach we have discussed does not achieve what we want. Indeed, the above approach does not really make sense as our simulator does not know for which of the candidates the simulator should be run.

We recall that our definition of combiner (Definition 8.5) states that if the combiner is not zero-knowledge then it should be possible to perform $n-t+1$ reductions to soundness or $n-t+1$ reductions to zero-knowledge of the candidates. In particular, if the combiner is not zero-knowledge, then every zero-knowledge simulator we design can be attacked. The intuition is that to prove the security of our scheme, we need to design multiple simulators, such that any time an adversary wins against one simulator, we can create a new reduction to a different input candidate. We construct these multiple simulators as follows.

Each simulator $\mathcal{S}^T$ is parameterized by a set T containing t distinct indices in $[n]$, and runs the simulators for the candidates whose indices are in T. We design a sequence of hybrid arguments in which we gradually replace all the sub-proofs generated by all the candidates in T, using real shares of the witness, with simulated sub-proofs. After these sub-proofs are simulated, we replace the $n - t$ remaining shares of the witness with random group elements. Because there cannot exist a good simulator, the output of $\mathcal{S}^T$ must be distinguishable, for all $T \subseteq [n]$ with $|T| = t$. But this means that there exists an adversary $\mathcal{A}$ that is able to distinguish two adjacent hybrids of the hybrid chain (say, $\mathcal{H}_{i^*-1}$ and $\mathcal{H}_{i^*}$), and we can therefore state that there exists one reduction to zero-knowledge of candidate Π_{i^*}.

Intuitively, whenever some $\mathcal{A}$ breaks zero-knowledge of candidate Π_{i^*} in $\mathcal{S}^T$ by distinguishing the output of a specific protocol Π_{i^*}, it also breaks all the other $\mathcal{S}^{T'}$ for $T \neq T'$ that contain candidate Π_{i^*} within their chain of hybrids. However, there still exist simulators that do not include Π_{i^*} as part of their hybrid chain. Therefore, (a possibly different) $\mathcal{A}$ has to break zero-knowledge of another candidate. By a combinatorics argument, the only way *all* $\binom{n}{t}$ simulators can be broken is by breaking zero-knowledge for at least $n-t+1$ candidates.

8.4.4 Proof sketch for the (ZK, AoK) case

We conclude the subsection by arguing that C_{HL} is also a (ZK, AoK)-combiner. Notice that, in proving zero-knowledge of the (ZK, Sound) case, we relied exclusively on zero-knowledge of the candidates. Hence, we can reuse exactly the same proof here. Therefore, we only need to prove that our combiner is indeed AoK. We provide a sketch of this proof in what follows, deferring the full details to Appendix C.5.2.

Argument-of-knowledge sketch. The proof for the AoK property is similar in spirit to the zero-knowledge one: we argue that, if AoK of the combiner is broken, we can design $n - t + 1$ reductions to AoK of the candidates. To do that, we construct an extractor $\mathcal{E}^T$ which is parameterized by a set T containing t distinct indices in $[n]$, and runs the extractors for the candidates in T. Then, we show that either $\mathcal{E}^T$ successfully extracts a witness, or there exists at least one reduction to AoK of one of the candidates whose indices are in T.

The first portion of the proof consists of an hybrid argument in which we gradually replace all the public parameters generated by setup algorithms of the candidates with ones generated by trapdoor setup algorithms. As for zero-knowledge, if some adversary detects any variation within the hybrid chain, we have a reduction to argument-of-knowledge of a specific candidate.

After that, the extractor attempts to run all the extractors for the primitives indexed by T to recompute a witness from an accepting proof (x, π) supplied by the adversary. For that, we argue that the only way the extractor may output the witness is by successfully running the extractor for all the candidates whose indices are in T. Hence, it suffices for one extractor of the candidates to fail to cause $\mathcal{E}^T$ to fail. For this scheme this is indeed the case, as each extractor extracts a share of the witness w_i, and the only way w can be extracted is by running the reconstruction procedure of Shamir's secret sharing scheme with these t shares.

The proof concludes with the same combinatorics argument used for the proof of zero-knowledge, with the only difference being that reductions may exist either because some adversary distinguishes the setup generation algorithm of one of the candidates, or because some adversary produces an accepting proof (x, π) for which a candidate fails to extract a share of the witness.

8.5 Combiner from recursive proofs

In this subsection we define a t-out-of-n NBB (ZK, AoK)-combiner, with $t > \lfloor \frac{n}{2} \rfloor$ and $n, t \in O(1)$, that may be used for all of NP and generates proofs by "nesting" proofs of the candidates. This procedure fundamentally entails two operations:

- **Computing the "innermost" proof.** The innermost proof is generated by using a proof system to prove knowledge of a witness w for a statement x with respect to language $\mathcal{L}$.
- **Computing an "intermediate" proof.** An intermediate proof consists of using a proof system to prove knowledge of an accepting proof π generated by another (possibly different) proof system. More formally, the NP-language for a scheme Π_j that proves knowledge of an accepting proof generated with scheme Π_{j-1} is defined as $\mathcal{L}_j = \{(\mathcal{V}_{j-1}, pp_{j-1}, x_{j-1}) \mid \exists \pi_{j-1} \; s.t. \; \mathcal{V}_{j-1}(pp_{j-1}, x_{j-1}, \pi_{j-1}) = 1\}$.

The second operation can be applied recursively, and the last intermediate proof will be output in the clear. Intuitively, if such a proof is accepting, it proves that the entire chain of proofs was generated correctly, given that it is possible to recursively extract the original witness[30]. Moreover, the presence of intermediate proofs is what makes the combiner non-black-box in its use of its candidates, as the construction is quantified after the specific instantiations of the candidates $\Pi_1, \ldots, \Pi_n$ are quantified.

Combiner description. The combiner takes as input n non-interactive protocols $\Pi_k = (\mathsf{setup}_k, \mathcal{P}_k, \mathcal{V}_k) \in \mathcal{F}_{\mathsf{NI}}$ as per Definition 2.16, for $k \in [n]$, which works for any NP language $\mathcal{L}$. From a high-level perspective, the prover generates $\binom{n}{t}$ proofs, each with a different subset of t out of n candidates. For each proof, the candidate with the lowest index generates the innermost proof by using statement x and witness w. After that, intermediate proofs are generated recursively by using all the other candidates in the specific subset. We formally describe our combiner $\mathsf{C}_{\mathsf{PoP}} = (\mathsf{C}_{\mathsf{setup}}, \mathsf{C}_{\mathsf{prove}}, \mathsf{C}_{\mathsf{vrfy}})$ in Figure 8.2.

Combiner $\mathsf{C}_{\mathsf{PoP}}$

- $\mathsf{C}_{\mathsf{setup}}(1^l, 1^\lambda)$: Compute $pp_k \leftarrow\!\!\$\; \mathsf{setup}_k(1^l, 1^\lambda)$, for $k \in [n]$. Output $pp \leftarrow \{pp_k\}_{k \in [n]}$.
- $\mathsf{C}_{\mathsf{prove}}(pp, x, w)$:
  - Index all possible subsets containing t of the n candidates with set $S = \{S^i\}_{i \in \binom{n}{t}}$.
  - For $S^i \in S$, denote $\{s_{i,1}, \ldots, s_{i,t}\}$ as the indices of the candidates for this specific subset sorted in lexicographic order. Then, compute proof π_i^{PoP} as follows:
    - Compute the innermost proof $\pi_{s_{i,1}} \leftarrow \mathcal{P}_{s_{i,1}}(pp_{s_{i,1}}, x, w)$. The language associated to this proof is $\mathcal{L}_{s_{i,1}} = \{x \mid \exists w \; s.t. \; (x, w) \in \mathscr{R}_{\mathcal{L}}\}$.
    - For $j \in [2, \ldots, t]$, let $x_{s_{i,j}} \leftarrow (\mathcal{V}_{s_{i,j-1}}, pp_{s_{i,j-1}}, x_{s_{i,j-1}})$. Compute intermediate proof $\pi_{s_{i,j}} \leftarrow \mathcal{P}_{s_{i,j}}(pp_{s_{i,j}}, x_{s_{i,j}}, \pi_{s_{i,j-1}})$. The language associated to the j-th proof is $\mathcal{L}_j = \{x_{s_{i,j}} \mid \exists \pi_{s_{i,j-1}} \; s.t. \; \mathcal{V}_{s_{i,j-1}}(pp_{s_{i,j-1}}, x_{s_{i,j-1}}, \pi_{s_{i,j-1}}) = 1\}$.
    - Set $\pi_i^{\mathsf{PoP}} \leftarrow \pi_{s_{i,t}}$.
  - Set $\pi \leftarrow \{\pi_i^{\mathsf{PoP}}\}_{i \in \binom{n}{t}}$. Output π.
- $\mathsf{C}_{\mathsf{vrfy}}(pp, x, \pi)$:
  - Parse π as $\{\pi_i^{\mathsf{PoP}}\}_{i \in \binom{n}{t}}$.
  - Index all possible subsets containing t of the n candidates with set $S = \{S^i\}_{i \in \binom{n}{t}}$.
  - For $S^i \in S$, denote $\{s_{i,1}, \ldots, s_{i,t}\}$ as the indices of the candidates for this specific subset sorted in lexicographic order. Then, run $\mathcal{V}_{s_{i,t}}(pp_{s_{i,t}}, x_{s_{i,t}}, \pi_i)$.
  - If any of the verifier runs above outputs 0, output 0. Otherwise, output 1.

Figure 8.2: Description of $\mathsf{C}_{\mathsf{PoP}}$.

[30] This is also the reason we consider only knowledge soundness within our combiner. Under the same rationale of [Val08], if one assumes the candidates to be only sound, an accepting intermediate proof would reveal nothing about the truthfulness of the original statement. This is the case, as intermediate proofs only reveal that "an accepting proof exists" - something that holds also for false statements (as soundness is imperfect).

The formal treatment of this protocol follows the same ideas presented in the sketch of the previous section, and is deferred to Appendix C.6.

8.6 Combiner from MPC-in-the-head

In this subsection we exhibit a t-out-of-n NB (ZK, Sound)-combiner, with $t > \lfloor \frac{2n}{3} \rfloor$, that relies on MPC-in-the-head techniques and may be used for all of NP. Similarly to Section 8.4, this combiner can also be instantiated as a t-out-of-n NBB (ZK, AoK)-combiner.

8.6.1 Preliminaries

We start by summarizing specific MPC notation that will be used specifically for this section, following the definitions of [IKOS07]. An n-party protocol $\Pi = (\mathsf{P}_1, \ldots, \mathsf{P}_n)$ is an n-tuple of PPT interactive Turing machines (ITMs). Each party P_i is initialized with a local input $w_i \in \{0,1\}^*$ and random coins $r_i \in \{0,1\}^*$. All the parties share a public input $x \in \{0,1\}^*$. We mainly focus on R-round MPC protocols that securely realize an n-party functionality f, where f maps the input $(x, w_1, \ldots, w_n)$ to an n-tuple of outputs (when only a single output is specified, this output is assumed to be given to all parties). Moreover, instead of letting every party share the input x, we hardcode x in the function f. All the MPC protocols we use have this structure; hence, this will be implicit.

Notation for MPC. Rather than viewing a protocol Π as an n-tuple of interactive Turing machines, it is convenient to view each Turing machine as a sequence of multiple algorithms: $\texttt{frst-msg}_i$, to compute P_i's first messages to its peers; $\texttt{nxt-msg}_i^k$, to compute P_i's k-th round messages for $2 \leq k \leq R$; and $\texttt{output}_i$, to compute P_i's output. Thus, a protocol Π can be defined as $\{(\texttt{frst-msg}_i, \texttt{nxt-msg}_i^k, \texttt{output}_i)\}_{i\in[n],k\in[R]\setminus\{1\}}$. The syntax of the algorithms is as follows:

- $\texttt{frst-msg}_i(w_i; r_i)$: Upon input w_i and some randomness r_i, output messages $(\mathsf{msg}^1_{i\to 1}, \ldots, \mathsf{msg}^1_{i\to n})$, *i.e.*, the first-round messages of party P_i to all other parties. If the message is sent over a broadcast channel, it holds that $\mathsf{msg}^1_{i\to 1} = \mathsf{msg}^1_{i\to 2} = \cdots = \mathsf{msg}^1_{i\to n}$. Note that a party's message to itself can be considered to be its state.
- $\texttt{nxt-msg}_i^k(w_i, \{\mathsf{msg}^l_{j\to i}\}_{j\in[n],l\in[k-1]}; r_i)$: Upon input w_i, the set of messages received from the other parties in all prior rounds, and some randomness r_i, output $(\mathsf{msg}^k_{i\to 1}, \ldots, \mathsf{msg}^k_{i\to n})$, *i.e.*, the k-th round messages of party P_i to all parties.
- $\texttt{output}_i(w_i, \{\mathsf{msg}^k_{j\to i}\}_{j\in[n],k\in[R]}; r_i)$: Upon input w_i, the set of messages received from the other parties in all prior rounds, and some randomness r_i, output y_i, *i.e.*, the output of party P_i.

We note that, unless needed, to not overburden the notation, we do not pass the random coin r as an explicit input of the cryptographic algorithms. In addition, we define the view of party P_i and the consistency of views as follows:

Definition 8.8 (View)**.** *The view* view_i *of party* P_i *is defined as the input* w_i*, the randomness* r_i*, and the all the messages* P_i *sends (denoted as* $\mathsf{msg}_{i,\mathsf{out}}$*, with* $\mathsf{msg}_{i,\mathsf{out}} \leftarrow \{\mathsf{msg}^k_{i\to j}\}_{j\in[n]\setminus\{i\},k\in[R]}$*) and receives (denoted as* $\mathsf{msg}_{i,\mathsf{in}}$*, with* $\mathsf{msg}_{i,\mathsf{in}} \leftarrow \{\mathsf{msg}^k_{j\to i}\}_{j\in[n]\setminus\{i\},k\in[R]}$*) during the execution of the MPC protocol.*

Definition 8.9 (View Consistency)**.** *A pair of views* view_i *and* view_j *are consistent (with respect to the protocol* Π *realizing* f_x*) if* $\{\mathsf{msg}^k_{i\to j}\}_{k\in[R]}$ *is identical to* $\{\mathsf{msg}^k_{j\to i}\}_{k\in[R]}$*.*

Finally, we report properties of MPC protocols from [IKOS07] that are needed within our construction.

Definition 8.10 (Correctness)**.** *Let* $f_x(w_1, \ldots, w_n) : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ *with a hardcoded value* $x \in \{0,1\}^*$ *be an* n*-party function. A protocol* Π *realizes the function* f_x *with* perfect correctness *if for all inputs* $w_1, \ldots, w_n$*, the probability that the output of some player is different from the output of* f_x *is* 0*, where the probability is over the independent choices of the randomness* $r_1, \ldots, r_n$*.*

Definition 8.11 ($\mathcal{K}$-Robustness)**.** *Let $f_x(w_1, \ldots, w_n) : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ with a hardcoded value $x \in \{0,1\}^*$ be an n-party function. A protocol Π realizes the function f_x with* perfect $\mathcal{K}$-Robustness *if it is perfectly correct (Definition 8.10) in the presence of a semi-honest adversary, and moreover for any computational unbounded malicious adversary corrupting a set $\mathcal{I}$ with at most size $\mathcal{K}$, and for any inputs $(w_1, \ldots, w_n)$, if there is no $(w'_1, \ldots, w'_n)$ s.t. $f_x(w'_1, \ldots, w'_n) = 1$, then the probability that some honest parties output 1 in an execution of Π realizing f_x is 0.*

Definition 8.12 ($\mathcal{K}$-Privacy)**.** *Let $f_x(w_1, \ldots, w_n) : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ with a hardcoded value $x \in \{0,1\}^*$ be an n-party function. We denote the joint view $\mathsf{view}_{\mathcal{I}}(w_1, \ldots, w_n)$ as the views of parties P_i ($i \in \mathcal{I}$), corresponding to the execution where every P_i's input is w_i. We denote $f_{x,\mathcal{I}}(w_1, \ldots, w_n)$ the output of function $f_x(w_1, \ldots, w_n)$ received by the parties whose indices are in the set $\mathcal{I}$. A protocol Π realizes the function f_x with* perfect $\mathcal{K}$-Privacy *if there is a PPT simulator $\mathcal{S}$ s.t. for all inputs $(w_1, \ldots, w_n)$, and every set of semi-honest players $\mathcal{I} \subseteq [n]$ with at most size $\mathcal{K}$, the joint view $\mathsf{view}_{\mathcal{I}}(w_1, \ldots, w_n)$ of parties in $\mathcal{I}$ is identically distributed to $\mathcal{S}(\mathcal{I}, x, \{w_k\}_{k\in[\mathcal{I}]}, f_{x,\mathcal{I}}(w_1, \ldots, w_n))$.*

8.6.2 Combiner description

We are now ready to define our combiner, which relies on the following building blocks:

- n non-interactive protocols $\Pi_k = (\mathsf{setup}_k, \mathcal{P}_k, \mathcal{V}_k) \in \mathcal{F}_{\mathsf{NI}}$ as per Definition 2.16, for $k \in [n]$, which work for any NP language $\mathcal{L}$.
- A perfectly secure t-out-of-n secret sharing scheme $\mathtt{SS} = (\mathtt{share}, \mathtt{reconstruct})$, as per Definitions 2.25, 2.26.
- A statistically binding and computationally hiding non-interactive string commitment $\mathtt{NIC} = (\mathsf{setup}, \mathsf{Com}, \mathsf{Open})$, as per Definitions 2.22, 2.23, 2.24.
- An MPC protocol $\Pi_{\mathsf{MPC}} = \{(\mathtt{frst\text{-}msg}_i, \mathtt{nxt\text{-}msg}_i^k, \mathtt{output}_i)\}_{i\in[n],k\in[R]\setminus\{1\}}$ that realizes the function f_x (described below) with perfect $\mathcal{K}$-robustness (as per Definition 8.11) in the malicious model and perfect $\mathcal{K}$-privacy (as per Definition 8.12) in the semi-honest model, where $\mathcal{K} = (n - t)$. f_x has hard-coded a statement x, takes as inputs $w_1, \ldots, w_n$, and outputs 1 if $(x, \mathtt{reconstruct}(\{w_j\}_{j\in[t]})) \in \mathscr{R}_{\mathcal{L}}$; otherwise, it outputs 0.

The construction is similar to [GJS19]. Intuitively, the prover secret shares the witness w, runs the MPC protocol Π_{MPC} to generate views of all parties, uses the non-interactive commitment scheme to commit to each of these views (which, again, include the input, randomness, and sent/received messages of the party they relate to). Then, for each party, it uses a NIZK candidate to prove that the commitments related to that party are computed correctly and the committed values are from an execution of Π_{MPC} for which the party's output is 1. The verifier checks whether all the sub-proofs are accepting, and checks whether the commitments corresponding to outgoing messages are consistent with the commitments corresponding to inbound messages. If all the checks succeed, the verifier outputs 1. Formally, we define our combiner $\mathsf{C}_{\mathsf{MPC}} = (\mathsf{C}_{\mathsf{setup}}, \mathsf{C}_{\mathsf{prove}}, \mathsf{C}_{\mathsf{vrfy}})$ in Figure 8.3.

Combiner $\mathsf{C}_{\mathsf{MPC}}$

- $\mathsf{C}_{\mathsf{setup}}(1^l, 1^\lambda)$: Compute $pp_k \leftarrow\$ \mathsf{setup}_k(1^l, 1^\lambda)$, for $k \in [n]$. Compute $pp_{\mathtt{NIC}} \leftarrow\$ \mathsf{setup}(1^\lambda)$. Output $pp \leftarrow \{pp_k\}_{k\in[n]} \cup \{pp_{\mathtt{NIC}}\}$.
- $\mathsf{C}_{\mathsf{prove}}(pp, x, w)$:
  - Compute $(w_1, \ldots, w_n) \leftarrow\$ \mathtt{share}(w)$.
  - Run MPC protocol Π_{MPC}, where every party P_i ($i \in [n]$) takes w_i as input, and uses uniformly randomly sampled value r_i as the randomness. Obtain $\mathsf{view}_1, \ldots, \mathsf{view}_n$ from the execution of Π_{MPC}.
  - For $i \in [n]$, $j \geq i$, compute $(com_{i,j}, o_{i,j}) \leftarrow\$ \mathsf{Com}(pp_{\mathtt{NIC}}, \{\mathsf{msg}_{i\to j}^k\}_{k\in[R]})$, and $(com_{j,i}, o_{j,i}) \leftarrow\$ \mathsf{Com}(pp_{\mathtt{NIC}}, \{\mathsf{msg}_{j\to i}^k\}_{k\in[R]})$.
  - For $i \in [n]$, set $x'_i \leftarrow (\{com_{i,j}\}_{j\in[n]}, \{com_{j,i}\}_{j\in[n]})$ and $w'_i \leftarrow (\mathsf{view}_i, \{o_{i,j}\}_{j\in[n]}, \{o_{j,i}\}_{j\in[n]})$.
  - For $k \in [n]$, compute $\pi_k = \mathcal{P}_k(pp_k, (pp_{\mathtt{NIC}}, x'_k), w'_k)$. The language associated to

the sub-proofs π_i is $\mathcal{L}_{i,\mathsf{MPC}} = \Big\{(pp_{\mathtt{NIC}}, \{com_{i,j}\}_{j\in[n]}, \{com_{j,i}\}_{j\in[n]}) \;\Big|\; \forall k \in [R] \setminus \{1\}, \exists(\mathsf{view}_i, \{o_{i,j}\}_{j\in[n]}, \{o_{j,i}\}_{j\in[n]}) \; s.t. \; \{\mathsf{msg}^1_{i\to j}\}_{j\in[n]} = \mathtt{frst\text{-}msg}_i(w_i; r_i) \wedge \{\mathsf{msg}^k_{i\to j}\}_{j\in[n]} = \mathtt{nxt\text{-}msg}^k_i(w_i, \{\mathsf{msg}^l_{j\to i}\}_{j\in[n],l\in[k-1]}; r_i) \wedge 1 = \mathtt{output}_i(w_i, \mathsf{msg}_{i,\mathsf{in}}; r_i) \wedge \forall j \in [n], \mathsf{Open}(com_{i,j}, o_{i,j}) = \{\mathsf{msg}^k_{i\to j}\}_{k\in[R]} \wedge \mathsf{Open}(com_{j,i}, o_{j,i}) = \{\mathsf{msg}^k_{j\to i}\}_{k\in[R]}\Big\}$.

  - Output $\pi \leftarrow (\{\pi_k\}_{k\in[n]}, \{x'_k\}_{k\in[n]})$.

- $\mathsf{C}_{\mathsf{vrfy}}(pp, x, \pi)$: Output 1 if all following conditions hold:
  - Compute $\mathcal{V}_k(pp_k, (pp_{\mathtt{NIC}}, x'_k), \pi_k)$ for $k \in [n]$, and all $\mathcal{V}_k$ outputs 1.
  - Parse $\{x'_k\}_{k\in[n]}$ as $\{(\{com_{k,j}\}_{j\in[n]}, \{com_{j,k}\}_{j\in[n]})\}_{k\in[n]}$. For $k \in [n], j \in [n]$, there exists $j \neq k$ s.t. $com_{k,j} = com_{j,k}$.

Figure 8.3: Description of $\mathsf{C}_{\mathsf{MPC}}$.

We remark that, even though all the proofs for the remainder of this section work for $t > \lfloor \frac{n}{2} \rfloor$, we rely on the instantiation of Π_{MPC} as in BGW [BGW88]. In [GMO16], the authors claim that such a protocol is $\lfloor \frac{n-1}{3} \rfloor$-private, perfect $\lfloor \frac{n-1}{3} \rfloor$-robust protocol, and use it to instantiate the protocol of [IKOS07]. The formal treatment of this protocol is deferred to Appendix C.7.

8.6.3 From NIZK candidates to non-interactive commitments

Among the primitives for this construction, we require a non-interactive commitment scheme that is computationally hiding and statistically binding. This is incompatible with our formalization of combiners, given that an adversary breaking the security of the combiner can now yield a reduction to the newly-introduced (computational) assumption, rather than $n - t + 1$ reductions to the security of the underlying candidates. In what follows, we show that it is possible to instantiate statistically binding and computationally hiding non-interactive string commitments by exclusively relying on the security of the candidates. Formally, we prove the following lemma:

Lemma 8.1. *Let $\{\Pi_k = (\mathsf{setup}_k, \mathcal{P}_k, \mathcal{V}_k)\}_{k\in[n]} \in \mathcal{F}_{\mathsf{NI}}$ be non-interactive protocols, as per Definition 2.16, for any NP language $\mathcal{L}$. Let t of the candidates be sound and t of the candidates be zero-knowledge, with $t > \lfloor \frac{n}{2} \rfloor$. Assuming that $\mathsf{NP} \subseteq \mathsf{ZKA}$ and that $\mathsf{ZKA} \not\subseteq \mathsf{ioP/poly}$, there exists an efficient construction of statistically binding and computationally hiding non-interactive string commitments in the CRS model.*

Proof. The high-level idea of the proof consists in expanding the following chain of implications: t-out-of-n NIZK candidates $\implies$ 1-out-of-n OWFs $\implies$ OWF $\implies$ PRG $\implies$ non-interactive commitments.

- **t-out-of-n NIZK candidates $\implies$ 1-out-of-n OWF candidates.** Let $\mathsf{S}_{\mathsf{snd}}$ be the set of sound candidates and S_{zk} be the set of ZK candidates. Since $t > \lfloor \frac{n}{2} \rfloor$, $\mathsf{S}_{\mathsf{snd}} \cap \mathsf{S}_{\mathsf{zk}} \neq \emptyset$. Hence, at least one candidate is both sound and zero-knowledge. Given that we assume $\mathsf{NP} \subseteq \mathsf{ZKA}$ and that $\mathsf{ZKA} \not\subseteq \mathsf{ioP/poly}$, we can invoke [LMP24, Theorem 1.1] on each of the candidates, obtaining a set of n candidate functions $\{f_i\}_{i\in[n]}$. Since at least one of the candidates is both sound and zero-knowledge, at least one of the functions will be a OWF.
- **1-out-of-n OWF candidates $\implies$ OWF.** [Her05, Lemma 4.1] shows a combiner that splits the input among the function candidates (*e.g.*, by XOR), and then simply concatenates the output of all the candidates. More explicitly, as long as there exists an index i for which f_i is a OWF, $f(x) = f_1(x_1)||\ldots||f_n(x_n)$ is also a OWF, with $x = x_1 \oplus x_2 \oplus \cdots \oplus x_n$.
- **OWF $\implies$ PRG.** By [HILL99], there exists a PRG that may be instantiated by a OWF.
- **PRG $\implies$ NI-Com.** Following [MP12], Naor's bit commitment scheme [Nao91] can be made non-interactive by moving the randomness of the receiver to the CRS that is statistically binding and computationally hiding. Once we have a non-interactive bit commitment scheme, we also have a non-interactive string commitment scheme.

□

Further remarks. Proving this lemma requires that $\mathsf{NP} \subseteq \mathsf{ZKA}$ and that $\mathsf{ZKA} \nsubseteq \mathsf{ioP/poly}$. As hinted throughout the technical overview, if the first assumption does not hold, it means that there exists a language in NP that does not admit zero-knowledge arguments. Hence, the problem of designing NIZK combiners becomes less interesting. From here, if the second assumption is falsified, it would imply that $\mathsf{NP} \subseteq \mathsf{ioP/poly}$.

Part V

Conclusions

9 Conclusions

In this composition, we investigated the problem of achieving secure multi-party computation in the presence of honest parties that might not fully trust their devices. We first discussed the framework of subversion-resilient Universal Composability (srUC) due to [CMNV22], which allows one to deploy cryptographic reverse firewalls to thwart adversarial implementations of protocols that attempt to exfiltrate sensitive information in a covert fashion. We then considered robust combiners: constructions that securely instantiate cryptographic primitives while distributing the trust placed on implementations or theoretical assumptions of different nature. In what follows we briefly summarize our contributions, highlighting research directions that might bring significant advances to the state of the art.

Subversion-resilient PAKE from oblivious transfer. In Part II, we presented the first Password-Authenticated Key Exchange (PAKE) protocol ever designed in the srUC framework. In the process, we introduced and instantiated different sanitation-friendly primitives (such as sanitizable oblivious transfer) and modelled framework extensions (such as split functionalities) that may find applicability for follow-up research.

Moreover, we considered for the first time the issue of sanitizing input-triggering attacks within the srUC framework, albeit in an informal fashion. Even though we provided further clarifications on the matter throughout this composition, properly formalizing this class of attacks remains an important open problem, as it would allow capturing even stronger (but realistic!) adversarial strategies (*e.g.*, software backdoors that alter their behaviour upon receiving a specific magic number).

Finally, we remark that our instantiation of the ideal functionality for sanitizable split-authenticated communication $s\mathcal{F}_{\mathsf{SAT}}$ requires a signature scheme with special sanitation-friendly properties that we realize using BLS signatures. Coming up with an alternative instantiation would allow avoiding the introduction of a random oracle as an additional setup assumption.

Subversion-resilient PAKE from smooth-projective hashing. In Part III, we showed a srUC PAKE protocol that runs in a single round while achieving better communication complexity compared to the PAKE described in the previous paragraph. Along the way, we presented advances to the field of malleable smooth-projective hash functions, and introduced the notion of malleable trapdoor smooth-projective hash functions, exhibiting an instantiation for the language of ElGamal ciphertexts. It is a natural question to extend the reach of this novel primitive to further applications, in particular by designing MTSPHFs for more general classes of languages (*e.g.*, linearly representable languages that support rerandomization).

Extending the reach of the srUC framework. The srUC framework is still in its early stages (EUROCRYPT'22), and our PAKE-from-OT protocol [CMMV24] (ASIACRYPT'24) marks the first published follow-up to the seminal paper. As one may expect, this leaves a plethora of research directions open, including the design of protocols for virtually any ideal functionality, as well as modelling extensions to the framework itself. The sky is the limit!

Robust NIZK combiners. In Part IV, we presented the first comprehensive characterization of robust NIZK combiners, showing definitions, tight impossibility results, as well as incomparable constructions.

While the rationale of our definitions is general enough to capture any setup assumption and any arbitrary security property, we exclusively consider NIZK in the CRS model, and discuss only zero-knowledge, soundness, and knowledge soundness. A natural extension to our work would entail capturing different setup assumptions (such as random oracles) and other folklore NIZK properties, including witness indistinguishability, simulation soundness, and simulation extractability.

Moreover, none of our three incomparable constructions manages to achieve optimality with respect to the properties we identified throughout this composition: namely, black-box access to the input candidates, support for a polynomial number of candidates in the security parameter, tight threshold ($t > \lfloor n/2 \rfloor$), functionality for any NP-language, and succinctness preservation. Designing alternative robust NIZK combiners

with all (or even a subset) of these properties remains an interesting research direction, especially if tailored for specific practical deployments.

Towards a real-world impact. The contents of this composition are almost entirely of theoretical nature. In an uncertain landscape where privacy is constantly threatened and anchors of trust are all but granted, making cryptographic reverse firewalls and robust combiners more accessible in the real world definitely warrants further discussion. Indeed, these solutions could find significant resonance within security-conscious users that are unwilling to place their entire trust in a single entity - be it a corporation, an operating system, a piece of hardware, or even a theoretical assumption.

References

[ABJ+19] Prabhanjan Ananth, Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. From FE combiners to secure MPC and back. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 199–228. Springer, Cham, December 2019.

[ABLZ17] Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, and Michal Zajac. A subversion-resistant SNARK. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 3–33. Springer, Cham, December 2017.

[ABMO23] Paula Arnold, Sebastian Berndt, Jörn Müller-Quade, and Astrid Ottenhues. Protection against subversion corruptions via reverse firewalls in the plain universal composability framework. Cryptology ePrint Archive, Report 2023/1951, 2023.

[AFMV19] Giuseppe Ateniese, Danilo Francati, Bernardo Magri, and Daniele Venturi. Public immunization against complete subversion without random oracles. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19International Conference on Applied Cryptography and Network Security*, volume 11464 of *LNCS*, pages 465–485. Springer, Cham, June 2019.

[AHH21] Michel Abdalla, Björn Haase, and Julia Hesse. Security analysis of CPace. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 711–741. Springer, Cham, December 2021.

[AJN+16] Prabhanjan Ananth, Aayush Jain, Moni Naor, Amit Sahai, and Eylon Yogev. Universal constructions and robust combiners for indistinguishability obfuscation and witness encryption. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 491–520. Springer, Berlin, Heidelberg, August 2016.

[AJS17] Prabhanjan Ananth, Aayush Jain, and Amit Sahai. Robust transforming combiners from indistinguishability obfuscation to functional encryption. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 91–121. Springer, Cham, April / May 2017.

[AK25] Benny Applebaum and Eliran Kachlon. How to share an NP statement or combiners for zero-knowledge proofs. Cryptology ePrint Archive, Paper 2025/334, 2025.

[ALSZ20] Behzad Abdolmaleki, Helger Lipmaa, Janno Siim, and Michal Zajac. On QA-NIZK in the BPK model. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 590–620. Springer, Cham, May 2020.

[AMV15] Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 364–375. ACM Press, October 2015.

[BB06] Dan Boneh and Xavier Boyen. On the impossibility of efficiently combining collision resistant hash functions. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 570–583. Springer, Berlin, Heidelberg, August 2006.

[BBC+13] Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. New smooth projective hash functions and one-round authenticated key exchange. Cryptology ePrint Archive, Report 2013/034, 2013.

[BCL+05] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 361–377. Springer, Berlin, Heidelberg, August 2005.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988.

[BFS16] Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 777–804. Springer, Berlin, Heidelberg, December 2016.

[BG24] Nir Bitansky and Nathan Geier. Amplification of non-interactive zero knowledge, revisited. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part IX*, volume 14928 of *LNCS*, pages 361–390. Springer, Cham, August 2024.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[BKP+24] Nir Bitansky, Chethan Kamath, Omer Paneth, Ron D. Rothblum, and Prashant Nalini Vasudevan. Batch proofs are statistically hiding. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *56th ACM STOC*, pages 435–443. ACM Press, June 2024.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Berlin, Heidelberg, December 2001.

[BP13] Fabrice Benhamouda and David Pointcheval. Trapdoor smooth projective hash functions. Cryptology ePrint Archive, Report 2013/341, 2013.

[BPR14] Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 1–19. Springer, Berlin, Heidelberg, August 2014.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[Can03] Ran Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Paper 2003/239, 2003.

[CDFR17] Ignacio Cascudo, Ivan Damgård, Oriol Farràs, and Samuel Ranellucci. Resource-efficient OT combiners with active security. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 461–486. Springer, Cham, November 2017.

[CDN20] Suvradip Chakraborty, Stefan Dziembowski, and Jesper Buus Nielsen. Reverse firewalls for actively secure MPCs. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 732–762. Springer, Cham, August 2020.

[CDVW12] Ran Canetti, Dana Dachman-Soled, Vinod Vaikuntanathan, and Hoeteck Wee. Efficient password authenticated key exchange via oblivious transfer. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 449–466. Springer, Berlin, Heidelberg, May 2012.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Berlin, Heidelberg, August 2001.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.

[CGPS21] Suvradip Chakraborty, Chaya Ganesh, Mahak Pancholi, and Pratik Sarkar. Reverse firewalls for adaptively secure MPC without setup. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 335–364. Springer, Cham, December 2021.

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 255–271. Springer, Berlin, Heidelberg, May 2003.

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, Berlin, Heidelberg, May 2005.

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 68–86. Springer, Berlin, Heidelberg, May 2003.

[CKLM12] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. Malleable proof systems and applications. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 281–300. Springer, Berlin, Heidelberg, April 2012.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM STOC*, pages 364–369. ACM Press, May 1986.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[CMMV24] Suvradip Chakraborty, Lorenzo Magliocco, Bernardo Magri, and Daniele Venturi. Key exchange in the post-snowden era: Universally composable subversion-resilient PAKE. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part V*, volume 15488 of *LNCS*, pages 101–133. Springer, Singapore, December 2024.

[CMNV22] Suvradip Chakraborty, Bernardo Magri, Jesper Buus Nielsen, and Daniele Venturi. Universally composable subversion-resilient cryptography. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 272–302. Springer, Cham, May / June 2022.

[CMVX25] Michele Ciampi, Lorenzo Magliocco, Daniele Venturi, and Yu Xia. Robust non-interactive zero-knowledge combiners. Cryptology ePrint Archive, Report 2025/240, 2025.

[CMY+16] Rongmao Chen, Yi Mu, Guomin Yang, Willy Susilo, Fuchun Guo, and Mingwu Zhang. Cryptographic reverse firewall via malleable smooth projective hash functions. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 844–876. Springer, Berlin, Heidelberg, December 2016.

[Cra96] Ronald Cramer. Modular design of secure yet practical cryptographic protocols. *Ph. D.-thesis, CWI and U. of Amsterdam*, 2, 1996.

[CRS+07] Ran Canetti, Ronald L. Rivest, Madhu Sudan, Luca Trevisan, Salil P. Vadhan, and Hoeteck Wee. Amplifying collision resistance: A complexity-theoretic treatment. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 264–283. Springer, Berlin, Heidelberg, August 2007.

[DFG+23] Yevgeniy Dodis, Niels Ferguson, Eli Goldin, Peter Hall, and Krzysztof Pietrzak. Random oracle combiners: Breaking the concatenation barrier for collision-resistance. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 514–546. Springer, Cham, August 2023.

[DFP15] Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. A more cautious approach to security against mass surveillance. In Gregor Leander, editor, *FSE 2015*, volume 9054 of *LNCS*, pages 579–598. Springer, Berlin, Heidelberg, March 2015.

[DHP+18] Pierre-Alain Dupont, Julia Hesse, David Pointcheval, Leonid Reyzin, and Sophia Yakoubov. Fuzzy password-authenticated key exchange. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 393–424. Springer, Cham, April / May 2018.

[DK05] Yevgeniy Dodis and Jonathan Katz. Chosen-ciphertext security of multiple encryption. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 188–209. Springer, Berlin, Heidelberg, February 2005.

[DMS16] Yevgeniy Dodis, Ilya Mironov, and Noah Stephens-Davidowitz. Message transmission with reverse firewalls—secure communication on corrupted machines. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 341–372. Springer, Berlin, Heidelberg, August 2016.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[FHNS16] Marc Fischlin, Amir Herzberg, Hod Bin Noon, and Haya Shulman. Obfuscation combiners. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 521–550. Springer, Berlin, Heidelberg, August 2016.

[FL08] Marc Fischlin and Anja Lehmann. Multi-property preserving combiners for hash functions. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 375–392. Springer, Berlin, Heidelberg, March 2008.

[FLP08] Marc Fischlin, Anja Lehmann, and Krzysztof Pietrzak. Robust multi-property combiners for hash functions revisited. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming*, pages 655–666, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[FLP14] Marc Fischlin, Anja Lehmann, and Krzysztof Pietrzak. Robust multi-property combiners for hash functions. *Journal of Cryptology*, 27(3):397–428, July 2014.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *31st FOCS*, pages 308–317. IEEE Computer Society Press, October 1990.

[FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple noninteractive zero knowledge proofs under general assumptions. *SIAM Journal on computing*, 29(1):1–28, 1999.

[FR24] Oriol Farràs and Jordi Ribes-González. One-out-of-q OT combiners. *IEEE Trans. Inf. Theory*, 70(4):2984–2998, 2024.

