BERGISCHE
UNIVERSITÄT
WUPPERTAL

Doctoral Dissertation

Towards Advanced Cryptographic Protocols for Real-World Applications

Jonas von der Heyden, M.Sc.

July 29, 2025

Submitted to the
School of Electrical, Information and Media Engineering
University of Wuppertal

for the degree of
Doktor-Ingenieur (Dr.-Ing.)

Jonas von der Heyden
Place of birth: Wuppertal, Germany
Author's contact information:
jvdh@uni-wuppertal.de

Thesis Advisor:	**Prof. Dr.-Ing. Tibor Jager** University of Wuppertal Wuppertal Germany
Second Examiner:	**Prof. Dr. phil. nat. Marc Fischlin** Darmstadt University of Technology Darmstadt Germany
Thesis submitted:	July 29, 2025
Thesis defense:	October 24, 2025
Last revision:	July 29, 2025

— Für meine Schwestern, Sara und Tabea,
und meine Eltern, Andrea und Gerhard

Acknowledgements

This journey took its course in 2012, when I was working as a sales associate for Google in Ireland, and I saw an email about a free online course on artificial intelligence, given by Sebastian Thrun and Peter Norvig. Having nothing better to do with my time, I signed up. One thing led to another, and soon I spent my nights and weekends learning about AI, taking Albert Hwang's excellent course on Python for non-engineers, and programming tools to support our team's sales work. I soon transferred to a somewhat more technical role, and—inspired by my flatmate Oliver—started studying computer science on the side.

I was fortunate to have managers like Sebastian and colleagues like Florian who always supported me in my unconventional endeavor. I continued to benefit from exceptionally generous advisors, supervisors and mentors, even after leaving Google: Marian Margraf supervised my bachelor and master theses and gave me a job at Fraunhofer AISEC that kept me afloat during my studies.

Through a stroke of good luck, I stumbled upon Tibor Jager's provable security course during the pandemic and participated online. I feel very grateful that he offered me to start a PhD in his group shortly afterwards, which, in the last consequence, now leads me to write these lines. Thank you, Tibor, for your trust, your support and, last but not least, your passion for research which inspires countless students (PhD and otherwise) to learn about and advance cryptography!

Thank you also to my amazing, talented and kind (former) colleagues Adrian Ackermann, Pascal Bemmann, Gareth T. Davies, Denis Diemert, Jan Drees, Amin Faez, Dennis Funke, Kai Gellert, Tobias Handirk, Raphael Heitjohann, Christian Holler, Jan Horning, Máté Horvath, Saqib Kakvi, Lin Lyu, Jutta Maerten, Tom Neuschulten, David Niehues and Marloes Venema.

I would especially like to thank Marloes, Máté, Raphael and Jan for exploring not only theoretical landscapes with me, but also venturing on beautiful hikes through the Bergisches Land, Sauerland, Eifel and Siebengebirge. Thank you for your friendship on and off the trail! I am also deeply grateful to Marloes, Máté, and Raphael for their invaluable help in proofreading this thesis. Marloes, thank you in particular for always being so incredibly helpful and generous with your time.

I am grateful to my co-authors Raphael, Tibor, Nils Schlüter, Philipp Binfet, Martin Asman, Moritz Schulze Darup, Markus Zdrallek, Marc Fischlin, Frank Morgner, Marian Margraf and Holger Bock for fruitful collaboration, and I addi-

tionally thank Marc Fischlin for being second examiner of my PhD thesis. I am also grateful to Adrian Ackermann, Sebastian Baum and Janis Gawrych, whose bachelor theses I had the honor to advise, and I want to thank them for their trust and our engaging discussions.

Lastly, I want to express deep gratitude to my friends and my family.

Abstract

This thesis aims to facilitate the deployment of advanced cryptographic primitives in the real world. It does so by designing and implementing post-quantum algorithms on electronic travel documents, and *multi-party computation* (MPC) on smart meters. We also introduce a novel pairing-based *key-and-message homomorphic encryption* (KMHE) scheme for more efficient outsourced MPC.

First, we present *PQ-EAC*, a post-quantum secure replacement for the Extended Access Control protocol used in electronic machine-readable travel documents (eMRTDs). By substituting Diffie-Hellman key exchange with post-quantum key encapsulation mechanisms, we design eight protocol variants offering different trade-offs between security and efficiency. Our implementation on an ARM SC300 chip demonstrates runtimes of under two seconds at typical data rates, confirming the practical feasibility of migrating eMRTDs to post-quantum security.

Second, we develop the first fully privacy-preserving power flow analysis (PFA) based on MPC. By adopting a Cartesian formulation of Newton's method and exploiting sparsity, we create an efficient MPC implementation that enables smart grid operations without compromising prosumer privacy. Our benchmarks show that for small grids with low network latency, online computation completes in under 30 seconds, demonstrating that MPC-based PFA is practical for preventive smart grid applications.

Third, we present a new KMHE scheme based on bilinear pairings that enables practical rerandomizable garbling schemes (RGS). Our construction reduces garbled gate size by 98.99% (from 133.43 MB to 1.35 MB) and garbling time by 99.998% (from 33 minutes to 0.04 seconds) compared to previous BHHO-based approaches. This four-order-of-magnitude improvement makes the SCALES protocol practically feasible for the first time, enabling constant-round outsourced computation with security against adaptive adversaries.

Together, these contributions demonstrate that advanced cryptographic techniques can be made practical through careful protocol design, algorithmic optimization, and implementation strategies tailored to specific application constraints.