[Fuc18] Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 315–347. Springer, Cham, March 2018.

[GHP18] Federico Giacon, Felix Heuer, and Bertram Poettering. KEM combiners. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 190–218. Springer, Cham, March 2018.

[GJS19] Vipul Goyal, Aayush Jain, and Amit Sahai. Simultaneous amplification: The case of non-interactive zero-knowledge. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 608–637. Springer, Cham, August 2019.

[GK10] Adam Groce and Jonathan Katz. A new framework for password-based authenticated key exchange. Cryptology ePrint Archive, Report 2010/147, 2010.

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 524–543. Springer, Berlin, Heidelberg, May 2003.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 1069–1083. USENIX Association, August 2016.

[GMV20] Chaya Ganesh, Bernardo Magri, and Daniele Venturi. Cryptographic reverse firewalls for interactive proof systems. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *ICALP 2020*, volume 168 of *LIPIcs*, pages 55:1–55:16. Schloss Dagstuhl, July 2020.

[GO07] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 323–341. Springer, Berlin, Heidelberg, August 2007.

[Gol01] Oded Goldreich. *The Foundations of Cryptography, vol. 1: Basic Tools*. Cambridge University Press, 2001.

[Gol04] Oded Goldreich. *The Foundations of Cryptography, vol. 2: Basic Applications*. Cambridge University Press, 2004.

[Goo47] Reuben Louis Goodstein. Transfinite ordinals in recursive number theory. *The Journal of Symbolic Logic*, 12(4):123–129, 1947.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 339–358. Springer, Berlin, Heidelberg, May / June 2006.

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3), jun 2012.

[Her05] Amir Herzberg. On tolerant cryptographic constructions. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 172–190. Springer, Berlin, Heidelberg, February 2005.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HKN+05] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. On robust combiners for oblivious transfer and other primitives. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 96–113. Springer, Berlin, Heidelberg, May 2005.

[HKNY23] Taiga Hiroka, Fuyuki Kitagawa, Ryo Nishimaki, and Takashi Yamakawa. Robust combiners and universal constructions for quantum cryptography. Cryptology ePrint Archive, Report 2023/1772, 2023.

[HN24] Shuichi Hirahara and Mikito Nanashima. One-way functions and zero knowledge. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *56th ACM STOC*, pages 1731–1738. ACM Press, June 2024.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

[JMS20] Aayush Jain, Nathan Manohar, and Amit Sahai. Combiners for functional encryption, unconditionally. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 141–168. Springer, Cham, May 2020.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, 2nd edition.* Chapman and Hall/CRC, 2014.

[KV11] Jonathan Katz and Vinod Vaikuntanathan. Round-optimal password-based authenticated key exchange. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 293–310. Springer, Berlin, Heidelberg, March 2011.

[Lin15] Yehuda Lindell. An efficient transform from sigma protocols to NIZK with a CRS and non-programmable random oracle. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 93–109. Springer, Berlin, Heidelberg, March 2015.

[Lin16] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046, 2016.

[LMP24] Yanyi Liu, Noam Mazor, and Rafael Pass. A note on zero-knowledge for NP and one-way functions. *Electron. Colloquium Comput. Complex.*, TR24-095, 2024.

[LPWW20] Benoît Libert, Alain Passelègue, Hoeteck Wee, and David J. Wu. New constructions of statistical NIZKs: Dual-mode DV-NIZKs and more. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 410–441. Springer, Cham, May 2020.

[LWC+25] Jiahao Liu, Yi Wang, Rongmao Chen, Xinyi Huang, Jinshu Su, and Moti Yung. Universally composable subversion-resilient authenticated key exchange. Cryptology ePrint Archive, Paper 2025/1521, 2025.

[Mau09] Ueli M. Maurer. Unifying zero-knowledge proofs of knowledge. In Bart Preneel, editor, *AFRICACRYPT 09*, volume 5580 of *LNCS*, pages 272–286. Springer, Berlin, Heidelberg, June 2009.

[Mit12] Arno Mittelbach. Hash combiners for second pre-image resistance, target collision resistance and pre-image resistance have long output. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 522–539. Springer, Berlin, Heidelberg, September 2012.

[Mit13] Arno Mittelbach. Cryptophia's short combiner for collision-resistant hash functions. In Michael J. Jacobson, Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13International Conference on Applied Cryptography and Network Security*, volume 7954 of *LNCS*, pages 136–153. Springer, Berlin, Heidelberg, June 2013.

[MP06] Remo Meier and Bartosz Przydatek. On robust combiners for private information retrieval and other primitives. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 555–569. Springer, Berlin, Heidelberg, August 2006.

[MP12] Mohammad Mahmoody and Rafael Pass. The curious case of non-interactive commitments - on the power of black-box vs. non-black-box use of primitives. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 701–718. Springer, Berlin, Heidelberg, August 2012.

[MP14] Bart Mennink and Bart Preneel. Breaking and fixing cryptophia's short combiner. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis G. Askoxylakis, editors, *CANS 14*, volume 8813 of *LNCS*, pages 50–63. Springer, Cham, October 2014.

[MPW07] Remo Meier, Bartosz Przydatek, and Jürg Wullschleger. Robuster combiners for oblivious transfer. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 404–418. Springer, Berlin, Heidelberg, February 2007.

[MS15] Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic reverse firewalls. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 657–686. Springer, Berlin, Heidelberg, April 2015.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, January 1991.

[Ore87] Yair Oren. On the cunning power of cheating verifiers: Some observations about zero knowledge proofs (extended abstract). In *28th FOCS*, pages 462–471. IEEE Computer Society Press, October 1987.

[Pie07] Krzysztof Pietrzak. Non-trivial black-box combiners for collision-resistant hash-functions don't exist. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 23–33. Springer, Berlin, Heidelberg, May 2007.

[Pie08] Krzysztof Pietrzak. Compression from collisions, or why CRHF combiners have a long output. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 413–432. Springer, Berlin, Heidelberg, August 2008.

[PR20] Bertram Poettering and Paul Rösler. Combiners for AEAD. *IACR Trans. Symm. Cryptol.*, 2020(1):121–143, 2020.

[Ps05] Rafael Pass and abhi shelat. Unconditional characterizations of non-interactive zero-knowledge. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 118–134. Springer, Berlin, Heidelberg, August 2005.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, Berlin, Heidelberg, August 2008.

[Rin23] Magnus Ringerud. Note on subversion-resilient key exchange. Cryptology ePrint Archive, Report 2023/749, 2023.

[RTYZ16] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Cliptography: Clipping the power of kleptographic attacks. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 34–64. Springer, Berlin, Heidelberg, December 2016.

[RTYZ17] Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Generic semantic security against a kleptographic adversary. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 907–922. ACM Press, October / November 2017.

[RX23] Lawrence Roy and Jiayu Xu. A universally composable PAKE with zero communication cost - (and why it shouldn't be considered UC-secure). In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 714–743. Springer, Cham, May 2023.

[Sim84] Gustavus J. Simmons. Authentication theory/coding theory. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 411–431. Springer, Berlin, Heidelberg, August 1984.

[Sim86] Gustavus J. Simmons. A secure subliminal channel (?). In Hugh C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 33–41. Springer, Berlin, Heidelberg, August 1986.

[Som06] Christian Sommer. *Robust Combiners for Cryptographic Primitives*. Master thesis, ETH Zurich, 2006.

[Sta25] Joint Statement. Joint statement of scientists and researchers on the eu presidency's new proposal for the child sexual abuse regulation. https://csa-scientist-open-letter.org/Sep2025, 2025.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

[Val08] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18. Springer, Berlin, Heidelberg, March 2008.

Part VI

Appendix

Part Overview

This portion of the composition completes the formal treatment of the contributions presented in the main body by providing complete security proofs and further technical details.

Organization. In Appendix A and Appendix B, we discuss the deferred security proofs and alternative modelling considerations stemming from our srUC PAKE protocols. In the former, we focus on the srUC PAKE from oblivious transfer presented in Section 5; in the latter, we consider the srUC PAKE from smooth-projective hashing presented in Section 7. In Appendix C, we provide further details pertaining our characterization of robust NIZK combiners presented in Section 8.

A Subversion-resilient PAKE from Oblivious Transfer

This section completes the formal treatment of Section 5.

A.1 Security proofs

In what follows, we report deferred security proofs.

A.1.1 Proof of Theorem 5.2

We first show that the firewalls are *strongly sanitizing*, meaning that no environment $\mathcal{Z}$ is able to distinguish a protocol run in which either core is speciously corrupted and its firewall is honest from a run in which the same core is incorruptible and its firewall is honest.

Lemma A.1. *The firewall* F_S *of the sender* C_S *in Figure 5.4 is strongly sanitizing.*

Proof (Lemma A.1). We show that no environment $\mathcal{Z}$, which allows specious corruptions for the core C_S but does not corrupt the respective firewall F_S, is able to distinguish between an execution of the protocol in Figure 5.4 with an incorruptible C_S from one that accepts the corruption $\tilde{\mathsf{C}}_\mathrm{S}$.

To do that, we show that $\tilde{\mathsf{C}}_\mathrm{S}$ is unable to leak any information. Since the only outbound message of the sender is a tuple of ciphertexts of a dual-mode encryption scheme (y_0, y_1), $\tilde{\mathsf{C}}_\mathrm{S}$ can modify how this tuple is generated. In order to establish a side-channel, the specious core may (i) produce signaling ciphertexts distributed as valid ciphertexts (*e.g.*, by means of rejection sampling), or (ii) produce a different message. $\mathcal{Z}$ is unable to determine whether the specious corruption had effect or not, as:

- In case (i), the ciphertext gets replaced by F_i with a fresh ciphertext that is side-channel free, as the random coin ρ used in $\mathsf{AlignEnc}$ and the internal coins tossed in the computation of (y'_0, y'_1) are sampled uniformly at random and are independent from each other.
- Case (ii) can never occur with probability better than negligible, as otherwise the core would not be specious.

□

Lemma A.2. *The firewall* F_R *of the receiver* C_R *in Figure 5.4 is strongly sanitizing.*

Proof (Lemma A.2). We show that no environment $\mathcal{Z}$, which allows specious corruptions for the core C_R but does not corrupt the respective firewall F_R, is able to distinguish between an execution of the protocol in Figure 5.4 with an incorruptible C_R from one that accepts the corruption $\tilde{\mathsf{C}}_\mathrm{R}$.

To do that, we show that $\tilde{\mathsf{C}}_\mathrm{R}$ is unable to leak any information. Since the only outbound message of the receiver is a public key pk of a dual-mode encryption scheme, $\tilde{\mathsf{C}}_\mathrm{R}$ can modify how pk is generated in order to exfiltrate information. The specious core may (i) produce a signaling key distributed as a valid key (*e.g.*, by means of rejection sampling), or (ii) produce a different message. $\mathcal{Z}$ is unable to determine whether the specious corruption had effect or not, as:

- In case (i), the key is mauled by F_R to a different key, as the firewall samples the random coin ρ used in MaulPK uniformly at random, and the output key is uniformly distributed.
- Case (ii) can never occur with probability better than negligible, as otherwise the core would not be specious.

□

Lemma A.3. *For every adversary* $\mathcal{A}$ *corrupting either party maliciously and the firewall of the other (honest) party semi-honestly in an execution of the protocol* Π *as in Figure 5.4 in the* $(\mathcal{F}_{\mathsf{SAT}}, \mathcal{F}_{\mathsf{crs}})$*-hybrid model, there exists a simulator* $\mathcal{S}$ *such that, for all environments* $\mathcal{Z}$*, the following holds:*

$$\textsc{Exec}^{(\mathcal{F}_{\mathsf{SAT}}, \mathcal{F}_{\mathsf{crs}})}_{\Pi, \mathcal{A}, \mathcal{Z}} \approx_c \textsc{Exec}_{\mathcal{F}_{\mathsf{sOT}}, \mathcal{S}, \mathcal{Z}}$$

Proof (Lemma A.3). The proof of this lemma follows closely the proof of the original generic framework [PVW08, Theorem 4.1]: the dual-mode encryption scheme induces statistical security for the sender in messy mode and statistical security for the receiver in decryption mode. The indistinguishability of modes induces computational security for the specific party in the protocol run.

In the context of our framework, for each corruption scenario, the adversary can semi-honestly corrupt the firewall of the matching honest core. Intuitively, revealing this information does not provide the adversary any advantage, as semi-honestly corrupting the firewall only reveals blinding factors and the matching core is honest.

We analyze corruption scenarios separately, simulating a run of the protocol Π for the core of the honest party by defining a simulator that interacts with $\mathcal{F}_{\mathsf{sOT}}$ and the environment $\mathcal{Z}$. We show the proof for $\mathsf{mode} = \mathsf{mes}$; the proof for $\mathsf{mode} = \mathsf{dec}$ only differs in the generation of the common reference strings.

Malicious Receiver. The receiver R is corrupted, so the core of the sender C_S is honest and its firewall F_S is semi-honest. $\mathcal{S}$ generates the CRS in messy mode, obtaining a trapdoor t that enables it to find a messy branch related to a (possibly malformed) public key pk. It then initializes the real-world adversary $\mathcal{A}$ with crs.

When $\mathcal{A}$ sends a message containing public key pk, $\mathcal{S}$ extracts a messy branch $b \leftarrow \mathsf{FindMessy}(\mathsf{crs}, t, pk)$. Then, upon receiving a $(\textsc{receipt}, \mathsf{sid})$ query from $\mathcal{F}_{\mathsf{sOT}}$ (signaling that the dummy sender's core has selected its two input messages), it sends to $\mathcal{F}_{\mathsf{sOT}}$ the following queries:

1. $(\textsc{blind}, \mathsf{sid}, (x'_0, x'_1))$ on S_S.
2. $(\textsc{receiver}, \mathsf{sid}, 1-b)$ on IO_R, receiving $\hat{x}_{1-b} = x_{1-b} * x'_{1-b}$.

Notice that the blind query on S_S is allowed, as the firewall of the sender F_S is semi-honestly corrupted, so its inputs (x'_0, x'_1) are known to $\mathcal{S}$.

After that, $\mathcal{S}$ simulates the behaviour of S by computing $x_{1-b} \leftarrow \hat{x}_{1-b} * (x'_{1-b})^{-1}$, $y_{1-b} \leftarrow\!\!\$\ \mathsf{Enc}(pk, 1-b, x_{1-b})$ and $y_b \leftarrow\!\!\$\ \mathsf{Enc}(pk, b, 0^n)$, and sending (y_{1-b}, y_b) to $\mathcal{A}$.

The only difference between the real world and ideal world is in the generation of y_b. Since b is the messy branch correctly identified by the trapdoor, $\mathsf{Enc}(pk, b, 0^n) \overset{s}{\approx} \mathsf{Enc}(pk, b, x_b)$. Hence, the two experiments are statistically indistinguishable.

Malicious Sender. The sender S is corrupted, so the core of the receiver C_R is honest and its firewall F_S is semi-honest. $\mathcal{S}$ generates the CRS in decryption mode, obtaining a tuple of keys (pk, sk_0, sk_1) with sk_b being the decryption key of the b-th branch for $b \in \{0,1\}$, and forwards pk to $\mathcal{A}$. Let us denote with ρ^R the random string used by (semi-honest) F_R to blind pk to $\widetilde{pk}$. This string is revealed to $\mathcal{S}$, as F_R is semi-honest.

Whenever $\mathcal{A}$ sends $(\tilde{y}_0, \tilde{y}_1)$ on behalf of C_S (*i.e.*, the core of the malicious sender), $\mathcal{S}$ computes $x_b \leftarrow \mathsf{Dec}(sk_b, \mathsf{AlignEnc}(\hat{y}_b, \rho^\mathrm{R}))$ for all $b \in \{0,1\}$, and sends to $\mathcal{F}_{\mathsf{sOT}}$ a $(\textsc{sender}, \mathsf{sid}, (x_0, x_1))$ query on IO_S. Then, upon receiving a (possibly different) tuple $(\tilde{y}''_0, \tilde{y}''_1)$ from $\mathcal{A}$ on behalf of the firewall of the (malicious) sender F_S, $\mathcal{S}$ computes $x'_b \leftarrow \mathsf{Dec}(sk_b, \mathsf{AlignEnc}(\tilde{y}''_b, \rho^\mathrm{R})) * (x_b)^{-1}$ for all $b \in \{0,1\}$, and forwards a $(\textsc{blind}, \mathsf{sid}, (x'_0, x'_1))$ query to $\mathcal{F}_{\mathsf{sOT}}$. Note that $\mathcal{S}$ can observe these adversarial interactions as it controls $\mathcal{F}_{\mathsf{SAT}}$.

Since the only difference between the real world and ideal world is in the generation of the keys, and $(pk, sk_\sigma) \overset{s}{\approx} \mathsf{KeyGen}(\sigma)$ by trapdoor key generation. However, crs was generated in decryption mode, whereas the protocol expects messy mode. Since $\mathsf{SetupMessy}_1 \overset{c}{\approx} \mathsf{SetupDec}_1$, the two experiments are only computationally indistinguishable. □

Proof (Theorem 5.2). The theorem statement follows by looking at the standard corruption translation table. Since the adversary maliciously corrupts either the sender or the receiver, the remaining honest party has either (i) a Honest core and a SemiHonest firewall, or (ii) a Specious core and a Honest firewall. Since both firewalls are strongly sanitizing, as shown in Lemma A.1 and Lemma A.2, the core in case (ii) can be considered Honest [CMNV22, Lemma 2]. Hence, the statement follows directly by Lemma A.3. We consider Isolate corruptions as Malicious corruptions. □

A.1.2 Proof of Theorem 5.3

First of all, we argue that the protocol is correct.

Lemma A.4. *The protocol in Figure 5.5 is correct.*

Proof (Lemma A.4). If $\mathsf{pw}^{\mathrm{I}} = \mathsf{pw}^{\mathrm{R}}$, then $K^{\mathrm{I}} * \widetilde{K}'' = \widetilde{K}' * K^{\mathrm{R}}$ (*i.e.*, parties output the same key):

$$\begin{aligned} K^{\mathrm{I}} * \widetilde{K}'' &= (r^{\mathrm{I}}_{\mathsf{pw}^{\mathrm{I}}} * \rho^{\mathrm{I}}_{\mathsf{pw}^{\mathrm{I}}}) * (r^{\mathrm{R}}_{\mathsf{pw}^{\mathrm{I}}} * \rho^{\mathrm{R}}_{\mathsf{pw}^{\mathrm{I}}}) \\ &= (r^{\mathrm{I}}_{\mathsf{pw}^{\mathrm{R}}} * \rho^{\mathrm{I}}_{\mathsf{pw}^{\mathrm{R}}}) * (r^{\mathrm{R}}_{\mathsf{pw}^{\mathrm{R}}} * \rho^{\mathrm{R}}_{\mathsf{pw}^{\mathrm{R}}}) \\ &= \widetilde{K}' * K^{\mathrm{R}}. \end{aligned}$$

□

Next, we prove strong sanitation.

Lemma A.5. *The firewall F_{P} in Figure 5.5, for $\mathsf{P} \in \{\mathrm{I}, \mathrm{R}\}$ is strongly sanitizing.*

Proof (Lemma A.5). We show that no environment $\mathcal{Z}$, which allows specious corruptions for the core C_{P} but does not corrupt the respective firewall F_{P}, is able to distinguish between an execution of the protocol in Figure 5.5 with an incorruptible C_{P} from one that accepts the corruption $\tilde{\mathsf{C}}_{\mathsf{P}}$. In particular, we show the proof for C_{I} is unable to leak any information (the proof is essentially the same for a specious receiver).

The initiator's outbound communication consists of sending its password pw^{I} to the first run of $\mathcal{F}_{\mathsf{sOT}}$, choosing two random strings $(r^{\mathrm{I}}_0, r^{\mathrm{I}}_1)$, and sending these strings through the second run of $\mathcal{F}_{\mathsf{sOT}}$. By security of $\mathcal{F}_{\mathsf{sOT}}$, the former reveals no information about pw^{I}. Hence, the only possible leakage lies in a receiver obtaining a (potentially biased) random string r^{I}_b. But the initiator's firewall participates to $\mathcal{F}_{\mathsf{sOT}}$ by blinding each r^{I}_b with a different uniformly-random string ρ^{I}_b.

□

Lemma A.6. *For every adversary $\mathcal{A}$ corrupting either party maliciously in an execution of the protocol Π as in Figure 5.4 in the $(\mathcal{F}_{\mathsf{sOT}}, \mathcal{F}_{\mathsf{SAT}})$-hybrid model, there exists a simulator $\mathcal{S}$ such that, for all environments $\mathcal{Z}$, the following holds:*

$$\textsc{Exec}^{(\mathcal{F}_{\mathsf{sOT}}, \mathcal{F}_{\mathsf{SAT}})}_{\Pi, \mathcal{A}, \mathcal{Z}} \approx_c \textsc{Exec}_{\mathcal{F}_{\mathsf{RE}}, \mathcal{S}, \mathcal{Z}}$$

Proof (Lemma A.6). Intuitively, the proof follows closely the proof of the non-sanitized protocol [CDVW12, Theorem 1], restricted to the case of static corruptions.

We analyze corruption scenarios separately, defining a simulator $\mathcal{S}$ interacting with $\mathcal{F}_{\mathsf{RE}}$ and the environment $\mathcal{Z}$, and simulating a run of the core of the honest party of the protocol Π.

Malicious Initiator. The initiator I is corrupted, so the core of the receiver C_{R} is honest and its firewall F_{R} is semi-honest. The simulator $\mathcal{S}$ can simulate the ideal functionality $\mathcal{F}_{\mathsf{sOT}}$, obtaining whatever the adversary $\mathcal{A}$ inputs on behalf of corrupted parties (honest parties are still internally simulated). More specifically, it simulates the first batch of OTs (in which the initiator I acts as the receiver) as follows: whenever $\mathcal{A}$ sends input pw to $\mathcal{F}_{\mathsf{sOT}}$ acting as a malicious initiator in the first OT, $\mathcal{S}$ samples a random value $r' \leftarrow\!\!\$\ \mathcal{I}$ and returns it to $\mathcal{A}$. It then simulates the second batch of OTs (in which the initiator I acts as the sender) as follows:

1. Whenever $\mathcal{A}$ sends inputs $(r^{\mathrm{I}}_0, r^{\mathrm{I}}_1)$ to $\mathcal{F}_{\mathsf{sOT}}$ acting as the core of a malicious initiator in the second OT, $\mathcal{S}$ records the tuple.
2. Whenever $\mathcal{A}$ sends inputs $(\rho^{\mathrm{I}}_0, \rho^{\mathrm{I}}_1)$ to $\mathcal{F}_{\mathsf{sOT}}$ acting as the firewall of the malicious initiator in the second OT, $\mathcal{S}$ records the tuple.

Notice that step (1) is still necessary, as $\mathcal{F}_{\mathsf{sOT}}$ expects $\mathcal{A}$ to interact both as a malicious core, as well as a malicious firewall.

After this preliminary phase, $\mathcal{S}$ sends message $(\textsc{NewSession}, \mathsf{sid}, \mathrm{I}, \mathrm{R}, \mathsf{pw})$ to $\mathcal{F}_{\mathsf{RE}}$. It also sends message $(\textsc{ok}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{RE}}$, prompting the (dummy) receiver to input its password. Then, it computes the key as the initiator would in the protocol by using the password pw it extracted from the adversary. More specifically, $K^{\mathrm{I}} = r^{\mathrm{I}}_{\mathsf{pw}} * \rho^{\mathrm{I}}_{\mathsf{pw}}$, and $\widetilde{K}'' = r'$. $\mathcal{S}$ then sends a $\textsc{NewKey}$ query containing $K = K^{\mathrm{I}} * \widetilde{K}''$ to $\mathcal{F}_{\mathsf{RE}}$ for both parties, receiving $\mathsf{skey}_{\mathrm{I}}$ (*i.e.*, the output of the corrupted initiator). $\mathsf{skey}_{\mathrm{I}}$ is forwarded to $\mathcal{A}$.

Malicious Responder. $\mathcal{S}$ initially waits for a message $(\mathsf{sid}, \mathrm{I}, \mathrm{R})$ from $\mathcal{F}_{\mathsf{RE}}$. As soon as the message is received, it inputs $(\textsc{ok}, \mathsf{sid})$ to $\mathcal{F}_{\mathsf{RE}}$, receiving a matching $(\textsc{wakeup}, \mathsf{sid}, \mathrm{I}, \mathrm{R})$ message. From this point on, it simulates the $\textsc{Respond}$ and $\textsc{NewKey}$ queries similarly to the simulation for the malicious initiator described above. □

Proof (Theorem 5.3). The theorem statement follows by looking at the standard corruption translation table. Since the adversary maliciously corrupts either the initiator or the responder, the remaining honest party has either (i) a $\textsc{Honest}$ core and a $\textsc{SemiHonest}$ firewall, or (ii) a $\textsc{Specious}$ core and a $\textsc{Honest}$ firewall. Both firewalls are strongly sanitizing, and as shown in Lemma A.5, the core in case (ii) can be considered to be $\textsc{Honest}$ [CMNV22, Lemma 2]. Hence, the statement follows directly by Lemma A.6. We consider $\textsc{Isolate}$ corruptions as $\textsc{Malicious}$ corruptions. □

A.1.3 Proof of Lemma 5.1

Proof (Lemma 5.1). Let $\Pi = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy}, \mathsf{MaulVK}, \mathsf{AlignSig}, \mathsf{UnAlignSig})$ be a KS-EUF-CMA signature scheme. Assume by contradiction the theorem statement does not hold. This implies the existence of an efficient adversary $\mathcal{A}'$ that breaks KS-EUF-CMA security of Π with non-negligible probability. Leveraging $\mathcal{A}'$, we build an efficient reduction $\mathcal{A}$ that breaks the EUF-CMA of Π with the same probability.

$\mathcal{A}$ acts as the challenger for KS-EUF-CMA security of $\mathcal{A}'$. Therefore, it has to (i) provide a verification key and randomness, and (ii) answer to signature queries. For (i), $\mathcal{A}$ simply mauls the verification key vk received from $\mathcal{C}$ to $\widetilde{\mathsf{vk}}$ via algorithm MaulVK by sampling a uniformly-distributed randomness ρ; for (ii), $\mathcal{A}$ forwards signature queries on m_i's to $\mathcal{C}$, and mauls signatures σ_i to $\widetilde{\sigma}_i$ by using $\mathsf{AlignSig}$ with input the same randomness ρ that was used to maul vk. Finally, $\mathcal{A}'$ produces a forged signature σ^* on $m^* \notin \mathcal{M}$ that verifies under $\widetilde{\mathsf{vk}}$. Using algorithm $\mathsf{UnAlignSig}$ with randomness ρ, $\mathcal{A}$ can translate this forgery to a forged signature σ for $m^* \notin \mathcal{M}$ that verifies under vk. This concludes the proof.

We depict pictorially the reduction in Figure A.1. □

A.1.4 Proof of Theorem 5.6

We first show that the firewalls are *strongly sanitizing*, meaning that no environment $\mathcal{Z}$ is able to distinguish a protocol run in which either core is speciously corrupted and its firewall is honest from a run in which the same core is incorruptible and their firewall is honest.

Lemma A.7. *The firewall F_i in Figures 5.7, 5.8 is strongly sanitizing.*

Proof (Lemma A.7). We show that no environment $\mathcal{Z}$, which allows specious corruptions for the core C_i but does not corrupt the respective firewall F_i, is able to distinguish between an execution of the protocol in Figures 5.7, 5.8 with an incorruptible C_i from one that accepts the corruption $\tilde{\mathsf{C}}_i$. We analyze the link initialization and the message authentication phases separately.

Link Initialization. The outbound communication of a core C_i is comprised of a verification key vk_i, a session ID $\widetilde{\mathsf{sid}}_i = (\widetilde{\mathsf{vk}}_i, \mathsf{vk}_j)$, and a signature σ_i. It is easy to see that the only potential source of leakage is vk_i, as $\widetilde{\mathsf{sid}}_i$ only contains $\widetilde{\mathsf{vk}}_i$ (alongside potentially adversarial inbound verification keys) and $\widetilde{\sigma}_i$ is a deterministic signature of a sanitized value. However, vk_i is sanitized to a fresh verification key by means of algorithm MaulVK with randomness ρ_i. Hence, no leakage is possible.

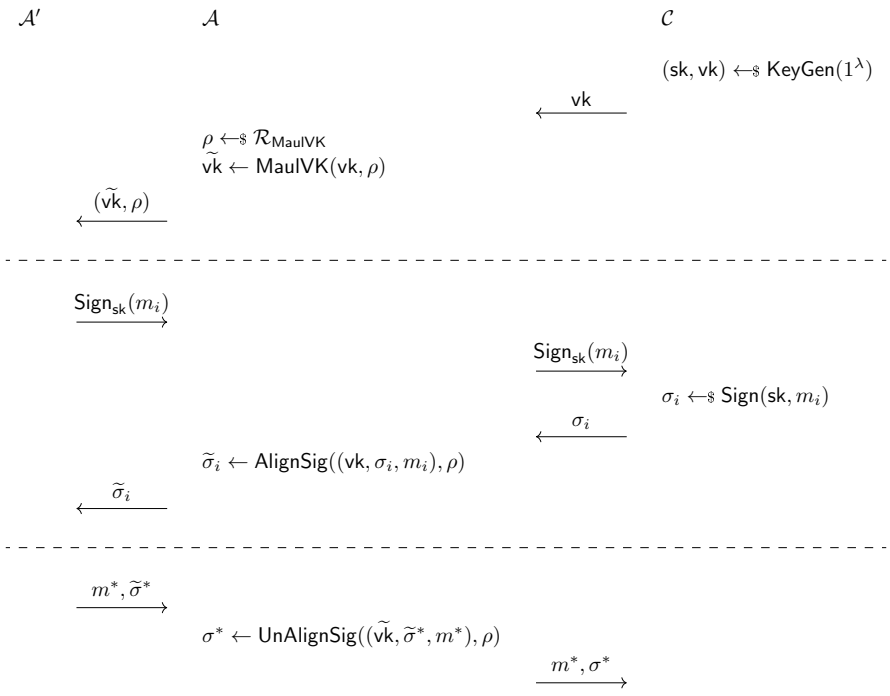


Figure A.1: A reduction proving Lemma 5.1. $\mathcal{A}'$ is the algorithm that breaks KS-EUF-CMA with non-negligible probability. $\mathcal{A}$ acts as the challenger for $\mathcal{A}'$ and as the adversary for EUF-CMA. $\mathcal{C}$ is the challenger for EUF-CMA. The middle portion between dashed lines denotes signature queries.

Message Authentication. The only outbound messages from C_i are signatures σ_i on messages $\tilde{m}_i$. Since the signature scheme is deterministic and the message is sanitized (by the calling protocol), no leakage is possible. □

Remark. *As highlighted in the above proof, requiring the KS-EUF-CMA signature scheme to be deterministic is necessary, as this ensures that (specious) cores cannot reveal any information from the signature itself (e.g., by performing rejection sampling). An alternative solution would consist in using a KS-EUF-CMA signature scheme that admits rerandomization.*

Lemma A.8. *For every adversary $\mathcal{A}$ corrupting any subset of parties maliciously and the firewall of honest parties semi-honestly in an execution of the protocol Π as in Figures 5.7, 5.8, there exists a simulator $\mathcal{S}$ such that, for all environments $\mathcal{Z}$, assuming secure channels between cores and their firewall and a deterministic KS-EUF-CMA signature scheme, the following holds:*

$$\text{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \text{EXEC}_{s\mathcal{F}_{\mathsf{SAT}},\mathcal{S},\mathcal{Z}}$$

Proof (Lemma A.8). The proof runs similarly to the one of (Proof of Theorem 11, [BCL+05]): the simulator $\mathcal{S}$ partitions parties according to the SID they compute, and the unforgeability of the signature scheme prevents the adversary from forcing inconsistent authentication sets among honest parties or injecting arbitrary messages in the channel.

Intuitively, the main differences consist in having (i) the adversary knowing the blinding factor used by the (semi-honest) firewall to sanitize the verification key and to align signatures, and (ii) the presence of additional communication between a core and its firewall. Item (i) is covered by assuming a KS-EUF-CMA signature scheme, so that knowing the blinding factor does not help towards producing a valid forgery. Item (ii) is covered by the assumption of secure channels between a core and its firewall.

For simplicity, we refer to the 2-party setting in which both parties are composed of an honest core and a semi-honest firewall. The simulator internally simulates the behaviour of the honest cores in the link

initialization phase by producing keys vk_i and vk_j and forwarding them to $\mathcal{A}$. The simulator also stores keys $\overline{\mathsf{vk}}_j$, $\overline{\mathsf{vk}}_i$ and SIDs $\widetilde{\mathsf{sid}}_i, \widetilde{\mathsf{sid}}_j$ received from $\mathcal{A}$. Now, the simulator checks whether $(\widetilde{\mathsf{vk}}_i, \overline{\mathsf{vk}}_j) = (\overline{\mathsf{vk}}_i, \widetilde{\mathsf{vk}}_j)$ (*i.e.*, whether the locally-received keys have been blinded consistently), and whether the signatures generated on $\widetilde{\mathsf{sid}}_i$ and $\widetilde{\mathsf{sid}}_j$ verify. If so, it places P_i and P_j in the same authentication set H. In any other case, it places P_i and P_j in different authentication sets.[31]

For what concerns the simulation of communication between firewalls, the strategy is the same as the original proof. The only way the simulation described in [BCL+05] fails in the message authentication phase is by having a RECEIVE message from party P_j to party P_i coming out of the blue from $\mathcal{A}$. This occurs only if a simulated P_i accepts a signature on a message $\tilde{m}$ that was not sent by a simulated P_j. In particular, the discrepancy only occurs by considering two honest cores with semi-honest firewalls, as otherwise the signature would have been generated by $\mathcal{S}$. In this case, if $\mathcal{A}$ forges a signature on behalf of the honest core C_j, it may be used to break KS-EUF-CMA of the signature scheme. Note that we reduce to KS-EUF-CMA security, as $\mathcal{A}$ knows the randomness used to maul the verification key of the (honest) C_j.

Simulating the communication between cores and their firewall is immediate, due to the presence of a secure channel between said components: SEND or RECEIVE messages coming from $s\mathcal{F}_{\mathsf{SAT}}$ (on behalf of dummy components) are forwarded to the appropriate component; SEND from (corrupted) cores or RECEIVE from (corrupted) firewalls messages are forwarded to $s\mathcal{F}_{\mathsf{SAT}}$. □

Proof (Theorem 5.6). The theorem statement follows by looking at the standard corruption translation table. Since both parties are honest, they are either composed of (i) a HONEST core and a SEMIHONEST firewall, or (ii) a SPECIOUS core and a HONEST firewall. Since both firewalls are strongly sanitizing, as shown in Lemma A.7, the core in case (ii) can be considered to be HONEST (Lemma 2 of [CMNV22]). Hence, the statement follows directly by Lemma A.8. We do not consider ISOLATE corruptions explicitly, as they collapse to MITM attacks. □

A.1.5 Proof of Lemma 5.2

Proof (Lemma 5.2). We follow the proof of [BCL+05] adjusting to our setting when necessary. Let $\Phi_{\mathcal{F}}$ be the protocol of [CMNV22] (*i.e.*, the GMW compiler in the srUC framework) realizing functionality $\mathcal{F}$ in the wsrUC model. The structure of $\Phi_{\mathcal{F}}$ can roughly be broken down into two main components: (1) a protocol $\pi_{\hat{\mathcal{F}}_{\mathsf{C\&P}}}$ for realizing a multi-instance commit-and-prove functionality $\hat{\mathcal{F}}_{\mathsf{C\&P}}$ that provides multiple independent commit-and-prove operations making (implicit) use of a single CRS functionality $\mathcal{F}_{\mathsf{crs}}$, and (2) a protocol $\Pi_{\mathcal{F}}$ for realizing functionality $\mathcal{F}$ assuming access to $\hat{\mathcal{F}}_{\mathsf{C\&P}}$. We can express the protocol $\Phi_{\mathcal{F}}$ as $\Phi_{\mathcal{F}} = \Pi_{\mathcal{F}}^{\pi_{\hat{\mathcal{F}}_{\mathsf{C\&P}}}}$. Observe that $\pi_{\hat{\mathcal{F}}_{\mathsf{C\&P}}}$ produces each commit-and-prove operation in a separate subroutine, and the only state that is shared is the access to the CRS. Thus, we conclude that for any n, running n independent instances of $\Phi_{\mathcal{F}}$ where each instance uses a different instance of $\hat{\mathcal{F}}_{\mathsf{C\&P}}$ is exactly the same as running the same n instances of $\Phi_{\mathcal{F}}$ where all instances use a single joint instance of $\hat{\mathcal{F}}_{\mathsf{C\&P}}$. Then, we can invoke the composition theorem of the srUC model and replace the the joint instance of $\hat{\mathcal{F}}_{\mathsf{C\&P}}$ by a single instance of a protocol that realizes $\hat{\mathcal{F}}_{\mathsf{C\&P}}$. We end up with a protocol that realizes n independent instances of $\mathcal{F}$ and uses a single instance of $\mathcal{F}_{\mathsf{crs}}$. □

A.1.6 Proof of Lemma 5.3

Proof (Lemma 5.3). The first important observation is that protocols realizing regular ideal functionalities exclusively allow cores to have an input or output. Given that, the proof is essentially the same as in [BCL+05], with particular care in adapting the communication between components of parties consistently. Towards this goal, we first provide a description of the compiler of [BCL+05] adapted to our setting.

Informally, protocol $\Pi_{\mathcal{F}}$ is described as follows. Each party first runs $s\mathcal{F}_{\mathsf{SAT}}$ to obtain sid_H. Next, it runs $\pi_{\mathcal{F}}$ with SID $= (\mathsf{sid}, \mathsf{sid}_H)$. We recall that, since $\pi_{\mathcal{F}}$ n-realizes $\mathcal{F}$, it internally runs n concurrent instances

[31] This simulation strategy correctly identifies the authentication sets in the 2-party setting. As remarked in the proof of [BCL+05], in the n-party setting, the adversary could force the simulator to place two honest parties that computed different SIDs into the same authentication set. But this may only occur by forging a signature.

of a protocol realizing $\mathcal{F}$, each identified by a different sub-session ID. We refer to each such instance as an "ITI" (Interactive Turing machine Instance). Each outgoing message of $\pi_{\mathcal{F}}$ is forwarded to $s\mathcal{F}_{\mathsf{SAT}}$, and each incoming message from $s\mathcal{F}_{\mathsf{SAT}}$ is forwarded to $\pi_{\mathcal{F}}$. As in [BCL+05], all participants of each instance of $\pi_{\mathcal{F}}$ interact with a single instance of $\mathcal{F}_{\mathsf{SAT}}$, identified by SID $\mathsf{sid}||0$. We proceed more formally in what follows.

Protocol $\Pi_{\mathcal{F}}$

Let $\pi_{\mathcal{F}}$ be a protocol n-realizing functionality $\mathcal{F}$, that has access to $\mathcal{G}$ and assumes authenticated communication (*i.e.*, $\mathcal{F}_{\mathsf{SAT}}$). Protocol $\Pi_{\mathcal{F}}$, with access to $\mathcal{G}$ and $s\mathcal{F}_{\mathsf{SAT}}$, interacts with a set of parties, each composed of a core C_i and a firewall F_i, as follows:

- Upon receiving input $(\textsc{Init}, \mathsf{sid})$ from party P, where $\mathsf{P} = (\mathsf{C}, \mathsf{F})$, $\mathsf{sid} = (\mathcal{P}, \mathsf{sid}')$, and $\mathcal{P}$ is a set of parties containing $\mathcal{P}$, call $s\mathcal{F}_{\mathsf{SAT}}$ with input $(\textsc{Init}, \mathsf{sid}||1)$.
- Upon receiving $(\textsc{Init}, \mathsf{sid}||1, \mathsf{sid}_H)$ from $s\mathcal{F}_{\mathsf{SAT}}$ with destination party P, store $(\mathsf{P}, \mathsf{sid}_H)$ and output $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_H)$ to P.
- Upon receiving $(\textsc{input}, \mathsf{P}_i, \mathsf{sid}, x)$ from core C_i of party P_i, input x to ITI $(\pi_{\mathcal{F}}, (\mathsf{sid}, \mathsf{sid}_H), \mathsf{P}_i)$, *i.e.*, to the ITI running $\pi_{\mathcal{F}}$ with SID $(\mathsf{sid}, \mathsf{sid}_H)$ and PID P_i on C_i.
- Handle communication as follows:
  - When core C_i of ITI $(\pi_{\mathcal{F}}, (\mathsf{sid}, \mathsf{sid}_H), \mathsf{P}_i)$ sends message $(\textsc{Send}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, m)$ using $\mathcal{F}_{\mathsf{SAT}}$ on interface IO_i, input the tuple to $s\mathcal{F}_{\mathsf{SAT}}$ on IO_i.
  - Upon receiving $(\textsc{Send}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, m)$ from $s\mathcal{F}_{\mathsf{SAT}}$ on interface S_i, forward the tuple to the firewall F_i of ITI $(\pi_{\mathcal{F}}, (\mathsf{sid}, \mathsf{sid}_H), \mathsf{P}_i)$ as if coming from interface S_i of $\mathcal{F}_{\mathsf{SAT}}$.
  - When firewall F_i of ITI $(\pi_{\mathcal{F}}, (\mathsf{sid}, \mathsf{sid}_H), \mathsf{P}_i)$ sends message $(\textsc{Send}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, m)$ using $\mathcal{F}_{\mathsf{SAT}}$ on interface S_i, input the tuple to $s\mathcal{F}_{\mathsf{SAT}}$ on S_i.
  - Upon receiving $(\textsc{Receive}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, m)$ from $s\mathcal{F}_{\mathsf{SAT}}$ on interface S_j of party P_j, forward the tuple to firewall F_j of ITI $(\pi_{\mathcal{F}}, (\mathsf{sid}, \mathsf{sid}_H), \mathsf{P}_j)$, as if coming from $\mathcal{F}_{\mathsf{SAT}}$ on interface S_j.
  - When firewall F_j of ITI $(\pi_{\mathcal{F}}, (\mathsf{sid}, \mathsf{sid}_H), \mathsf{P}_j)$ sends message $(\textsc{Receive}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, m)$ using $\mathcal{F}_{\mathsf{SAT}}$ on interface S_j, input the tuple to $s\mathcal{F}_{\mathsf{SAT}}$ on S_j.
  - Upon receiving $(\textsc{Receive}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, m)$ from $s\mathcal{F}_{\mathsf{SAT}}$ on interface IO_j of party P_j, forward $(\textsc{Receive}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j, m)$ to core C_j of ITI $(\pi_{\mathcal{F}}, (\mathsf{sid}, \mathsf{sid}_H), \mathsf{P}_j)$ as if coming from $\mathcal{F}_{\mathsf{SAT}}$ on interface IO_j.
- When core C_i of ITI $(\pi_{\mathcal{F}}, (\mathsf{sid}, \mathsf{sid}_H), \mathsf{P}_i)$ outputs a value y, output y.

Figure A.2: Description of the generic protocol transforming any protocol n-realizing $\mathcal{F}$ assuming $\mathcal{F}_{\mathsf{SAT}}$ into a protocol realizing $s\mathcal{F}$ assuming $s\mathcal{F}_{\mathsf{SAT}}$.

The goal of this proof consists in showing that $\Pi_{\mathcal{F}}$ wsrUC realizes $s\mathcal{F}$, *i.e.*:

$$\textsc{Exec}^{s\mathcal{F}_{\mathsf{SAT}}}_{\Pi_{\mathcal{F}},\mathcal{A},\mathcal{Z}} \approx \textsc{Exec}_{s\mathcal{F},\mathcal{S},\mathcal{Z}}$$

To this extent, we build a series of intermediate experiments relating the unauthenticated execution of protocol $\Pi_{\mathcal{F}}$ to the authenticated concurrent execution.

Intermediate experiment 1. Given an adversary $\mathcal{A}$ that interacts with a single instance of protocol $\Pi_{\mathcal{F}}$ (running on unauthenticated channels), we construct an adversary $\mathcal{A}'$ that interacts with n instances of protocol $\pi_{\mathcal{F}}$ (running on authenticated channels). We first build an experiment $\textsc{Exec}^{\mathcal{F}_{\mathsf{SAT}}}_{\pi_{\mathcal{F}},\mathcal{A}',\mathcal{Z}'}$ for the authenticated concurrent of protocol $\pi_{\mathcal{F}}$, and argue that the following holds:

$$\textsc{Exec}^{s\mathcal{F}_{\mathsf{SAT}}}_{\Pi_{\mathcal{F}},\mathcal{A},\mathcal{Z}} \equiv \textsc{Exec}^{\mathcal{F}_{\mathsf{SAT}}}_{\pi_{\mathcal{F}},\mathcal{A}',\mathcal{Z}'}$$

The environment $\mathcal{Z}'$ internally runs the environment in a black-box manner $\mathcal{Z}$, and the adversary $\mathcal{A}'$ internally runs the adversary in a black-box manner $\mathcal{A}$ (*i.e.*, objects of the unauthenticated execution of protocol $\Pi_{\mathcal{F}}$), emulating $s\mathcal{F}_{\mathsf{SAT}}$ for $\mathcal{Z}$ and $\mathcal{A}$. We proceed to describe $\mathcal{Z}'$ and $\mathcal{A}'$ more formally.

Environment $\mathcal{Z}'$ with black-box access to $\mathcal{Z}$.

Link initialization phase

- Whenever $\mathcal{Z}$ wishes to input $(\text{INIT}, \mathsf{sid})$ to party $\mathsf{P} = (\mathsf{C}, \mathsf{F})$, $\mathcal{Z}'$ forwards $(\text{INTERNAL-INIT}, \mathsf{P}, \mathsf{sid})$ to $\mathcal{A}'$.
- Whenever $\mathcal{Z}'$ receives an output $(\text{INTERNAL-INIT}, \mathsf{sid}, \mathsf{P}, H, \mathsf{sid}_H)$ from $\mathcal{A}'$, store $(\mathsf{P}, \mathsf{sid}, \mathsf{sid}_H, H)$ and forward the tuple to $\mathcal{Z}$ as if it came from P with SID sid.

Computation

- Whenever $\mathcal{Z}$ wishes to input $(\text{INPUT}, \mathsf{sid}, x)$ to party P of SID sid, $\mathcal{Z}'$ checks whether there exists a stored tuple $(\mathsf{P}, \mathsf{sid}, \mathsf{sid}_H, H)$. If so, $\mathcal{Z}'$ inputs x to party P of SID $(\mathsf{sid}, \mathsf{sid}_H)$ of $\pi_{\mathcal{F}}$. Otherwise, $\mathcal{Z}'$ ignores the message.
- Whenever $\mathcal{Z}$ wishes to input any other value to P, $\mathcal{Z}'$ ignores the message.
- Whenever $\mathcal{Z}$ wishes to input any value to $\mathcal{A}$, $\mathcal{Z}$ forwards this message to $\mathcal{A}'$.
- Whenever $\mathcal{Z}'$ receives any message from $\mathcal{A}'$, it forwards the message to $\mathcal{Z}$.
- Whenever $\mathcal{Z}'$ receives an output from party P with SID $(\mathsf{sid}, \mathsf{sid}_H)$, the message is forwarded to $\mathcal{Z}$ as if it came from P with SID sid.
- Whenever $\mathcal{Z}$ halts outputting z, $\mathcal{Z}'$ also halts outputting z.

Figure A.3: Description of the environment $\mathcal{Z}'$ interacting with n concurrent runs of protocol $\pi_{\mathcal{F}}$.

Essentially, $\mathcal{Z}'$ coordinates with $\mathcal{A}'$ in order to initialize the session IDs and sub-session IDs of parties in the authenticated protocol run consistently with the behaviour mandated by $\mathcal{A}$ in the unauthenticated protocol run. Note that, since we only restrict our attention to static corruptions, no adaptive corruption message is present. If a party is corrupted in $\Pi_{\mathcal{F}}$ (according to the corruption translation table in Table 4.1) before the protocol run, the party will also be corrupted in the respective run of $\pi_{\mathcal{F}}$. This implicitly includes also specious corruptions of cores, as per Table 4.1. In particular, if the environment $\mathcal{Z}$ is specious (*i.e.*, it forwards a SPECIOUS instruction to the adversary as the first thing it does), $\mathcal{Z}'$ is also specious, as it simply forwards the SPECIOUS instruction received from $\mathcal{Z}$ to $\mathcal{A}'$ as the first thing it does.

Adversary $\mathcal{A}'$ with black-box access to $\mathcal{A}$.

Link initialization phase

- Whenever $\mathcal{A}'$ receives $(\text{INTERNAL-INIT}, \mathsf{P}, \mathsf{sid})$ from $\mathcal{Z}'$, $\mathcal{A}'$ initializes an instance of $s\mathcal{F}_{\mathsf{SAT}}$ with input $(\text{INIT}, \mathsf{sid}||1)$ as if coming from party $(\mathsf{P}, \mathsf{sid})$. Messages from $s\mathcal{F}_{\mathsf{SAT}}$ to the adversary are delivered to $\mathcal{A}$.
- Whenever $\mathcal{A}$ outputs $(\text{INIT}, \mathsf{sid}||1, \mathsf{P}, H, \mathsf{sid}_H)$ to $s\mathcal{F}_{\mathsf{SAT}}$, $\mathcal{A}'$ forwards the tuple to the internal instance of $s\mathcal{F}_{\mathsf{SAT}}$.
- Whenever the internal instance of $s\mathcal{F}_{\mathsf{SAT}}$ outputs $(\text{INIT}, \mathsf{sid}||1, H, \mathsf{sid}_H)$ to P, $\mathcal{A}'$ outputs $(\text{INTERNAL-INIT}, \mathsf{sid}, \mathsf{P}, H, \mathsf{sid}_H)$ to $\mathcal{Z}'$.

Inbound communication from $\mathcal{F}_{\mathsf{SAT}}$

- Whenever $\mathcal{A}'$ receives a message $(\text{SEND}, \mathsf{P}_i, \mathsf{P}_j, m)$ from $\mathcal{F}_{\mathsf{SAT}}$ with destination F_i, it inputs $(\text{SEND}, \mathsf{sid}||1, \mathsf{P}_i, \mathsf{P}_j, m)$ to $s\mathcal{F}_{\mathsf{SAT}}$ as if coming from C_i.
- Whenever $\mathcal{A}'$ receives a message $(\text{DELIVER}, (\text{SEND}, \mathsf{sid}||0, \mathsf{P}_i, \mathsf{P}_j, m))$ from $\mathcal{F}_{\mathsf{SAT}}$ (*i.e.*, the tuple that is leaked to the adversary), it inputs $(\text{SEND}, \mathsf{sid}||1, \mathsf{P}_i, \mathsf{P}_j, m)$ to $s\mathcal{F}_{\mathsf{SAT}}$ as if coming from F_i.
- Whenever $\mathcal{A}'$ receives a message $(\text{RECEIVE}, \mathsf{P}_i, \mathsf{P}_j, m)$ from $\mathcal{F}_{\mathsf{SAT}}$ with destination C_j, it inputs $(\text{RECEIVE}, \mathsf{sid}||1, \mathsf{P}_i, \mathsf{P}_j, m)$ to $s\mathcal{F}_{\mathsf{SAT}}$ as if coming from F_j.

Outbound communication from $\mathcal{A}$

- Whenever $\mathcal{A}$ wishes to send $(\text{SEND}, \mathsf{P}_i, \mathsf{P}_j, m)$ to $s\mathcal{F}_{\mathsf{SAT}}$ on IO_i, $\mathcal{A}'$ forwards the tuple to $s\mathcal{F}_{\mathsf{SAT}}$ on IO_i. Upon receiving $(\text{SEND}, \mathsf{P}_i, \mathsf{P}_j, m)$ from $s\mathcal{F}_{\mathsf{SAT}}$ with destination (uncorrupted) F_i, $\mathcal{A}'$ forwards the message $(\text{SEND}, \mathsf{P}_i, \mathsf{P}_j, m)$ to $\mathcal{F}_{\mathsf{SAT}}$ on IO_i.[a]
- Whenever $\mathcal{A}$ wishes to send $(\text{DELIVER}, (\text{SEND}, \mathsf{sid}||1, \mathsf{P}_i, \mathsf{P}_j, m))$ to $s\mathcal{F}_{\mathsf{SAT}}$, $\mathcal{A}'$ forwards the tuple to $s\mathcal{F}_{\mathsf{SAT}}$. Whenever $s\mathcal{F}_{\mathsf{SAT}}$ outputs $(\text{RECEIVE}, \mathsf{sid}||1, \mathsf{P}_i, \mathsf{P}_j, m)$ with destination (uncorrupted) F_j, $\mathcal{A}'$ sends $(\text{DELIVER}, (\text{SEND}, \mathsf{sid}||0, \mathsf{P}_i, \mathsf{P}_j, m))$ to $\mathcal{F}_{\mathsf{SAT}}$.

- Whenever $\mathcal{A}$ wishes to send $(\text{RECEIVE}, \mathsf{P}_i, \mathsf{P}_j, m)$ to $s\mathcal{F}_{\mathsf{SAT}}$ on S_j, $\mathcal{A}'$ forwards the tuple to $s\mathcal{F}_{\mathsf{SAT}}$ on S_j. Upon receiving $(\text{RECEIVE}, \mathsf{P}_i, \mathsf{P}_j, m)$ from $s\mathcal{F}_{\mathsf{SAT}}$ with destination (uncorrupted) C_j, $\mathcal{A}'$ forwards the message $(\text{RECEIVE}, \mathsf{P}_i, \mathsf{P}_j, m)$ to $\mathcal{F}_{\mathsf{SAT}}$ on S_j.
- Whenever $\mathcal{A}$ wishes to deliver any other message to an uncorrupted component, the message is ignored.

[a]Note that if F_i is corrupted, $s\mathcal{F}_{\mathsf{SAT}}$ already internally returns this tuple to $\mathcal{A}$.

Figure A.4: Description of the adversary $\mathcal{A}'$ interacting with n concurrent runs of protocol $\pi_{\mathcal{F}}$.

Intuitively, the fact that $\mathcal{A}'$ is receiving inbound messages from $\mathcal{F}_{\mathsf{SAT}}$ means that some honest dummy component has sent them on $\mathcal{F}_{\mathsf{SAT}}$ with destination a corrupted component. For that, $\mathcal{A}'$ reproduces the query made by the honest component by making a corresponding query to $s\mathcal{F}_{\mathsf{SAT}}$. Conversely, $\mathcal{A}$ may want to deliver messages to uncorrupted components. For that, $\mathcal{A}'$ first inputs them to the local instance of $s\mathcal{F}_{\mathsf{SAT}}$. If $s\mathcal{F}_{\mathsf{SAT}}$ returns a matching tuple to $\mathcal{A}'$, it means that it has as destination an uncorrupted component. In this case, the tuple is forwarded on $\mathcal{F}_{\mathsf{SAT}}$ on the appropriate interface[32].

It directly follows from the construction of $\mathcal{A}'$ and $\mathcal{Z}'$ that the view of $\mathcal{Z}$ running within $\mathcal{Z}'$ is distributed identically to the view of $\mathcal{Z}$ in the execution of $\Pi_{\mathcal{F}}$.

Intermediate experiment 2. Since $\pi_{\mathcal{F}}$ wsrUC-securely n-realizes $\mathcal{F}$, we have that for every adversary $\mathcal{A}'$ there exists a simulator $\mathcal{S}'$ such that, for every environment $\mathcal{Z}'$ that opens at most a single session of $\mathcal{F}$, the following holds:

$$\text{EXEC}^{\mathcal{F}_{\mathsf{SAT}}}_{\pi_{\mathcal{F}},\mathcal{A}',\mathcal{Z}'} \overset{c}{\approx} \text{EXEC}_{\mathcal{F},\mathcal{S}',\mathcal{Z}'}$$

The descriptions of $\mathcal{A}'$ and $\mathcal{Z}'$ are defined above.

Intermediate experiment 3. Finally, we construct simulator $\mathcal{S}$ from $\mathcal{S}'$. In particular, we prove that:

$$\text{EXEC}_{\mathcal{F},\mathcal{S}',\mathcal{Z}'} \overset{s}{\approx} \text{EXEC}_{s\mathcal{F},\mathcal{S},\mathcal{Z}}$$

Essentially $\mathcal{S}$ internally incorporates $\mathcal{S}'$, and mimics for $\mathcal{S}'$ the interaction with environment $\mathcal{Z}'$ by using environment $\mathcal{Z}$. Intuitively, the goal of $\mathcal{S}$ consists of (i) initializing $\mathcal{S}'$ consistently with $s\mathcal{F}_{\mathsf{SAT}}$, (ii) forward outbound messages with destination the concurrent execution of $\mathcal{F}$ with SSID sid_H to $s\mathcal{F}_{\mathsf{SAT}}$, and (iii) forward inbound messages from $s\mathcal{F}_{\mathsf{SAT}}$ to corrupted parties as if coming from $\mathcal{F}$. We describe $\mathcal{S}$ formally next.

Simulator $\mathcal{S}$ with black-box access to $\mathcal{S}'$.

- Whenever $\mathcal{S}$ receives a message $(\text{INIT}, \mathsf{sid}, \mathsf{P})$ from $s\mathcal{F}$, it forwards $(\text{INTERNAL-INIT}, \mathsf{sid}, \mathsf{P})$ to $\mathcal{S}'$ as if it came from $\mathcal{Z}'$.
- Whenever $\mathcal{S}'$ wishes to send $(\text{INTERNAL-INIT}, \mathsf{sid}, \mathsf{P}, H, \mathsf{sid}_H)$ to $\mathcal{Z}'$, $\mathcal{S}$ forwards $(\text{INIT}, \mathsf{sid}, \mathsf{P}, H, \mathsf{sid}_H)$ to $s\mathcal{F}$.
- Whenever $\mathcal{S}'$ wishes to send input x to the instance of $\mathcal{F}$ with SID $(\mathsf{sid}, \mathsf{sid}_H)$, $\mathcal{S}$ sends $(\text{INPUT}, \mathsf{sid}, H, x)$ to $s\mathcal{F}$ (the set H associated with sid_H can be determined from the content of the respective INTERNAL-INIT message received from $\mathcal{S}'$).
- Upon receiving from $s\mathcal{F}$ an output v directed at party P in instance H, $\mathcal{S}$ forwards to $\mathcal{S}'$ an output v for party P as if coming from $\mathcal{F}$ (in this case, P is corrupted, according to the corruption translation table in Table 4.1).

Figure A.5: Description of the simulator $\mathcal{S}$ interacting with ideal functionality $s\mathcal{F}$.

As in the original proof, the only difference is that $s\mathcal{F}$ might ignore certain inputs and not forward them to the respective instance of $\mathcal{F}$. This happens only if any of the following occurs:

- Upon receiving a message from an uncorrupted party that is not part of any H.

[32]Otherwise, it means that the communication is among corrupted components and does not need to be forwarded to $\mathcal{F}_{\mathsf{SAT}}$, as it is already locally emulated.

- Upon receiving a message $(\text{INIT}, \mathsf{sid}, \mathsf{P}_i, H, \mathsf{sid}_H)$ from $\mathcal{S}$ that is malformed. This happens if:
  - P_i has not previously sent any $(\text{INIT}, \mathsf{sid})$ message.
  - P_i is not part of set H.
  - $s\mathcal{F}$ has previously received another pair $(\mathsf{sid}_{H'}, H')$ such that either $H = H'$ and $H \cap H'$ contains uncorrupted parties, or $H = H'$ but $\mathsf{sid}_H \neq \mathsf{sid}_{H'}$.

The first event never occurs, as these inputs are also ignored in the execution of $\mathcal{Z}'$ and $\mathcal{S}'$ with (the multi-session) $\mathcal{F}$. In particular, $\mathcal{Z}'$ ignores all inputs that $\mathcal{Z}$ wishes to send to parties P_i that are not of a previously-recorded set H.

We analyze the second event next. By construction of $\mathcal{A}'$, $\mathcal{A}'$ never outputs a malformed message $(\text{INTERNAL-INIT}, \mathsf{sid}, \mathsf{P}, H, \mathsf{sid}_H)$ to $\mathcal{Z}'$: indeed, $\mathcal{A}'$ stores a tuple (sid_H, H) only if $s\mathcal{F}_{\mathsf{SAT}}$ would have done so, which implies that only well-formed INTERNAL-INIT messages are output to $\mathcal{Z}'$. Since for every $\mathcal{A}'$ there exists a valid simulator $\mathcal{S}'$, $\mathcal{S}'$ will output a malformed INTERNAL-INIT message with negligible probability. By construction of $\mathcal{S}$, the probability that $\mathcal{S}$ sends a malformed $(\text{INIT}, \mathsf{sid}, P, H, \mathsf{sid}_H)$ message to $s\mathcal{F}$ is negligible (as this occurs only if $\mathcal{S}'$ outputs an INTERNAL-INIT that is malformed). Since the execution of $\mathcal{S}'$ and $\mathcal{Z}'$ interacting with a multi-session $\mathcal{F}$ is otherwise identically distributed to an execution of $\mathcal{S}$ and $\mathcal{Z}$ interacting with $s\mathcal{F}$, the two executions are statistically indistinguishable, concluding the experiment.

Putting everything together. We conclude the proof by putting the experiments together:

$$\text{EXEC}^{s\mathcal{F}_{\mathsf{SAT}}}_{\Pi_{\mathcal{F}},\mathcal{A},\mathcal{Z}} \equiv \text{EXEC}^{\mathcal{F}_{\mathsf{SAT}}}_{\pi_{\mathcal{F}},\mathcal{A}',\mathcal{Z}'} \overset{c}{\approx} \text{EXEC}_{\mathcal{F},\mathcal{S}',\mathcal{Z}'} \overset{s}{\approx} \text{EXEC}_{s\mathcal{F},\mathcal{S},\mathcal{Z}}$$

□

A.1.7 Proof of Theorem 5.8

Proof (Theorem 5.8). We adapt the proof of [DHP+18, Appendix D] to the $\mathcal{F}_{\mathsf{RE}}$ and $\mathcal{F}_{\mathsf{PAKE}}$ reported in Figures 4.8, 4.10. First, we define a trivial protocol[33] Π_{PAKE} that instantiates $\mathcal{F}_{\mathsf{PAKE}}$ in the $s\mathcal{F}_{\mathsf{RE}}$-hybrid model. Then, we show a one-to-one mapping of the adversarial behaviour within the two functionalities.

Definition and simulation of Π_{PAKE}. Compared to the original proof, in Π_{PAKE} we only distinguish explicitly the initiator I from the responder R to match the description of our ideal functionalities.

Protocol Π_{PAKE}

- Upon receiving $(\mathsf{sid}, \mathsf{pw}^{\mathsf{P}})$ from party $\mathsf{P} \in \{\text{I}, \text{R}\}$, proceed as follows:
  - Send $(\text{INIT}, \mathsf{sid})$ to $s\mathcal{F}_{\mathsf{RE}}$ on behalf of P.
  - If $\mathsf{P} = \text{I}$: Send $(\text{NEWSESSION}, \mathsf{sid}, \text{I}, \text{R}, \mathsf{pw}^{\text{I}})$ to $s\mathcal{F}_{\mathsf{RE}}$.
  - If $\mathsf{P} = \text{R}$: Upon receiving $(\text{WAKEUP}, \mathsf{sid}, \text{I}, \text{R})$ from $s\mathcal{F}_{\mathsf{RE}}$, send $(\text{RESPOND}, \mathsf{sid}, \text{R}, \text{I}, \mathsf{pw}^{\text{R}})$ to $s\mathcal{F}_{\mathsf{RE}}$.
- Upon receiving $\mathsf{skey}_{\mathsf{P}}$ from $s\mathcal{F}_{\mathsf{RE}}$, output $\mathsf{skey}_{\mathsf{P}}$ to P.

Figure A.6: Description of Π_{PAKE} instantiating $\mathcal{F}_{\mathsf{PAKE}}$ in the $s\mathcal{F}_{\mathsf{RE}}$-hybrid model.

Next, we define a simulator $\mathcal{S}_{\mathsf{PAKE}}$ for Π_{PAKE}, similarly to [DHP+18].

Simulator $\mathcal{S}_{\mathsf{PAKE}}$

Initialization

- Upon receiving $(\text{INIT}, \mathsf{sid})$ from $\mathcal{A}$ on behalf of any corrupted party, $\mathcal{S}_{\mathsf{PAKE}}$ does nothing.
- Upon receiving $(\text{INIT}, \mathsf{sid}, \mathsf{P}_i, H, \mathsf{sid}_H)$ from $\mathcal{A}$, $\mathcal{S}_{\mathsf{PAKE}}$ records the partition set H for party P.

[33] We say it is a trivial protocol, as it exclusively makes parties interact with $s\mathcal{F}_{\mathsf{RE}}$.

$\mathcal{A}$ **places** I **and** R **in the same authentication set** H

- If both I and R are corrupted, $\mathcal{S}_{\mathsf{PAKE}}$ internally simulates the behaviour of $s\mathcal{F}_{\mathsf{RE}}$ without interacting with $\mathcal{F}_{\mathsf{PAKE}}$.
- Else (*i.e.*, at most one among I and R are corrupted), $\mathcal{S}_{\mathsf{PAKE}}$ proceeds as follows:
  - Upon receiving $(\textsc{NewSession}, \mathsf{sid}, \mathrm{I}, \mathrm{R}, \mathsf{pw}^{\mathrm{I}})$ from $\mathcal{A}$, where I is corrupted, $\mathcal{S}_{\mathsf{PAKE}}$ forwards the query to $\mathcal{F}_{\mathsf{PAKE}}$.
  - Upon receiving $(\textsc{Respond}, \mathsf{sid}, \mathrm{R}, \mathrm{I}, \mathsf{pw}^{\mathrm{R}})$ from $\mathcal{A}$, where R is corrupted, $\mathcal{S}_{\mathsf{PAKE}}$ forwards the query to $\mathcal{F}_{\mathsf{PAKE}}$.
  - Upon receiving $(\textsc{NewKey}, \mathsf{sid}, \mathsf{P}, K)$ from $\mathcal{A}$, $\mathcal{S}_{\mathsf{PAKE}}$ forwards the query to $\mathcal{F}_{\mathsf{PAKE}}$.
  - Upon receiving $(\mathsf{sid}, \cdot)$ from $\mathcal{F}_{\mathsf{PAKE}}$, $\mathcal{S}_{\mathsf{PAKE}}$ forwards the tuple to $\mathcal{A}$.

$\mathcal{A}$ **places** I **and** R **in different authentication sets**

- If an authentication set contains exclusively corrupted parties, $\mathcal{S}_{\mathsf{PAKE}}$ internally simulates the behaviour of $s\mathcal{F}_{\mathsf{RE}}$ for that authentication set without interacting with $\mathcal{F}_{\mathsf{PAKE}}$.
- Else, $\mathcal{S}_{\mathsf{PAKE}}$ proceeds as follows:
  - Upon receiving $(\textsc{NewSession}, \mathsf{sid}, \mathrm{I}, \mathrm{R}, \mathsf{pw}^{\mathrm{I}})$ from $\mathcal{A}$, where I is corrupted, $\mathcal{S}_{\mathsf{PAKE}}$ forwards $(\textsc{TestPwd}, \mathsf{sid}, \mathrm{R}, w_{\mathrm{I}})$ to $\mathcal{F}_{\mathsf{PAKE}}$.
  - Upon receiving $(\textsc{Respond}, \mathsf{sid}, \mathrm{R}, \mathrm{I}, \mathsf{pw}^{\mathrm{R}})$ from $\mathcal{A}$, where R is corrupted, $\mathcal{S}_{\mathsf{PAKE}}$ forwards $(\textsc{TestPwd}, \mathsf{sid}, \mathrm{I}, \mathsf{pw}^{\mathrm{R}})$ to $\mathcal{F}_{\mathsf{PAKE}}$.
  - Upon receiving $(\textsc{NewKey}, \mathsf{sid}, \mathsf{P}, K)$ from $\mathcal{A}$, $\mathcal{S}_{\mathsf{PAKE}}$ forwards the query to $\mathcal{F}_{\mathsf{PAKE}}$.
  - Upon receiving $(\mathsf{sid}, \cdot)$ from $\mathcal{F}_{\mathsf{PAKE}}$, $\mathcal{S}_{\mathsf{PAKE}}$ forwards the tuple to $\mathcal{A}$.

Figure A.7: Description of a simulator $\mathcal{S}_{\mathsf{PAKE}}$ for Π_{PAKE}.

Whenever $\mathcal{A}$ places both parties in the same authentication set, no TestPwd query can occur, as the parties were allowed to mutually authenticate and the adversary can no longer man-in-the-middle. On the other hand, if $\mathcal{A}$ is splitting parties, an authenticated set can contain at most one honest party (as the intersection of distinct authentication sets can only contain corrupted parties). This way, the adversary is effectively performing two distinct TestPwd queries against honest I and honest R, as they are in distinct authentication and the adversary can impersonate their peer.

Mapping between $s\mathcal{F}_{\mathsf{RE}}$ and $\mathcal{F}_{\mathsf{PAKE}}$. It remains to show that the output of $\mathcal{S}_{\mathsf{PAKE}}$ is correctly distributed. To this extent, we enumerate all possible adversarial behaviours in $s\mathcal{F}_{\mathsf{RE}}$ and map them to the status of records in $\mathcal{F}_{\mathsf{PAKE}}$ in Table A.1. When the adversary "attacks" party P in $s\mathcal{F}_{\mathsf{RE}}$, it first partitions parties in different authentication sets, and then runs the individual $\mathcal{F}_{\mathsf{RE}}$ on behalf of its peer by using an arbitrary password. Intuitively, the attack is successful if the arbitrary password matches with pw^{P}. For all rows, the output of both $s\mathcal{F}_{\mathsf{RE}}$ and $\mathcal{F}_{\mathsf{PAKE}}$ matches by inspecting the behaviour of the functionalities for that specific adversarial behaviour. For the sake of clarity, we discuss the meaning of some of the rows in a verbose fashion.

Row 1. Parties input pw^{I} and pw^{R} such that $\mathsf{pw}^{\mathrm{I}} = \mathsf{pw}^{\mathrm{R}}$, and the adversary does not MITM in $s\mathcal{F}_{\mathsf{RE}}$. This means that the parties are placed within the same authentication set. This behaviour corresponds to both records being fresh in $\mathcal{F}_{\mathsf{PAKE}}$, as no TestPwd queries can occur. If both parties are honest (*i.e.*, $C = \emptyset$), the output of both functionalities is a random shared key. If party R is corrupted, the adversary can set the key it outputs arbitrarily, and since it knows the password used by the honest peer I, it may set the key I outputs to an arbitrary value (as intuitively, it is able to run the protocol on behalf of R with the correct password).

Row 4. Parties input pw^{I} and pw^{R} and the adversary *does* MITM in $s\mathcal{F}_{\mathsf{RE}}$ by isolating parties and impersonating their matching peer. In particular, the adversary supplies password $\widetilde{\mathsf{pw}}^{\mathrm{R}}$ on behalf of R in a run against I, and $\widetilde{\mathsf{pw}}^{\mathrm{I}}$ on behalf of I in a run against R. In this row, $\mathsf{pw}^{\mathrm{I}} = \widetilde{\mathsf{pw}}^{\mathrm{R}}$ and $\widetilde{\mathsf{pw}}^{\mathrm{I}} \neq \mathsf{pw}^{\mathrm{R}}$. This means that the adversary successfully guessed the password of I but failed to guess the password of R. In $\mathcal{F}_{\mathsf{PAKE}}$, this behaviour corresponds to the record of I being $\mathsf{compromised}$ and the record of R being $\mathsf{interrupted}$. Therefore, the adversary can set the key output by I.

$s\mathcal{F}_{\mathsf{RE}}$			$\mathcal{F}_{\mathsf{PAKE}}$		Output of $s\mathcal{F}_{\mathsf{RE}}$ and $\mathcal{F}_{\mathsf{PAKE}}$		
No MITM	Attack I	Attack R	Record of I	Record of R	$C = \emptyset$	$C = \{\mathrm{R}\}$	$C = \{\mathrm{I}, \mathrm{R}\}$
$\mathsf{pw}^{\mathrm{I}} = \mathsf{pw}^{\mathrm{R}}$			fresh	fresh	r, r	K, K'	K, K'
$\mathsf{pw}^{\mathrm{I}} \neq \mathsf{pw}^{\mathrm{R}}$			fresh	fresh	r, s	r, K'	K, K'
	$\mathsf{pw}^{\mathrm{I}} = \widetilde{\mathsf{pw}}^{\mathrm{R}}$	$\widetilde{\mathsf{pw}}^{\mathrm{I}} = \mathsf{pw}^{\mathrm{R}}$	compromised	compromised	K, K'	K, K'	K, K'
	$\mathsf{pw}^{\mathrm{I}} = \widetilde{\mathsf{pw}}^{\mathrm{R}}$	$\widetilde{\mathsf{pw}}^{\mathrm{I}} \neq \mathsf{pw}^{\mathrm{R}}$	compromised	interrupted	K, s	K, K'	K, K'
	$\mathsf{pw}^{\mathrm{I}} = \widetilde{\mathsf{pw}}^{\mathrm{R}}$		compromised	fresh	K, s	K, K'	K, K'
	$\mathsf{pw}^{\mathrm{I}} \neq \widetilde{\mathsf{pw}}^{\mathrm{R}}$	$\widetilde{\mathsf{pw}}^{\mathrm{I}} = \mathsf{pw}^{\mathrm{R}}$	interrupted	compromised	r, K'	r, K'	K, K'
	$\mathsf{pw}^{\mathrm{I}} \neq \widetilde{\mathsf{pw}}^{\mathrm{R}}$	$\widetilde{\mathsf{pw}}^{\mathrm{I}} \neq \mathsf{pw}^{\mathrm{R}}$	interrupted	interrupted	r, s	r, K'	K, K'
	$\mathsf{pw}^{\mathrm{I}} \neq \widetilde{\mathsf{pw}}^{\mathrm{R}}$		interrupted	fresh	r, s	r, K'	K, K'
		$\widetilde{\mathsf{pw}}^{\mathrm{I}} = \mathsf{pw}^{\mathrm{R}}$	fresh	compromised	r, K'	r, K'	K, K'
		$\widetilde{\mathsf{pw}}^{\mathrm{I}} \neq \mathsf{pw}^{\mathrm{R}}$	fresh	interrupted	r, s	r, K'	K, K'

Table A.1: Mapping between the adversarial behaviour in $s\mathcal{F}_{\mathsf{RE}}$ and in $\mathcal{F}_{\mathsf{PAKE}}$. C is the set of corrupted parties. r, s are randomly generated keys. K, K' are adversarially generated keys. Passwords denoted as pw^{P} are inputs to the protocol. Passwords denoted as $\widetilde{\mathsf{pw}}^{\mathsf{P}}$ are supplied by the adversary while performing MITM. The column for $C = \{\mathrm{I}\}$ is omitted, as it is symmetrical with respect to the column for $C = \{\mathrm{R}\}$.

□

A.2 Discussing input-triggering attacks in the protocols of [CMMV24]

In Section 4.3.6 we observed that, even though the protocols described in [CMMV24] attempt to sanitize inbound messages to ensure resiliency against input-triggering attacks, the srUC framework does not formally account for this threat model. In this section, we review the protocols of [CMMV24], highlighting formal weaknesses and proposing informal correcting measures.

A.2.1 Sanitizable Oblivious Transfer

Defining oblivious transfer while accounting for input triggers (in addition to specious subversions) is a tricky task, as the (non-sanitized) functionality would allow a (specious) receiver to obtain exactly one of the inputs of the sender, which may act as a trigger if sampled maliciously. Similarly, it would allow a (specious) sender to sample the inputs in a leaky manner and send them over to a corrupted party. For that, *both* firewalls should be allowed to blind the sender's inputs. This way, the sender's firewall can sanitize the sender's randomly chosen inputs, and the receiver's firewall can sanitize the inbound inputs. The result is depicted in Figure A.8.

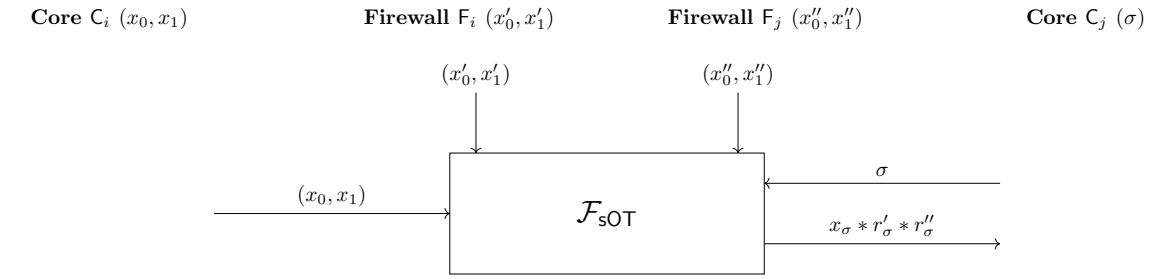


Figure A.8: The sanitizable OT functionality $\mathcal{F}_{\mathsf{sOT}}$ presented in [CMMV24], with $*$ being a blinding operation for the input domain.

Note that, in contrast to the $\mathcal{F}_{\mathsf{sOT}}$ presented in Section 5, this functionality does no longer require the sender's firewall to return its blinding factors to the respective core. Indeed, these blinding factors would now be insufficient to ensure that both parties have a consistent view of the protocol run, given that the blinding factors used by the firewall of the receiver to sanitize inbound messages are unknown to the sender.

A formal description. In what follows, we provide a description of the $\mathcal{F}_{\mathsf{sOT}}$ functionality presented in [CMMV24].

Functionality $\mathcal{F}_{\mathsf{sOT}}$

$\mathcal{F}_{\mathsf{sOT}}$ is a sanitizable ideal functionality that interacts with the sender $\mathrm{S} = (\mathsf{C}_\mathrm{S}, \mathsf{F}_\mathrm{S})$ and the receiver $\mathrm{R} = (\mathsf{C}_\mathrm{R}, \mathsf{F}_\mathrm{R})$, parameterized by input domain $\mathcal{I} \subseteq \{0,1\}^n$ and a blinding operation $* : \mathcal{I}^2 \to \mathcal{I}$.

Interface IO_i:

Upon receiving a query $(\textsc{sender}, \mathsf{sid}, (x_0, x_1))$ **from** C_S **on** IO_S**:**

Record $(\textsc{sender}, \mathsf{sid}, (x_0, x_1))$ and forward the tuple on S_i. Ignore subsequent commands of the form $(\textsc{sender}, \mathsf{sid}, \cdot)$.