Contents

Acronyms

ABB arithmetic black box

AKE authenticated key exchange

APDU application protocol data unit

BAC Basic Access Control

BPL broadband over power lines

BSI German Federal Office for Information Security

CA Chip Authentication

CSCA Country Signing Certificate Authority

CVCA Country Verifying Certificate Authority

DDH decisional Diffie-Hellman

DH Diffie-Hellman

DP differential privacy

DTC Digital Travel Credential

EAC Extended Access Control

eMRTD electronic machine-readable travel document

ePassport electronically-enabled passport

FHE fully homomorphic encryption

GMRES generalized minimal residual method

HE homomorphic encryption

ICAO International Civil Aviation Organization

KEM key encapsulation mechanism

KMHE key-and-message homomorphic encryption

LU lower-upper

MAC message authentication code

MPC multi-party computation

MRZ machine-readable zone

NIST National Institute of Standards and Technology

OCR optical character recognition

OT oblivious transfer

PACE Password-Authenticated Connection Establishment

PAKE password-authenticated key exchange

PFA power flow analysis

PKI public-key infrastructure

RFID radio-frequency identification

RGS rerandomizable garbling schemes

RTT round-trip time

SCALES MPC with small clients and larger ephemeral servers

SXDH symmetric external Diffie-Hellman

TA Terminal Authentication

TTP trusted third party

UC universal composability

YOSO you only speak once

1 Introduction

Driven by the rapid development of both theoretical and applied cryptography, primitives that were considered to be of purely theoretical interest a decade ago are now at the threshold of practical feasibility or beyond. At the same time, there has only been a muted adoption of advanced cryptographic techniques such as *multi-party computation* (MPC) or *fully homomorphic encryption* (FHE) "in the wild", and *post-quantum* cryptography has yet to become the default for securing internet connections. This dissertation aims to bridge this gap through a twofold approach: Firstly, by demonstrating the practical feasibility of such advanced cryptographic techniques for specific applications such as post-quantum electronic passports and privacy-preserving smart grids. Secondly, by introducing novel ideas that address fundamental barriers to their practical deployment, particularly for constant-round outsourced MPC with security against adaptive adversaries.

As of December 2024, only about 13% of TLS 1.3 traffic routed through Cloudflare's network was protected by post-quantum secure schemes[1]. The security of the remaining 87% rests on assumptions related to the hardness of integer factorization and discrete logarithms. Due to Peter Shor's polynomial-time quantum algorithm for these problems [Sho94] those assumptions are now considered to be false. While existing quantum computers are not powerful enough to endanger modern cryptography, recent progress [A+25] shows that eventually, practical quantum attacks are a realistic scenario. To mitigate this threat, discrete-logarithm- and factoring-based cryptography must be replaced with post-quantum alternatives such as lattice-based constructions.

The quantum threat especially impacts embedded devices: their long service lifetimes and limited firmware-update options demand early migration to post-quantum cryptography, yet their constrained processing power and memory often hinder the deployment of these advanced cryptographic algorithms.

This problem is particularly acute for *electronic machine-readable travel document*s (eMRTDs)—such as ePassports, identity cards, and refugee travel documents—which store sensitive biometric data and have service lives of ten years or more, while lengthy regulatory processes can further delay the deployment of security updates—see Figure 1.1. Access to this data is currently protected by the *Extended Access*

[1] `https://blog.cloudflare.com/radar-2024-year-in-review/#13-0-of-tls-1-3-traffic-is-using-post-quantum-encryption`

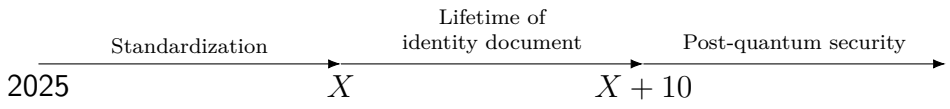

Figure 1.1: Timeline for post-quantum security.

Control (EAC) protocol, which uses a discrete-logarithm-based *Diffie-Hellman* (DH) key exchange. Consequently, the design of a successor to the EAC protocol that is both quantum-resistant and efficient enough for deployment on the constrained hardware of eMRTDs is an urgent priority. This leads us to our first research question:

Can we devise an efficient and post-quantum secure replacement for the EAC protocol on eMRTDs?

Innovative cryptographic techniques can be "too new" for practice. The difficulty in their adoption is often not a technical failing, but a conceptual one: their potential can be hard to recognize when viewed from the perspective of existing problems and solutions. Much like Henry Ford's customers wanted faster horses instead of a car, the focus today is often on ostensible conflicts between the protection of sensitive data and its exploitation for research, business or other purposes, rather than embracing new paradigms that reconcile privacy with analysis.

Consequently, opportunities for research breakthroughs, better governance, and the generation of wealth are being missed. A concrete example is that of intelligent energy networks ("smart grids"), which enable more efficient and sustainable energy distribution but whose adoption is hindered due to privacy concerns. This dissertation addresses this specific gap by demonstrating the feasibility of MPC for *power flow analysis* (PFA), a fundamental algorithm for control and optimization in smart grids. This leads to our second research question:

Is MPC from secret sharing efficient enough to enable privacy-preserving power flow analysis on smart meters?

While our results show that secure PFA is practical for some applications, they also highlight a key bottleneck: In many cases the limiting factor is not just the resources of the smart meter, but also network latency. This is because in MPC from *secret sharing* the number of required communication rounds grows proportionally with the depth of the circuit to be computed. Hence, secure computation of complex functions such as PFA quickly becomes infeasible with increased network latency. This limitation motivates the exploration of constant-round MPC, where the number of communication rounds is independent of the circuit depth. To

further sidestep performance limitations of resourced-constrained smart meters, we consider outsourced MPC where heavy computation is offloaded from weak clients to powerful external servers.

This outsourced architecture, however, introduces a critical vulnerability: an adaptive adversary could observe the protocol start and then corrupt the entire, small server committee. Guaranteeing security against such an adversary—who can choose corruption targets mid-protocol—is therefore a paramount challenge. Addressing this challenge brings us to our third and final research question:

How can we design constant-round, adaptively secure outsourced MPC protocols with concrete efficiency sufficient for practical deployment?

We will introduce these research questions in more detail in Section 1.1, Section 1.2, and Section 1.3 respectively.

1.1 Post-Quantum Security for eMRTDs

In the following we will introduce the EAC protocol and show how it currently secures eMRTDs, before summarizing our improvements to the protocol, especially in the context of post-quantum security.