Upon receiving a query $(\textsc{receiver}, \mathsf{sid}, \sigma)$ **from** C_R **on** IO_R**:**

Check if a record $(\textsc{sender}, \mathsf{sid}, (\hat{x}_0, \hat{x}_1))$ exists. If this is the case, check the following:

- The message $(\textsc{blind}, \mathsf{sid}, \cdot)$ was sent to $\mathcal{F}_{\mathsf{sOT}}$ on S_S. If S is malicious according to the corruption translation table, mark this check as passed.
- The message $(\textsc{blind}, \mathsf{sid}, \cdot)$ was sent to $\mathcal{F}_{\mathsf{sOT}}$ on S_R. If R is malicious according to the corruption translation table, mark this check as passed.

If the conditions above hold, output $(\mathsf{sid}, \hat{x}_\sigma)$ to R, sid to the adversary $\mathcal{S}$, and halt. Otherwise, send nothing to R but continue running.

Interface S_i:

Upon receiving a query $(\textsc{blind}, \mathsf{sid}, (x'_0, x'_1))$ **from** F_S **on** S_S**:**

If S is malicious according to the corruption translation table, do nothing. Otherwise, check if a record $(\textsc{sender}, \mathsf{sid}, (x_0, x_1))$ exists. If so, update the tuple to $(\textsc{sender}, \mathsf{sid}, (\tilde{x}_0, \tilde{x}_1))$, with $\tilde{x}_b = x_b * x'_b$. Otherwise, do nothing. Ignore future commands of the form $(\textsc{blind}, \mathsf{sid}, \cdot)$ on S_S.

Upon receiving a query $(\textsc{blind}, \mathsf{sid}, (x''_0, x''_1))$ **from** F_R **on** S_R**:**

If R is malicious according to the corruption translation table, do nothing. Otherwise, check the following:

- A record $(\textsc{sender}, \mathsf{sid}, (\tilde{x}_0, \tilde{x}_1))$ exists.
- A message $(\textsc{blind}, \mathsf{sid}, \cdot)$ was sent to $\mathcal{F}_{\mathsf{sOT}}$ on S_S. If S is malicious according to the corruption translation table, mark this check as passed.

If the conditions above hold, update the tuple to $(\textsc{sender}, \mathsf{sid}, (\hat{x}_0, \hat{x}_1))$, with $\hat{x}_b = \tilde{x}_b * x''_b$. Otherwise, do nothing. Ignore future commands of the form $(\textsc{blind}, \mathsf{sid}, \cdot)$ on S_R.

Figure A.9: Description of the sanitizable oblivious transfer functionality $\mathcal{F}_{\mathsf{sOT}}$ of [CMMV24].

Discussion. In describing our revised $\mathcal{F}_{\mathsf{sOT}}$, we introduce a different formalization technique compared to the sanitizable functionalities defined in the seminal framework. Namely, $\mathcal{F}_{\mathsf{sOT}}$ allows each firewall to explicitly contribute to the sanitation, unless the overall party related to that firewall is malicious. From a formal standpoint this is allowed, as there exists a corruption translation table that maps corruptions of individual components of a party to a corruption for the entire party, and currently the srUC framework only supports static corruptions, so the functionality knows in advance which parties are corrupted. This also makes sense for what concerns simulation: once we have mapped the behaviour of components to that of a malicious party, we should not simulate anything that occurs *within* that malicious party. As an example, while handling a malicious sender in a protocol realizing $\mathcal{F}_{\mathsf{sOT}}$, it suffices for the simulator to only forward to $\mathcal{F}_{\mathsf{sOT}}$ the malicious sender's input messages.

Instantiating $\mathcal{F}_{\mathsf{sOT}}$. In light of the above observations, instantiating the revised $\mathcal{F}_{\mathsf{sOT}}$ can be achieved by slightly modifying our instantiation presented in Figure 5.4. In particular, both firewalls now introduce a layer of sanitation by sanitizing the public key as well as the ciphertexts; hence, sanitizing all inbound and outbound messages with independent random strings. We describe the resulting protocol in Figure A.10, which matches the result presented in [CMMV24].

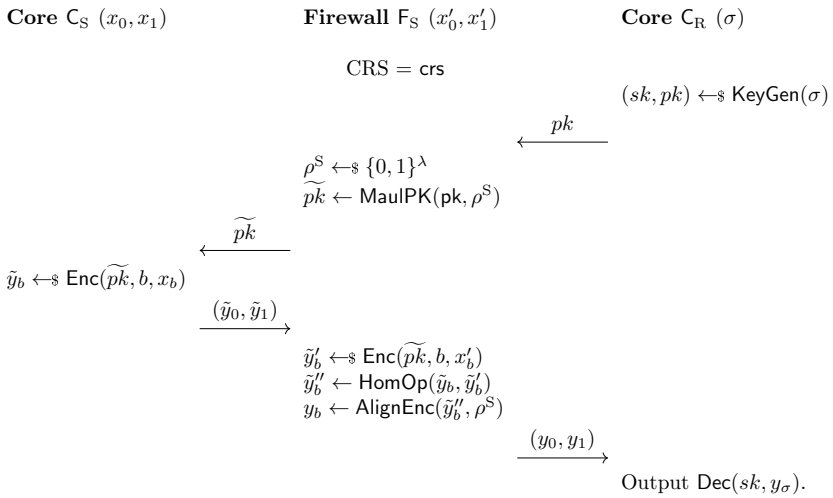


Figure A.10: The sanitizing protocol for $\mathcal{F}_{\mathsf{sOT}}$ of [CMMV24]. The firewall of the receiver is omitted, as it runs the same code as F_S.

The simulation is quite similar to what we presented in Theorem 5.2, and crucially leverages on the fact that the simulation of the firewalls of malicious parties is disregarded by $\mathcal{F}_{\mathsf{sOT}}$. More specifically: in simulating against a malicious receiver, the CRS is generated in messy mode. However, the simulator would also have to extract the blinding factors selected by the adversary on behalf of the malicious receiver's firewall - something that may be achieved only by having the CRS being generated in decryption mode. By our description of $\mathcal{F}_{\mathsf{sOT}}$, extracting these blinding factors is not required.

A.2.2 Randomized Equality

With our revised protocol for $\mathcal{F}_{\mathsf{sOT}}$, we are ready to consider the instantiation of $\mathcal{F}_{\mathsf{RE}}$ of [CMMV24], which is depicted in Figure A.11. In spite of the two parties being unable to agree on the same view of a protocol run, the trick to preserve correctness leverages the symmetrical structure of the protocol: namely, random strings used for the i-th OT in which a core acts as the sender are re-used for the i-th OT in which the same core acts as the receiver.

A security re-analysis. As remarked throughout the proof of [CMMV24], this protocol successfully sanitizes all inbound and outbound messages as long as the adversary does *not* successfully guess the password. While one may argue that this security is still meaningful in the context of PAKE, given that a successful adversarial password guess jeopardizes any security that would be provided by the functionality, we cannot claim that the protocol fully adheres with the formalism of the srUC framework. Concretely, a correct password guess against a party P should only imply that the adversary can impersonate P's peer, and not that P also exfiltrates information or triggers. We discuss the attack in what follows, alongside a potential fix.

Attacking a specious responder. Let us consider a malicious initiator, and let us assume that both parties use pw as their input password. A specious responder may sample $(r_0^\mathrm{R}, r_1^\mathrm{R})$ in a biased manner, but its firewall

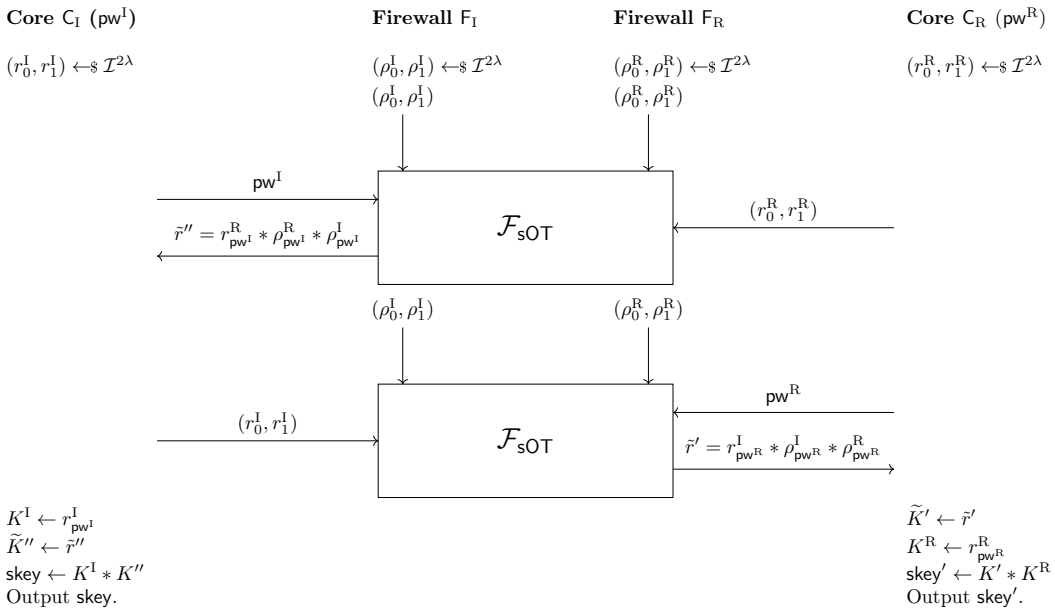


Figure A.11: The sanitizing protocol for $\mathcal{F}_{\mathsf{RE}}$ of [CMMV24].

correctly sanitizes these values with uniformly random values $(\rho_0^\mathrm{R}, \rho_1^\mathrm{R})$ that were never used before. This means that a specious receiver is secure against leakage of information. Note however that the adversary is allowed to control the (non-sanitized) values $(r_0^\mathrm{R}, r_1^\mathrm{R})$; hence, upon receiving $\tilde{r}'' = r^\mathrm{R}_\mathsf{pw} * \rho^\mathrm{R}_\mathsf{pw}$ from the firewall of the responder, the adversary can retrieve $\rho^\mathrm{R}_\mathsf{pw}$ and compute $(\rho^\mathrm{R}_\mathsf{pw})^{-1}$. Next, we show that this information is sufficient to mount an input-triggering attack.

In the second $\mathcal{F}_{\mathsf{sOT}}$, the (malicious) initiator may craft r^I_pw in such a way that the specious core of the receiver computes an adversarially-known key K which causes a trigger ($r^\mathrm{I}_{1-\mathsf{pw}}$ is not relevant to this discussion, and it may be any arbitrary value). To do so, the (malicious) initiator computes $r^\mathrm{I}_\mathsf{pw} \leftarrow a*(\rho^\mathrm{R}_\mathsf{pw})^{-1}$, where a is part of the auxiliary information revealed to the adversary in the setup phase of the specious responder. In particular, the specious core of the responder is designed to crash upon receiving a. Now, whenever the core of the responder queries the second $\mathcal{F}_{\mathsf{sOT}}$, it will retrieve the value $\tilde{r}' = r^\mathrm{I}_\mathsf{pw} * \rho^\mathrm{R}_\mathsf{pw} = \rho^\mathrm{R}_\mathsf{pw} * a * (\rho^\mathrm{R}_\mathsf{pw})^{-1} = a$, causing it to crash.

With the above attack, it is always trivially possible for an environment $\mathcal{Z}$ to win the strong sanitation game. Indeed, a specious core equipped with an honest firewall would crash, whereas an honest core equipped with an honest firewall would output the adversarially-chosen key K.

Attacking a specious initiator. With a similar strategy, one can show that it is possible to exfiltrate information from a specious initiator. In this case, the adversary knows in advance some value a that is part of the auxiliary information stemming from the setup of the specious core. Now, the (malicious) responder inputs to the first $\mathcal{F}_{\mathsf{sOT}}$ the value $r^\mathrm{R}_\mathsf{pw} = a^{-1}$. This way, whenever the (specious) core of the initiator receives $\tilde{r}' = r^\mathrm{R}_\mathsf{pw} * \rho^\mathrm{I}_\mathsf{pw}$, it may retrieve the blinding factor $\rho^\mathrm{I}_\mathsf{pw}$ that was used by its (honest) firewall to sanitize the inbound message. With that, the specious core can evaluate $r^\mathrm{R}_\mathsf{pw} \leftarrow e * (\rho^\mathrm{I}_\mathsf{pw})^{-1}$, where e is a value obtained by running any arbitrary undetectable exfiltration strategy (*e.g.*, the one described in Section 1.3). This information is immediately retrieved by the adversary upon receiving $\tilde{r}'' = \rho^\mathrm{I}_\mathsf{pw} * r^\mathrm{R}_\mathsf{pw} = \rho^\mathrm{I}_\mathsf{pw} * e * (\rho^\mathrm{I}_\mathsf{pw})^{-1} = e$.

A potential fix. The fundamental reason behind the aforementioned attacks lies in (i) having a firewall reusing the same randomness twice, and (ii) the fact that the adversary is able to determine in advance the randomness that will be used by the specious core's (honest) firewall by observing an inbound protocol

message. This shortcoming can be remedied by introducing commitments with malleability and rerandomizability properties in a similar fashion to [LWC+25]. Concretely, core C_R would commit to its two local random strings $(r_0^\mathrm{R}, r_1^\mathrm{R})$ and send the commitments to C_I via $\mathcal{F}_{\mathsf{sOT}}$. Each core's firewall, in turn, leverages the rerandomizability and malleability properties of the commitment scheme to sanitize the commitments, as well as the respective committed messages. Therefore, C_I retrieves a *commitment* to $r_{\mathsf{pw}}^\mathrm{R} * \rho_{\mathsf{pw}}^\mathrm{R} * \rho_{\mathsf{pw}}^\mathrm{I}$ instead of the respective plaintext. Hence, a specious C_I can no longer compute its random strings $(r_0^\mathrm{I}, r_1^\mathrm{I})$ as a function of its firewall's randomness. Of course, C_R should open the commitment once it has received $r_{\mathsf{pw}}^\mathrm{I}$ from the second run of $\mathcal{F}_{\mathsf{sOT}}$ to ensure that the protocol can run to completion.

A.2.3 Sanitizable Split-Authenticated communication

Finally, we consider the protocol that realizes $s\mathcal{F}_{\mathsf{SAT}}$ presented in [CMMV24]. In particular, this protocol's link initialization phase relies on the determination of session IDs via (identifying) verification keys of parties, which are sanitized by firewalls in different directions by using the same randomness (to ensure correctness). For instance, in the 2-party setting, core C_i has access to vk_i and $\widetilde{\mathsf{vk}}_j$, and core C_j has access to $\widetilde{\mathsf{vk}}_i$ and vk_j, with $\widetilde{\mathsf{vk}}_i$, $\widetilde{\mathsf{vk}}_j$ being appropriate sanitations of vk_i, vk_j using the same randomness ρ_i. For this reason, in [CMMV24] we additionally define a generic algorithm that allows to combine these keys either way to output the same value, which may be used as a session ID.

Definition A.1 (Consistent identity combinability)**.** *A key-sanitizable signature scheme has consistent identity combinability if it supports an algorithm* IDComb *with the following property:*

$$\mathsf{IDComb}(\mathsf{vk}_i, \mathsf{MaulVK}(\mathsf{vk}_j, \rho)) = \mathsf{IDComb}(\mathsf{MaulVK}(\mathsf{vk}_i, \rho), \mathsf{vk}_j).$$

The BLS signature scheme can be trivially extended to capture this property as follows:

$$\mathsf{IDComb}(\mathsf{vk}_i, \mathsf{vk}_j) = \hat{e}(\mathsf{vk}_i, \mathsf{vk}_j).$$

Proving this new property follows by inspection:

$$\begin{aligned}\mathsf{IDComb}(\mathsf{vk}_i, \mathsf{MaulVK}(\mathsf{vk}_j, \rho)) &= \hat{e}(g^{x_i}, g^{x_j\rho}) \\ &= \hat{e}(g^{x_i\rho}, g^{x_j}) \\ &= \mathsf{IDComb}(\mathsf{MaulVK}(\mathsf{vk}_i, \rho), \mathsf{vk}_j).\end{aligned}$$

Discussion. Similarly to the protocol for randomized equality presented in [CMMV24], this protocol also reuses the same randomness ρ_i to blind both outbound and inbound verification keys for the signature scheme. While one can show that this protocol still does not lead to any exfiltration of information (and hence, is secure as per [CMNV22]), the adversary can predict the value $\widetilde{\mathsf{sid}}_i$ that will be computed by C_i by exclusively observing $\widetilde{\mathsf{vk}}_i$. In particular, the adversary can mount an input-triggering attack by sampling a vk_j such that the output of $\mathsf{IDComb}(\widetilde{\mathsf{vk}}_i, \mathsf{vk}_j) = \mathsf{IDComb}(\mathsf{vk}_i, \widetilde{\mathsf{vk}}_j)$ is a trigger. Nonetheless, by applying the fix we sketched for the previous protocol, this attack would also be foiled: the adversary would have to pick to its vk_j *before* retrieving $\widetilde{\mathsf{vk}}_i$ (which would be committed).

Further limitations. Differently from the protocol described in Section 5.4.1, the construction we show here is inherently restricted to the 2-party setting due to requiring the signature scheme to achieve consistent identity combinability. This, in turn, implies that our porting of the split-authenticated framework of [BCL+05] would be restricted to only 2-party split functionalities, unless a proper extension of this property is formalized (and instantiated).

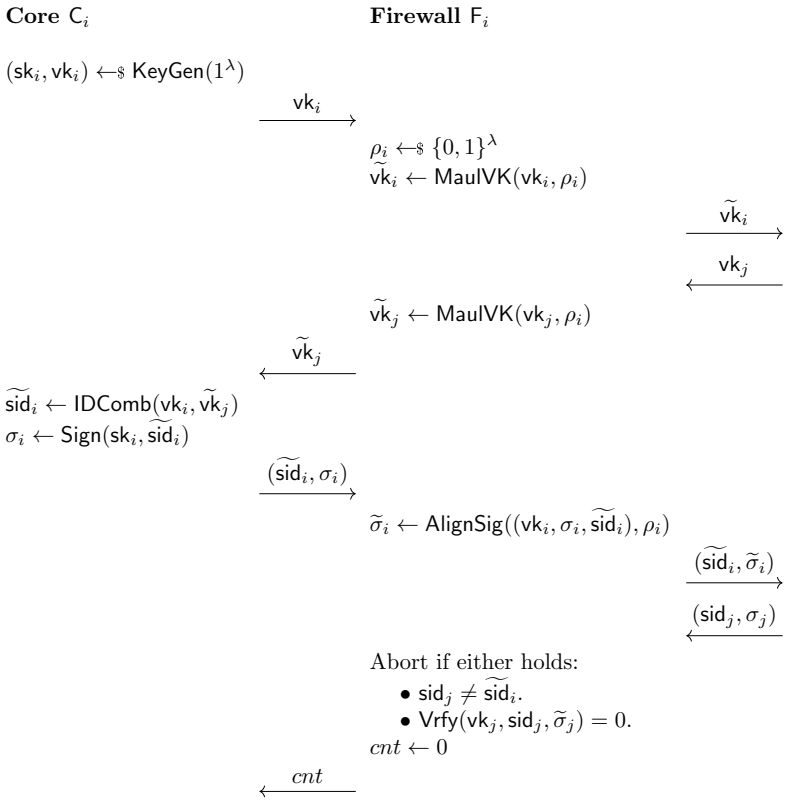


Figure A.12: The sanitizing protocol of [CMMV24] implementing the link initialization phase of $s\mathcal{F}_{\mathsf{SAT}}$.

B Subversion-resilient PAKE from Smooth-Projective Hashing

This section completes the formal treatment of Section 7.

B.1 Complete proof of the PAKE protocol

Theorem B.1. *The protocol in Figure 7.1 is correct.*

Proof. Let us denote $\mathsf{full\text{-}aux}_i = (\mathsf{pp}, \mathsf{crs}', \mathsf{aux}_i)$. Given that $\mathsf{pw}_i = \mathsf{pw}_{1-i}$, $\mathsf{full\text{-}aux}_i = \mathsf{full\text{-}aux}_{1-i}$; hence, we omit $\mathsf{full\text{-}aux}$ from all algorithm calls for compactness of notation. We analyze the two portions of the key $\mathsf{skey}^1, \mathsf{skey}^2$ separately:

$$\begin{aligned}
\mathsf{skey}^1_i &= \mathsf{Hash}(\mathsf{hk}_i, \tilde{c}_{1-i}) \oplus \mathsf{MaulH}(\mathsf{hp}_i, \tilde{r}_i) \\
&= \mathsf{Hash}(\widetilde{\mathsf{hk}}_i, \tilde{c}_{1-i}) && \text{[Projection Consistency]} \\
&= \mathsf{Hash}(\widetilde{\mathsf{hk}}_i, c_{1-i}) \oplus \mathsf{ReranH}(c_{1-i}, \tilde{w}_{1-i}) && \text{[Rerandomization Consistency]} \\
&= \mathsf{ProjHash}(\widetilde{\mathsf{hp}}_i, c_{1-i}, w_{1-i}) \oplus \mathsf{ReranH}(c_{1-i}, \tilde{w}_{1-i}) && \text{[Correctness of SPHF]} \\
&= \mathsf{skey}^2_{1-i}.
\end{aligned}$$

$$\begin{aligned}
\mathsf{skey}^2_i &= \mathsf{ProjHash}(\widetilde{\mathsf{hp}}_{1-i}, c_i, w_i) \oplus \mathsf{ReranH}(c_i, \widetilde{w}_i) \\
&= \mathsf{Hash}(\widetilde{\mathsf{hk}}_{1-i}, c_i) \oplus \mathsf{ReranH}(c_i, \widetilde{w}_i) && \text{[Correctness of SPHF]} \\
&= \mathsf{Hash}(\widetilde{\mathsf{hk}}_{1-i}, \widetilde{c}_i) && \text{[Rerandomization Consistency]} \\
&= \mathsf{Hash}(\mathsf{hk}_{1-i}, \widetilde{c}_i) \oplus \mathsf{MaulH}(\mathsf{hp}_{1-i}, \widetilde{r}_{1-i}) && \text{[Projection Consistency]} \\
&= \mathsf{skey}^1_{1-i}.
\end{aligned}$$

We have that $\mathsf{skey}^1_i \oplus \mathsf{skey}^2_i = \mathsf{skey}^2_{1-i} \oplus \mathsf{skey}^1_{1-i}$, as desired. □

Lemma B.1. *The firewalls in Figure 7.1 are strongly sanitizing.*

Proof. In order to prove strong sanitation, we show that no environment $\mathcal{Z}$, which allows specious corruptions for the core C but does not corrupt the respective firewall F, is able to distinguish between an execution of the protocol in Figure 7.1 with an incorruptible C from one that accepts the corruption $\tilde{\mathsf{C}}$. To this extent, we invoke the individual malleability properties of the primitives used for the three types of messages exchanged in the protocol: projection keys, ciphertexts, and NIZKs.

A specious $\tilde{\mathsf{C}}$ might generate the outbound tuple (hp, c, π) in a biased manner. The projection key hp gets mauled to a new key by F with the MaulK algorithm, which outputs a fresh key $\widetilde{\mathsf{hp}}$ due to key indistinguishability of the MTSPHF. The ciphertext c gets rerandomized using the ReranE algorithm, which outputs a fresh ciphertext $\widetilde{c}$ by element indistinguishability of the MTSPHF. The proof π gets mauled to a fresh proof $\widetilde{\pi}$ due to strong derivation privacy of the CM-NIZK scheme. □

Lemma B.2. *For every adversary $\mathcal{A}$ corrupting either party maliciously in an execution of protocol Π as in Figure 7.1 in the $\mathcal{F}_{\mathsf{crs}}$-hybrid model, there exists a simulator $\mathcal{S}$ such that, for all environments $\mathcal{Z}$, the following holds:*

$$\textsc{Exec}^{\mathcal{F}_{\mathsf{crs}}}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \textsc{Exec}_{\mathcal{F}_{\mathsf{PAKE}},\mathcal{S},\mathcal{Z}}$$

Proof. In the following proof, we denote $\mathsf{full\text{-}aux} \leftarrow (\mathsf{pp}, \mathsf{crs}', \mathsf{aux})$ for compactness of notation. We provide a description of the simulator in what follows.

Description of the simulator. The simulator $\mathcal{S}$ first generates the CRS knowing its trapdoors (*i.e.*, $\mathsf{dk}, \tau_{\mathsf{CM\text{-}NIZK}}, \tau_{\mathsf{MTSPHF}}$). Then:

- **Upon receiving** (NEWSESSION, $\mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j$) **from** $\mathcal{F}_{\mathsf{PAKE}}$**:**
  $\mathcal{S}$ simulates the behaviour of party P_i. In particular, since F_i is semi-honest, $\mathcal{S}$ should only simulate the behaviour of C_i. For that, it computes a key pair $(\mathsf{hk}, \mathsf{hp})$ for the MTSPHF, an encryption c of dummy password 1 using ek, and a simulated proof π on statement c by using trapdoor $\tau_{\mathsf{CM\text{-}NIZK}}$. Finally, since F_i is semi-honest, $\mathcal{S}$ has access to the randomness $\widetilde{w}$ used to rerandomize c to $\widetilde{c}$, and to the randomness $\widetilde{r}$ used to rerandomize hp to $\widetilde{\mathsf{hp}}$. In the remainder of the proof, these random strings will be omitted while referring to inbound messages for compactness of representation.
- **Upon receiving** (hp', c', π') **from** $\mathcal{A}$ **with destination honest party** P_i**:**
  $\mathcal{S}$ checks whether $\mathsf{Vrfy}(\mathsf{crs}, c', \pi') = 1$. If so, it checks whether (hp', c') match a tuple $(\widetilde{\mathsf{hp}}, \widetilde{c})$ that was output by $\mathcal{S}$ on behalf of uncorrupted C_j and rerandomized by semi-honest F_j. Then, it proceeds as follows:
  - If the above check succeeds, the tuple is deemed to be "oracle-generated", meaning that the adversary did allow uncorrupted P_i to receive the protocol messages originating from uncorrupted P_j. In this case, $\mathcal{S}$ sends $(\text{NEWKEY}, \mathsf{sid}, \mathsf{P}_i, \perp)$ to $\mathcal{F}_{\mathsf{PAKE}}$. Since no TESTPWD queries were performed on this record, the dummy P_i will receive a random session key if it terminates before P_j or if the input passwords differed; otherwise, the same session key as the one sent to P_j.
  - If the above check fails, the tuple is deemed to be "non-oracle-generated", meaning that the adversary has actively interfered with the transmission of messages among uncorrupted parties. In this case, $\mathcal{S}$ uses $\tau_{\mathsf{CM\text{-}NIZK}}$ to extract from π' the tuple $((\mathsf{pw}', w'), c'', T)$. Then:
    - If $(\mathsf{pw}', w') = \perp$ and $(c'', T) \neq \perp$, it means $\mathcal{A}$ rerandomized a simulated proof originally generated by $\mathcal{S}$. In this case, even though $\mathcal{A}$ exhibits an accepting proof for a statement c', it does not prove knowledge of the password or the randomness used to generate c' (and hence, it cannot derive the same session key as the target uncorrupted P_i as per the protocol description). For that, $\mathcal{S}$ sends queries $(\text{TESTPWD}, \mathsf{sid}, \mathsf{P}_i, \perp)$ and $(\text{NEWKEY}, \mathsf{sid}, \mathsf{P}_i, \perp)$ to $\mathcal{F}_{\mathsf{PAKE}}$, which marks the record as "interrupted" and makes P_i receive a random session key.
    - If $(\mathsf{pw}', w') \neq \perp$, it means $\mathcal{A}$ produced a fresh accepting proof on statement c'. In this case, $\mathcal{S}$ sends query $(\text{TESTPWD}, \mathsf{sid}, \mathsf{P}_i, \mathsf{pw}')$ to $\mathcal{F}_{\mathsf{PAKE}}$. If the reply is "correct guess", then $\mathcal{S}$ computes the session key as in the real-world protocol run with password pw'. Importantly, $\mathcal{S}$ uses THash with trapdoor τ_{MTSPHF} to evaluate H_1 instead of $\mathsf{ProjHash}$. This is to ensure H_1 matches with the counterpart computed by $\mathcal{A}$, given that c was originally an encryption of 1 instead of pw'. In particular, $\mathcal{S}$ computes $\mathsf{aux} \leftarrow \mathsf{pw}'$, $H_1 \leftarrow \mathsf{THash}(\mathsf{full\text{-}aux}, \mathsf{hp}', c, \tau_{\mathsf{MTSPHF}})$, $H_2 \leftarrow \mathsf{ReranH}(\mathsf{full\text{-}aux}, c, \widetilde{w})$, $H_1' \leftarrow \mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}, c')$, $H_2' \leftarrow \mathsf{MaulH}(\mathsf{full\text{-}aux}, \mathsf{hp}, \widetilde{r})$. Finally, $\mathcal{S}$ computes $\mathsf{skey} \leftarrow H_1 \oplus H_2 \oplus H_1' \oplus H_2'$ and sends $(\text{NEWKEY}, \mathsf{sid}, \mathsf{P}_i, \mathsf{sk})$ to $\mathcal{F}_{\mathsf{PAKE}}$, which makes P_i receive sk. If the reply is "wrong guess", $\mathcal{S}$ sends $(\text{NEWKEY}, \mathsf{sid}, \mathsf{P}_i, \perp)$ to $\mathcal{F}_{\mathsf{PAKE}}$, which makes P_i receive a random session key.

We now proceed to incrementally build our simulator via a chain of hybrids, starting from G_0 (the real world) until G_{10}, whose distribution is identical to the simulator described above. Compared to the original proof, introducing CM-NIZKs requires simulating up to q proofs and extracting from up to q proofs (where q is the number of activated parties). Hence, we achieve the same security loss as described in [BP13], with an additional loss bounded by $q \cdot (\epsilon_{\mathsf{ZK}} + \epsilon_{\mathsf{CM\text{-}SSE}})$, where ϵ_{ZK} is the distinguishing advantage of the adversary against zero-knowledge of the CM-NIZK, and $\epsilon_{\mathsf{CM\text{-}SSE}}$ is the advantage of the adversary in winning the CM-SSE game of the CM-NIZK.

Game G_0. This game matches the real world, where the CRS is generated without knowledge of trapdoors and all flows of honest components are generated by $\mathcal{S}$ as per the protocol description, with $\mathcal{S}$ knowing the inputs of each player. More precisely:

- Upon receiving $(\text{NEWSESSION}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j)$ for an honest P_i, $\mathcal{S}$ runs the protocol as prescribed by generating hashing and projection keys $(\mathsf{hk}, \mathsf{hp})$, a ciphertext c encrypting password pw with randomness w, and a proof π with statement c and witness (pw, w). Then, it sends (hp, c, π) to $\mathcal{A}$.
- Upon receiving (hp', c', π') from $\mathcal{A}$ (supposedly generated by C_j and rerandomized by F_j), $\mathcal{S}$ computes $H_1 \leftarrow \mathsf{ProjHash}(\mathsf{full\text{-}aux}, \mathsf{hp}', c, r)$, $H_2 \leftarrow \mathsf{ReranH}(\mathsf{full\text{-}aux}, c, \tilde{w})$, $H_1' \leftarrow \mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}, c')$, $H_2' \leftarrow \mathsf{MaulH}(\mathsf{full\text{-}aux}, \mathsf{hp}, \tilde{r})$, where $\mathsf{aux} = \mathsf{pw}$. Finally, it computes $\mathsf{skey} \leftarrow H_1 \oplus H_2 \oplus H_1' \oplus H_2'$ and provides sk to P_i.

Game G_{1a}. $\mathcal{S}$ generates the trapdoor of the CM-NIZK and of the CPA-secure encryption scheme, and it simulates all the proofs output by honest cores. This change is computationally indistinguishable due to zero-knowledge of the CM-NIZK.

Game G_{1b}. Upon receiving (hp', c', π') from $\mathcal{A}$ that is non-oracle generated on behalf of P_j, $\mathcal{S}$ extracts a tuple $((\mathsf{pw}', w'), c'', T)$ from the proof. If any of the following conditions occur, output abort: (i) $(\mathsf{pw}, w') \neq \bot \wedge (c', w') \notin R_{\mathcal{L}_{\mathsf{lpp}}}$, (ii) $(c'', T) \neq (\bot, \bot) \wedge (c'' \notin Q_x \vee c' \neq T_x(c'') \vee T \notin \mathcal{T})$, or (iii) $((\mathsf{pw}', w'), c'', T) = (\bot, \bot, \bot)$. In essence, unless $\mathcal{S}$ aborts, it correctly extracts either a password or a previously-queried statement and transformation, enabling the distinguishing of oracle-generated queries from non-oracle generated queries, as well as their respective handling, following the description of the ideal-world simulator. This event occurs with negligible probability due to controlled-malleable simulation sound extractability of the CM-NIZK. Note that, even though the reduction does no longer have access to the simulation trapdoor required to simulate proofs as per G_{1a}, it may query the oracle of the challenger of CM-SSE to obtain simulated proofs.

Game G_2. Upon receiving (hp', c', π') from $\mathcal{A}$ that is either (i) oracle-generated by $\mathcal{S}$ on behalf of P_j with password pw', or (ii) non-oracle generated by $\mathcal{A}$ on behalf of P_j with a proof that extracts to pw', the simulator checks whether $\mathsf{pw} \neq \mathsf{pw}'$. If so, it replaces the quantity $H_1' = \mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}, c')$ with a random value. This variation is computationally indistinguishable due to the smoothness of the MTSPHF.

Game G_3. Upon receiving (hp', c', π') from $\mathcal{A}$ that is either (i) oracle-generated by $\mathcal{S}$ on behalf of P_j with password pw', or (ii) non-oracle generated by $\mathcal{A}$ on behalf of P_j with a proof that extracts to pw', the simulator checks whether $\mathsf{pw} \neq \mathsf{pw}'$. If so, it replaces the session key skey of party P_i with a random value. This is identically distributed to the previous hybrid, as sk is obtained by combining H_1, H_2, H_1', H_2', and H_1' is random by the previous hybrid.

Game G_4. Upon receiving (hp', c', π') from $\mathcal{A}$ that is either (i) oracle-generated by $\mathcal{S}$ on behalf of P_j with password pw' and hashing key hk', or (ii) non-oracle generated by $\mathcal{A}$ on behalf of P_j with a proof that extracts to pw', $\mathcal{S}$ checks whether $\mathsf{pw} = \mathsf{pw}'$. If so, it redefines the computation of hash values H and H'. First, $\mathcal{S}$ introduces an initially-empty set S_L, and then it evaluates the hash values as follows:

- Upon computation of H' with hashing key hk, check if there exists a record $(\mathsf{aux}, \mathsf{hk}, c'; h)$ in S_L. If so, return h. Otherwise, retrieve the projection key hp associated with hashing key hk, compute $h \leftarrow \mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}, c') \oplus \mathsf{MaulK}(\mathsf{full\text{-}aux}, \mathsf{hp}, \tilde{r})$, and append $(\mathsf{aux}, \mathsf{hk}, c'; h)$ to S_L.
- Upon computation of H with projection key hp', retrieve the hashing key hk' associated with hp'[34], and check if there exists a record $(\mathsf{hk}', \mathsf{aux}, c; h)$ in S_L. If this is the case, return h. Otherwise, compute $h \leftarrow \mathsf{ProjHash}(\mathsf{full\text{-}aux}, \mathsf{hp}', c, r) \oplus \mathsf{ReranH}(\mathsf{full\text{-}aux}, c, \tilde{w})$ and append $(\mathsf{aux}, \mathsf{hk}', c; h)$ to S_L.

Intuitively, whenever two uncorrupted (simulated) cores mutually receive a tuple originating from their (simulated) peer, the set S_L enforces consistency between the output hashes without explicitly computing Hash or $\mathsf{ProjHash}$ more than once[35]. If the adversary instead guesses the password on behalf of P_j, the set S_L ensures that P_i still computes the session key by running the protocol honestly. This game is identically

[34] If hp' is non-oracle generated, set $\mathsf{hk}' \leftarrow \bot$, as the hashing key used by the adversary is unknown to $\mathcal{S}$.
[35] This mechanism is equivalent to algorithms HashL, $\mathsf{ProjHashL}$ used within the original security proof.

distributed to the previous one as, whenever $\mathsf{pw} = \mathsf{pw}'$, the outputs of the hashes match by the correctness properties of the MTSPHF.

Game G_5. Upon receiving (hp', c', π') from $\mathcal{A}$ on behalf of P_j that was oracle-generated by $\mathcal{S}$ on behalf of (simulated) P_j with password pw' and hashing key hk', $\mathcal{S}$ checks whether $\mathsf{pw} = \mathsf{pw}'$. If so, $\mathcal{S}$ replaces the evaluation of H' with a random value whenever S_L does not contain a record for hashing key hk (instead of evaluating $\mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}, c') \oplus \mathsf{MaulK}(\mathsf{full\text{-}aux}, \mathsf{hp}, \tilde{r})$).

To prove the indistinguishability of this game, we show a series of sub-hybrids $G(0)$ to $G(q)$, where in sub-hybrid $G(i)$ $\mathcal{S}$ modifies the evaluation of H' as specified above for all j oracle-generated flows (hp, c', π') received from $\mathcal{A}$ in which parties had the same password, with $j \leq i$. In sub-hybrid $G(i)$, we denote the i-th oracle-generated tuple (hp', c', π') as the critical flow, with P_a being the originator and P_b the receiver (where a, b are the indexes of NewSession queries used to initialize players $\mathsf{P}_a, \mathsf{P}_b$). For each sub-hybrid, we further define a series of sub-games G_a to G_e.

- **Game G_a.** In this game, $\mathcal{S}$ guesses the index a identifying the a-th NewSession query. This succeeds with probability $1/q$, where q is the number of activated players. As in the original proof, if the guess fails, the reduction simply outputs a random guess in the distinguishing game.
- **Game G_b.** In this game, whenever P_a receives its flow, it replaces the evaluation of the h within H as follows. Whenever no record containing hk' exists in S_L, $\mathcal{S}$ computes $h \leftarrow \mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}', c) \oplus \mathsf{ReranH}(\mathsf{full\text{-}aux}, c, \tilde{w})$ (instead of $h \leftarrow \mathsf{ProjHash}(\mathsf{full\text{-}aux}, \mathsf{hp}', c, r) \oplus \mathsf{ReranH}(\mathsf{full\text{-}aux}, c, \tilde{w})$). This game is perfectly indistinguishable from the previous one by correctness of the MTSPHF. In particular, $\mathsf{ProjHash}(\mathsf{full\text{-}aux}, \mathsf{hp}', c, r)$ is replaced with $\mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}', c)$, and in this game we are only considering projection keys that were oracle-generated on a valid ciphertext.
- **Game G_c.** For the a-th NewSession query, $\mathcal{S}$ computes c as an encryption of dummy password 1 instead of pw. This game is computationally indistinguishable from the previous one due to CPA-security of the encryption scheme. Within the reduction, random coins r used to generate the ciphertext are no longer used for the computation of the hash for this case. Moreover, the reduction can still handle non-oracle generated tuples consistently with the ideal-world simulator by extracting information from the respective proof (rather than relying on the decryption oracle of CCA-security of the original security proof).
- **Game G_d.** In this game, $\mathcal{S}$ guesses the index b identifying the b-th NewSession query, with P_b being the recipient of the critical flow. When P_b receives its flow, $\mathcal{S}$ computes H as in Game G_4 (*i.e.*, by computing it as a random value instead of $\mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}', c) \oplus \mathsf{MaulK}(\mathsf{full\text{-}aux}, \mathsf{hp}, \tilde{r})$ whenever h is unset). This change is computationally indistinguishable due to smoothness of the MTSPHF, as c is now an encryption of a dummy password (instead of the real password).
- **Game G_e.** For the a-th NewSession query, $\mathcal{S}$ reverts the variations introduced in Game G_c, computing c as an encryption of the real password pw instead of the dummy password 1. As for Game G_c, this modification is computationally indistinguishable due to CPA-security of the encryption scheme.
- **Game G_f.** For the a-th NewSession query, $\mathcal{S}$ reverts the variations introduced in Game G_b, replacing the evaluation of $\mathsf{Hash}(\mathsf{full\text{-}aux}, \mathsf{hk}', c)$ within h back with $\mathsf{ProjHash}(\mathsf{full\text{-}aux}, \mathsf{hp}', c, r)$ whenever no record containing hk' exists in S_L. This game is perfectly indistinguishable from the previous one by correctness of the MTSPHF, as we are only considering projection keys that were oracle-generated on a valid ciphertext.