1.1.1 State of the art

EAC provides authenticated key establishment between a terminal (as used in border control) and an eMRTD, also called *chip*, communicating via *radio-frequency identification* (RFID). EAC includes two authentication mechanisms: *Chip Authentication* (CA) and *Terminal Authentication* (TA): CA implicitly authenticates the chip and establishes a shared key for secure communication; TA prevents the reading of biometric data from unauthorized terminals. While EAC originated in the context of travel documents, it is used beyond passports–for example, to secure German eID transactions such as online identification and e-government services. Even though we focus on the *electronically-enabled passport* (ePassport) setting, most of our results apply more generally.

In the context of border control EAC is preceded by other protocols as shown in Figure 1.2. Both *Basic Access Control* (BAC) and *Password-Authenticated Connection Establishment* (PACE) use the *machine-readable zone* (MRZ), which is read out-of-band via *optical character recognition* (OCR) from the ePassport, to derive a shared secret key for further secure communication via RFID. Additionally, *Passive Authentication* ensures the integrity of the chip's data and binds the data to the chip's public key. For reading non-sensitive data from the ePassport, execution of BAC or PACE is deemed to be sufficient to establish a weak form

of authentication. TA is only mandatory if the terminal wants to read sensitive biometric data.

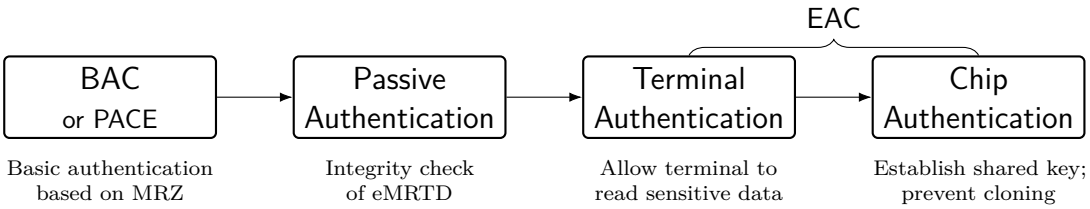

Figure 1.2: EAC and its subprotocols.

Since post-quantum *key encapsulation mechanisms* (KEMs) standardized by the *National Institute of Standards and Technology* (NIST) cannot serve as a direct drop-in replacement for the DH key exchange in the EAC protocol, a revised post-quantum secure version of EAC is needed. Such a protocol must balance several competing requirements:

– The protocol must complete in 2-3 seconds to maintain border control throughput, even on resource-constrained devices and under non-ideal conditions such as suboptimal placement of the booklet on the reader.

– The protocol must guarantee the integrity of the eMRTD's data and be resistant to cloning attacks, where an adversary attempts to create a functional copy of the chip.

– The holder's sensitive biometric data must be protected from unauthorized access, and the protocol must prevent the eMRTD from being tracked over time and across different locations.

Although the literature [SSW20, HNS+21, ADH+22] contains numerous proposals for transforming DH-based protocols to use KEMs, generic approaches are not directly applicable to the unique context of EAC. Moreover, the way EAC interacts with BAC, Passive Authentication and PACE to achieve non-standard security goals such as unlinkability make a generic mechanism unlikely to apply. Finally, in order to ensure backwards compatibility, a new version of EAC should conform to the specification of EAC in standard TR-03110 [Bun16]. We therefore pursue a tailored post-quantum design rather than a simple substitution.

Additionally, the protocol must account for the tight resources of identity-document chips. Unfortunately, post-quantum KEMs and signature schemes such as Kyber and Dilithium impose substantial time and memory overhead in comparison to their classical counterparts. To ensure sufficiently short execution time, the protocol must avoid expensive operations, and an accompanying implementation

should provide versions of Kyber and Dilithium that are optimized for the specific chip architecture, e.g. ARM SC300.

Table 1.1: Options for PQ-EAC.

Option	**Description**	**Advantage**	**Disadvantage**
Round-reduced	Combines the TA and CA steps of EAC.	Saves one round-trip.	Chip sends public key before TA (except in KEM-only variant).
Forward-secure	Uses ephemeral keys.	Compromised long-term key does not reveal past session keys.	Requires extra key generation and decapsulation.
KEM-only	Uses KEMs for authentication.	Avoids expensive post-quantum signatures; chip's public key can be sent encrypted.	—

1.1.2 Advancements to the state of the art

Chapter 3 advances the state of the art as follows:

– We devise a post-quantum secure EAC protocol, called *PQ-EAC*. More precisely, we give a base protocol which can optionally be round-reduced (merging the formerly separate TA and CA), forward-secure, and KEM-only (i.e., KEMs are also used for authentication). Hence, in total there are eight different versions of PQ-EAC. This is to give standardizing bodies sufficient flexibility in balancing various trade-offs between security and efficiency. For more details, see Table 1.1.

– We implement the combined version of PQ-EAC with signatures and without forward secrecy on an ARM SC300 chip that is designed for use in identity documents and a Bundesdruckerei VISOCORE terminal for border control, and benchmark it under realistic conditions. In a second experiment, we also benchmark PQ-EAC using a smartcard emulation device for tighter control over parameters such as data rate, and in order to be able to inspect and benchmark the card's internal workflow on a more granular level.

Our results show that, at a typical data rate of 848 kbit/s, all versions of PQ-EAC run in well below two seconds. We therefore conclude that eMRTDs can be migrated to PQ-EAC.

1.2 Privacy-Preserving Smart Grids

An increasing share of our electricity is generated by intermittent renewable energy sources such as solar and wind power. Unlike conventional energy sources, power generation from these renewables cannot be scaled flexibly as required by demand. Usually, this makes it necessary to complement wind and solar power with secondary power plants using, for example, natural gas. *Smart grids*, defined by a multilateral exchange of energy (e.g. from solar panels, batteries and electric vehicles) and data (e.g. demand forecasts and consumption data) between households (so-called *prosumers*) and a grid operator are a potential solution to mitigate the challenges arising from intermittent energy sources and to reduce dependency on secondary power plants. In comparison to conventional grids, smart grids grant operators more ability to plan ahead, and to optimize the distribution of energy. In order to reduce dependency on highly flexible energy supply, smart grids also enable the proactive management of demand, for example by means of dynamic pricing or flexibility markets, where prosumers are incentivized to provide electricity (or to avoid consuming electricity) during peak times [ATB17].