Game G_6. Upon receiving (hp', c', π') from $\mathcal{A}$ that was oracle-generated by $\mathcal{S}$ on behalf of (simulated) P_j with password pw' and hashing key hk', $\mathcal{S}$ checks whether $\mathsf{pw} = \mathsf{pw}'$. If so, $\mathcal{S}$ chooses skey at random, and it will provide the same key to its peer whenever it receives (hp, c, π) from $\mathcal{A}$ on behalf of P_i.

This game is perfectly indistinguishable from the previous one, as skey is computed by combining H with H', and H is a random value. Moreover, by the computation of the hash defined in Games G_4 and G_5, the set S_L will enforce consistency among these random values.

Game G_7. $\mathcal{S}$ fully generates the CRS by also knowing the trapdoor τ_{MTSPHF} for the MTSPHF. As in the original proof, we remark that this cannot be done before, as smoothness of the MTSPHF is not guaranteed to hold if τ_{MTSPHF} is known. This game is identically distributed to the previous one.

Game G_8. Upon receiving (hp', c', π') from $\mathcal{A}$ on behalf of P_j that is non-oracle generated for which the proof extracts to pw', $\mathcal{S}$ checks whether $\mathsf{pw} = \mathsf{pw}'$. If so, $\mathcal{S}$ computes H_1 as $\mathsf{THash}(\mathsf{full\text{-}aux}, \mathsf{hp}', c, \tau_{\mathsf{MTSPHF}})$ instead of $\mathsf{ProjHash}(\mathsf{full\text{-}aux}, \mathsf{hp}', c, r)$. This game is indistinguishable from the previous one due to soundness of the MTSPHF.

Game G_9. Upon receiving $(\textsc{NewSession}, \mathsf{sid}, \mathsf{P}_i, \mathsf{P}_j)$ for an honest C_i, $\mathcal{S}$ computes c as an encryption of dummy password 1 instead of pw. This game is computationally indistinguishable from the previous one due to CPA-security of the encryption scheme. As before, the random coins r used to generate the ciphertext are no longer used for the computation of the hash. Moreover, as in Game G_c, the reduction can still handle non-oracle generated tuples consistently with the ideal-world simulator by extracting information from the respective proof (rather than relying on the decryption oracle of CCA-security of the original security proof).

Game G_{10}. In this game, we introduce $\mathcal{F}_{\mathsf{PAKE}}$ and remove access of $\mathcal{S}$ to the input passwords. In particular, $\mathcal{S}$ runs the simulator described above. □

B.2 A MTSPHF for ElGamal supporting the original syntax of malleability

In this section, we design a MTSPHF for the language of ElGamal ciphertexts that adheres to the original syntax of malleability due to [CMY+16].

B.2.1 A malleable SPHF for ElGamal ciphertexts

By inspecting the instantiation provided in the main body (Construction 7.1), we observe that a single $\widetilde{\alpha}_1$ is sufficient to maul hp to $\widetilde{\mathsf{hp}}$ while retaining key indistinguishability (instead of using a pair of random elements $(\widetilde{\alpha}_1, \widetilde{\alpha}_2)$). This modification allows us to remove the dependency of aux from MaulH.

Construction B.1 (A MSPHF for $\mathsf{L}^{\mathsf{ElGamal}}$)**.**

- $\mathsf{Setup}, \mathsf{HashKG}, \mathsf{ProjKG}, \mathsf{Hash}, \mathsf{ProjHash}$ *are the same as Construction 6.2.*
- $\mathsf{SampR}(\mathsf{pp})$*: Output* $\widetilde{r} \leftarrow \widetilde{\alpha}_1 \leftarrow\!\!\$\ \mathbb{Z}_{\mathsf{q}}$.
- $\mathsf{MaulK}(\mathsf{pp}, \mathsf{hp}, \widetilde{r})$*: Output* $\widetilde{\mathsf{hp}} \leftarrow \mathsf{hp} \cdot g^{\widetilde{\alpha}_1}$
- $\mathsf{MaulH}(\mathsf{pp}, \mathsf{hp}, c = (c_1, c_2), \widetilde{r})$*: Output* $\widetilde{\mathsf{hv}} \leftarrow c_1^{\widetilde{\alpha}_1}$.
- $\mathsf{MaulHK}(\mathsf{pp}, \mathsf{crs}', \mathsf{hk} = \alpha_1, \widetilde{r} = \widetilde{\alpha}_1)$*: Output* $\widetilde{\mathsf{hk}} \leftarrow (\alpha_1 + \widetilde{\alpha}_1, \alpha_2)$.
- $\mathsf{SampW}(\mathsf{pp})$*: Output* $\widetilde{w} \leftarrow \mathbb{Z}_{\mathsf{q}}$.
- $\mathsf{ReranE}(\mathsf{pp}, c, \widetilde{w})$*: Output* $\widetilde{c} = (c_1 \cdot g^{\widetilde{w}}, c_2 \cdot h^{\widetilde{w}})$.
- $\mathsf{ReranH}(\mathsf{pp}, \mathsf{hp}, c, \widetilde{w})$*: Output* $\mathsf{hp}^{\widetilde{w}}$.

The security proof is essentially the same as Construction 6.8, with the observation that $\mathsf{hp} \cdot g^{\widetilde{\alpha}_1}$ is uniformly distributed in $\mathbb{G}$.

B.2.2 A different TSPHF

Now that we have a MSPHF for ElGamal ciphertexts, we focus again on introducing a trapdoor component in a similar vein to the TSPHF for linearly representable languages of [BP13] (Construction 6.4). By casting that construction to the language of ElGamal ciphertexts, we observe that projection keys would be evaluated as $\mathsf{hp} = (g_0^{\alpha_1} h^{\alpha_2}, (g_1^{\tau\alpha_1}, g_1^{\tau\alpha_2}))$. In order to make the construction compatible with our malleability

properties (and, in particular, with key indistinguishability), we would have to maul all the components of such projection keys to prevent leakage. However, τ is a fixed setup, and we can only sanitize α_1 with some $\widetilde{\alpha}_1$ (due to the structure of our new MSPHF). Concretely, this means that $g_1^{\tau\alpha_2}$ can be leveraged to exfiltrate information (*e.g.*, by performing rejection sampling on α_2). Due to this constraint, we pivot instead towards an alternative (less efficient) generic construction of TSPHFs provided by [BP13, Section 4.1], which we briefly summarize in what follows.

A generic TSPHF from SPHFs and knowledge-sound NIZKs. Intuitively, it is possible to obtain a TSPHF from any SPHF by having the ProjKG algorithm of the SPHF additionally output a proof-of-knowledge showing that hp has been correctly generated from hk. With that, it is possible to perform trapdoor evaluation by extracting hk associated with hp from π (using the extraction trapdoor τ'). In the notation of Construction 6.4, the resulting projection key $\mathsf{hp} = (\gamma, \chi)$ is such that γ is the projection key of the SPHF, and $\chi = \pi$.

The authors remark this approach would be very inefficient: by instantiating the aforementioned knowledge sound NIZK with Groth-Sahai proofs[36], one would have to run a bit-by-bit proof due to the lack of assumptions on the domain of hk (for example, we do not assume that hk belongs to a group). As we have seen in the previous constructions, hk lies in $\mathbb{Z}_{\mathsf{q}}^n$, while Groth-Sahai proofs only allow the extraction of group elements.

Nonetheless, it is possible to overcome this limitation by introducing a bilinear pairing. With that, one may encode hk in $\mathbb{G}_1$ (in place of hk), and evaluate hashes by leveraging the bilinear pairing (similarly to Construction 6.4). For instance, instead of using $(\alpha_1, \alpha_2) \in \mathbb{Z}_{\mathsf{q}}^2$, one would compute π by using $(g_1^{\alpha_1}, g_1^{\alpha_2})$ as a witness. Hash, on input $x = (c_1, c_2)$, would be computed by evaluating $\hat{e}(c_1^{\alpha_1} \cdot c_2^{\alpha_2}, g_1)$, whereas THash, on input $x = (c_1, c_2)$, $\mathsf{hp} = (\gamma, \pi)$, and the trapdoor τ', would be computed by extracting $g_1^{\alpha_1}, g_1^{\alpha_2}$ from π (using τ') and evaluating $\hat{e}(c_1, g_1^{\alpha_1}) \cdot \hat{e}(c_2, g_1^{\alpha_2})$.

[BP13, Appendix F.1] remarks that the resulting construction, although less efficient than the generic framework given by Construction 6.4, is still reasonably efficient.

B.2.3 A malleable TSPHF for ElGamal ciphertexts

Finally, we equip the generic TSPHF described above with the malleability properties described in Construction B.1. This yields a MTSPHF for the language of ElGamal ciphertexts that adheres to the original syntax of malleability of [CMY+16], hence supporting applications that may require transparent RFs.

One crucial detail lies in the introduction of CM-NIZKs in place of knowledge sound NIZKs. This is necessary to ensure that, whenever an outbound projection key is mauled, the proof included in the projection key still verifies under the new mauled statement. In what follows, we formalize the NP-language that captures SPHF keys, alongside a set of allowable transformations that is consistent with the key rerandomization described in Construction B.1.

Definition B.1 (The language of SPHF keys $\mathcal{L}_{\mathsf{SPHF}}$ and set of allowable transformations $\mathcal{T}_{\mathsf{SPHFRerand}}$)**.**

- *Let $\mathcal{R}_{\mathsf{SPHF}}$ denote the relation of tuples $((g_0, h_0, \gamma, g_1), (\chi_1, \chi_2))$ such that $\hat{e}(g_0, \chi_1) \cdot \hat{e}(h_0, \chi_2) = \hat{e}(\gamma, g_1)$. Let $\mathcal{L}_{\mathsf{SPHF}}$ be the corresponding NP language. In what follows, when g_0, h_0 and g_1 are clear from the context, the statement (g_0, h_0, γ, g_1) is denoted as $\mathsf{statement}_\gamma$.*
- *Let $\mathcal{T}_{\mathsf{SPHFRerand}}$ denote the set of transformations $T_{\widetilde{\alpha}}$ over L_{SPHF} that map a statement (g_0, h_0, γ, g_1) to $(g_0, h_0, g_0^{\widetilde{\alpha}}\gamma, g_1)$, for $\widetilde{\alpha} \in \mathbb{Z}_{\mathsf{q}}$. Note that if (χ_1, χ_2) is a witness for (g_0, h_0, γ, g_1), then $(\chi_1 g_0^{\widetilde{\alpha}}, \chi_2)$ is a witness for $(g_0, h_0, g_0^{\widetilde{\alpha}}\gamma, g_1)$.*

Construction B.2 (A MTSPHF for $\mathsf{L}^{\mathsf{ElGamal}}$)**.** *Let $(\mathsf{CRSSetup}, \mathcal{P}, \mathcal{V}, (\mathcal{E}_0, \mathcal{E}_1), (\mathcal{S}_0, \mathcal{S}_1), \mathsf{ZKEval})$ be a CM-NIZK for $\mathcal{L}_{\mathsf{SPHF}}$ that is malleable with respect to $\mathcal{T}_{\mathsf{SPHFRerand}}$. We define a malleable TSPHF for $\mathsf{L}^{\mathsf{ElGamal}}$ as follows:*

- ***Trapdoor component.***

[36] We focus our attention on Groth-Sahai proofs, given that these objects are also used to instantiate CM-NIZKs, which will be required to obtain a MTSPHF.

- $\mathsf{Setup}(1^\lambda)$: *Compute* $(\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T, g_0, g_1, q) \leftarrow\!\!\$\ \mathsf{BilGroupGen}(1^\lambda)$. *Compute* $s \leftarrow\!\!\$\ \mathbb{Z}_\mathsf{q}$, *and evaluate* $h_0 = g_0^s$. *Compute* $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{CRSSetup}(1^\lambda)$. *Output* $\mathsf{pp} \leftarrow (1^\lambda, \mathsf{crs}, \mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T, g_0, g_1, q, h_0)$.
- $\mathsf{TSetup}(\mathsf{pp})$: *Output* $(\mathsf{crs}', \tau') \leftarrow (\sigma, \xi) \leftarrow\!\!\$\ \mathcal{E}_0(1^\lambda)$.
- $\mathsf{HashKG}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux})$: *Output* $\mathsf{hk} \leftarrow (\alpha_1, \alpha_2) \leftarrow\!\!\$\ \mathbb{Z}_\mathsf{q}^2$.
- $\mathsf{ProjKG}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hk} = (\alpha_1, \alpha_2))$: *Compute* $\gamma \leftarrow g_0^{\alpha_1} h_0^{\alpha_2}$ *and* $\chi \leftarrow (g_1^{\alpha_1}, g_1^{\alpha_2})$. *Compute* $\pi \leftarrow\!\!\$\ \mathcal{P}(\mathsf{crs}', \mathsf{statement}_\gamma, (\chi_1, \chi_2))$. *Output* $\mathsf{hp} \leftarrow (\gamma, \pi)$.
- $\mathsf{Hash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hk} = (\alpha_1, \alpha_2), x = (c_1, c_2))$: *Output* $\hat{e}(c_1^{\alpha_1}(c_2/\mathsf{aux})^{\alpha_2}, g_1)$.
- $\mathsf{ProjHash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp} = (\gamma, \pi), x, w)$: *Output* $\hat{e}(\gamma^w, g_1)$.
- $\mathsf{THash}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \tau', \mathsf{hp} = (\gamma, \pi), x = (c_1, c_2))$: *Compute* $(\chi_1, \chi_2) \leftarrow \mathcal{E}_1(\mathsf{crs}', \tau', \mathsf{statement}_\gamma, \pi)$. *Output* $\hat{e}(c_1, \chi_1) \cdot \hat{e}(c_2/\mathsf{aux}, \chi_2)$.
- $\mathsf{VerHP}(\mathsf{pp}, \mathsf{crs}', \mathsf{aux}, \mathsf{hp} = (\gamma, \pi))$: *Output* $\mathcal{V}(\mathsf{crs}', \mathsf{statement}_\gamma, \pi)$.

- ***Malleability component.***
  - $\mathsf{SampR}(\mathsf{pp}, \mathsf{crs}')$: *Output* $\widetilde{r} \leftarrow \widetilde{\alpha}_1 \leftarrow\!\!\$\ \mathbb{Z}_\mathsf{q}$.
  - $\mathsf{MaulK}(\mathsf{pp}, \mathsf{crs}', \mathsf{hp} = (\gamma, \pi), \widetilde{r} = \widetilde{\alpha}_1)$: *Let* $T_{\widetilde{\alpha}_1}$ *be the transformation that maps* $\mathsf{statement}_\gamma$ *to* $\mathsf{statement}_{g_0^{\widetilde{\alpha}_1}\gamma}$. *Compute* $\widetilde{\pi} \leftarrow\!\!\$\ \mathsf{ZKEval}(\mathsf{crs}', T_{\widetilde{\alpha}_1}, \mathsf{statement}_\gamma, \pi)$. *Output* $\widetilde{\mathsf{hp}} \leftarrow (g_0^{\widetilde{\alpha}_1}\gamma, \widetilde{\pi})$.
  - $\mathsf{MaulH}(\mathsf{pp}, \mathsf{crs}', \mathsf{hp}, x, \widetilde{r})$: *Output* $\widetilde{\mathsf{hv}} \leftarrow \hat{e}(c_1^{\widetilde{\alpha}_1}, g_1)$.
  - $\mathsf{MaulHK}(\mathsf{pp}, \mathsf{crs}', \mathsf{hk} = (\alpha_1, \alpha_2), \widetilde{r} = \widetilde{\alpha}_1)$: *Output* $\widetilde{\mathsf{hk}} \leftarrow (\alpha_1 + \widetilde{\alpha}_1, \alpha_2)$.
  - $\mathsf{SampW}(\mathsf{pp}, \mathsf{crs}')$: *Outputs* $\widetilde{w} \leftarrow\!\!\$\ \mathbb{Z}_\mathsf{q}$.
  - $\mathsf{ReranE}(\mathsf{pp}, \mathsf{crs}', x = (c_1, c_2), \widetilde{w})$: *Output* $\widetilde{x} \leftarrow (g_0^{\widetilde{w}} c_1, h_0^{\widetilde{w}} c_2)$.
  - $\mathsf{ReranH}(\mathsf{pp}, \mathsf{crs}', \mathsf{hp}, x, \widetilde{w})$: *Output* $\widetilde{\mathsf{hv}} \leftarrow e(\gamma^{\widetilde{w}}, g_1)$.

Theorem B.2. *Construction B.2 is a malleable TSPHF for* $\mathsf{L}^{\mathsf{ElGamal}}$.

Proof (Sketch). TSPHF properties are proved in [BP13]. The correctness properties for malleability follow by inspection. The element rerandomizability property follows from the DDH assumption in $\mathbb{G}_0$ (similarly to Construction 7.1). The key rerandomizability property follows by observing that $\gamma^{\widetilde{\alpha}_1}$ is uniformly distributed over $\mathbb{G}_0$, and that $\widetilde{\pi}$ is indistinguishable from a freshly-simulated proof (by strong derivation privacy of the proof system). □

C Robust NIZK Combiners

This section completes the formal treatment of Section 8.

C.1 A comparison with [HKNY23]

In [HKNY23], the authors exhibit a 1-out-of-2 robust combiner for quantum commitments - an inherently multi-property primitive that admits computational and statistical variants. By adapting our characterization of robustness to commitments, one would expect to be able to reuse the same proof strategies adopted for our impossibility results to rule out such combiners, hence contradicting the existence of their positive result. The reason of this mismatch lies in a fundamentally different characterization of robustness in [HKNY23], which is weaker compared to ours as it grants the combiner with the power to *create* security. We provide more details in what follows.

The definition of [HKNY23] requires the combined commitment to *always* be statistically binding and computationally hiding, regardless of whether the input schemes have statistical or computational security. Our definition (in the case of commitments) instead would not put any restriction on whether the binding or the hiding is statistical/computational. This indeed will depend on the input candidates. In particular, our notion allows the design of a combiner that, for example, on input statistically hiding commitments, yields a statistically hiding commitment.

The reason why the combiner of [HKNY23] does not satisfy our security notion is that the combiner of [HKNY23] always returns a statistical binding commitment, even if the input candidates are computationally binding and computationally hiding. This means that an unbounded adversary breaking the statistical binding of the combined scheme can never be turned into an adversary meaningfully breaking the security of any of the input schemes (given that the security of the candidates is only computational). This, in turn, means that the combiner can create security (statistical binding); something that does should not be captured by the spirit of robustness. Indeed, as per our definitions, the security of a combiner *must* derive from the security of the input primitives.

C.2 Combiners for knowledge soundness and zero-knowledge

In this section, we expand the informal definition provided in Section 8.2.4.

C.2.1 Security break for argument-of-knowledge

Following Section 8.2.2, we define $\mathcal{R}_{\mathsf{AoK}}$ in a similar fashion to $\mathcal{R}_{\mathsf{ZK}}$. In particular, a tuple *(protocol, (extractor, adversary))* belongs to $\mathcal{R}_{\mathsf{AoK}}$ if the adversary is able to either distinguish the introduction of a trapdoor setup by the extractor or craft an accepting proof for which the extractor fails to extract any witness. More formally:

Definition C.1 ($\mathcal{R}_{\mathsf{AoK}}$)**.** *Let $\Pi \in \mathcal{F}_{\mathsf{NI}}$, let $\mathcal{A}$ be an algorithm, and let $\mathcal{E} = (\mathcal{E}_0, \mathcal{E}_1)$ be a PPT algorithm. We say that $(\Pi, (\mathcal{A}, \mathcal{E})) \in \mathcal{R}_{\mathsf{AoK}}$ if the following holds:*

$$\Pr\left[\mathcal{A}(pp) = 1 \mid pp \leftarrow\$ \ \mathsf{setup}(1^{|x|}, 1^\lambda)\right] \not\approx_c \Pr\left[\mathcal{A}(pp) = 1 \mid (pp, \xi) \leftarrow\$ \ \mathcal{E}_0(1^{|x|}, 1^\lambda)\right],$$

or

$$\Pr\left[(x, w) \in \mathscr{R}_{\mathcal{L}}, \mathcal{V}(pp, x, \pi) = 1 \;\middle|\; \begin{array}{l}(pp, \xi) \leftarrow\$ \ \mathcal{E}_0(1^n, 1^\lambda); (x, \pi) \leftarrow \mathcal{A}(pp) \\ w \leftarrow \mathcal{E}_1(pp, \xi, x, \pi)\end{array}\right] < 1 - \mathsf{negl}(\lambda).$$

$\mathcal{R}_{\mathsf{AoK}}$ by itself is insufficient to capture AoK, as it only refers to a specific adversary breaking a specific extractor. We proceed similarly to ZK, and discuss black-box AoK in what follows (non-black-box AoK is reported in Appendix C.3).

Breaking black-box argument-of-knowledge. The standard definition of black-box argument-of-knowledge (reported in Definition 2.19) states that a protocol Π is secure if there exists a (universal) extractor $\mathcal{E}$ that is secure against every efficient adversary $\mathcal{A}$ (*i.e.*, $\exists \mathcal{E}$ *s.t.* $\forall PPT \mathcal{A}, (\Pi, (\mathcal{A}, \mathcal{E})) \notin \mathcal{R}_{\mathsf{AoK}}$). Breaking black-box argument-of-knowledge therefore means that there cannot exist any such extractor (which, in turn, means that *every* possible extractor can be attacked). This is formalized as follows:

$$\forall \mathcal{E} \exists PPT \mathcal{A} \text{ s.t. } (\Pi, (\mathcal{A}, \mathcal{E})) \in \mathcal{R}_{\mathsf{AoK}}.$$

The same rationale of Section 8.2.2 can be applied to proof-of-knowledge by allowing unbounded adversaries.

C.2.2 Combiners for argument-of-knowledge

In what follows, we describe (ZK, AoK)-combiners. All our definitions have a label $\alpha\beta\gamma \in \{\mathsf{B}, \mathsf{N}\}^3$ that specifies whether the combiner has black-box access ($\alpha = \mathsf{B}$) or non-black-box access ($\alpha = \mathsf{N}$) to its candidates, whether the combiner relies on black-box zero-knowledge ($\beta = \mathsf{B}$) or non-black-box zero-knowledge ($\beta = \mathsf{N}$) of the candidates, and whether the combiner relies on black-box argument-of-knowledge ($\gamma = \mathsf{B}$) or non-black-box argument-of-knowledge ($\gamma = \mathsf{N}$). For instance, a NBN combiner has non-black-box access to its candidates, and considers candidates satisfying black-box zero-knowledge (*i.e.*, the secure candidates admit a universal simulator) and non-black-box argument of knowledge (*i.e.*, the secure candidates are such that for every adversary there exists an extractor). For a cleaner presentation, we again present separate definitions for argument-of-knowledge and zero-knowledge.

Argument-of-knowledge of (ZK, AoK)-combiners. Argument-of-knowledge uses exactly the same structure as our (ZK, Sound)-combiners (intuitively, with extractors in place of simulators). For instance, for $*\,*\,\mathsf{B}$ (ZK, AoK)-combiners, we say that C is black-box argument-of-knowledge if the following holds: if C is not black-box argument-of-knowledge (*i.e.*, no universal extractor exists), then it is possible to design $n - t + 1$ reductions to one of the security properties of distinct candidates. Hence, if the combiner is not argument-of-knowledge, it means that a security property is violated for at least $n - t + 1$ of the candidates.

Definition C.2 (Argument-of-knowledge of t-out-of-n $\mathsf{type} \in \{\mathsf{B}, \mathsf{N}\}^3$ (ZK, AoK)-combiners)**.** *Let* C *be a correct* n*-candidate* $\mathcal{F}_{\mathsf{NI}}$*-combiner. We say a* t*-out-of-*n $\mathsf{type} \in \{\mathsf{B}, \mathsf{N}\}^3$ *(ZK, AoK)-combiner* C *is argument-of-knowledge if, after ordering the quantifiers according to* type *in Tables C.1, C.2, the following holds (*$\mathsf{combAoKQuant}$ *is also defined in Table C.1):*

$$\mathsf{combAoKQuant} \text{ s.t. } (\mathsf{C}^{f_1,\ldots,f_n}, (\mathcal{A}^{f_1,\ldots,f_n}, \mathcal{E})) \in \mathcal{R}_{\mathsf{AoK}} \implies \textit{any of the following:}$$

- $\mathbf{n - t + 1}$ *reductions to black-box (resp. non-black-box) zero-knowledge of the candidates if* $\mathsf{type} = *\mathsf{B}*$ *(resp.* $\mathsf{type} = *\mathsf{N}*$*), i.e.:*

$$\exists I = \{i_1, \ldots, i_{n-t+1}\} \subseteq [n] \text{ s.t. the following holds:}$$
$$\forall i_j \in I, \mathsf{candZKQuant} \text{ s.t. } (f_{i_j}, (\mathsf{Red}_{i_j}, \mathcal{S}_{i_j})) \in \mathcal{R}_{\mathsf{ZK}}.$$

- $\mathbf{n - t + 1}$ *reductions to black-box (resp. non-black-box) argument-of-knowledge of the candidates if* $\mathsf{type} = **\mathsf{B}$ *(resp.* $\mathsf{type} = **\mathsf{N}$*), i.e.:*

$$\exists I = \{i_1, \ldots, i_{n-t+1}\} \subseteq [n] \text{ s.t. the following holds:}$$
$$\forall i_j \in I, \mathsf{candAoKQuant} \text{ s.t. } (f_{i_j}, (\mathsf{Red}_{i_j}, \mathcal{E}_{i_j})) \in \mathcal{R}_{\mathsf{AoK}}.$$

type	*Structural quantifiers*	combAoKQuant
$\mathsf{B} * \mathsf{B}$	$\exists \mathsf{C} \forall f_1, \ldots, f_n \in \mathcal{F}$	$\forall \mathcal{E} \exists \mathcal{A}$
$\mathsf{B} * \mathsf{N}$	$\exists \mathsf{C} \forall f_1, \ldots, f_n \in \mathcal{F}$	$\exists \mathcal{A} \forall \mathcal{E}$
$\mathsf{N} * \mathsf{B}$	$\forall f_1, \ldots, f_n \in \mathcal{F} \exists \mathsf{C}$	$\forall \mathcal{E} \exists \mathcal{A}$
$\mathsf{N} * \mathsf{N}$	$\forall f_1, \ldots, f_n \in \mathcal{F} \exists \mathsf{C}$	$\exists \mathcal{A} \forall \mathcal{E}$

Table C.1: Classification of argument-of-knowledge for (ZK, AoK)-combiners.

type	candZKQuant
$*\mathsf{B}*$	$\forall \mathcal{S}_{i_j}\ \exists \mathsf{Red}_{i_j}$
$*\mathsf{N}*$	$\exists \mathsf{Red}_{i_j} \forall \mathcal{S}_{i_j}$

type	candAoKQuant
$**\mathsf{B}$	$\forall \mathcal{E}_{i_j}\ \exists \mathsf{Red}_{i_j}$
$**\mathsf{N}$	$\exists \mathsf{Red}_{i_j} \forall \mathcal{E}_{i_j}$

Table C.2: Quantifiers for ZK and AoK of candidates. $\mathcal{S}_{i_j}$, $\mathcal{E}_{i_j}$ are PPT algorithms. Red_{i_j} runs in time $\mathsf{poly}(\tau)$.

Connections with the *standard* AoK notion. Following the same arguments used for the zero-knowledge case, we can claim that an AoK combiner executed on input t primitives that satisfy the *standard* notion of AoK also satisfies the *standard* AoK notion. For the case of black-box AoK, this implies that a secure combiner admits a universal extractor, and hence it may be used in any application in which a black-box AoK protocol is required.

Zero-knowledge of (ZK, AoK)-combiners. The definition of zero-knowledge of a (ZK, AoK)-combiner is essentially the same as the one described for (ZK, Sound) combiner, with the only difference being that we can obtain reductions to ZK or AoK of the candidates.

Definition C.3 (Zero-knowledge of t-out-of-n $\mathsf{type} \in \{\mathsf{B},\mathsf{N}\}^3$ (ZK, AoK)-combiners)**.** *Let* C *be a correct* n*-candidate* $\mathcal{F}_{\mathsf{NI}}$*-combiner. We say a* t*-out-of-*n $\mathsf{type} \in \{\mathsf{B},\mathsf{N}\}^3$ *(ZK, AoK)-combiner* C *is zero-knowledge if, after ordering the quantifiers according to* type *in Tables C.2, C.3, the following holds (*combZKQuant *is also defined in Table C.3):*

$$\mathsf{combZKQuant}\ s.t.\ (\mathsf{C}^{f_1,\ldots,f_n}, (\mathcal{A}^{f_1,\ldots,f_n}, \mathcal{S})) \in \mathcal{R}_{\mathsf{ZK}} \implies \textit{any of the following:}$$

- $\mathbf{n-t+1}$ *reductions to black-box (resp. non-black-box) zero-knowledge of the candidates if* $\mathsf{type} = *\mathsf{B}*$ *(resp.* $\mathsf{type} = *\mathsf{N}*$*), as in Definition C.2.*
- $\mathbf{n-t+1}$ *reductions to black-box (resp. non-black-box) argument-of-knowledge of the candidates if* $\mathsf{type} = **\mathsf{B}$ *(resp.* $\mathsf{type} = **\mathsf{N}$*), as in Definition C.2.*

type	Structural quantifiers	combZKQuant
$\mathsf{BB}*$	$\exists \mathsf{C} \forall f_1,\ldots,f_n \in \mathcal{F}$	$\forall \mathcal{S} \exists \mathcal{A}$
$\mathsf{BN}*$	$\exists \mathsf{C} \forall f_1,\ldots,f_n \in \mathcal{F}$	$\exists \mathcal{A} \forall \mathcal{S}$
$\mathsf{NB}*$	$\forall f_1,\ldots,f_n \in \mathcal{F} \exists \mathsf{C}$	$\forall \mathcal{S} \exists \mathcal{A}$
$\mathsf{NN}*$	$\forall f_1,\ldots,f_n \in \mathcal{F} \exists \mathsf{C}$	$\exists \mathcal{A} \forall \mathcal{S}$

Table C.3: Classification of zero-knowledge for (ZK, AoK)-combiners.

C.3 Non-black-box ZK/AoK and related security breaks

In this section, we provide non-black-box variants of standard definitions for zero-knowledge and argument-of-knowledge, and show, in a similar fashion to Section 8.2.2, how to formalize security breaks by using the respective relations $\mathcal{R}_{\mathsf{ZK}}$ (described in Definition 8.3) and $\mathcal{R}_{\mathsf{AoK}}$ (described in Definition C.1).

Non-black-box zero-knowledge. First, let us consider non-black-box zero-knowledge.

Definition C.4 (Non-Black-Box Non-Interactive Zero-Knowledge)**.** *Let* $\Pi \in \mathcal{F}_{\mathsf{NI}}$*. Let* `REAL` *and* `IDEAL` *be two experiments defined as follows:*

- $\texttt{REAL} = \{(pp, x, w, \pi) \mid pp \leftarrow\!\$\ \mathsf{setup}(1^{|x|}, 1^\lambda);\ (x, w) \leftarrow\!\$\ \mathcal{A}(pp);\ \pi \leftarrow\!\$\ \mathcal{P}(pp, x, w)\}_{\lambda\in\mathbb{N}}$
- $\texttt{IDEAL}_{\mathcal{S}} = \{(pp, x, w, \pi) \mid (pp, \tau) \leftarrow\!\$\ \mathcal{S}^0(1^{|x|}, 1^\lambda);\ (x, w) \leftarrow\!\$\ \mathcal{A}(pp);\ \pi \leftarrow\!\$\ \mathcal{S}^1(pp, \tau, x)\}_{\lambda\in\mathbb{N}}$.

We say that Π *satisfies non-black-box zero-knowledge if, for all PPT algorithms* $\mathcal{A}$*, there exists a PPT algorithm* $\mathcal{S} = (\mathcal{S}^0, \mathcal{S}^1)$ *for which the following holds:*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{l} b \leftarrow\$ \{0,1\} \\ b' \leftarrow \mathcal{A}(pp, x, w, \pi) \end{array}; (pp, x, w, \pi) \leftarrow\$ \begin{cases} \texttt{REAL} & \text{if } b = 0 \\ \texttt{IDEAL}_{\mathcal{S}} & \text{if } b = 1 \end{cases}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Breaking non-black-box zero-knowledge. The standard definition of (computational) non-black-box zero-knowledge (reported above) states that a protocol Π is secure if, for every efficient adversary $\mathcal{A}$, there exists an efficient simulator $\mathcal{S}$ (*i.e.*, $\forall PPT\mathcal{A}\exists\mathcal{S}$ *s.t.* $(\Pi, (\mathcal{A}, \mathcal{S})) \notin \mathcal{R}_{\mathsf{ZK}}$). Breaking non-black-box zero-knowledge therefore means that there exists an efficient adversary for which no secure simulator can be designed (*i.e.*, this adversary breaks *every* possible simulator). This is formalized as follows:

$$\exists PPT\mathcal{A} \; s.t. \; \forall \mathcal{S}, (\Pi, (\mathcal{A}, \mathcal{S})) \in \mathcal{R}_{\mathsf{ZK}}.$$

As discussed in Section 8.2.2, it is possible to obtain a statistical variant of the above by lifting the requirement of PPT adversaries.

Non-black-box argument-of-knowledge. Next, we consider non-black-box argument-of-knowledge.

Definition C.5 (Non-Black-Box Argument-of-Knowledge)**.** *Let* $\Pi \in \mathcal{F}_{\mathsf{NI}}$*. We say that* Π *satisfies non-black-box argument-of-knowledge if, for all PPT algorithms* $\mathcal{A}$*, there exists a PPT algorithm* $\mathcal{E} = (\mathcal{E}^0, \mathcal{E}^1)$ *for which the following holds:*

$$\Pr\left[\mathcal{A}(pp) = 1 \mid pp \leftarrow\$ \; \mathsf{setup}(1^{|x|}, 1^{\lambda})\right] \approx_c \Pr\left[\mathcal{A}(pp) = 1 \mid (pp, \xi) \leftarrow\$ \; \mathcal{E}^0(1^{|x|}, 1^{\lambda})\right],$$

and

$$\Pr\left[(x, w) \in \mathscr{R}_{\mathcal{L}}, \mathcal{V}(pp, x, \pi) = 1 \;\middle|\; \begin{array}{l} (pp, \xi) \leftarrow\$ \; \mathcal{E}^0(1^n, 1^{\lambda}); (x, \pi) \leftarrow \mathcal{A}(pp) \\ w \leftarrow \mathcal{E}^1(pp, \xi, x, \pi) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Breaking non-black-box argument-of-knowledge. The standard definition of non-black-box argument-of-knowledge (reported above) states that a protocol Π is secure if, for every efficient adversary $\mathcal{A}$, there exists an efficient extractor $\mathcal{E}$ (*i.e.*, $\forall PPT\mathcal{A}\exists\mathcal{E}$ *s.t.* $(\Pi, (\mathcal{A}, \mathcal{E})) \notin \mathcal{R}_{\mathsf{AoK}}$). Breaking non-black-box argument-of-knowledge therefore means that there exists an efficient adversary for which no secure extractor can be designed (*i.e.*, this adversary breaks *every* possible extractor). This is formalized as follows:

$$\exists PPT\mathcal{A} \; s.t. \; \forall \mathcal{E}, (\Pi, (\mathcal{A}, \mathcal{E})) \in \mathcal{R}_{\mathsf{AoK}}.$$

Again, it is possible to obtain a definition applicable to proof-of-knowledge by considering unbounded adversaries.

C.4 Concrete instantiations of homomorphic languages and $\overline{\texttt{reconstruct}}_{\mathbb{H}}$

In this section, we report concrete instantiations of $\overline{\texttt{reconstruct}}_{\mathbb{H}}$ for some of the homomorphic languages presented in [Mau09].

We also note that, in the following examples, the witness is either in $\mathbb{Z}_{\mathsf{q}}$ or $\mathbb{Z}_{\mathsf{q}} \times \mathbb{Z}_{\mathsf{q}}$. When the witness is in $\mathbb{Z}_{\mathsf{q}}$, we know $\mathbb{Z}_{\mathsf{q}}$ is actually a finite field, and the secret sharing scheme ($\texttt{share}_{\mathbb{G}}$, $\texttt{reconstruct}_{\mathbb{G}}$) in Section 8.4 satisfies correctness (as per Definition 8.6) and perfect security (as per Definition 8.7). Regarding $\mathbb{Z}_{\mathsf{q}} \times \mathbb{Z}_{\mathsf{q}}$, if we consider the addition operator as the component-wise addition, and the multiplication operator as the component-wise multiplication, then $\mathbb{Z}_{\mathsf{q}} \times \mathbb{Z}_{\mathsf{q}}$ is a finite field, and the correctness and perfect security of ($\texttt{share}_{\mathbb{G}}$, $\texttt{reconstruct}_{\mathbb{G}}$) also holds. Intuitively, for this case, we secret share each component of the witness and perform the reconstruction component-wise.