Privacy challenges in smart grids. While conventional grids record only the aggregate monthly electricity consumption of a household, smart grids typically employ *smart meters*, which provide fine-grained usage measurements. Unfortunately, these measurements can leak sensitive private information. Figure 1.3 shows that even without detailed *a priori* knowledge of household appliances, statistical analysis enables identification of complex usage patterns from smart meter data, including household occupancy, number of residents, and daily routines [MSF+10]. Smart meter data allows inferences not only about which household appliances are being used, but also what they are being used for. For example, variations in power consumption when displaying bright and dark scenes in audiovisual content enable the detection of specific movies being watched [GGJL12]. Since the private information leaked by smart meters can be exploited for burglary, mass surveillance, or targeted advertising, the adoption of smart grids is often restricted via legal means and hampered by limited user acceptance. Chapter 4 addresses these challenges by developing a privacy-preserving version of PFA, which provides crucial information about the power system's performance and can thereby help to mitigate issues such as voltage instability and overloads [AC92]. In smart grids, PFA is an important component of many essential control algorithms for congestion management, state

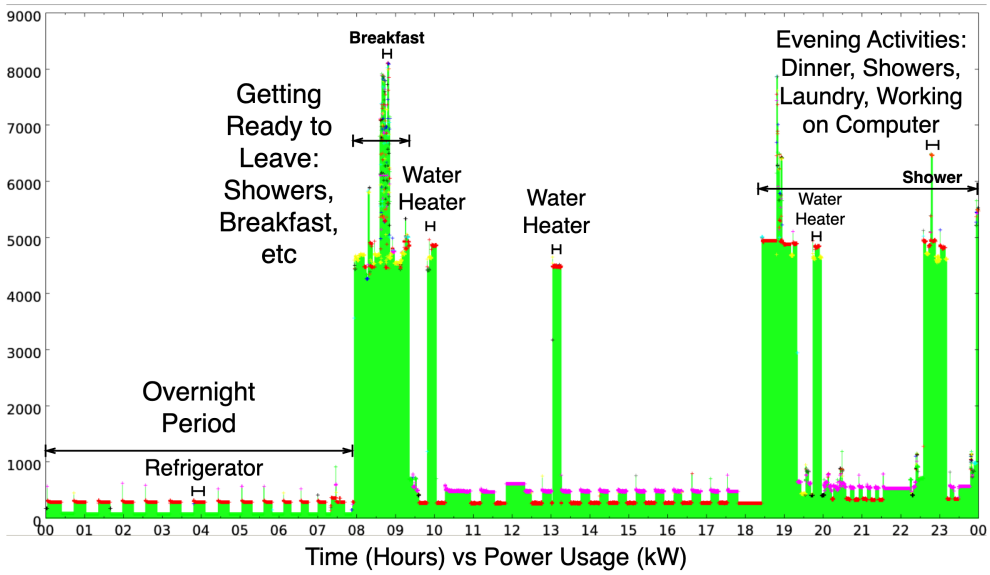

Figure 1.3: Using power consumption data to infer the daily routine of household residents (by Molina-Markham et al. [MSF+10]).

estimation, and flexibility markets.

1.2.1 State of the art

Privacy-preservation in smart grids has been addressed in the literature before. We briefly summarize related works and point out conceptual differences to our approach in Table 1.2. The listed works can be distinguished by applied cryptographic technique, the application, whether a *trusted third party* (TTP) is required, and whether some crucial data is leaked. We also provide a coarse classification of the problem complexity, which reflects the difficulty of implementing the task at hand securely, without making use of a TTP or leaking data. In the following, we discuss some related works in more detail.

Common attempts to alleviate privacy problems include restricting the measurement periods to 15 minutes or more, limiting the data storage duration, and anonymization [EK10]. However, these methods prevent leveraging all available information, require trust, and provide limited security guarantees. Besides MPC, other conceivable cryptographic techniques with robust security guarantees are *differential privacy* (DP) and *homomorphic encryption* (HE). The main goal of these techniques is to support some form of function evaluation while maintaining the privacy of the function's inputs.

In DP, carefully calibrated noise is added to the data, to hide information while some computation is still possible. Consequently, a trade-off between accuracy and

Table 1.2: Overview of related work.

Work	Crypt. technique	Application	Problem complexity	No TTP	No leakage
[EK10]	Escrow	Real-time metering data	High	✗	✗
[ÁC11]	DP	Data aggregation	Low	✓	✗
[SDT15]	DP	State estimation	Medium	✓	✗
[FMH20]	DP	Power line obfuscation	Medium	✓	✗
[GSA+21]	DP	Optimal power flow	High	✗	✗
[TTM+23]	DP	Load forecast	Medium	✓	✗
[HHMW22]	DP	Load profile synthesis	Medium	✓	✗
[GJ10, LLL10, JKL16]	HE	Data aggregation	Low	✓	✓
[CLL22]	HE	Economic dispatch	Medium	✓	✗
[WZZ21]	HE	Optimal power flow	High	✗	✗
[BCIV17]	HE	Load forecast	Medium	✓	✓
[KDK11] [DFKZB13, MCAA19]	MPC	Data aggregation	Low	✓	✓
[TGSZ22]	MPC	Economic dispatch	Medium	✓	✓
[SZW+23]	MPC	Optimal power flow	High	✗	✗
[AACM16, ZWG+23]	MPC	Energy markets	Medium	✓	✓
[MDS+22]	MPC	Energy markets	Medium	✓	✗
Ours	MPC	Power flow analysis	High	✓	✓

security arises. Applications of differential privacy for smart grids are numerous and include meter data aggregation [ÁC11], clustering, and state estimation [SDT15]. More recent works have applied it to optimal power flow [FMH20, GSA+21], neural network-based voltage forecasting [TTM+23], and load profile synthesis [HHMW22].