DLOG. In DLOG, $(\mathbb{G}, \star) = (\mathbb{Z}_q, +)$, and $(\mathbb{H}, \otimes) = (\mathbb{H}, \cdot)$, with $\mathbb{H}$ being a group of prime order q. The homomorphic group operation is defined as:

$$\mathbb{G} \to \mathbb{H} : w \to f(w) = g^w$$

Therefore, $\{2\star\} = \cdot$, and $\{2\otimes\} = \exp$. More specifically:

- $f(w_1) \cdot f(w_2) = g^{w_1} \cdot g^{w_2} = g^{w_1+w_2} = f(w_1 + w_2)$
- $f(w) \exp v = g^w \exp v = \underbrace{g^w \cdot g^w \cdot \cdots \cdot g^w}_{v \text{ copies of } g^w} = g^{w \cdot v} = f(w \cdot v)$

For completeness, we also instantiate the generic $\mathtt{share}_{\mathbb{G}}$ of Section 8.4 with the operators defined above:

- $\mathtt{share}_{\mathbb{G}}(w)$: $a_1, \ldots, a_{t-1} \leftarrow\!\!\$\ \mathtt{Gen}_{\mathbb{G}}(1^\lambda)$. Output $\{w_i \leftarrow w + \sum_{j=1}^{t-1}(a_j \cdot i^j)\}_{i\in[n]}$
- $\overline{\mathtt{reconstruct}}_{\mathbb{G}}(\{s_i\}_{i\in I}, l)$: Output $w_l \leftarrow \sum_{j\in I}(s_i \cdot (\prod_{k\in I, k\neq j} \frac{l-k}{j-k}))$

We expand $\overline{\mathtt{reconstruct}}_{\mathbb{H}}$ in terms of $\overline{\mathtt{reconstruct}}_{\mathbb{G}}$, as in Section 8.4.

$$\begin{aligned}
x'_l &= \bigotimes_{j\in I}(x_j\{2\otimes\}(\prod_{k\in I,k\neq j}\frac{l-k}{j-k})) \\
&= \prod_{j\in I}(g^{w_j}\exp(\prod_{k\in I,k\neq j}\frac{l-k}{j-k})) \\
&= \prod_{j\in I}(g^{w_j\cdot(\prod_{k\in I,k\neq j}\frac{l-k}{j-k})}) \\
&= g^{\sum_{j\in I} w_j\cdot(\prod_{k\in I,k\neq j}\frac{l-k}{j-k})} \\
&= g^{w_l} = f(\overline{\mathtt{reconstruct}}_{\mathbb{G}}(\{w_j\}_{j\in I}, l)) = f(w_l) = x_l.
\end{aligned}$$

DH tuples. In DH, $f(w) = (g^w, h^w)$, $(\mathbb{G}, \star) = (\mathbb{Z}_q, +)$, and $(\mathbb{H}, \otimes) = (\mathbb{H} \times \mathbb{H}, \odot)$, with $\mathbb{H}$ being a group of prime order q. The homomorphic group operation is defined as:

$$\mathbb{G} \to \mathbb{H} \times \mathbb{H} : w \to f(w) = (g^w, h^w)$$

The group operation of $\mathbb{H}$ $\odot$ is the Hadamard product (*i.e.*, the component-wise product), with its matching hyperoperation $\{2\odot\}$ being the component-wise exponentiation. More specifically:

- $f(w_1) \odot f(w_2) = (g^{w_1} \cdot g^{w_2}, h^{w_1} \cdot h^{w_2}) = (g^{w_1+w_2}, h^{w_1+w_2}) = f(w_1 + w_2)$
- $f(w)\{2\odot\}v = \underbrace{f(w) \odot f(w) \odot \ldots \odot f(w)}_{v \text{ copies of } f(w_i)} = (g^{w\cdot v}, h^{w\cdot v}) = f(w \cdot v)$

The algorithms $\mathtt{share}_{\mathbb{G}}$, $\mathtt{reconstruct}_{\mathbb{G}}$ are exactly the same as in the previous example. We expand $\overline{\mathtt{reconstruct}}_{\mathbb{H}}$ in terms of $\overline{\mathtt{reconstruct}}_{\mathbb{G}}$, as in Section 8.4.

$$\begin{aligned}
x'_l &= \bigotimes_{j\in I}(x_j\{2\otimes\}(\prod_{k\in I,k\neq j}\frac{l-k}{j-k})) \\
&= \bigodot_{j\in I}((g^{w_j}, h^{w_j})\{2\odot\}(\prod_{k\in I,k\neq j}\frac{l-k}{j-k})) \\
&= \bigodot_{j\in I}(g^{w_j\cdot(\prod_{k\in I,k\neq j}\frac{l-k}{j-k})}, h^{w_j\cdot(\prod_{k\in I,k\neq j}\frac{l-k}{j-k})}) \\
&= (g^{\sum_{j\in I} w_j\cdot(\prod_{k\in I,k\neq j}\frac{l-k}{j-k})}, h^{\sum_{j\in I} w_j\cdot(\prod_{k\in I,k\neq j}\frac{l-k}{j-k})}) \\
&= (g^{w_l}, h^{w_l}) = f(\overline{\mathtt{reconstruct}}_{\mathbb{G}}(\{w_j\}_{j\in I}, l)) = f(w_l) = x_l.
\end{aligned}$$

Pedersen commitments. In Pedersen commitments, $f(w_1, w_2) = h_1^{w_1} h_2^{w_2}$, $(\mathbb{G}, \star) = (\mathbb{Z}_\mathsf{q} \times \mathbb{Z}_\mathsf{q}, \star)$, with $\star$ being the component-wise addition, with its matching hyperoperation $\{2\star\}$ being the component-wise multiplication, and $(\mathbb{H}, \otimes) = (\mathbb{H}, \cdot)$, with $\mathbb{H}$ being a group of prime order q. The homomorphic group operation is defined as:

$$\mathbb{Z}_\mathsf{q} \times \mathbb{Z}_\mathsf{q} \to \mathbb{H} : (w_1, w_2) \to f(w_1, w_2) = h_1^{w_1} h_2^{w_2}$$

We expand the homomorphic operations as follows:

- $f(a) \otimes f(b) = (h_1^{a_1} h_2^{a_2}) \cdot (h_1^{b_1} h_2^{b_2}) = h_1^{a_1+b_1} h_2^{a_2+b_2} = f(a \star b)$
- $f(a)\{2\otimes\}v = \underbrace{(h_1^{a_1} h_2^{a_2}) \cdot (h_1^{a_1} h_2^{a_2}) \ldots (h_1^{a_1} h_2^{a_2})}_{v \text{ copies of } a = (a_1, a_2)} = f(h_1^{va_1} h_2^{va_2}) = f(a\{2\star\}v)$

For completeness, we also report $\mathtt{share}_\mathbb{G}$ instantiated with the operators defined above. Intuitively, we secret share each component of the witness on a different polynomial, and perform the reconstruction component-wise:

- $\mathtt{share}_\mathbb{G}(w = (w_1, w_2))$: $(a_{1,1}, a_{2,1}) \ldots, (a_{1,t-1}, a_{2,t-1}) \leftarrow\$\ \mathtt{Gen}_\mathbb{G}(1^\lambda)$. Output $\{(w_{1,i}, w_{2,i}) \leftarrow (w_1 + \sum_{j=1}^{t-1}(a_{1,j} \cdot i^j), w_2 + \sum_{j=1}^{t-1}(a_{2,j} \cdot i^j))\}_{i \in [n]}$
- $\overline{\mathtt{reconstruct}}_\mathbb{G}(\{(s_{1,i}, s_{2,i})\}_{i \in I}, l)$:
  Output $w = (w_1, w_2) \leftarrow (\sum_{j \in I}(s_{1,i} \cdot (\prod_{k \in I, k \neq j} \frac{l-k}{j-k})), \sum_{j \in I}(s_{2,i} \cdot (\prod_{k \in I, k \neq j} \frac{l-k}{j-k})))$

We expand $\overline{\mathtt{reconstruct}}_\mathbb{H}$ in terms of $\overline{\mathtt{reconstruct}}_\mathbb{G}$, as in Section 8.4.

$$\begin{aligned}
x'_l &= \bigotimes_{j \in I}(x_j\{2\otimes\}(\prod_{k \in I, k \neq j} \frac{l-k}{j-k})) \\
&= \prod_{j \in I}(h_1^{a_{1,j}} h_2^{a_{2,j}}) \exp(\prod_{k \in I, k \neq j} \frac{l-k}{j-k})) \\
&= \prod_{j \in I}(h_1^{a_{1,j} \cdot (\prod_{k \in I, k \neq j} \frac{l-k}{j-k})} h_2^{a_{2,j} \cdot (\prod_{k \in I, k \neq j} \frac{l-k}{j-k})} \\
&= h_1^{\sum_{j \in I} a_{1,j} \cdot (\prod_{k \in I, k \neq j} \frac{l-k}{j-k})} h_2^{\sum_{j \in I} a_{2,j} \cdot (\prod_{k \in S, k \neq j} \frac{l-k}{j-k})} \\
&= f(\overline{\mathtt{reconstruct}}_\mathbb{G}(\{(a_{1,j}, a_{2,j})\}_{j \in I}), l) = f(a_l) = x_l.
\end{aligned}$$

C.5 Combiner for homomorphic languages: formal treatment

In this section, we provide formal theorems and proofs for the combiner introduced in Section 8.4.

C.5.1 C_HL is a (ZK, Sound)-combiner

In order to claim that C_HL (as in Figure 8.1) is a t-out-of-n BB (ZK, Sound)-combiner, we need to argue the following:

- C_HL is a correct n-candidate $\mathcal{F}_\mathsf{NI}$-combiner for the language $\mathcal{L}_\mathsf{HL}$, as per Definition 8.1.
- C_HL is a sound t-out-of-n BB (ZK, Sound)-combiner for the language $\mathcal{L}_\mathsf{HL}$, as per Definition 8.4.
- C_HL is a zero-knowledge t-out-of-n BB (ZK, Sound)-combiner for the language $\mathcal{L}_\mathsf{HL}$, as per Definition 8.5.

Below, we provide formal theorems and proofs for each of the aforementioned items.

Theorem C.1. *The construction in Figure 8.1 is a correct n-candidate $\mathcal{F}_\mathsf{NI}$-combiner for the language $\mathcal{L}_\mathsf{HL}$, as per Definition 8.1, for $n, t \in \mathsf{poly}(\lambda)$.*

Proof (Theorem C.1). In order for the construction to be an n-candidate $\mathcal{F}_\mathsf{NI}$-combiner, the following has to hold:

$$\Pr\left[\mathsf{C}_\mathsf{vrfy}(pp, x, \mathsf{C}_\mathsf{prove}(pp, x, w)) = 1 \mid pp \leftarrow\$\ \mathsf{C}_\mathsf{setup}(1^{|x|}, 1^\lambda)\right] \geq 1 - \mathsf{negl}(\lambda).$$

Intuitively, this happens if both (i) all sub-proofs are accepting, and (ii) all reconstructed values are consistent with the same polynomial, which evaluates to x in 0. We explore the two events separately.

Event 1. Let $\mathsf{E}_{\mathsf{corr,k}}$ as the event that the k-th sub-proof is accepting. We denote $\mathsf{E}_{\mathsf{corr}}$ as the event that all sub-proofs are accepting, *i.e.*:

$$\Pr[\mathsf{E}_{\mathsf{corr}}] = \prod_{k=1}^{n} \Pr[\mathsf{E}_{\mathsf{corr,k}}].$$

For $t \in \mathsf{poly}(\lambda)$, the aforementioned quantity is overwhelming. By Bernoulli's inequality (*i.e.*, $(1+x)^n \geq 1+nx$), the following holds:

$$\begin{aligned}
\Pr[\mathsf{E}_{\mathsf{corr}}] &\geq (\min_k \{\Pr[\mathsf{E}_{\mathsf{corr,k}}]\})^n \\
&\geq (1 - \mathsf{negl}(\lambda))^{\mathsf{poly}(\lambda)} \\
&\geq 1 - \mathsf{negl}(\lambda)\mathsf{poly}(\lambda) \\
&\geq 1 - \mathsf{negl}(\lambda).
\end{aligned}$$

Event 2. The verifier computes $x'_l = \overline{\texttt{reconstruct}}_{\mathbb{H}}(\{x_j\}_{j\in[t]}, l)$, for $l \in [n] \setminus [t] \cup \{0\}$. Following Section 8.4, this is equivalent to $f(\overline{\texttt{reconstruct}}_{\mathbb{G}}(\{s_j\}_{j\in[t]}, l))$. By correctness of Shamir's secret sharing (as per Definition 8.6), the verifier successfully reconstructs the sub-statement $x'_l = f(w_l) = x_l$ with probability 1. □

Theorem C.2. *The construction in Figure 8.1 is a sound t-out-of-n BB (ZK, Sound)-combiner for the language $\mathcal{L}_{\mathsf{HL}}$, as per Definition 8.4, for $t > \lfloor \frac{n}{2} \rfloor$ and $n, t \in \mathsf{poly}(\lambda)$.*

Proof (Theorem C.2). Assume by contradiction that the construction C_{HL} is not sound. Then, there exists an algorithm $\mathcal{A}$ running in time τ for which $(\mathsf{C}_{\mathsf{HL}}, \mathcal{A}) \in \mathcal{R}_{\mathsf{Sound}}$ (as per Definition 8.4). This means that $\mathcal{A}$ outputs an accepting proof π for the combiner for a statement $x \notin \mathcal{L}_{\mathsf{HL}}$ with non-negligible probability. The verifier parses (x, π) as $\{x_k, \pi_k\}_{k\in[n]}$, and behaves according to the protocol description. In particular, the verifier checks that (i) each sub-proof π_i verifies against the verifier of candidate i $\mathcal{V}_i$, and (ii) each sub-statement x_i is an evaluation of the same polynomial of degree $t-1$ in i.

In order to prove soundness for our combiner, it suffices to show that any accepting proof for an $x \notin \mathcal{L}_{\mathsf{HL}}$ must yield a set of indices $I = \{i_1, \ldots, i_{n-t+1}\}$ such that $(\Pi_{i_k}, \mathsf{Red}_{i_k}) \in \mathcal{R}_{\mathsf{Sound}}$ for all $i_k \in I$, where Red_{i_k} runs in time $\mathsf{poly}(\tau)$. We prove that this fact holds in Lemma C.1, and specify the behaviour of Red_{i_k} next.

Reduction Red_{i_j} to soundness of the i_k-th candidate

- Receive public parameters pp'_{i_k} from the challenger $\mathcal{C}$.
- Compute $pp \leftarrow\!\!\$\ \mathsf{C}_{\mathsf{setup}}(1^l, 1^\lambda)$. Parse pp as $\{pp_j\}_{j\in[n]}$. Replace pp_{i_k} with pp'_{i_k}.
- Forward pp to $\mathcal{A}$.
- Upon receiving (x, π) from $\mathcal{A}$, Parse π as $\{\pi_k, x_k\}_{k\in[n]}$.
- Forward (x_{i_k}, π_{i_k}) to $\mathcal{C}$.

Figure C.1: Reduction Red_{i_j} to soundness of the i_k-th candidate.

□

Lemma C.1. *For $t > \lfloor \frac{n}{2} \rfloor$, any accepting proof π for the combiner related to a statement $x \notin \mathcal{L}_{\mathsf{HL}}$ must include at least $n-t+1$ distinct indices $I = \{i_1, \ldots, i_{n-t+1}\}$ for which $(\Pi_{i_k}, \mathsf{Red}_{i_k}) \in \mathcal{R}_{\mathsf{Sound}}$ for all $i_k \in I$.*

Proof. Assume by contradiction that the lemma statement does not hold. Then, there exists an accepting proof π for the combiner related to a statement $x \notin \mathcal{L}_{\mathsf{HL}}$ that includes at most $n-t$ distinct indices

$I = \{i_1, \ldots, i_{n-t}\}$ for which π_{i_k} is an accepting proof for a statement $x_{i_k} \notin \mathcal{L}_{\mathsf{HL}}$ for candidate Π_{i_k}, with $i_k \in I$.

Since the verifier of the combiner accepts proof π, all sub-proofs are accepting. In particular, this means that there exists a subset I' containing sub-proofs $\{\pi_{i_k}\}_{i_k \notin I}$ that do not yield reductions to soundness. Given that these proofs are accepting, they relate to sub-statements that are in the language. We observe that, by assumption, the cardinality of set I' is $n-(n-t) = t$. In what follows, we denote the set of shares identified by I' as $S = \{x_i\}_{i \in I'}$.

In order for the verifier of the combiner to accept, it should be the case that $x = \overline{\texttt{reconstruct}}_{\mathbb{H}}(\{x_k\}_{k \in I'}, 0)$, *i.e.*:

$$x = \bigotimes_{j \in I'} (x_j \{2\otimes\} (\prod_{k \in I', k \neq j} \frac{k}{k-j})).$$

Since all the sub-statements in S are in the language (*i.e.*, $\forall j \in I', x_j = f(w_j)$), the following holds:

$$x = \bigotimes_{j \in I'} (f(w_j) \{2\otimes\} (\prod_{k \in I', k \neq j} \frac{k}{k-j})).$$

Then, we apply the homomorphism of f by invoking Fact 1 and Fact 2:

$$\begin{aligned} x &= \bigotimes_{j \in I'} (f(w_j) \{2\otimes\} (\prod_{k \in I', k \neq j} \frac{k}{k-j})) \\ &= f(\bigstar_{j \in I'} (w_j \{2\star\} (\prod_{k \in I', k \neq j} \frac{k}{k-j}))) \\ &= f(\overline{\texttt{reconstruct}}_{\mathbb{G}}(\{w_j\}_{j \in I'}, 0)) \\ &= f(w). \end{aligned}$$

Note that, by the assumption of $t > \lfloor \frac{n}{2} \rfloor$, the reconstruction of the secret out of the n shares is uniquely determined by any set of size t. Hence, set S uniquely determines the secret $f(w)$. This is a contradiction to the assumption that $x \notin \mathcal{L}_{\mathsf{HL}}$, as $x = f(w)$. Therefore, any accepting proof π for the combiner for a statement $x \notin \mathcal{L}_{\mathsf{HL}}$ implies the existence of at least $n - t + 1$ reductions to soundness of the underlying candidates.

□

Theorem C.3. *The construction in Figure 8.1 is a zero-knowledge t-out-of-n BB (ZK, Sound)-combiner for the language $\mathcal{L}_{\mathsf{HL}}$, as per Definition 8.5, for $t > \lfloor \frac{n}{2} \rfloor$ and $n, t \in \mathsf{poly}(\lambda)$.*

Proof (Theorem C.3). Following the sketch provided in Section 8.4.3, we construct a simulator $\mathcal{S}^T$ and prove in Lemma C.2 that either its output is indistinguishable from the output of the real prover, or there exists at least one reduction to zero-knowledge of one of the candidates whose indices are in T. Then, we argue in Lemma C.6 that the existence of adversaries who break zero-knowledge of the combiner implies the existence of sufficiently-many reductions to zero-knowledge of the underlying candidates, concluding the proof. □

Lemma C.2. *There exists a simulator $\mathcal{S}^T$ for which either $\{\texttt{REAL}\} \approx_c \{\texttt{IDEAL}_{\mathcal{S}^T}\}$, or there exists a reduction to zero-knowledge of at least one of the candidates $\{\Pi_k\}_{k \in T}$.*

Proof. Let $T = \{i_1, \ldots, i_t\}$ be a set of indices sorted in lexicographic order identifying candidates $\Pi_{i_1}, \ldots, \Pi_{i_t}$. We denote T_j as the set containing the first j indices of T, *i.e.*, $T_j = \{i_1, \ldots, i_j\}$. $\mathcal{S}^T = (\mathcal{S}^{T,0}, \mathcal{S}^{T,1})$ is constructed as in Figure C.2.

Simulator $\mathcal{S}^T$ for C_{HL}

$\mathcal{S}^{T,0}(1^l, 1^\lambda)$:

- For $l \in T$, compute $(pp_l, \tau_l) \leftarrow\$ \mathcal{S}_l^0(1^l, 1^\lambda)$.
- For $i \in [n] \setminus T$, compute $pp_i \leftarrow\$ \mathsf{setup}_i(1^l, 1^\lambda)$.
- Compute $pp \leftarrow \{pp_k\}_{k\in[n]}$, $\tau \leftarrow \{\tau_k\}_{k\in T}$.
- Return (pp, τ).

$\mathcal{S}^{T,1}(pp, \tau, x)$:

- Sample $\{r_i\}_{i\in[n]\setminus T\cup T_{2t-n-1}} \leftarrow\$ \mathtt{Gen}_{\mathbb{G}}(1^\lambda)$.
- For $i \in [n] \setminus T \cup T_{2t-n-1}$, compute $x_i' \leftarrow f(r_i)$. Let $x_0' \leftarrow x$.
- For $l \in T \setminus T_{2t-n-1}$, compute $x_l' \leftarrow \overline{\mathtt{reconstruct}}_{\mathbb{H}}(\{x_j'\}_{j\in[n]\setminus T\cup T_{2t-n-1}\cup\{0\}}, l)$.
- For $i \in [n] \setminus T$, compute $\pi_i \leftarrow\$ \mathcal{P}_i(pp_i, x_i', r_i)$. For $l \in T$, compute $\pi_l \leftarrow\$ \mathcal{S}_l^1(pp_l, \tau_l, x_l')$.
- Return $(\{\pi_k\}_{k\in[n]}, \{x_k'\}_{k\in[n]})$.

Figure C.2: Description of $\mathcal{S}^T$ for $\mathsf{C_{HL}}$.

The proof of this lemma goes through $t+3$ hybrid experiments, where hybrid $\mathcal{H}_0$ is the same as the real world, and hybrid $\mathcal{H}_{t+2}$ is the same as the ideal world. We denote the output of the adversary in hybrid $\mathcal{H}_i$ as $\mathsf{out}^{\mathcal{H}_i}$, for $i \in \{0, 1, \ldots, t+2\}$. Formally we prove that, for $j \in \{1, \ldots, t+2\}$ and for any algorithm $\mathcal{A}$ running in time τ, the following holds:

$$\left|\Pr[\mathcal{A}(\mathsf{out}^{\mathcal{H}_{j-1}}) = 1] - \Pr[\mathcal{A}(\mathsf{out}^{\mathcal{H}_j}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

We define the hybrids in Figure C.7.

Hybrid chain for zero-knowledge of $\mathsf{C_{HL}}$

$\mathcal{H}_0$, $\boxed{\mathcal{H}_j}$, $\boxed{\mathcal{H}_{t+1}}$, $\boxed{\mathcal{H}_{t+2}}$

1. $pp_k \leftarrow\$ \mathsf{setup}_k(1^l, 1^\lambda)$ for $k \in [n]$.
   $pp \leftarrow \{pp_k\}_{k\in[n]}$.

1. $(pp_k, \tau_k) \leftarrow\$ \mathcal{S}_k^0(1^l, 1^\lambda)$ for $k \in T_j$. $pp_i \leftarrow\$ \mathsf{setup}_i(1^l, 1^\lambda)$ for $i \in [n] \setminus T_j$.
   $pp \leftarrow \{pp_k\}_{k\in[n]}$.

2. Compute $(x, w) \leftarrow\$ \mathcal{A}(pp)$

3. Compute $\{w_i\}_{i\in[n]} \leftarrow\$ \mathtt{share}_{\mathbb{G}}(w)$.
   For $i \in [n]$, compute $x_i \leftarrow f(w_i)$.

3. Sample $\{r_i\}_{i\in[n]\setminus T\cup T_{2t-n-1}} \leftarrow\$ \mathtt{Gen}_{\mathbb{G}}(1^\lambda)$.
   For $i \in [n] \setminus T \cup T_{2t-n+1}$, compute $x_i' \leftarrow f(r_i)$. Let $r_0 \leftarrow w$.
   For $l \in T \setminus T_{2t-n-1}$, compute $w_l \leftarrow \overline{\mathtt{reconstruct}}_{\mathbb{G}}(\{r_i\}_{j\in[n]\setminus T\cup T_{2t-n-1}\cup\{0\}}, l)$. Compute $x_l \leftarrow f(w_l)$.

3. Sample $\{r_i\}_{i\in[n]\setminus T\cup T_{2t-n-1}} \leftarrow\$ \mathtt{Gen}_{\mathbb{G}}(1^\lambda)$.
   For $i \in [n] \setminus T \cup T_{2t-n-1}$, compute $x_i' \leftarrow f(r_i)$. Let $x_0' \leftarrow x$.
   For $l \in T \setminus T_{2t-n-1}$, compute $x_l' \leftarrow \overline{\mathtt{reconstruct}}_{\mathbb{H}}(\{x_j'\}_{j\in[n]\setminus T\cup T_{2t-n-1}\cup\{0\}}, l)$.

4. $\pi_k \leftarrow\$ \mathcal{P}_k(pp_k, x_k, w_k)$ for $k \in [n]$.
   $\pi \leftarrow (\{\pi_k\}_{k\in[n]}, \{x_k\}_{k\in[n]})$.

4. $\pi_k \leftarrow\$ \mathcal{S}_k^1(pp_k, \tau_k, x_k)$ for $k \in T_j$. $\pi_i \leftarrow\$ \mathcal{P}_i(pp_i, x_i, w_i)$ for $i \in [n] \setminus T_j$.
   $\pi \leftarrow (\{\pi_k\}_{k\in[n]}, \{x_k\}_{k\in[n]})$.

4. $\pi_k \leftarrow\$ \mathcal{S}_k^1(pp_k, \tau_k, x_k)$ for $k \in T$. $\pi_i \leftarrow\$ \mathcal{P}_i(pp_i, x'_i, r_i)$ for $i \in [n] \setminus T$.
   $\pi \leftarrow (\{\pi_k\}_{k\in[n]}, \{x_k\}_{k\in T\setminus T_{2t-n-1}} || \{x'_k\}_{k\in[n]\setminus T\cup T_{2t-n-1}})$.

4. $\pi_k \leftarrow\$ \mathcal{S}_k^1(pp_k, \tau_k, x'_k)$ for $k \in T$. $\pi_i \leftarrow\$ \mathcal{P}_i(pp_i, x'_i, r_i)$ for $i \in [n] \setminus T$.
   $\pi \leftarrow (\{\pi_k\}_{k\in[n]}, \{x'_k\}_{k\in[n]})$.

5. Run $\mathcal{A}(pp, x, \pi)$.

Figure C.3: Hybrid experiments $\mathcal{H}_0$, $\mathcal{H}_j$, $\mathcal{H}_{t+1}$, $\mathcal{H}_{t+2}$, for $1 \le j \le t$, for zero-knowledge of C_{HL}.

Intuitively, in the first t hybrids, the real proofs generated by the candidates in T are gradually replaced with simulated proofs generated by their respective simulator. Note that, in principle, it could be the case that some of these candidates are not zero-knowledge. Hence, the adversary may be able to distinguish the output of two adjacent hybrids. This is acceptable as per the lemma statement, since in this case we would have a reduction to ZK of that specific candidate.

In $\mathcal{H}_{t+1}$, we replace $t-1$ shares of the real witness w with $t-1$ random group elements in $\mathbb{G}$. This is possible as shares of the real witness are used in at most $t-1$ of the prover algorithms of the candidates[37]. Hence, any adversary can leak at most $t-1$ of the shares, and it is therefore unable to determine whether these are shares of w or random group elements. Since this argument is information theoretical, the two hybrids are identically distributed. Finally, the last hybrid is just a syntactic change that derives from homomorphic operations. Formally, this lemma is proved by having Lemma C.3, Lemma C.4, and Lemma C.5. □

Lemma C.3 (Transition from $\mathcal{H}_{j-1}$ to $\mathcal{H}_j$, for $1 \le j \le t$)**.** *If there exists an algorithm $\mathcal{A}$ running in time τ for which $\mathsf{out}^{\mathcal{H}_{j-1}} \not\approx_c \mathsf{out}^{\mathcal{H}_j}$, there exists an algorithm Red_{i_j} running in time $\mathsf{poly}(\tau)$ that breaks zero-knowledge of the i_j-th scheme.*

Proof. Given $\mathcal{A}$ as in the theorem statement, we describe Red_{i_j} in what follows:

Reduction Red_{i_j} to zero-knowledge of the i_j-th candidate

- Receive public parameters pp_{i_j} from the challenger $\mathcal{C}$.
- Compute $(pp_k, \tau_k) \leftarrow\$ \mathcal{S}_k^0(1^l, 1^\lambda)$ for $k \in T_{j-1}$. Compute $pp_i \leftarrow\$ \mathsf{setup}_i(1^l, 1^\lambda)$ for $i \in [n] \setminus T_j$. Set $pp \leftarrow \{pp_k\}_{k\in[n]}$.
- Compute $(x, w) \leftarrow\$ \mathcal{A}(pp)$.
- Compute $\{w_i\}_{i\in[n]} \leftarrow \mathtt{share}_{\mathbb{G}}(w)$. Compute $\{x_i \leftarrow f(w_i)\}_{i\in[n]}$.
- Send $\{x_{i_j}, w_{i_j}\}$ to $\mathcal{C}$, and receive proof π_{i_j}. This proof is generated either as $\mathcal{P}_{i_j}(pp_{i_j}, x_{i_j}, w_{i_j})$ or $\mathcal{S}_{i_j}^1(pp_{i_j}, \tau_{i_j}, x_{i_j})$.
- Compute $\pi_k \leftarrow\$ \mathcal{S}_k^1(pp_k, \tau_k, x_k)$ for $k \in T_{j-1}$. Compute $\pi_i \leftarrow\$ \mathcal{P}_i(pp_i, x_i, w_i)$ for $i \in [n] \setminus T_j$.
- Set $\pi \leftarrow (\{\pi_k\}_{k\in[n]}, \{x_k\}_{k\in[n]})$.
- Run $\mathcal{A}(pp, x, \pi)$ and output whatever $\mathcal{A}$ outputs.

Figure C.4: Reduction Red_{i_j} to zero-knowledge of the i_j-th candidate.

If $\mathcal{C}$ provides a real proof, we are in hybrid $\mathcal{H}_{j-1}$. Otherwise, we are in hybrid $\mathcal{H}_j$. Therefore, Red_{i_j} retains the same distinguishing advantage of $\mathcal{A}$, breaking ZK of candidate Π_{i_j}. □

[37] More precisely, the real prover is used exactly in $n-t$ of the candidates. Therefore, $\mathcal{H}_{t+1}$ removes the dependency on the witness from all the proofs generated by real provers, as well as the first $t-1-(n-t) = 2t-n-1$ elements of the set T (*i.e.*, T_{2t-n-1}). From the honest majority assumption, the cardinality of T_{2t-n-1} is always greater or equal to 0. Looking ahead, this enables that the reconstruction procedure in $\mathcal{H}_{t+2}$ to be carried out with sufficiently-many shares.

Lemma C.4 (Transition from $\mathcal{H}_t$ to $\mathcal{H}_{t+1}$)**.** $\mathsf{out}^{\mathcal{H}_t}$ *and* $\mathsf{out}^{\mathcal{H}_{t+1}}$ *are identically distributed.*

Proof. Assume by contradiction there exists an unbounded algorithm $\mathcal{A}$ that can distinguish $\mathsf{out}^{\mathcal{H}_t}$ from $\mathsf{out}^{\mathcal{H}_{t+1}}$. Then, we can construct the following adversary $\overline{\mathcal{A}}^{\mathsf{SSS}}$ breaking security (as per Definition 8.6) of the t-out-of-n Shamir's secret sharing scheme SSS:

Reduction $\overline{\mathcal{A}}^{\mathsf{SSS}}$ to perfect security of Shamir's secret sharing scheme SSS

- Compute $(pp_k, \tau_k) \leftarrow\$ \mathcal{S}_k^0(1^l, 1^\lambda)$ for $k \in T$. $pp_i \leftarrow\$ \mathsf{setup}_i(1^l, 1^\lambda)$ for $i \in [n] \setminus T$. $pp \leftarrow \{pp_k\}_{k\in[n]}$.
- Compute $(x, w) \leftarrow\$ \mathcal{A}(pp)$, and forward w to the challenger $\mathcal{C}$.
- Receive $t-1$ shares $\{s_i\}_{i\in[n]\setminus T\cup T_{2t-n+1}}$ from $\mathcal{C}$ which are either shares of the real witness w or random group elements $\{r_i\}_{i\in[n]\setminus T\cup T_{2t-n-1}} \leftarrow\$ \mathtt{Gen}_{\mathbb{G}}(1^\lambda)$.
- Set $s_0 \leftarrow w$.
- For $l \in T \setminus T_{2t-n-1}$, compute $w_l \leftarrow \overline{\mathtt{reconstruct}}_{\mathbb{G}}(\{s_i\}_{i\in[n]\setminus T\cup T_{2t-n-1}\cup\{0\}}, l)$.
- For $l \in T \setminus T_{2t-n-1}$, compute $x_l \leftarrow f(w_l)$. For $i \in [n] \setminus T \cup T_{2t-n-1} \cup \{0\}$, compute $x_i \leftarrow f(s_i)$.
- For $l \in T$, compute $\pi_l \leftarrow\$ \mathcal{S}_l^1(pp_l, \tau_l, x_l)$. For $i \in [n] \setminus T$, compute $\pi_i \leftarrow\$ \mathcal{P}_i(pp_i, x_i, s_i)$.
- Set $\pi \leftarrow \{(\pi_k, x_k)\}_{k\in[n]}$.
- Run $\mathcal{A}(pp, x, \pi)$ and output whatever $\mathcal{A}$ outputs.

Figure C.5: Reduction $\overline{\mathcal{A}}^{\mathsf{SSS}}$ to perfect security of Shamir's secret sharing scheme SSS.

First, observe that $\overline{\mathcal{A}}^{\mathsf{SSS}}$ uses w alongside the shares s_i output by the challenger $\mathcal{C}$ in order to reconstruct shares of the real w for $i \in T \setminus T_{2t-n-1}$. This is necessary, as both $\mathcal{H}_t$ and $\mathcal{H}_{t+1}$ simulate proofs for $i \in T \setminus T_{2t-n-1}$ by supplying $x_i = f(w_i)$ as the input to the simulators of the candidates whose indices are in $T \setminus T_{2t-n-1}$.

We now argue that $\overline{\mathcal{A}}^{\mathsf{SSS}}$ correctly reproduces the behavior of hybrids $\mathcal{H}_t$ and $\mathcal{H}_{t+1}$ according to the challenger's behaviour. If the challenger shares the real witness w, all the proofs in $[n] \setminus T \cup T_{2t-n-1}$ are generated exactly as in $\mathcal{H}_t$ (*i.e.*, using shares of w). If the challenger generates random group elements in $\mathbb{G}$, all the proofs in $[n] \setminus T \cup T_{2t-n-1}$ are generated exactly as in $\mathcal{H}_{t+1}$ (*i.e.*, using random group elements in $\mathbb{G}$). Therefore, $\overline{\mathcal{A}}^{\mathsf{SSS}}$ retains the same distinguishing advantage of $\mathcal{A}$, breaking security of Shamir's secret sharing scheme. This is a contradiction; therefore, $\mathsf{out}^{\mathcal{H}_t} \equiv \mathsf{out}^{\mathcal{H}_{t+1}}$. □

Lemma C.5 (Transition from $\mathcal{H}_{t+1}$ to $\mathcal{H}_{t+2}$)**.** $\mathsf{out}^{\mathcal{H}_{t+1}}$ *and* $\mathsf{out}^{\mathcal{H}_{t+2}}$ *are identically distributed.*

Proof. The proof follows by comparing how the sub-statements are generated and by applying the property of homomorphic language. More precisely:

- **Comparing** $\{x_l\}_{l\in[n]\setminus T\cup T_{2t-n-1}}$**.** In both experiments, these sub-statements are generated as $f(r_l)$ for a random $r_l \in \mathbb{G}$.
- **Comparing** x_0**.** In $\mathcal{H}_{t+1}$, $x_0 = f(r_0)$, with $r_0 = w$. In $\mathcal{H}_{t+2}$, $x_0 = x$. Since $f(w) = x$ by assumption, the two sub-statements are identical.
- **Comparing** $\{x_l\}_{l\in T\setminus T_{2t-n-1}}$**.**
  In $\mathcal{H}_{t+1}$, $x_l = f(w_l)$, with $w_l = \overline{\mathtt{reconstruct}}_{\mathbb{G}}(\{r_j\}_{j\in[n]\setminus T\cup T_{2t-n-1}\cup\{0\}}, l)$.
  In $\mathcal{H}_{t+2}$, $x_l = \overline{\mathtt{reconstruct}}_{\mathbb{H}}(\{x_j\}_{j\in[n]\setminus T\cup T_{2t-n-1}\cup\{0\}}, l)$, with $x_j = f(r_j)$.
  We show that these sub-statements are identical by applying the homomorphic property of the lan-

guage, similarly to the proof for Theorem C.1:

$$\begin{aligned}
f(w_l) &= f(\overline{\texttt{reconstruct}}_{\mathbb{G}}(\{r_j\}_{j\in[n]\setminus T\cup T_{2t-n-1}\cup\{0\}}, l) \\
&= f(\bigstar_{j\in[n]\setminus T\cup T_{2t-n-1}\cup\{0\}} (r_j\{2\star\}(\prod_{k\in T\setminus T_{2t-n-1}, k\neq j} \frac{l-k}{j-k}))) \\
&= \bigotimes_{j\in[n]\setminus T\cup T_{2t-n-1}\cup\{0\}} (f(r_j)\{2\otimes\}(\prod_{k\in T\setminus T_{2t-n-1}, k\neq j} \frac{l-k}{j-k})) \\
&= \overline{\texttt{reconstruct}}_{\mathbb{H}}(\{f(r_j)\}_{j\in[n]\setminus T\cup T_{2t-n-1}\cup\{0\}}, l) \\
&= \overline{\texttt{reconstruct}}_{\mathbb{H}}(\{x_j\}_{j\in[n]\setminus T\cup T_{2t-n-1}\cup\{0\}}, l)
\end{aligned}$$

□

Lemma C.6. *The existence of adversaries breaking zero-knowledge of all possible simulators $\mathcal{S}^T$, as described in Lemma C.2, yields reductions to zero-knowledge of at least $n-t+1$ distinct candidates.*

Proof. Let $\mathtt{S}_0$ be a set of cardinality $\binom{n}{t}$ containing all the possible instantiations of $\mathcal{S}^T$ as described in Lemma C.2 (*i.e.*, by considering all the possible choices of T out of the n candidates). The fact that the combiner is not zero-knowledge (as per Definition 8.5) implies that no simulator in $\mathtt{S}_0$ can be such that $\{\mathtt{REAL}\} \approx_c \{\mathtt{IDEAL}_{\mathcal{S}^T}\}$. Moreover, Lemma C.2 prescribes that if $\{\mathtt{REAL}\} \not\approx_c \{\mathtt{IDEAL}_{\mathcal{S}^T}\}$, there exists a reduction to at least one of the candidate schemes identified by an index in T. Let $i_1 \in [n]$ be such an index. This means that there exists an adversary $\mathcal{A}$ that is able to break zero-knowledge for scheme Π_{i_1}, *i.e.*, $\mathcal{A}$ breaks all possible simulators of Π_{i_1}. Observe that *any* simulator that is parameterized by a T containing i_1 is such that $\{\mathtt{REAL}\} \not\approx_c \{\mathtt{IDEAL}_{\mathcal{S}^T}\}$. However, there still exists a set $\mathtt{S}_1$ of cardinality $\binom{n-1}{t}$ containing all the simulators parameterized by a T that do *not* contain i_1. Since there should exist (a possibly different) $\mathcal{A}$ that breaks all the simulators in $\mathtt{S}_1$, an index $i_2 \in [n]$ distinct from i_1 exists. This means that $\mathcal{A}$ is able to break zero-knowledge for scheme Π_{i_2}. Generalizing the above, we denote I_j as the set of indices of cardinality j for which some $\mathcal{A}$ is able to break zero-knowledge of the respective candidate Π_{i_k}. We further denote $\mathtt{S}_j$ as the set of simulators that are parameterized by a T that does not contain any $i_k \in I_j$. The cardinality of $\mathtt{S}_j$ is $\binom{n-j}{t}$. Since $\mathcal{A}$ must also break all the simulators in $\mathtt{S}_j$, there exists another index $i^* \in [n] \setminus I_j$ related to a scheme Π_{i^*} for which $\mathcal{A}$ breaks its ZK property.

In order to have $|\mathtt{S}_j| = 0$, it suffices to show that $t > n-j$. The minimum j for which this condition holds is $j = n-t+1$, as $\binom{t-1}{t} = 0$. Hence, the existence of adversaries breaking zero-knowledge of the combiner implies the existence of at least $n-t+1$ reductions to zero-knowledge of distinct candidates, concluding the proof. □

C.5.2 C_{HL} is a (ZK, AoK)-combiner

In order to claim that C_{HL} (as in Figure 8.1) is a t-out-of-n BBB (ZK, AoK)-combiner, we need to argue the following:

- C_{HL} is a correct n-candidate $\mathcal{F}_{\mathsf{NI}}$-combiner for the language $\mathcal{L}_{\mathsf{HL}}$, as per Definition 8.1.
- C_{HL} is an argument-of-knowledge t-out-of-n BBB (ZK, AoK)-combiner for the language $\mathcal{L}_{\mathsf{HL}}$, as per Definition C.2.
- C_{HL} is a zero-knowledge t-out-of-n BBB (ZK, AoK)-combiner for the language $\mathcal{L}_{\mathsf{HL}}$, as per Definition C.3.

Note that the first item already holds by Theorem C.1. In what follows, we formalize the remaining items.

Theorem C.4. *The construction in Figure 8.1 is an argument-of-knowledge t-out-of-n BBB (ZK, AoK)-combiner for the language $\mathcal{L}_{\mathsf{HL}}$, as per Definition C.2, for $t > \lfloor \frac{n}{2} \rfloor$ and $n, t \in \mathsf{poly}(\lambda)$.*

Proof (Theorem C.4). Following Section 8.4.3, we construct an extractor $\mathcal{E}^T$ and prove in Lemma C.7 that either it successfully extracts a witness, or there exists at least one reduction to AoK of one of the candidates whose indices are in T. Then, we argue in Lemma C.9 that the existence of adversaries who break argument-of-knowledge of the combiner implies the existence of sufficiently-many reductions to argument-of-knowledge of the underlying candidates, concluding the proof. The structure of this argument is similar to Lemma C.6. □

Lemma C.7. *There exists an extractor $\mathcal{E}^T$ for which either a witness w is successfully extracted from an accepting proof, or there exists a reduction to argument-of-knowledge of at least one of the candidates $\{\Pi_k\}_{k\in T}$.*

Proof. Let $T = \{i_1, \ldots, i_t\}$ be a set of indices sorted in lexicographic order identifying candidates $\Pi_{i_1}, \ldots, \Pi_{i_t}$. The extractor $\mathcal{E}^T = (\mathcal{E}^{T,0}, \mathcal{E}^{T,1})$ is constructed in Figure C.6.

Extractor $\mathcal{E}^T$ for C_{HL}

$\mathcal{E}^{T,0}(1^l, 1^\lambda)$:

- For $k \in T$, compute $(pp_k, \xi_k) \leftarrow\!\$\ \mathcal{E}^0_k(1^l, 1^\lambda)$.
- Compute $pp \leftarrow \{pp_k\}_{k\in[n]}$, $\xi \leftarrow \{\xi_k\}_{k\in T}$.
- Return (pp, ξ).

$\mathcal{E}^{T,1}(pp, \xi, x, \pi)$:

- For $k \in T$, parse the k-th component of π as (x_k, π_k). Compute $w_k \leftarrow \mathcal{E}^1_k(pp_k, \xi_k, x_k, \pi_k)$.
- Output $w \leftarrow \mathtt{reconstruct}_{\mathbb{G}}(\{w_j\}_{j\in T})$.

Figure C.6: Description of $\mathcal{E}^T$ for C_{HL}.

As per Definition C.1, an adversary may break argument-of-knowledge either by detecting the introduction of trapdoor setups, or by outputting an accepting proof that causes any extractor to fail. If either occurs, we exhibit a reduction to AoK of that specific candidate.

Introducing trapdoor setups. We define t hybrid experiments: in hybrid $\mathcal{H}_0$ all public parameters for the candidates in T are generated by setup algorithms; in hybrid $\mathcal{H}_t$, all public parameters for the candidates in T are generated by trapdoor setup algorithms. We denote the output of the adversary in hybrid $\mathcal{H}_i$ as $\mathsf{out}^{\mathcal{H}_i}$, for $i \in \{0, 1, \ldots, t\}$. Formally, for $j \in \{1, \ldots, t\}$ and for any algorithm $\mathcal{A}$ running in time τ, the following holds:

$$\left|\Pr[\mathcal{A}(\mathsf{out}^{\mathcal{H}_{j-1}}) = 1] - \Pr[\mathcal{A}(\mathsf{out}^{\mathcal{H}_j}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

We define the hybrids in Figure C.7, and prove that the above holds in Lemma C.8.

Hybrid chain for argument-of-knowledge of C_{HL}

$\mathcal{H}_0$, $\boxed{\mathcal{H}_j}$

1. $pp_k \leftarrow\!\$\ \mathsf{setup}_k(1^l, 1^\lambda)$ for $k \in [n]$. $pp \leftarrow \{pp_k\}_{k\in[n]}$.

1. $(pp_k, \tau_k) \leftarrow\!\$\ \mathcal{S}^0_k(1^l, 1^\lambda)$ for $k \in T_j$. $pp_i \leftarrow\!\$\ \mathsf{setup}_i(1^l, 1^\lambda)$ for $i \in [n] \setminus T_j$. $pp \leftarrow \{pp_k\}_{k\in[n]}$.

2. Run $\mathcal{A}(pp)$.

Figure C.7: Hybrid experiments $\mathcal{H}_0$, $\mathcal{H}_j$, for $1 \leq j \leq t$ for argument-of-knowledge of C_{HL}.

Extractor $\mathcal{E}^T$ fails to extract a witness. If any reduction occurred throughout the chain of hybrids, the existence of the extractor in Figure C.6 already implies a reduction to AoK for one of the candidates, as prescribed by the lemma statement. Indeed, if the adversary is able to distinguish whether any of the public parameters of candidates in T is generated by a trapdoor setup algorithm (*i.e.*, $\mathcal{E}_k^0$ for some $k \in T$), its behaviour may change arbitrarily when interacting with $\mathcal{E}^T$.

Conversely, if no reduction occurred so far, the adversary has at most an advantage of $t \cdot \mathsf{negl}(\lambda)$ in discerning the generation algorithm used for $pp = \{pp_k\}_{k\in[n]}$. Hence, it will always output an accepting proof upon input pp except for negligible probability. Given that, the proof continues only if no reduction was possible in the chain of hybrids, as the adversary interacts with an extractor that controls the trapdoors used to generate pp.

We observe that the only way $\mathcal{E}^T$ can reconstruct w is by successfully extracting all the sub-witnesses in T and using the $\texttt{reconstruct}_{\mathbb{G}}$ algorithm of Shamir's secret sharing scheme. Since the verifier $\mathsf{C}_{\mathsf{vrfy}}$ runs $\overline{\texttt{reconstruct}}_{\mathbb{H}}$ as a consistency check, when all the checks pass, it means that all sub-statements are shares of the statement $x = f(w)$, which in turn are all consistent with the witness w computed by reconstructing the extracted sub-witnesses (by the property of $\overline{\texttt{reconstruct}}_{\mathbb{H}}$ we have in Section 8.4). Hence, by correctness of Shamir's secret sharing, we reconstruct w successfully.

Therefore, in order for $\mathcal{E}^T$ to fail, it suffices for one of the sub-extraction procedures to fail, as otherwise the reconstruction of Shamir's secret sharing would have been successful and $\mathcal{E}^T$ would not have failed. Let i^* be an index for which $\mathcal{E}_{i^*}^1$ fails to extract from (x_{i^*}, π_{i^*}). Any adversary that outputs an accepting proof (x, π) for the combiner that contains (x_{i^*}, π_{i^*}) in position i^* immediately causes $\mathcal{E}^T$ to fail, yielding a reduction to AoK of scheme Π_{i^*}. We describe Red_{i^*} in Figure C.8.

Reduction Red_{i^*} to argument-of-knowledge of the i^*-th candidate

- Receive public parameters pp'_{i^*} from the challenger $\mathcal{C}$.
- Compute $pp \leftarrow\!\$\ \mathsf{C}_{\mathsf{setup}}(1^l, 1^\lambda)$. Parse pp as $\{pp_j\}_{j\in[n]}$. Replace pp_{i^*} with pp'_{i^*}.
- Forward pp to $\mathcal{A}$.
- Upon receiving (x, π) from $\mathcal{A}$, Parse π as $\{\pi_k, x_k\}_{k\in[n]}$.
- Forward (x_{i^*}, π_{i^*}) to $\mathcal{C}$.

Figure C.8: Reduction Red_{i^*} to argument-of-knowledge of the i^*-th candidate.

□

Lemma C.8 (Transition from $\mathcal{H}_{j-1}$ to $\mathcal{H}_j$, for $1 \le j \le t$)**.** *If there exists an algorithm $\mathcal{A}$ running in time τ for which $\mathsf{out}^{\mathcal{H}_{j-1}} \not\approx_c \mathsf{out}^{\mathcal{H}_j}$, there exists an algorithm Red_{i_j} running in time $\mathsf{poly}(\tau)$ that breaks argument-of-knowledge of the i_j-th scheme.*

Proof. Given $\mathcal{A}$ as in the theorem statement, we describe Red_{i_j} in Figure C.9.

Reduction Red_{i_j} to argument-of-knowledge of the i_j-th candidate

- Compute $(pp_k, \tau_k) \leftarrow\!\$\ \mathcal{E}_k^0(1^l, 1^\lambda)$ for $k \in T_{j-1}$. Compute $pp_i \leftarrow\!\$\ \mathsf{setup}_i(1^l, 1^\lambda)$ for $i \in [n] \setminus T_j$.
- Receive public parameters pp_{i_j} from the challenger $\mathcal{C}$. These public parameters are generated either by using $\mathsf{setup}_{i_j}(1^l, 1^\lambda)$ or $\mathcal{E}_{i_j}^0(1^l, 1^\lambda)$.
- Set $pp \leftarrow \{pp_k\}_{k\in[n]}$.
- Run $\mathcal{A}(pp)$ and output whatever $\mathcal{A}$ outputs.

Figure C.9: Reduction Red_{i_j} to argument-of-knowledge of the i_j-th candidate.

If $\mathcal{C}$ generates the public parameters using the setup algorithm, we are in hybrid $\mathcal{H}_{j-1}$. Otherwise, we are in hybrid $\mathcal{H}_j$. Therefore, Red_{i_j} retains the same distinguishing advantage of $\mathcal{A}$, breaking AoK of candidate Π_{i_j}. □

Lemma C.9. *The existence of adversaries breaking argument-of-knowledge of $\mathcal{E}^T$ as described in Lemma C.7 yields reductions to argument-of-knowledge for at least $n-t+1$ distinct candidates.*

Proof. The proof uses the same combinatorics argument described for the proof of Lemma C.6. In Lemma C.7, we define $\binom{n}{t}$ extractors, and the existence of adversaries breaking AoK for the combiner can also be used to break the security of all the extractors. In particular, there exists an index i^* associated with candidate Π_{i^*} for which an adversary either detects the introduction of the trapdoor generation algorithm, or makes sub-extractor $\mathcal{E}^1_{i^*}$ fail upon input an accepting sub-proof. In order to break AoK for all the possible extractors, there should exist $n-t+1$ such indices that are distinct, *i.e.*, there should exist adversaries breaking AoK for $n-t+1$ of the underlying schemes. The existence of these adversaries implies the existence of at least $n-t+1$ reductions to AoK of distinct candidates, concluding the proof. □

Theorem C.5. *The construction in Figure 8.1 is a zero-knowledge t-out-of-n BBB (ZK, AoK)-combiner for the language $\mathcal{L}_{\mathsf{HL}}$, as per Definition C.3, for $t > \lfloor \frac{n}{2} \rfloor$ and $n, t \in \mathsf{poly}(\lambda)$.*

Proof. Within the proof for Theorem C.3, we showed that it is possible to break zero-knowledge of the combiner by solely relying on zero-knowledge of the candidates (*i.e.*, by obtaining $n-t+1$ reductions to zero-knowledge of the candidates). Hence, we reuse exactly the same proof. □

C.6 Combiner from recursive proofs: formal treatment

In this section, we provide formal theorems and proofs for the combiner introduced in Section 8.5.

C.6.1 $\mathsf{C}_{\mathsf{PoP}}$ is a (ZK, AoK)-combiner

In order to claim that $\mathsf{C}_{\mathsf{PoP}}$ (as in Figure 8.2) is a t-out-of-n NBB (ZK, AoK)-combiner, we need to argue the following:

- $\mathsf{C}_{\mathsf{PoP}}$ is a correct n-candidate $\mathcal{F}_{\mathsf{NI}}$-combiner for any NP language $\mathcal{L}$, as per Definition 8.1.
- $\mathsf{C}_{\mathsf{PoP}}$ is an argument-of-knowledge t-out-of-n NBB (ZK, AoK)-combiner for any NP language $\mathcal{L}$, as per Definition C.2.
- $\mathsf{C}_{\mathsf{PoP}}$ is a zero-knowledge t-out-of-n NBB (ZK, AoK)-combiner for any NP language $\mathcal{L}$, as per Definition C.3.

Below, we provide formal theorems and proofs for each of the aforementioned items.

Theorem C.6. *The construction in Figure 8.2 is a correct n-candidate $\mathcal{F}_{\mathsf{NI}}$-combiner for any NP language $\mathcal{L}$, as per Definition 8.1, for $n, t \in O(1)$.*

Proof (Theorem C.6). In order for the construction to be an n-candidate $\mathcal{F}_{\mathsf{NI}}$-combiner, the following has to hold:

$$\Pr\left[\mathsf{C}_{\mathsf{vrfy}}(pp, x, \mathsf{C}_{\mathsf{prove}}(pp, x, w)) = 1 \mid pp \leftarrow\!\!\$\ \mathsf{C}_{\mathsf{setup}}(1^{|x|}, 1^{\lambda})\right] \geq 1 - \mathsf{negl}(\lambda).$$

The verifier of the combiner accepts a proof if all the sub-proofs are accepting. Denoting $\mathsf{E}_{\mathsf{corr,i}}$ as the event that the i-th sub-proof is accepting, we first argue that $\Pr[\mathsf{E}_{\mathsf{corr,i}}] \geq 1 - \mathsf{negl}(\lambda)$ for all $i \in \binom{n}{t}$. Given that π_i is generated by nesting t candidates, the success probability of generating an accepting π_i depends on the event that all the sub-proofs generated in the chain of proofs are accepting. Let $\mathsf{E}_{\mathsf{corr,i_k}}$ be the event that the i_k-th proof internally generated as part of π_i is accepting. Since $t \in O(1)$, we apply Bernoulli's inequality

(*i.e.*, $(1+x)^n \geq 1+nx$) to argue that $\Pr[\mathsf{E}_{\mathsf{corr},\mathsf{i}}]$ is overwhelming:

$$\begin{aligned}\Pr[\mathsf{E}_{\mathsf{corr},\mathsf{i}}] &= \prod_{k=1}^{t} \Pr[\mathsf{E}_{\mathsf{corr},\mathsf{i}_k}] \\ &\geq (\min_k \{\Pr[\mathsf{E}_{\mathsf{corr},\mathsf{i}_k}]\})^t \\ &\geq (1-\mathsf{negl}(\lambda))^{O(1)} \\ &\geq 1-\mathsf{negl}(\lambda)\cdot O(1) \\ &\geq 1-\mathsf{negl}(\lambda).\end{aligned}$$

Finally, we observe that, for $n, t \in O(1)$, $\binom{n}{t} \in O(1)$. Denoting $\mathsf{E}_{\mathsf{corr}}$ as the event that all sub-proofs are accepting, we argue that $\Pr[\mathsf{E}_{\mathsf{corr}}]$ is overwhelming:

$$\begin{aligned}\Pr[\mathsf{E}_{\mathsf{corr}}] &= \prod_{i=1}^{\binom{n}{t}} \Pr[\mathsf{E}_{\mathsf{corr},\mathsf{i}}] \\ &\geq (\min_i \{\Pr[\mathsf{E}_{\mathsf{corr},\mathsf{i}}]\})^{\binom{n}{t}} \\ &\geq (1-\mathsf{negl}(\lambda))^{O(1)} \\ &\geq 1-\mathsf{negl}(\lambda).\end{aligned}$$

□

Theorem C.7. *The construction in Figure 8.2 is a zero-knowledge t-out-of-n NBB (ZK, AoK)-combiner for any NP language $\mathcal{L}$, as per Definition C.3, for $t > \lfloor \frac{n}{2} \rfloor$ and $n, t \in O(1)$.*

Proof (Theorem C.7). The structure of this proof is the same as the one sketched in Section 8.4.3. First, we construct a simulator $\mathcal{S}^T$, and argue that either its output is indistinguishable from the output of the real prover, or there exists at least one reduction to zero-knowledge of one of the candidates whose indices are in T. We describe such a $\mathcal{S}^T$ in Lemma C.10. With that, we can use exactly the same the combinatorics argument used to prove Lemma C.6. □

Lemma C.10. *There exists a simulator $\mathcal{S}^T$ for which either $\{\mathtt{REAL}\} \approx_c \{\mathtt{IDEAL}_{\mathcal{S}^T}\}$, or there exists a reduction to zero-knowledge of at least one of the candidates $\{\Pi_k\}_{k\in T}$.*

Proof (Lemma C.10). Let $T = \{i_1, \ldots, i_t\}$ be a set of indices sorted in lexicographic order identifying candidates $\Pi_{i_1}, \ldots, \Pi_{i_t}$. $\mathcal{S}^T = (\mathcal{S}^{T,0}, \mathcal{S}^{T,1})$ is constructed in Figure C.10.

Simulator $\mathcal{S}^T$ for $\mathsf{C}_{\mathsf{PoP}}$

$\mathcal{S}^{T,0}(1^l, 1^\lambda)$:

- For $l \in T$, compute $(pp_l, \tau_l) \leftarrow\$ \mathcal{S}_l^0(1^l, 1^\lambda)$.
- For $i \in [n] \setminus T$, compute $pp_i \leftarrow\$ \mathsf{setup}_i(1^l, 1^\lambda)$.
- Compute $pp \leftarrow \{pp_k\}_{k\in[n]}$, $\tau \leftarrow \{\tau_k\}_{k\in[t]}$.
- Return (pp, τ).

$\mathcal{S}^{T,1}(pp, \tau, x)$:

- Index all possible subsets containing t of the n candidates with set $S = \{S^i\}_{i\in\binom{n}{t}}$.
- Denote S^{T_k} as the set of all proofs containing candidate i_k but not candidates in T_{k-1} (*i.e.*, $S^{T_k} = \{S^i \mid \forall i \in \binom{n}{t}\ s.t.\ i_k \in S^i \wedge S^i \cap T_{k-1} = \emptyset\}$). Denote every $S^i \in S^{T_k}$ as $S^{T_k,i}$.
- For $k \in [t]$, for $S^{T_k,i} \in S^{T_k}$, simulate as follows:
  - Compute $\pi_{T_k,i} \leftarrow\$ \mathcal{S}_{i_k}^1(pp_{i_k}, \tau_{i_k}, x_{T_k,i})$, where $x_{T_k,i} \leftarrow (\{\mathcal{V}_l, pp_l\}_{l\in S^{T_k,i} \wedge l < i_k}, x)$, and proceed as follows:

* If $\pi_{T_k,i}$ is the outermost proof, set $\pi^{\mathsf{PoP}}_{T_k,i} \leftarrow \pi_{T_k,i}$.
* Otherwise, obtain $\pi^{\mathsf{PoP}}_{T_k,i}$ by recursively running the prover algorithm $\{\mathcal{P}_l\}_{l \in S^{T_k,i} \wedge l > i_k}$, using $\pi_{T_k,i}$ as a valid witness (*i.e.*, an accepting proof), as per the combiner description.

Figure C.10: Description of $\mathcal{S}^T$ for $\mathsf{C}_{\mathsf{PoP}}$.

The proof of this lemma goes through $t+1$ hybrid experiments, where hybrid $\mathcal{H}_0$ is the same as the real world, and hybrid $\mathcal{H}_t$ is the same as the ideal world. We denote the output of the adversary in hybrid $\mathcal{H}_i$ as $\mathsf{out}^{\mathcal{H}_i}$, for $i \in \{0, 1, \ldots, t\}$. Formally we prove that, for $j \in \{1, \ldots, t\}$ and for any algorithm $\mathcal{A}$ running in time τ, the following holds:

$$\left| \Pr[\mathcal{A}(\mathsf{out}^{\mathcal{H}_{j-1}}) = 1] - \Pr[\mathcal{A}(\mathsf{out}^{\mathcal{H}_j}) = 1] \right| \leq \mathsf{negl}(\lambda).$$

We define the hybrids in Figure C.11.

Hybrid chain for zero-knowledge of $\mathsf{C}_{\mathsf{PoP}}$

$\mathcal{H}_0$, $\boxed{\mathcal{H}_j}$

1. $pp \leftarrow\!\$\ \mathsf{C}_{\mathsf{setup}}(1^l, 1^\lambda)$.

1. $(pp_k, \tau_k) \leftarrow\!\$\ \mathcal{S}^0_k(1^l, 1^\lambda)$ for $i \in [j]$. $pp_i \leftarrow\!\$\ \mathsf{setup}_i(1^l, 1^\lambda)$ for $i \in [n] \setminus [j]$. $pp \leftarrow \{pp_i\}_{i \in [n]}$.

2. Index all possible subsets containing t of the n candidates with set $S = \{S^i\}_{i \in \binom{n}{t}}$.

3. For $S^i \in S$, compute π_i as the honest prover.

3. Denote $S^{T_k} = \{S^i \mid \forall i \in \binom{n}{t}\ s.t.\ i_k \in S^i \wedge S^i \cap T_{k-1} = \emptyset\}$. We denote every $S^i \in S^{T_k}$ as $S^{T_k,i}$.
   For $k \in [j]$, for $S^{T_k,i} \in S^{T_k}$, simulate as follows:
   - Compute $\pi_{T_k,i} \leftarrow\!\$\ \mathcal{S}^1_{i_k}(pp_{i_k}, \tau_{i_k}, x_{T_k,i})$, where $x_{T_k,i} \leftarrow (\{\mathcal{V}_l, pp_l\}_{l \in S^{T_k,i} \wedge l < i_k}, x)$, and proceed as follows:
     - If $\pi_{T_k,i}$ is the outermost proof, set $\pi^{\mathsf{PoP}}_{T_k,i} \leftarrow \pi_{T_k,i}$.
     - Otherwise, obtain $\pi^{\mathsf{PoP}}_{T_k,i}$ by recursively running the prover algorithm $\{\mathcal{P}_l\}_{l \in S^{T_k,i} \wedge l > i_k}$, using $\pi_{T_k,i}$ as a valid witness (*i.e.*, an accepting proof), as per the combiner description.

   For $S^i \in S \setminus \{S^{T_k}\}_{k \in [j]}$, compute π_i as the honest prover of the combiner.

4. Set $\pi \leftarrow \{\pi^{\mathsf{PoP}}_i\}_{i \in \binom{n}{t}}$.

5. Run $\mathcal{A}(pp, x, \pi)$.

Figure C.11: Hybrid experiments $\mathcal{H}_0$, $\mathcal{H}_j$, for $1 \leq j \leq t$, for zero-knowledge of $\mathsf{C}_{\mathsf{PoP}}$.

Intuitively, in $\mathcal{H}_j$ we simulate all the proofs containing candidate i_j that were not already simulated in any previous hybrid. If the proof to be simulated is the outermost one, the dependency on the witness is removed by running the simulator for the outermost proof system. In any other case, the simulation generates an accepting intermediate proof without knowing its respective witness, which is used as a valid witness by recursively running the prover algorithm as per the combiner description.

Finally, we argue that the simulator is indeed independent from the witness (*i.e.*, that all proofs were simulated) through a simple combinatorics argument. Hence, if no reduction occurred throughout the hybrid chain, the simulator successfully removes the dependency of the witness from all the $\binom{n}{t}$ proofs. We formalize the above in Lemma C.11 and Lemma C.12.

On the indistinguishability of the hybrid chain. If no reduction occurred, each simulated proof yields an advantage to the adversary of $\mathsf{negl}(\lambda)$. Hence, the advantage of the adversary from $\mathcal{H}_0$ to $\mathcal{H}_t$ is upper bounded by $\binom{n}{t} \cdot \mathsf{negl}(\lambda)$. In order for this quantity to be negligible, it should be the case that $\binom{n}{t} \in \mathsf{poly}(\lambda)$. Because we have $n, t \in O(1)$, $\binom{n}{t} \in O(1)$, satisfying our requirement. □

Lemma C.11 (Transition from $\mathcal{H}_{j-1}$ to $\mathcal{H}_j$, for $1 \leq j \leq t$)**.** *If there exists an algorithm $\mathcal{A}$ running in time τ for which $\mathsf{out}^{\mathcal{H}_{j-1}} \not\approx_c \mathsf{out}^{\mathcal{H}_j}$, there exists an algorithm Red_{i_j} running in time $\mathsf{poly}(\tau)$ that breaks zero-knowledge of the i_j-th scheme.*

Proof (Lemma C.11). We describe a reduction Red_{i_j} given $\mathcal{A}$ as in the theorem statement:

Reduction Red_{i_j} to zero-knowledge of the i_j-th candidate

- Receive public parameters pp_{i_j} from the challenger $\mathcal{C}$.
- Compute $(pp_k, \tau_k) \leftarrow\$ \mathcal{S}_k^0(1^l, 1^\lambda)$ for $k \in T_{j-1}$. Compute $pp_i \leftarrow\$ \mathsf{setup}_i(1^l, 1^\lambda)$ for $i \in [n] \setminus T_j$. Set $pp \leftarrow \{pp_k\}_{k\in[n]}$.
- Compute $(x, w) \leftarrow\$ \mathcal{A}(pp)$.
- Index all possible subsets containing t of the n candidates with set $S = \{S^i\}_{i\in\binom{n}{t}}$.
- Denote $S^{T_k} = \{S^i \mid \forall i \in \binom{n}{t}\ s.t.\ i_k \in S^i \wedge S^i \cap T_{k-1} = \emptyset\}$. We denote every $S^i \in S^{T_k}$ as $S^{T_k,i}$.
- For $k \in [j-1]$, for $S^{T_k,i} \in S^{T_k}$, simulate proofs as per the description of $\mathcal{H}_{j-1}$:
- Send (x, w) to $\mathcal{C}$, and receive proofs $\{\pi_l^{\mathsf{PoP}}\}_{l\in S^{T_j}}$. These proofs are either generated by running the honest prover of the combiner, or by running $\mathcal{S}_{i_j}^1$, as described in $\mathcal{H}_j$.
- For $S^i \in S \setminus \{S^{T_k}\}_{k\in[j]}$, compute π_i as the honest prover of the combiner.
- Set $\pi \leftarrow \{\pi_i^{\mathsf{PoP}}\}_{i\in\binom{n}{t}}$.
- Run $\mathcal{A}(pp, x, \pi)$ and output whatever $\mathcal{A}$ outputs.

Figure C.12: Reduction Red_{i_j} to zero-knowledge of the i_j-th candidate.

If $\mathcal{C}$ provides real proofs, we are in hybrid $\mathcal{H}_{j-1}$. Otherwise, we are in hybrid $\mathcal{H}_j$. Therefore, Red_{i_j} retains the same distinguishing advantage of $\mathcal{A}$, breaking multi-proof ZK of candidate Π_{i_j}[38].

□

Lemma C.12. *The output of $\mathcal{S}^T$ is independent from the witness.*

Proof (Lemma C.12). First, we observe that $\mathcal{S}^T$ exclusively simulates all the proofs that include any candidate $\{\Pi_k\}_{k\in T}$ as part of their generation procedure. In order to show that the output of $\mathcal{S}^T$ is independent from the witness, we define a set $\bar{S}$ containing proofs that do not include any of the candidates in T as part of their generation procedure. The cardinality of $\bar{S}$ is $\binom{n-t}{t}$. Since $t > \lfloor \frac{n}{2} \rfloor$ by assumption, it trivially holds that $t > n - t$. This implies that $|\bar{S}| = 0$. Therefore, all proofs are simulated. □

Theorem C.8. *The construction in Figure 8.2 is an argument-of-knowledge t-out-of-n NBB (ZK, AoK)-combiner for any NP language $\mathcal{L}$, as per Definition C.2, for $t > \lfloor \frac{n}{2} \rfloor$ and $n, t \in O(1)$.*

Proof (Theorem C.8). Following Section 8.4.3, we construct an extractor $\mathcal{E}^T$ and prove in Lemma C.13 that either it successfully extracts a witness, or there exists at least one reduction to AoK of one of the candidates whose indices are in T. With that, we can use exactly the same the combinatorics argument used to prove Lemma C.9. □

Lemma C.13. *There exists an extractor $\mathcal{E}^T$ for which either a witness w is successfully extracted from an accepting proof, or there exists a reduction to argument-of-knowledge of at least one of the candidates $\{\Pi_k\}_{k\in T}$.*

Proof (Lemma C.13). Let $T = \{i_1, \ldots, i_t\}$ be a set of indices sorted in lexicographic order identifying candidates $\Pi_{i_1}, \ldots, \Pi_{i_t}$. The extractor $\mathcal{E}^T = (\mathcal{E}^{T,0}, \mathcal{E}^{T,1})$ is constructed as follows:

[38] For convenience, we use multi-proof zero-knowledge. We remark that this is for compactness of notation in the setup generation algorithm of the combiner, and comes without loss of generality. Indeed, the same can be achieved without multi-proof ZK by generating a different trapdoor for each individual proof.

Extractor $\mathcal{E}^T$ for $\mathsf{C}_{\mathsf{PoP}}$

$\mathcal{E}^{T,0}(1^l, 1^\lambda)$:

- For $k \in T$, compute $(pp_k, \xi_k) \leftarrow\!\!\$\ \mathcal{E}_k^0(1^l, 1^\lambda)$.
- Compute $pp \leftarrow \{pp_k\}_{k\in[n]}$, $\xi \leftarrow \{\xi_k\}_{k\in T}$.
- Return (pp, ξ).

$\mathcal{E}^{T,1}(pp, \xi, x, \pi)$:

- Parse π as $\{\pi_i^{\mathsf{PoP}}\}_{i\in\binom{n}{t}}$.
- Index all possible subsets containing t of the n candidates with set $S = \{S^i\}_{i\in\binom{n}{t}}$.
- Select the proof π for which $S_i = T$, and set $\pi_{i_t} \leftarrow \pi$.
- For $k \in \{0, \ldots, t-2\}$, compute $\pi_{i_{t-k-1}} \leftarrow \mathcal{E}^1_{i_{t-k}}(pp_{i_{t-k}}, \xi_{i_{t-k}}, x_{i_{t-k}}, \pi_{i_{t-k}})$.
- Output $w \leftarrow \mathcal{E}^1_{i_1}(pp_{i_1}, \xi_{i_1}, x, \pi_{i_1})$

Figure C.13: Description of $\mathcal{E}^T$ for $\mathsf{C}_{\mathsf{PoP}}$.

Intuitively, the extractor picks the specific proof that was generated using all the candidates in T, and runs the extractors in T recursively until a witness is extracted. Since π is parsed as a set of $\binom{n}{t}$ proofs, and these proofs are enumerated by considering all the subsets of size t out of n, such a proof always exists.

As per Definition C.1, an adversary may break argument-of-knowledge either by detecting the introduction of trapdoor setups, or by outputting an accepting proof that causes any extractor to fail. If either occurs, we exhibit a reduction to AoK of that specific candidate. The former step is the same for all our combiners, and is formally proved in Lemma C.8. Intuitively, we replace one setup at a time, and if the adversary distinguishes two adjacent hybrids we have a reduction.

Given that no reduction occurred after introducing the trapdoor setups, we proceed with analyzing $\mathcal{E}^T$. The only way $\mathcal{E}^T$ can output w is by recursively extracting from the proof π that was generated by running all the schemes in T. Because all candidates work for any NP language $\mathcal{L}$, they also work with the specific languages we have in Figure 8.2. Therefore, in order for $\mathcal{E}^T$ to fail, it suffices for one of the sub-extraction procedures to fail. Let i^* be an index for which $\mathcal{E}^1_{i^*}$ fails to extract from (x_{i^*}, π_{i^*}). Any adversary that outputs an accepting proof (x, π) for the combiner for which the extraction procedure, upon reaching scheme Π_{i^*}, fails to extract from (x_{i^*}, π_{i^*}), immediately causes $\mathcal{E}^T$ to fail. This yields a reduction to AoK of scheme Π_{i^*}. □

C.7 Combiner from MPC-in-the-head: formal treatment

In this section, we present formal theorems and proofs for our combiner presented in Section 8.6.

C.7.1 $\mathsf{C}_{\mathsf{MPC}}$ is a (ZK, Sound)-combiner

In order to claim that $\mathsf{C}_{\mathsf{MPC}}$ (as in Figure 8.3) is a t-out-of-n NB (ZK, Sound)-combiner, we need to argue the following:

- $\mathsf{C}_{\mathsf{MPC}}$ is a correct n-candidate $\mathcal{F}_{\mathsf{NI}}$-combiner for any NP language $\mathcal{L}$, as per Definition 8.1.
- $\mathsf{C}_{\mathsf{MPC}}$ is a sound t-out-of-n NB (ZK, Sound)-combiner for any NP language $\mathcal{L}$, as per Definition 8.4.
- $\mathsf{C}_{\mathsf{MPC}}$ is a zero-knowledge t-out-of-n NB (ZK, Sound)-combiner for any NP language $\mathcal{L}$, as per Definition 8.5.

Below, we provide formal theorems and proofs for each of the aforementioned items.

Theorem C.9. *The construction in Figure 8.3 is a correct n-candidate $\mathcal{F}_{\mathsf{NI}}$-combiner for any NP language $\mathcal{L}$, as per Definition 8.1, for $n, t \in \mathsf{poly}(\lambda)$.*

Proof (Theorem C.9). In order for the construction to be an n-candidate $\mathcal{F}_{\mathsf{NI}}$-combiner, the following has to hold:

$$\Pr\left[\mathsf{C}_{\mathsf{vrfy}}(pp, x, \mathsf{C}_{\mathsf{prove}}(pp, x, w)) = 1 \mid pp \leftarrow\!\!\$\ \mathsf{C}_{\mathsf{setup}}(1^{|x|}, 1^{\lambda})\right] \geq 1 - \mathsf{negl}(\lambda).$$

Intuitively, this happens if (i) all sub-proofs are accepting, and (ii) the commitment corresponding to the outgoing messages are consistent with the commitment corresponding to the inbound messages. Because Π_{MPC} is honestly executed and the commitment are computed honestly, then (ii) holds. Hence, it only remains to show that all the sub-proofs are accepting:

1. By perfect correctness of Π_{MPC}, f_x (as described in Section 8.6) obtains $w' \leftarrow \mathtt{reconstruct}(\{w_j\}_{j\in[t]})$, where $(w_1, \ldots, w_n) \leftarrow\!\!\$\ \mathtt{share}(w)$. By correctness of secret sharing scheme $\mathtt{SS}$, we have $w' = w$. When $(x, w) \in \mathscr{R}_{\mathcal{L}}$, f_x outputs 1.

2. By Definition 2.16, for $k \in [n]$, $\mathcal{V}_k$ will output 1 with probability at least $1 - \mathsf{negl}(\lambda)$ if and only if $(x'_k, w'_k) \in \mathscr{R}_{k,\mathcal{L}_{\mathsf{MPC}}}$ and Π_k is for $\mathcal{L}_{k,\mathsf{MPC}}$. Since $\Pi_k \in \mathcal{F}_{\mathsf{NI}}$ works for any NP language, it will also work for $\mathcal{L}_{k,\mathsf{MPC}}$. By correctness of Π_{MPC}, $(x'_k, w'_k) \in \mathscr{R}_{\mathcal{L}_{k,\mathsf{MPC}}}$. Therefore, all $\mathcal{V}_k$ will output 1 with probability at least $1 - \mathsf{negl}(\lambda)$.

Hence, when $(x, w) \in \mathscr{R}_{\mathcal{L}}$, the probability $\mathsf{C}_{\mathsf{vrfy}}$ accepts is the same as the probability of having all sub-proofs that are accepting, which is exactly the same as Event 1 in the proof of Theorem C.1. □

Theorem C.10. *The construction in Figure 8.3 a sound t-out-of-n NB (ZK, Sound)-combiner for any NP language $\mathcal{L}$ for $t > \lfloor \frac{n}{2} \rfloor$ and $n, t \in \mathsf{poly}(\lambda)$.*

Proof (Theorem C.10). The proof is very similar to the proof of Theorem C.2. The only difference is that we need to argue that there must be $n-t+1$ reductions to the soundness of the candidates by relying on the statistical binding of the non-interactive commitment scheme and on the $\mathcal{K}$-robustness of the MPC protocol.

Formally, assume by contradiction that there are at most $n - t$ reductions. This means there will be $n-(n-t) = t$ sub-statements x'_k that are in the language $\mathcal{L}_{k,\mathsf{MPC}}$, which means that there are t parties that output 1. By assumption, we know $x \notin \mathcal{L}$. This means that $(x, w') \notin \mathscr{R}_{\mathcal{L}}$ holds for any w'. In this case, $f_x(w_1, \ldots, w_n)$ must output 0.

Because the verifier accepts, the commitments corresponding to the outgoing messages are consistent with the commitments corresponding to the inbound messages. In this case, we claim that all pairs of views are consistent. Let $\mathsf{E}_{\mathsf{bad}}$ be the event for an efficient malicious prover $\tilde{\mathcal{P}}$ produces two consistent commitments that open to inconsistent views. This means that there exist two openings o_0, o_1 for the same commitment that open to the same value. Given that $\mathtt{NIC}$ is statistically binding, this event happens only with negligible probability.

Hence, conditioning on $\mathsf{E}_{\mathsf{bad}}$ not happening, all views come from the same execution of Π_{MPC} realizing f_x. In such an execution, by perfect $\mathcal{K}$-robustness, when there are at most $\mathcal{K}$ corrupted parties (controlled by a malicious adversary) that output 1, the other t parties must output 0. Since perfect $\mathcal{K}$-robustness holds unconditionally, we have a contradiction, as in this case the verification procedure of the combiner must have been rejecting, because some parties are outputting 0. This is the case as, in order to break $\mathcal{K}$-robustness, at least one more party should output 1. But the existence of such a party would imply the existence of more reductions to the underlying primitives, contradicting the assumption of having at most $n - t$ reductions.