Unlike DP, MPC and HE come with stronger cryptographic security guarantees while enabling high accuracy. These properties make them attractive for a scalable solution with improved utility. HE refers to cryptosystems that allow for (restricted) computation on ciphertexts. In the context of smart grids, HE is extensively used for smart meter data aggregation [GJ10, LLL10], where fraud detection is additionally provided by [JKL16]. Further, economic dispatch [CLL22] and optimal power flow [WZZ21] have been addressed. However, these approaches rely on intermediate decryption, which causes information leakage. Privacy-preserving load forecasting has also been considered in [BCIV17]

In contrast to HE, the bottleneck in MPC is usually not computational, but communication complexity. For smart grids, [MCAA19] and [KDK11] use secret sharing-based aggregation protocols, and [DFKZB13] implements a simple statistical analysis for metering data. [TGSZ22] consider an economic dispatch problem, where the optimization is solved in a partially private manner using

(Shamir's) secret sharing. Similarly, energy flow in radial alternating current grids is considered in [SZW+23], where the authors employ a TTP to establish consensus between the participants; however, the more complex subproblems involving the power flow equations are solved using plaintext computations. Finally, [AACM16, MDS+22, ZWG+23] utilize MPC to devise privacy-preserving energy markets.

1.2.2 Advancements to the state of the art

We present the first realization of a fully privacy-preserving PFA based on Newton's method, leveraging the strong security guarantees provided by MPC protocols. In more detail, the main contributions of Chapter 4 can be summarized as follows:

– While MPC allows for arbitrary computations, a careless choice of protocols and parameters or naive implementations of algorithms typically lead to poor performance. We provide an efficient MPC implementation tailored for PFA by adopting a suitable Cartesian formulation and by exploiting sparsity. Additionally, we adapt solvers for systems of linear equations—such as *lower-upper* (LU) decomposition and *generalized minimal residual method* (GMRES)—and integrate line search techniques. We share our implementation for reproducibility.

– We provide a security analysis of our algorithm in the *universal composability* (UC) framework [Can01]. This is especially relevant since PFA is an essential subtask in applications such as flexibility markets, demand response, detection of voltage range violations, state estimation, and optimal power flow. UC-security means that our algorithm can be securely used as a component of other secure programs, e.g., for aforementioned complex applications.

– We benchmark our algorithm (using both LU and GMRES solvers) with several MPC protocols for two grids of varied topology and size. In addition, we break out runtime by network latency. Since the employed MPC protocols provide differing security guarantees, we can also show the efficiency of our algorithm for various threat models. This yields insights into the practicality of MPC in the context of PFA, and allows us to substantiate the claim that MPC-based privacy-preserving PFA can be practical for many threat models and real-world settings.

1.3 Efficient Constant-Round Outsourced MPC

At a high level, protocols enabling constant-round MPC can be broadly categorized into two families: FHE-based MPC and MPC from garbled circuits. In FHE-based

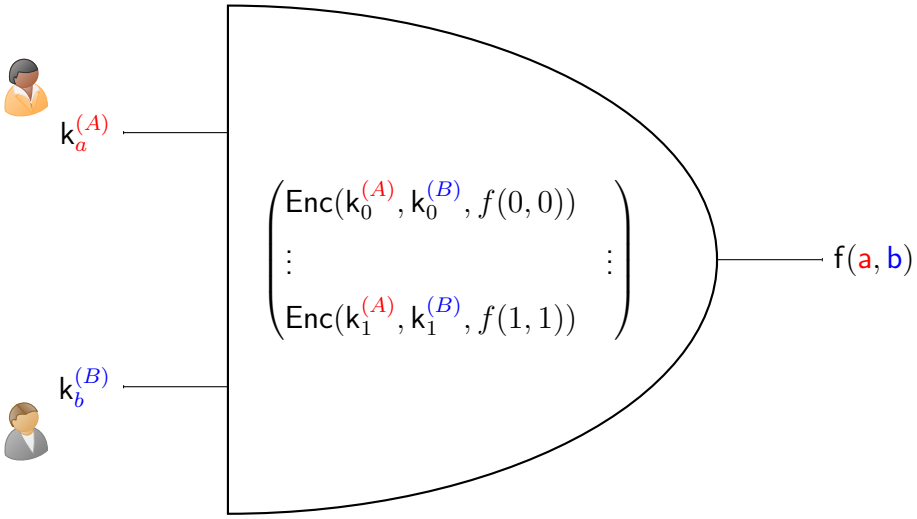

Figure 1.4: Garbled circuit for a 2-input gate that computes f.

MPC, participants use some form of threshold FHE scheme to encrypt their inputs and send the ciphertexts to the computing party, which homomorphically evaluates a function on them. The resulting ciphertext is then decrypted by the parties in a collaborative manner. This approach has the advantage of minimizing the computational load on the input parties, as the intensive task of homomorphic evaluation can be outsourced to a semi-honest high-performance server [Sma23]. Here, we will focus on MPC from garbled circuits which we introduce below.

1.3.1 MPC from garbled circuits.

Garbled circuits, introduced by Yao [Yao86], are a fundamental technique for secure *two-party* computation. The central idea is to represent a function f as a Boolean circuit and then "garble" it, creating an encrypted version in that can be evaluated without revealing intermediate values–see Figure 1.4. The protocol involves a garbler (Alice) and an evaluator (Bob). For each wire in the circuit, Alice generates two random cryptographic keys called labels, one corresponding to the bit value 0 and the other to 1. For each gate, she prepares a (randomly permuted) garbled table containing four ciphertexts. Each ciphertext encrypts an output wire label under a key derived from the two corresponding input wire labels.