Therefore, any accepting proof π for the combiner for a statement $x \notin \mathcal{L}$ implies the existence of at least $n - t + 1$ reductions to soundness of the underlying candidates. Once we have at least $n - t + 1$ reductions, the proof is the same as the proof of Theorem C.2. □

Theorem C.11. *The construction in Figure 8.3 is a zero-knowledge t-out-of-n NB (ZK, Sound)-combiner for any NP language $\mathcal{L}$, as per Definition 8.5, for $t > \lfloor \frac{n}{2} \rfloor$ and $n, t \in \mathsf{poly}(\lambda)$.*

Proof (Theorem C.11). The structure of this proof is nearly the same as the one of Theorem C.3: in order to obtain $n - t + 1$ reductions to zero-knowledge of the candidates, we first construct a simulator $\mathcal{S}^T$ and then use exactly the same the combinatorics argument used to prove Lemma C.6. Again, $\mathcal{S}^T$ should be

such that either its output is indistinguishable from the output of the real prover, or there exists at least one reduction to zero-knowledge of one of the candidates whose indices are in T. However, this construction additionally requires the computational hiding of the non-interactive commitment scheme, which relies both the soundness as well as the zero-knowledge of $n - t + 1$ of the candidates. Hence, within the proof, we additionally argue that, if the computational hiding of the commitment is violated, we would have at least $n-t+1$ reductions to either soundness or zero-knowledge of the input candidates. We expand this argument and describe $\mathcal{S}^T$ in Lemma C.14. □

Lemma C.14. *There exists a simulator $\mathcal{S}^T$ of $\mathsf{C}_{\mathsf{MPC}}$ for which either $\{\mathtt{REAL}\} \approx_c \{\mathtt{IDEAL}_{\mathcal{S}^T}\}$, or there exists a reduction to zero-knowledge of at least one of the candidates $\{\Pi_k\}_{k\in[T]}$.*

Proof. Let $T = \{i_1, \ldots, i_t\}$ be a set of indices sorted in lexicographic order identifying candidates $\Pi_{i_1}, \ldots, \Pi_{i_t}$. Assume $\mathtt{Gen}$ is an algorithm that can sample uniformly random elements in the space of shares of w. $\mathcal{S}^T = (\mathcal{S}^{T,0}, \mathcal{S}^{T,1})$ is constructed as follows:

Simulator $\mathcal{S}^T$ for $\mathsf{C}_{\mathsf{MPC}}$

$\mathcal{S}^{T,0}(1^l, 1^\lambda)$:

- For $l \in T$, compute $(pp_l, \tau_l) \leftarrow\!\$\ \mathcal{S}_l^0(1^l, 1^\lambda)$.
- For $i \in [n] \setminus T$, compute $pp_i \leftarrow\!\$\ \mathsf{setup}_i(1^l, 1^\lambda)$.
- Compute $pp_{\mathtt{NIC}} \leftarrow\!\$\ \mathsf{setup}(1^\lambda)$.
- Compute $pp \leftarrow \{pp_k\}_{k\in[n]} \cup \{pp_{\mathtt{NIC}}\}$, $\tau \leftarrow \{\tau_k\}_{k\in T}$.
- Return (pp, τ).

$\mathcal{S}^{T,1}(pp, \tau, x)$:

- Sample $\{w_j\}_{j\in[n]\setminus T} \leftarrow\!\$\ \mathtt{Gen}(1^\lambda)$. Run the $\mathcal{K}$-privacy simulator $\mathcal{S}_{\mathsf{MPC}}([n] \setminus T, x, \{w_j\}_{j\in[n]\setminus T}, 1)$ of Π_{MPC} to obtain the simulated views $\{\mathsf{view}_j\}_{j\in[n]\setminus T}$.
- For $i \in T$, for $j \in [n]$, set $\{\mathsf{msg}^k_{i\to j}\}_{k\in[R]} = 0$ in view_i and view_j. Complete view_i to be consistent with $\{\mathsf{view}_j\}_{j\in[n]\setminus T}$.
- For $i \in [n]$, $j \in [n]$, compute $(com_{i,j}, o_{i,j}) \leftarrow\!\$\ \mathsf{Com}(pp_{\mathtt{NIC}}, \{\mathsf{msg}^k_{i\to j}\}_{k\in[R]})$, and $(com_{j,i}, o_{j,i}) \leftarrow\!\$\ \mathsf{Com}(pp_{\mathtt{NIC}}, \{\mathsf{msg}^k_{j\to i}\}_{k\in[R]})$.
- For $i \in [n]$, set $x'_i \leftarrow (\{com_{i,j}\}_{j\in[n]}, \{com_{j,i}\}_{j\in[n]})$. For $l \in [n] \setminus T$, $w'_l \leftarrow (\mathsf{view}_l, \{o_{l,j}\}_{j\in[n]}, \{o_{j,l}\}_{j\in[n]})$.
- For $l \in T$, compute $\pi_l \leftarrow\!\$\ \mathcal{S}_l^1(pp_l, \tau_l, (pp_{\mathtt{NIC}}, x'_l))$. For $i \in [n] \setminus T$, compute $\pi_i \leftarrow\!\$\ \mathcal{P}_i(pp_i, (pp_{\mathtt{NIC}}, x'_i), w'_i)$.
- Return $(\{\pi_k\}_{k\in[n]}, \{x'_k\}_{k\in[n]})$.

Figure C.14: Description of $\mathcal{S}^T$ for $\mathsf{C}_{\mathsf{MPC}}$.

The proof of this lemma goes through $t + 4$ hybrid experiments, where hybrid $\mathcal{H}_0$ is the same as the real world, and hybrid $\mathcal{H}_{t+3}$ is the same as the ideal world. We denote the output of the adversary in hybrid $\mathcal{H}_i$ as $\mathsf{out}^{\mathcal{H}_i}$, for $i \in \{0, 1, \ldots, t + 3\}$. Formally we prove that, for $j \in \{1, \ldots, t + 3\}$ and for any algorithm $\mathcal{A}$ running in time τ, the following holds:

$$\left|\Pr[\mathcal{A}(\mathsf{out}^{\mathcal{H}_{j-1}}) = 1] - \Pr[\mathcal{A}(\mathsf{out}^{\mathcal{H}_j}) = 1]\right| \leq \mathsf{negl}(\lambda).$$

We define the hybrids in Figure C.15.

Hybrid chain for zero-knowledge of $\mathsf{C}_{\mathsf{MPC}}$

$\mathcal{H}_0$, $\boxed{\mathcal{H}_j}$, $\boxed{\mathcal{H}_{t+1}}$, $\boxed{\mathcal{H}_{t+2}}$, $\boxed{\mathcal{H}_{t+3}}$

1. $pp_k \leftarrow\!\$\ \mathsf{setup}_k(1^l, 1^\lambda)$ for $k \in [n]$. $pp_{\mathtt{NIC}} \leftarrow\!\$\ \mathsf{setup}(1^\lambda)$. $pp \leftarrow \{pp_k\}_{k\in[n]} \cup \{pp_{\mathtt{NIC}}\}$.

1. $(pp_k, \tau_k) \leftarrow\$ \mathcal{S}_k^0(1^l, 1^\lambda)$ for $k \in T_j$. $pp_i \leftarrow\$ \mathsf{setup}_i(1^l, 1^\lambda)$ for $i \in [n] \setminus T_j$. $pp_{\mathtt{NIC}} \leftarrow\$ \mathsf{setup}(1^\lambda)$. $pp \leftarrow \{pp_k\}_{k\in[n]} \cup \{pp_{\mathtt{NIC}}\}$.
2. Compute $(x, w) \leftarrow\$ \mathcal{A}(pp)$
3. $\{w_i\}_{i\in[n]} \leftarrow\$ \mathtt{share}(w)$. Run MPC protocol Π_{MPC} with input $(w_1, \ldots, w_n)$ to obtain $\{\mathsf{view}_i\}_{i\in[n]}$.

3. • $\{w_i\}_{i\in[n]} \leftarrow\$ \mathtt{share}(w)$. Run MPC protocol Π_{MPC} with input $(w_1, \ldots, w_n)$ to obtain $\{\mathsf{view}_i\}_{i\in[n]}$.
   • For $i \in T$, for $j \in [n]$, set $\{\mathsf{msg}_{i\to j}^k\}_{k\in[R]} = 0$ in view_i and view_j.

   3. • $\{w_i\}_{i\in[n]} \leftarrow\$ \mathtt{share}(w)$. Run the $\mathcal{K}$-privacy simulator $\mathcal{S}_{\mathsf{MPC}}([n] \setminus T, x, \{s_j\}_{j\in[n]\setminus T}, 1)$ of Π_{MPC} to obtain the simulated views $\{\mathsf{view}_j\}_{j\in[n]\setminus T}$.
      • For $i \in T$, for $j \in [n]$, set $\{\mathsf{msg}_{i\to j}^k\}_{k\in[R]} = 0$ in view_i and view_j. Complete view_i to be consistent with $\{\mathsf{view}_j\}_{j\in[n]\setminus T}$.

      3. • Sample $\{w_j\}_{j\in[n]\setminus T} \leftarrow\$ \mathtt{Gen}(1^\lambda)$. Run the $\mathcal{K}$-privacy simulator $\mathcal{S}_{\mathsf{MPC}}([n] \setminus T, x, \{s_j\}_{j\in[n]\setminus T}, 1)$ of Π_{MPC} to obtain the simulated views $\{\mathsf{view}_j\}_{j\in[n]\setminus T}$.
         • For $i \in T$, for $j \in [n]$, set $\{\mathsf{msg}_{i\to j}^k\}_{k\in[R]} = 0$ in view_i and view_j. Complete view_i to be consistent with $\{\mathsf{view}_j\}_{j\in[n]\setminus T}$.

4. For $i \in [n]$, $j \in [n]$, compute $(com_{i,j}, o_{i,j}) \leftarrow\$ \mathsf{Com}(pp_{\mathtt{NIC}}, \{\mathsf{msg}_{i\to j}^k\}_{k\in[R]})$, and $(com_{j,i}, o_{j,i}) \leftarrow \mathsf{Com}(pp_{\mathtt{NIC}}, \{\mathsf{msg}_{j\to i}^k\}_{k\in[R]})$.
5. For $i \in [n]$, set $x'_i \leftarrow (\{com_{i,j}\}_{j\in[n]}, \{com_{j,i}\}_{j\in[n]})$ and $w'_i \leftarrow (\mathsf{view}_i, \{o_{i,j}\}_{j\in[n]}, \{o_{j,i}\}_{j\in[n]})$.
6. $\pi_k \leftarrow\$ \mathcal{P}_k(pp_k, (pp_{\mathtt{NIC}}, x'_k), w'_k)$ for $k \in [n]$. $\pi \leftarrow (\{\pi_k\}_{k\in[n]}, \{x'_k\}_{k\in[n]})$.

6. $\pi_k \leftarrow\$ \mathcal{S}_k^1(pp_k, \tau_k, (pp_{\mathtt{NIC}}, x'_k))$ for $k \in T_j$. $\pi_i \leftarrow\$ \mathcal{P}_i(pp_i, (pp_{\mathtt{NIC}}, x'_i), w'_i)$ for $i \in [n] \setminus T_j$. $\pi \leftarrow (\{\pi_k\}_{k\in[n]}, \{x'_k\}_{k\in[n]})$.

   6. $\pi_k \leftarrow\$ \mathcal{S}_k^1(pp_k, \tau_k, (pp_{\mathtt{NIC}}, x'_k))$ for $k \in T$. $\pi_i \leftarrow\$ \mathcal{P}_i(pp_i, (pp_{\mathtt{NIC}}, x'_i), w'_i)$ for $i \in [n] \setminus T$. $\pi \leftarrow (\{\pi_k\}_{k\in[n]}, \{x'_k\}_{k\in[n]})$.

7. Run $\mathcal{A}(pp, x, \pi)$.

Figure C.15: Hybrid experiments $\mathcal{H}_0$, $\mathcal{H}_j$, $\mathcal{H}_{t+1}$, $\mathcal{H}_{t+2}$, $\mathcal{H}_{t+3}$, for $1 \le j \le t$, for zero-knowledge of $\mathsf{C}_{\mathsf{MPC}}$.

Intuitively, in the first t hybrids, the real proofs generated by the candidates in T are gradually replaced with simulated proofs generated by their respective simulator. In $\mathcal{H}_{t+1}$, we use the computational hiding property of $\mathtt{NIC}$ to replace the messages sent by honest parties with the message 0, and update the views accordingly so the commitments remain consistent. In $\mathcal{H}_{t+2}$, we run $\mathcal{S}_{\mathsf{MPC}}([n] \setminus T, x, \{w_j\}_{j\in[n]\setminus T}, 1)$ in place of the MPC protocol to obtain simulated views for the semi-honest parties. By $\mathcal{K}$-privacy of Π_{MPC}, such a simulator exists. Finally, in $\mathcal{H}_{t+3}$, we replace $n-t$ shares of the real witness w with $n-t$ random elements in the same space of shares of w. This is possible as shares of the real witness are used in at most $n-t$ of the prover algorithms of the candidates. Because $t > \lfloor \frac{n}{2} \rfloor$, the adversary can leak at most $\lfloor \frac{n}{2} \rfloor$ of the shares, and is unable to determine whether these are shares of w or random elements. Formally, this lemma is proved in Lemma C.15, Lemma C.16, Lemma C.17, and Lemma C.18. □

Lemma C.15 (Transition from $\mathcal{H}_{j-1}$ to $\mathcal{H}_j$, for $1 \le j \le t$)**.** *If there exists an algorithm $\mathcal{A}$ running in time τ for which $\mathsf{out}^{\mathcal{H}_{j-1}} \not\approx_c \mathsf{out}^{\mathcal{H}_j}$, there exists an algorithm Red_{i_j} running in time $\mathsf{poly}(\tau)$ that breaks zero-knowledge of the i_j-th scheme.*

Proof. Given $\mathcal{A}$ as in the theorem statement, we describe Red_{i_j} in what follows:

Reduction Red_{i_j} to zero-knowledge of the i_j-th candidate

- Receive public parameters pp_{i_j} from the challenger $\mathcal{C}$.

- Compute $(pp_k, \tau_k) \leftarrow\$ \mathcal{S}_k^0(1^l, 1^\lambda)$ for $k \in T_{j-1}$. Compute $pp_i \leftarrow\$ \mathsf{setup}_i(1^l, 1^\lambda)$ for $i \in [n] \setminus T_j$. Compute $pp_{\mathtt{NIC}} \leftarrow\$ \mathsf{setup}(1^\lambda)$. Set $pp \leftarrow \{pp_k\}_{k\in[n]} \cup \{pp_{\mathtt{NIC}}\}$.
- Compute $(x, w) \leftarrow\$ \mathcal{A}(pp)$.
- Compute $\{w_i\}_{i\in[n]} \leftarrow\$ \mathtt{share}(w)$. Run MPC protocol Π_{MPC} with input $(w_1, \ldots, w_n)$ to obtain $\{\mathsf{view}_i\}_{i\in[n]}$.
- For $i \in [n]$, $j \in [n]$, compute $(com_{i,j}, o_{i,j}) \leftarrow\$ \mathsf{Com}(pp_{\mathtt{NIC}}, \{\mathsf{msg}_{i\to j}^k\}_{k\in[R]})$, and $(com_{j,i}, o_{j,i}) \leftarrow\$ \mathsf{Com}(pp_{\mathtt{NIC}}, \{\mathsf{msg}_{j\to i}^k\}_{k\in[R]})$.
- For $i \in [n]$, set $x'_i \leftarrow (\{com_{i,j}\}_{j\in[n]}, \{com_{j,i}\}_{j\in[n]})$ and $w'_i \leftarrow (\mathsf{view}_i, \{o_{i,j}\}_{j\in[n]}, \{o_{j,i}\}_{j\in[n]})$.
- Send $\{x'_{i_j}, w'_{i_j}\}$ to the challenger $\mathcal{C}$, and receive proof π_{i_j}. This proof is generated by the challenger $\mathcal{C}$ either as $\mathcal{P}_{i_j}(pp_{i_j}, (pp_{\mathtt{NIC}}, x'_{i_j}), w'_{i_j})$ or $\mathcal{S}_{i_j}^1(pp_{i_j}, \tau_{i_j}, (pp_{\mathtt{NIC}}, x'_{i_j}))$.
- For $k \in T_{j-1}$, compute $\pi_k \leftarrow\$ \mathcal{S}_k^1(pp_k, \tau_k, (pp_{\mathtt{NIC}}, x'_k))$. For $i \in [n] \setminus T_j$, compute $\pi_i \leftarrow\$ \mathcal{P}_i(pp_i, (pp_{\mathtt{NIC}}, x'_i), w'_i)$.
- Set $\pi \leftarrow (\{\pi_k\}_{k\in[n]}, \{x'_k\}_{k\in[n]})$.
- Run $\mathcal{A}(pp, x, \pi)$ and output whatever $\mathcal{A}$ outputs.

Figure C.16: Reduction Red_{i_j} to zero-knowledge of the i_j-th candidate.

If $\mathcal{C}$ provides a real proof, we are in hybrid $\mathcal{H}_{j-1}$. Otherwise, we are in hybrid $\mathcal{H}_j$. Therefore, Red_{i_j} retains the same distinguishing advantage of $\mathcal{A}$, breaking ZK of candidate Π_{i_j}. □

Lemma C.16 (Transition from $\mathcal{H}_t$ to $\mathcal{H}_{t+1}$)**.** $\mathsf{out}^{\mathcal{H}_t} \approx_c \mathsf{out}^{\mathcal{H}_{t+1}}$.

Proof. The indistinguishability goes through nt intermediate hybrids $\mathcal{H}_{s,e}$, with $s \in [t]$ and $e \in [n]$. The difference between two intermediate hybrids is that in $\mathcal{H}_{s,e-1}$, $(com_{i_s,i_e}, o_{i_s,i_e}) \leftarrow\$ \mathsf{Com}(pp_{\mathtt{NIC}}, \{\mathsf{msg}_{i_s\to i_e}^k\}_{k\in[R]})$, whereas in $\mathcal{H}_{s,e}$, $(com_{i_s,i_e}, o_{i_s,i_e}) \leftarrow\$ \mathsf{Com}(pp_{\mathtt{NIC}}, 0)$. This way, we effectively replace the set of messages from party i_s to party i_e (*i.e.*, $\{\mathsf{msg}_{i_s\to i_e}^k\}$) with 0. We note that, for $s \neq 1$, $\mathcal{H}_{s,0}$ is $\mathcal{H}_{s-1,n}$. We also note that $\mathcal{H}_{1,0}$ is $\mathcal{H}_t$, and $\mathcal{H}_{t,n}$ is $\mathcal{H}_{t+1}$.

Assume by contradiction there exists a PPT algorithm $\mathcal{A}$ that can distinguish $\mathsf{out}^{\mathcal{H}_{s,e-1}}$ from $\mathsf{out}^{\mathcal{H}_{s,e}}$. Then, we can construct the following adversary $\overline{\mathcal{A}}_{s,e}^{\mathtt{NIC}}$ breaking computational hiding of $\mathtt{NIC}$.

Reduction $\overline{\mathcal{A}}_{s,e}^{\mathtt{NIC}}$ to hiding of $\mathtt{NIC}$ (s-th party, e-th outbound message)

- Receive public parameters $\overline{pp}$ from the challenger $\mathcal{C}$.
- Compute $(pp_k, \tau_k) \leftarrow\$ \mathcal{S}_k^0(1^l, 1^\lambda)$ for $k \in T$. $pp_i \leftarrow\$ \mathsf{setup}_k(1^l, 1^\lambda)$ for $i \in [n] \setminus T$. Set $pp \leftarrow \{pp_k\}_{k\in[n]} \cup \{\overline{pp}\}$.
- Compute $(x, w) \leftarrow\$ \mathcal{A}(pp)$.
- Compute $\{w_i\}_{i\in[n]} \leftarrow\$ \mathtt{share}(w)$. Run MPC protocol Π_{MPC} with input $(w_1, \ldots, w_n)$ to obtain $\{\mathsf{view}_i\}_{i\in[n]}$.
- For $i \in T_s$:
  - if $i \neq i_s$, for $j \in [n]$, set $\{\mathsf{msg}_{i\to j}^k\}_{k\in[R]} = 0$ in view_i and view_j.
  - if $i = i_s$, for $j \in T_{e-1}$, set $\{\mathsf{msg}_{i\to j}^k\}_{k\in[R]} = 0$ in view_i and view_j.
- For $i \in [n]$, $j \in [n]$, compute $(com_{i,j}, o_{i,j}) \leftarrow\$ \mathsf{Com}(pp_{\mathtt{NIC}}, \{\mathsf{msg}_{i\to j}^k\}_{k\in[R]})$, and $(com_{j,i}, o_{j,i}) \leftarrow\$ \mathsf{Com}(pp_{\mathtt{NIC}}, \{\mathsf{msg}_{j\to i}^k\}_{k\in[R]})$.
- Send 0 and $\{\mathsf{msg}_{i_s\to i_e}^k\}_{k\in[R]}$ to the challenger $\mathcal{C}$, and receive the commitment com' that could either be a commitment of 0 or a commitment of the messages. Set $com_{i_s,i_e} \leftarrow com'$.
- For $i \in [n]$, set $x'_i \leftarrow (\{com_{i,j}\}_{j\in[n]}, \{com_{j,i}\}_{j\in[n]})$ and $w'_i \leftarrow (\mathsf{view}_i, \{o_{i,j}\}_{j\in[n]}, \{o_{j,i}\}_{j\in[n]})$.
- For $k \in T$, compute $\pi_k \leftarrow\$ \mathcal{S}_k^1(pp_k, \tau_k, (pp_{\mathtt{NIC}}, x'_k))$. For $i \in [n] \setminus T$, compute $\pi_i \leftarrow\$ \mathcal{P}_i(pp_i, (pp_{\mathtt{NIC}}, x'_i), w'_i)$.
- Set $\pi \leftarrow (\{\pi_k\}_{k\in[n]}, \{x'_k\}_{k\in[n]})$.
- Run $\mathcal{A}(pp, x, \pi)$ and output whatever $\mathcal{A}$ outputs.

Figure C.17: Reduction $\overline{\mathcal{A}}_{s,e}^{\mathtt{NIC}}$ to hiding of $\mathtt{NIC}$ (s-th party, e-th outbound message).

If $\mathcal{C}$ provides a real commitment, we are in intermediate hybrid $\mathcal{H}_{s,e-1}$. Otherwise, we are in intermediate hybrid $\mathcal{H}_{s,e}$. Therefore, $\overline{\mathcal{A}}_{s,e}^{\mathtt{NIC}}$ retains the same distinguishing advantage of $\mathcal{A}$, breaking the computational

hiding of $\mathtt{NIC}$. This is a contradiction; therefore, $\mathsf{out}^{\mathcal{H}_{s,e-1}} \approx_c \mathsf{out}^{\mathcal{H}_{s,e}}$. By applying nt intermediate hybrids, we have $\mathsf{out}^{\mathcal{H}_t} \approx_c \mathsf{out}^{\mathcal{H}_{t+1}}$.

Finally we observe that, by Lemma 8.1, there exists a construction of computationally hiding commitments that exclusively relies on the soundness and zero-knowledge of at least $t > \lfloor \frac{n}{2} \rfloor$ candidates. More specifically: in order to break the computational hiding property of the commitment, it must be that the security of the OWF obtained from the 1-out-of-n combiner from [Her05, Lemma 4.1] is broken. This is possible if an adversary is able to break the security of all OWF candidates. By the construction in Lemma 8.1, breaking a OWF candidate implies having a reduction either to the soundness of the corresponding input NIZK candidate, or the zero-knowledge of the corresponding input NIZK candidate, or both. Given that $t > \lfloor \frac{n}{2} \rfloor$, it must be the case that there exist either $n - t + 1$ reductions to soundness of distinct NIZK candidates, or $n - t + 1$ reductions to zero-knowledge of distinct NIZK candidates, or both. Therefore, in any of the cases above, we have enough reductions as desired. We argue this fact in what follows.

Assume by contradiction the above is not true. Then, there would exist an adversary breaking the security of all OWF candidates by using at most $n - t$ reductions to soundness and $n - t$ reductions to zero-knowledge of distinct candidates. In particular, the adversary could attack the security of each OWF candidate by using either a reduction to soundness or a reduction to zero-knowledge. This yields a set of broken OWF candidates of size at most $2n - 2t$. Since we assumed that $t > \lfloor \frac{n}{2} \rfloor$, we have that $2n - 2t < n$. But this implies that there still exists a OWF for which both soundness and ZK hold (*i.e.*, a OWF that is secure).

□

Lemma C.17 (Transition from $\mathcal{H}_{t+1}$ to $\mathcal{H}_{t+2}$)**.** $\mathsf{out}^{\mathcal{H}_{t+1}}$ *and* $\mathsf{out}^{\mathcal{H}_{t+2}}$ *are identically distributed, if* Π_{MPC} *has perfect* $\mathcal{K}$*-privacy.*

Proof. Assume by contradiction there exists an unbounded algorithm $\mathcal{A}$ that can distinguish $\mathsf{out}^{\mathcal{H}_{t+1}}$ from $\mathsf{out}^{\mathcal{H}_{t+2}}$. Then, we can construct the following adversary $\overline{\mathcal{A}}^{\mathsf{MPC}}$ breaking perfect $\mathcal{K}$-privacy of Π_{MPC}:

Reduction $\overline{\mathcal{A}}^{\mathsf{MPC}}$ to perfect $\mathcal{K}$-privacy of Π_{MPC}

- Compute $(pp_k, \tau_k) \leftarrow\$ \mathcal{S}_k^0(1^l, 1^\lambda)$ for $k \in T$. $pp_i \leftarrow\$ \mathsf{setup}_k(1^l, 1^\lambda)$ for $i \in [n] \setminus T$. Compute $pp_{\mathtt{NIC}} \leftarrow\$ \mathsf{setup}(1^\lambda)$. Set $pp \leftarrow \{pp_k\}_{k\in[n]} \cup \{pp_{\mathtt{NIC}}\}$.
- Compute $(x, w) \leftarrow\$ \mathcal{A}(pp)$.
- Compute $\{w_i\}_{i\in[n]} \leftarrow\$ \mathtt{share}(w)$. Send $\{w_i\}_{i\in[n]}$ to the challenger $\mathcal{C}$.
- The challenger $\mathcal{C}$ for $\mathcal{K}$-privacy computes $\{\mathsf{view}_j\}_{j\in[n]\setminus T}$ either by running a real execution of the protocol or by using the $\mathcal{K}$-privacy simulator $\mathcal{S}(\mathcal{I}, x, \{w_k\}_{k\in[\mathcal{I}]}, 1)$, with $\mathcal{I} = [n] \setminus T$.
- For $i \in T$, for $j \in [n]$, set $\{\mathsf{msg}^k_{i\to j}\}_{k\in[R]} = 0$ in view_i and view_j. Complete view_i to be consistent with $\{\mathsf{view}_j\}_{j\in[n]\setminus T}$.
- For $i \in [n]$, $j \in [n]$, compute $(com_{i,j}, o_{i,j}) \leftarrow\$ \mathsf{Com}(pp_{\mathtt{NIC}}, \{\mathsf{msg}^k_{i\to j}\}_{k\in[R]})$, and $(com_{j,i}, o_{j,i}) \leftarrow\$ \mathsf{Com}(pp_{\mathtt{NIC}}, \{\mathsf{msg}^k_{j\to i}\}_{k\in[R]})$.
- For $i \in [n]$, set $x'_i \leftarrow (\{com_{i,j}\}_{j\in[n]}, \{com_{j,i}\}_{j\in[n]})$. For $l \in [n] \setminus T$, $w'_l \leftarrow (\mathsf{view}_l, \{o_{l,j}\}_{j\in[n]}, \{o_{j,l}\}_{j\in[n]})$.
- For $l \in T$, compute $\pi_l \leftarrow\$ \mathcal{S}_l^1(pp_l, \tau_l, (pp_{\mathtt{NIC}}, x'_l))$. For $i \in [n] \setminus T$, compute $\pi_i \leftarrow\$ \mathcal{P}_i(pp_i, (pp_{\mathtt{NIC}}, x'_i), w'_i)$.
- Set $\pi \leftarrow (\{\pi_k\}_{k\in[n]}, \{x'_k\}_{k\in[n]})$.
- Run $\mathcal{A}(pp, x, \pi)$ and output whatever $\mathcal{A}$ outputs.

Figure C.18: Reduction $\overline{\mathcal{A}}^{\mathsf{MPC}}$ to perfect $\mathcal{K}$-privacy of Π_{MPC}

If $\mathcal{C}$ produces views coming from real parties, we are in hybrid $\mathcal{H}_{t+1}$. Otherwise, we are in hybrid $\mathcal{H}_{t+2}$. Therefore, $\overline{\mathcal{A}}^{\mathsf{MPC}}$ retains the same distinguishing advantage of $\mathcal{A}$, breaking perfect $\mathcal{K}$-privacy of Π_{MPC}. □

Lemma C.18 (Transition from $\mathcal{H}_{t+2}$ to $\mathcal{H}_{t+3}$)**.** $\mathsf{out}^{\mathcal{H}_{t+2}}$ *and* $\mathsf{out}^{\mathcal{H}_{t+3}}$ *are identically distributed.*

Proof. Assume by contradiction there exists an unbounded algorithm $\mathcal{A}$ that can distinguish $\mathsf{out}^{\mathcal{H}_{t+2}}$ from $\mathsf{out}^{\mathcal{H}_{t+3}}$. Then, we can construct the following adversary $\overline{\mathcal{A}}^{\mathsf{SS}}$ breaking perfect security of the t-out-of-n secret sharing scheme SS:

Reduction $\overline{\mathcal{A}}^{\mathsf{SS}}$ to perfect security of the t-out-of-n secret sharing scheme SS

- Compute $(pp_k, \tau_k) \leftarrow\$ \mathcal{S}_k^0(1^l, 1^\lambda)$ for $k \in T$. $pp_i \leftarrow\$ \mathsf{setup}_k(1^l, 1^\lambda)$ for $i \in [n] \setminus T$. Compute $pp_{\mathtt{NIC}} \leftarrow\$ \mathsf{setup}(1^\lambda)$. Set $pp \leftarrow \{pp_k\}_{k\in[n]} \cup \{pp_{\mathtt{NIC}}\}$.
- Compute $(x, w) \leftarrow\$ \mathcal{A}(pp)$, and forward w to the challenger $\mathcal{C}$.
- Receive $n - t$ shares $\{s_j\}_{j\in[n]\setminus T}$ from the challenger $\mathcal{C}$ which are either shares of the real witness w or random elements $\{r_j\}_{j\in[n]\setminus T} \leftarrow\$ \mathtt{Gen}(1^\lambda)$.
- Run the $\mathcal{K}$-privacy simulator $\mathcal{S}_{\mathsf{MPC}}([n]\setminus T, x, \{s_j\}_{j\in[n]\setminus T}, 1)$ of Π_{MPC} to obtain the simulated views $\{\mathsf{view}_j\}_{j\in[n]\setminus T}$.
- For $i \in T$, for $j \in [n]$, set $\{\mathsf{msg}^k_{i\to j}\}_{k\in[R]} = 0$ in view_i and view_j. Complete view_i to be consistent with $\{\mathsf{view}_j\}_{j\in[n]\setminus T}$.
- For $i \in [n]$, $j \in [n]$, compute $(com_{i,j}, o_{i,j}) \leftarrow\$ \mathsf{Com}(pp_{\mathtt{NIC}}, \{\mathsf{msg}^k_{i\to j}\}_{k\in[R]})$, and $(com_{j,i}, o_{j,i}) \leftarrow\$ \mathsf{Com}(pp_{\mathtt{NIC}}, \{\mathsf{msg}^k_{j\to i}\}_{k\in[R]})$.
- For $i \in [n]$, set $x'_i \leftarrow (\{com_{i,j}\}_{j\in[n]}, \{com_{j,i}\}_{j\in[n]})$. For $l \in [n] \setminus T$, $w'_l \leftarrow (\mathsf{view}_l, \{o_{l,j}\}_{j\in[n]}, \{o_{j,l}\}_{j\in[n]})$.
- For $l \in T$, compute $\pi_l \leftarrow\$ \mathcal{S}_l^1(pp_l, \tau_l, (pp_{\mathtt{NIC}}, x'_l))$. For $i \in [n] \setminus T$, compute $\pi_i \leftarrow\$ \mathcal{P}_i(pp_i, (pp_{\mathtt{NIC}}, x'_i), w'_i)$.
- Set $\pi \leftarrow (\{\pi_k\}_{k\in[n]}, \{x'_k\}_{k\in[n]})$.
- Run $\mathcal{A}(pp, x, \pi)$ and output whatever $\mathcal{A}$ outputs.

Figure C.19: Reduction $\overline{\mathcal{A}}^{\mathsf{SS}}$ to perfect security of the t-out-of-n secret sharing scheme SS.

We now argue that $\overline{\mathcal{A}}^{\mathsf{SS}}$ correctly reproduces the behavior of hybrids $\mathcal{H}_{t+2}$ and $\mathcal{H}_{t+3}$ according to the challenger's behaviour. If the challenger shares the real witness w, all the proofs in $[n] \setminus T$ are generated exactly as in $\mathcal{H}_{t+2}$ (*i.e.*, using shares of w). If the challenger generates random elements by using $\mathtt{Gen}$, all the proofs in $[n] \setminus T$ are generated exactly as in $\mathcal{H}_{t+3}$. Also, because we have $t > \lfloor \frac{n}{2} \rfloor$, $n - t \leq t - 1$. Therefore, $\overline{\mathcal{A}}^{\mathsf{SS}}$ retains the same distinguishing advantage of $\mathcal{A}$, breaking perfect security of the t-out-of-n secret sharing scheme. This is a contradiction; therefore, $\mathsf{out}^{\mathcal{H}_{t+2}}$ and $\mathsf{out}^{\mathcal{H}_{t+3}}$ and identically distributed. □

C.7.2 $\mathsf{C}_{\mathsf{MPC}}$ is a (ZK, AoK)-combiner

In order to claim that $\mathsf{C}_{\mathsf{MPC}}$ (as in Figure 8.3) is a t-out-of-n NBB (ZK, AoK)-combiner, we need to argue the following:

- $\mathsf{C}_{\mathsf{MPC}}$ is a correct n-candidate $\mathcal{F}_{\mathsf{NI}}$-combiner for any NP language $\mathcal{L}$, as per Definition 8.1.
- $\mathsf{C}_{\mathsf{MPC}}$ is an argument-of-knowledge t-out-of-n NBB (ZK, AoK)-combiner for any NP language $\mathcal{L}$, as per Definition C.2.
- $\mathsf{C}_{\mathsf{MPC}}$ is a zero-knowledge t-out-of-n NBB (ZK, AoK)-combiner for any NP language $\mathcal{L}$, as per Definition C.3.

Note that the first item already holds by Theorem C.9. In what follows, we formalize the remaining items.

Theorem C.12. *The construction in Figure 8.3 is an argument-of-knowledge t-out-of-n NBB (ZK, AoK)-combiner for any NP language $\mathcal{L}$ for $t > \lfloor \frac{n}{2} \rfloor$ and $n, t \in \mathsf{poly}(\lambda)$.*

Proof (Theorem C.12). The structure of this proof is the same as the one of Theorem C.4 and uses exactly the same the combinatorics argument used to prove Lemma C.9. It therefore suffices to construct an extractor $\mathcal{E}^T$, and argue that either $\mathcal{E}^T$ successfully extracts a witness, or there exists at least one reduction to AoK of one of the candidates whose indices are in T. We describe such an $\mathcal{E}^T$ in Lemma C.19. □

Lemma C.19. *There exists an extractor $\mathcal{E}^T$ for which either a witness w is successfully extracted from an accepting proof, or there exists a reduction to AoK of at least one of the candidates $\{\Pi_k\}_{k\in T}$.*

Proof. Let $T = \{i_1, \ldots, i_t\}$ be a set of indices sorted in lexicographic order identifying candidates $\Pi_{i_1}, \ldots, \Pi_{i_t}$. The extractor $\mathcal{E}^T = (\mathcal{E}^{T,0}, \mathcal{E}^{T,1})$ is constructed as follows:

Extractor $\mathcal{E}^T$ for $\mathsf{C}_{\mathsf{MPC}}$

$\mathcal{E}^{T,0}(1^l, 1^\lambda)$:

- For $k \in T$, compute $(pp_k, \xi_k) \leftarrow\!\!\$\ \mathcal{E}_k^0(1^l, 1^\lambda)$.
- Compute $pp_{\mathtt{NIC}} \leftarrow\!\!\$\ \mathsf{setup}(1^\lambda)$.
- Compute $pp \leftarrow \{pp_k\}_{k\in[n]} \cup \{pp_{\mathtt{NIC}}\}$, $\xi \leftarrow \{\xi_k\}_{k\in T}$.
- Return (pp, ξ).

$\mathcal{E}^{T,1}(pp, \xi, x, \pi)$:

- For $k \in T$, parse the k-th component of π as (x'_k, π_k). Compute $w'_k \leftarrow \mathcal{E}_k^1(pp_k, \xi_k, (pp_{\mathtt{NIC}}, x'_k), \pi_k)$. Parse w'_k as $(\mathsf{view}_k, \{o_{k,j}\}_{j\in[n]}, \{o_{j,k}\}_{j\in[n]})$. Take w_k from view_k.
- Output $w \leftarrow \mathtt{reconstruct}(\{w_k\}_{k\in T})$.

Figure C.20: Description of $\mathcal{E}^T$ for $\mathsf{C}_{\mathsf{MPC}}$.

An adversary may break AoK either by detecting the introduction of trapdoor setups, or by outputting an accepting proof that causes any extractor to fail. If either occurs, we exhibit a reduction to AoK of that specific candidate.

Introducing trapdoor setups. This step is the same as the proof of Lemma C.7, and is therefore omitted.

Extractor $\mathcal{E}^T$ fails to extract a witness. We conclude the proof exactly as the proof of Lemma C.7: given that no reduction occurred, we proceed with analyzing $\mathcal{E}^T$.

We observe that the only way $\mathcal{E}^T$ can reconstruct w is by successfully extracting all the sub-witnesses in T, and using the $\mathtt{reconstruct}$ algorithm of the t-out-of-n secret sharing scheme. Because the verifier $\mathsf{C}_{\mathsf{vrfy}}$ performs checks for the consistency of the commitments, when all the checks pass, it means that all pair of views are consistent with the execution of Π_{MPC} because $\mathtt{NIC}$ is statistically binding. The proof is the same as what we have in the proof of Theorem C.10.

When all the sub-extraction procedures succeed, we can extract sub-witnesses from t sub-proofs successfully. This implies that these sub-statements x'_k are in the language $\mathcal{L}_{k,\mathsf{MPC}}$, meaning that the output of parties $\{\mathsf{P}_k\}_{k\in T}$ is 1. It also means there are at most $n - t$ parties that output 0, satisfying $\mathcal{K}$-robustness. Hence, the statement x is in the language, and using these t sub-witnesses as the input of the t-out-of-n secret sharing scheme allows to reconstruct w successfully.

Therefore, in order for $\mathcal{E}^T$ to fail, it suffices for one of the sub-extraction procedures to fail. Let i^* be an index for which $\mathcal{E}_{i^*}^1$ fails to extract from (x'_{i^*}, π_{i^*}). Any adversary that outputs an accepting proof (x, π) for the combiner that contains (x'_{i^*}, π_{i^*}) in position i^* immediately causes $\mathcal{E}^T$ to fail. As for the proof of Lemma C.7, this yields a reduction to AoK of scheme Π_{i^*}.

□

Theorem C.13. *The construction in Figure 8.3 is a zero-knowledge t-out-of-n NBB (ZK, AoK)-combiner for any NP language $\mathcal{L}$ for $t > \lfloor \frac{n}{2} \rfloor$ and $n, t \in \mathsf{poly}(\lambda)$.*

Proof. The proof for zero-knowledge of Theorem C.11 relies both on soundness as well as zero-knowledge of the candidates (due to the presence of the commitment scheme). However, given that AoK implies soundness, we can reuse the same proof in this setting. □