As shown in Figure 1.5, the protocol proceeds as follows. First, Alice sends the entire garbled circuit G to Bob. To provide Bob with the labels for the input wires, they engage in an *oblivious transfer* (OT) protocol. For Bob's input bit b, OT allows him to learn the corresponding label $\mathsf{k}_b^{(B)}$ from Alice, without revealing b to her and without him learning the other label label $\mathsf{k}_{\bar{b}}^{(B)}$. Alice simply sends

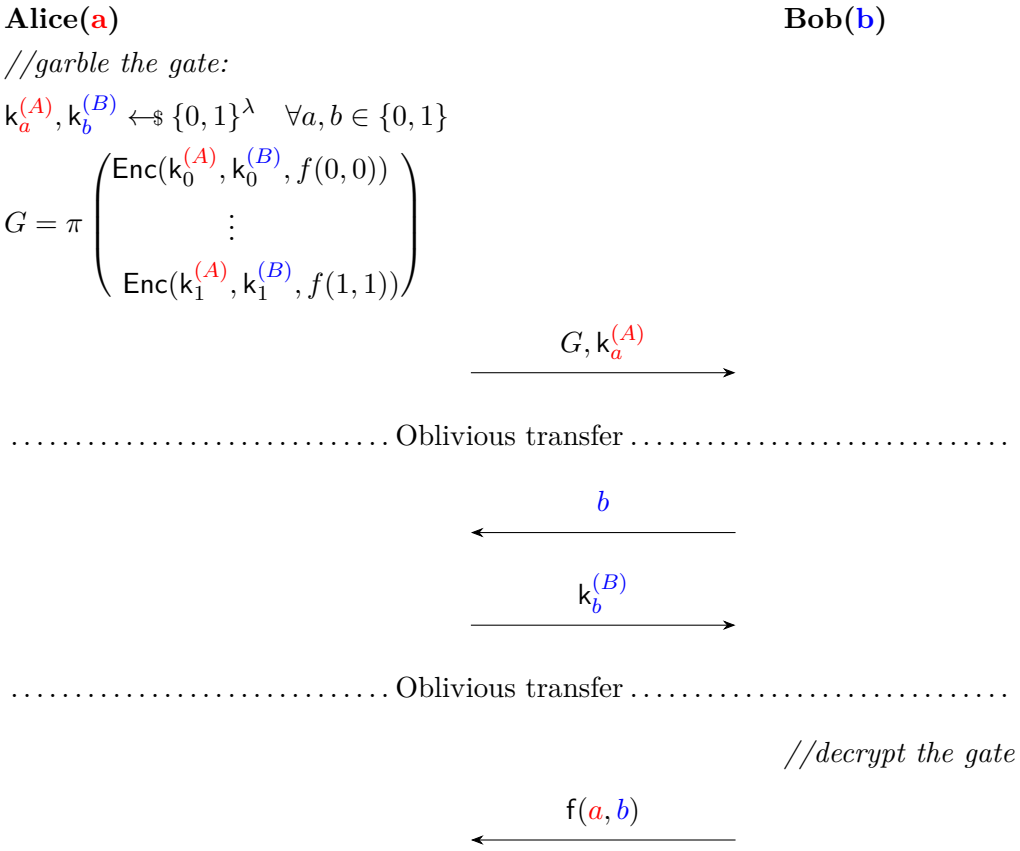

Figure 1.5: Garbled circuit protocol for a 2-input gate that computes f.

the labels for her own inputs directly. With the garbled circuit and the initial labels, Bob can evaluate the circuit gate-by-gate, using his input labels to decrypt the correct entry in each garbled table to obtain the next wire's label. Finally, Alice provides a mapping from the output wire labels to the plaintext output bits, allowing Bob to learn the result $f(a, b)$ while all other values remain hidden.

BMR. The first constant-round protocol for *multi-party* computation was the BMR protocol [BMR90] which utilizes a generic secret sharing protocol to securely garble the circuit to be computed during an (expensive) offline phase. Then, during the (cheap) online phase, the parties holding shares for the garbling can evaluate the circuit by selectively opening the input labels to the respective input parties. Thanks to a series of significant optimizations [LPSY19, HSS20], this technique has become highly practical; for instance, garbling an AES circuit (a common benchmark) can now be achieved in under 0.5 seconds.

Outsourced MPC from garbled circuits. More recently, Acharya et al. [AHKP22] presented a novel technique for constant-round MPC from *rerandomizable* garbled circuits, called *MPC with small clients and larger ephemeral servers* (SCALES), which we will explore in more detail below.

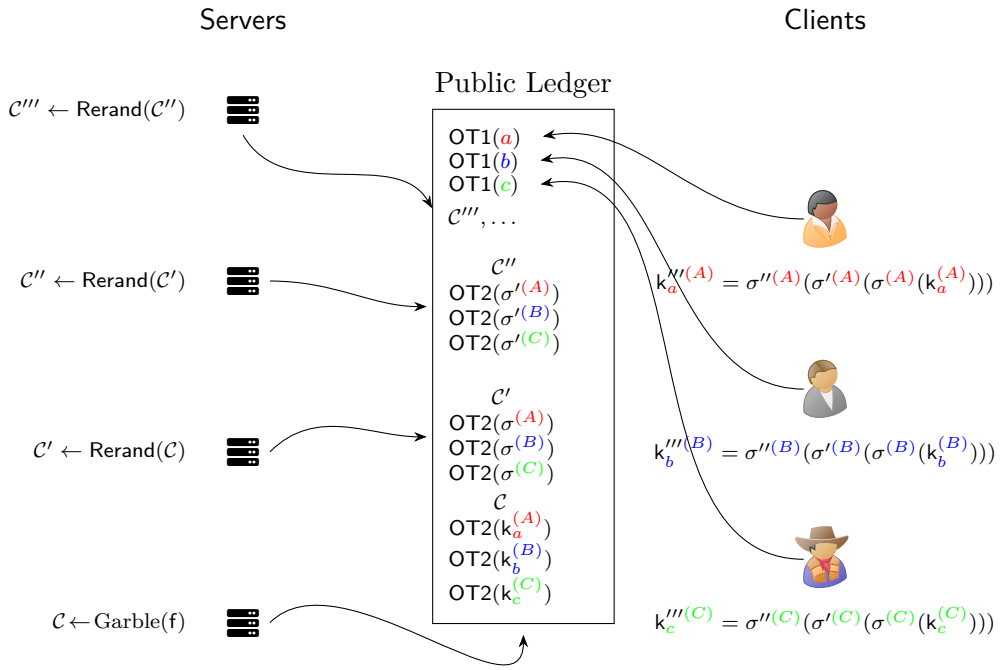

Figure 1.6: The SCALES protocol.

In SCALES, a potentially large group of *clients* supply the inputs while a smaller number of *servers* carry out the computation. As illustrated in Figure 1.6, each client first posts their inputs—cryptographically hidden via OT—to a public ledger. The first server garbles the circuit f and publishes the garbling along with the corresponding input labels, which are also hidden via OT, to the ledger. Note that OT guarantees that the clients can only learn the input labels corresponding to the inputs they posted in their first message. Every subsequent server rerandomizes the garbling, adds new OT responses for the transformed labels, and posts the result on the ledger. After the final rerandomization, each client locally derives its input label for the final circuit and broadcasts it to all other clients. Any entity can now evaluate the final circuit on these labels. Privacy is preserved as long as at least one server is honest.

Although less efficient, SCALES' advantage over BMR is that it provides outsourced MPC in the *you only speak once* (YOSO) setting. This implies that the

computational work can be securely outsourced to a small committee of servers. Since the servers can be selected spontaneously, e.g. via a blockchain, have to communicate only once, and then disappear, SCALES withstands even an adaptive adversary whose corruption budget may greatly exceed the number of servers.

The SCALES implementation presented in [AHKP22] utilizes garbled circuits based on the BHHO *key-and-message homomorphic encryption* (KMHE) scheme [BHHO08]. Since BHHO is highly inefficient, garbling an AES circuit on 16 cores, each running at four GHz, takes two years. The third contribution in this dissertation—presented in Chapter 5—is a new, more efficient KMHE construction, which reduces the garbling time by four orders of magnitude, thus enabling the first practical SCALES implementation.

Rerandomizable garbling schemes. Rerandomizable garbled circuits, also called *rerandomizable garbling schemes* (RGS), were first defined by Gentry et al. [GHV10] and provide the following functionality: Firstly, it is possible to rerandomize the garbled circuit such that it is indistinguishable from a fresh garbled circuit with the same topology and output on a given input. Hence, the underlying encryption scheme needs to support such rerandomization, which we call *KMH rerandomizability*.

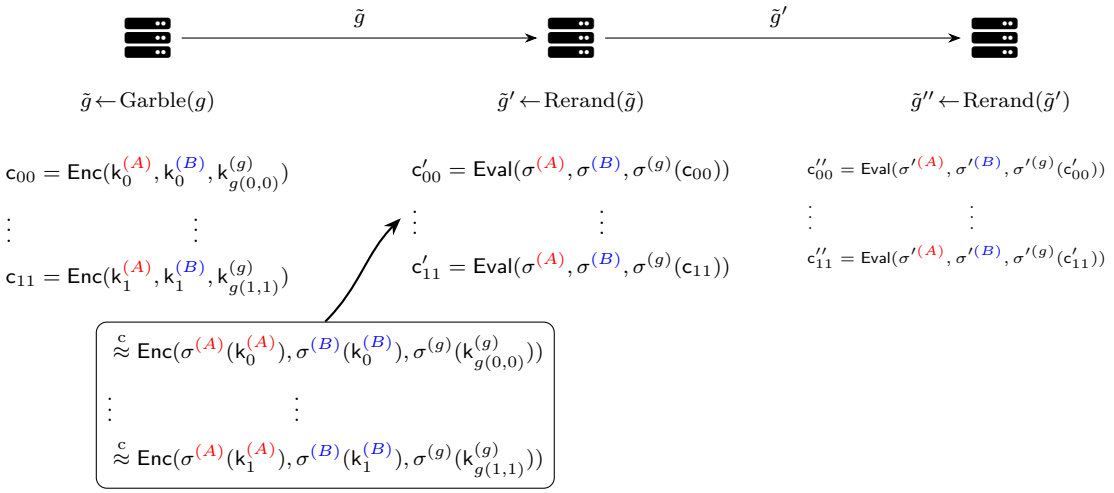

Figure 1.7: Rerandomizing a garbled gate $\tilde{g}$ by homomorphically evaluating the keys and messages in the gate's ciphertexts.

Secondly, as part of the rerandomization, we can apply a key-and-message homomorphism to the ciphertexts in connected gates, such that the labels on the output wire of the topologically earlier gate are consistently transformed with the labels encrypting the ciphertexts in the topologically later gate(s). Such a

homomorphic operation yields a garbled circuit consisting of completely transformed labels and rerandomized ciphertexts, while maintaining correctness. This requires the underlying encryption scheme to be key-and-message homomorphic, with the same homomorphism in the key and the message space. We illustrate the rerandomization of a garbled circuit consisting of a single garbled gate $\tilde{g}$ in Figure 1.7.

Key-and-message homomorphic encryption. Due to a subtle reason we require the underlying KMHE to fulfill a specific property, which is called *key privacy under leakage*. Consider the setting of SCALES and assume for simplicity that we have two servers of which the second one is honest, and two clients, Alice with input bit a and Bob with input bit b, where Alice is honest. Since the first server and Bob collude, at the end of the SCALES execution they know the input labels $\mathsf{k}_a^{(A)}$, $\mathsf{k}_{\overline{a}}^{(A)}, \sigma^{(A)}(\mathsf{k}_a^{(A)}), \mathsf{k}_b^{(B)}, \mathsf{k}_{\overline{b}}^{(B)}, \sigma^{(B)}(\mathsf{k}_b^{(B)}), \sigma^{(B)}(\mathsf{k}_{\overline{b}}^{(B)})$, and all ciphertexts in the garbled circuit (hence the adversary also learns all active labels in intermediate gates, and the function output). We now need two things to hold:

(i) For garbling privacy to hold, the adversary must still have only negligible advantage in breaking the urity of each KMHE ciphertext encrypted under an inactive label, e.g. $\mathsf{Enc}(\sigma^{(\ell)}(\mathsf{k}_{\overline{\mathsf{b}^{(\ell)}}}^{(\ell)}), \sigma^{(r)}(\mathsf{k}_{\mathsf{b}^{(r)}}^{(r)}), \mathsf{k}^{(i)})$. Hence, knowledge of $\mathsf{k}_{\mathsf{b}^{(\ell)}}^{(\ell)}, \sigma^{(\ell)}(\mathsf{k}_{\mathsf{b}^{(\ell)}}^{(\ell)})$ must reveal only very little about $\sigma^{(\ell)}$.

(ii) The adversary must not be able to distinguish whether $\sigma^{(i)}(\mathsf{k}_{\mathsf{b}^{(i)}}^{(i)})$ is a transformation of $\mathsf{k}_{\mathsf{b}^{(i)}}^{(i)}$ or a transformation of $\mathsf{k}_{\overline{\mathsf{b}}^{(i)}}^{(i)}$, for otherwise he would be able to learn the output of an intermediate gate.

We refer to properties (i) and (ii) as key privacy under leakage and KMH rerandomizability, respectively.

The difficulty in finding a suitable homomorphism σ is that simple operations such as addition and multiplication clearly do not support key privacy under leakage. The problem with more complex operations is that they are often not supported by the underlying encryption scheme. Therefore, until our work, the only known scheme to provide the required functional and security properties was BHHO [BHHO08] which relies on a permutation homomorphism over a space with an even number of Zero- and One-bits.

1.3.2 State of the art

The original SCALES paper [AHKP22] utilizes the expanded version of BHHO to build RGS. We begin by defining the key, message, and ciphertext spaces used in the scheme.

– The secret key space $\mathcal{K}$ is the set of all $(s_1, \ldots, s_\kappa) \in \{0,1\}^\kappa$ with Hamming weight $\kappa/2$.

– The message space is the set $\mathcal{M} = \mathbb{G}^\kappa$ where $\mathbb{G}$ is a cyclic group.

– The ciphertext space is the set $\mathcal{C} = \mathbb{G}^{(2\kappa+1)\times\kappa}$.

– The key homomorphism is $\mathcal{F}_{\mathsf{key}} = \mathbb{S}^\kappa$ where $\mathbb{S}^\kappa$ is the set of all permutations on κ positions and $f \in \mathcal{F}$ is represented by the binary sparse matrix M_f and the message homomorphism is $\mathcal{F}_{\mathsf{msg}} = \mathbb{S}^\kappa$ where $g \in \mathcal{F}$ is represented by the binary sparse matrix M_g.

Note that we use additive notation in this construction, i.e.

$$\vec{s} \cdot A \;=\; \begin{bmatrix} s_1 & \ldots & s_m \end{bmatrix} \begin{bmatrix} A_{1,1} & \ldots & A_{1,n} \\ \vdots & \ddots & \vdots \\ A_{m,1} & \ldots & A_{m,n} \end{bmatrix} \;:=\; \left[\prod_{j=1}^{m} A_{j,1}^{s_j}, \ldots, \prod_{j=1}^{m} A_{j,n}^{s_j} \right]$$

Construction (Expanded BHHO)**.** The expanded version of BHHO is defined as follows:

$\mathsf{KeyGen}(1^\kappa)$**:** Sample $\mathsf{sk} \leftarrow\!\!\$\ \mathcal{K}$;
output $\mathsf{sk} = (s_1, \ldots, s_\kappa)$.

$\mathsf{pkKeyGen}(\mathsf{sk})$**:** Sample a random matrix $\Psi \leftarrow\!\!\$\ \mathbb{G}^{(\kappa+1)\times\kappa}$ of rank κ;
output $\mathsf{pk} = [\Psi | -\Psi \cdot \mathsf{sk}] \in \mathbb{G}^{(\kappa+1)\times(\kappa+1)}$.

$\mathsf{Enc}(\mathsf{sk}, m)$**:** Compute $\mathsf{pk} \leftarrow\!\!\$\ \mathsf{pkKeyGen}(\mathsf{sk})$.
Then for all $m_i \in m$, sample $r_i \leftarrow\!\!\$\ \mathbb{Z}_q^{1\times(\kappa+1)}$ and compute $c_i = r_i \cdot \mathsf{pk} + [0^{1\times\kappa} | m_i]$.
Output $c = (c_1, \ldots, c_\kappa, \mathsf{pk})$.

$\mathsf{Dec}(\mathsf{sk}, c)$ For each c_i, compute $m_i = c_i^\top \cdot [\mathsf{sk}|1]^\top$.
Output $m = (m_1, \ldots, m_\kappa)$.

$\mathsf{Eval}(c, f, g)$**:** First rerandomize the ciphertext by sampling $r_i \leftarrow\!\!\$\ \mathbb{Z}_q^{1\times(\kappa+1)}$ for each c_i, and compute $c_i' = r_i \cdot \mathsf{pk} + c_i$.
To apply the key and message homomorphism, compute $c' = M_g \cdot c$ and $c'' = c' \cdot M_f^{-1}$ as well as $\mathsf{pk}' = \mathsf{pk} M_f^{-1}$. Finally, rerandomize the public key by sampling $r \leftarrow\!\!\$\ \mathbb{Z}_q^{(\kappa+1)\times(\kappa+1)}$ and compute $\mathsf{pk}'' = r \cdot \mathsf{pk}'$.
Output (c'', pk'').

The advantage of this KMHE scheme is that it is perfectly rerandomizable and therefore allows for a strong version of RGS rerandomization. The disadvantage is that one encryption holds $2\kappa^2 + 3\kappa + 1$ group elements and requires $\kappa^3 + \kappa^2$ exponentiations, which renders RGS based on this KMHE scheme impractical even for simple circuits.

1.3.3 Advancements to the state of the art

Our work replaces the inefficient BHHO scheme with a new, highly efficient KMHE scheme. While this new scheme does not achieve the perfect rerandomizability of BHHO, we show its security is sufficient for our purposes. To do so, we build RGS upon carefully defined, weaker security notions——namely key privacy under leakage and KMH rerandomizability——which our KMHE scheme is tailored to efficiently satisfy. In summary, our contributions are as follows:

– For the first time since BHHO, we present a new KMHE scheme with the same homomorphism in the key and message space. Our use of bilinear pairings makes it possible to encode the encryption randomness in one instead of κ group elements. We additionally introduce *gate KMHE*—which allows for reuse of another 2κ group elements between the ciphertexts of a garbled gate—and give a new construction for RGS which uses gate KMHE and the new security notions *key privacy under leakage* and *KMH rerandomizability*.

– Our RGS construction shrinks the size of a garbled gate by 98.99 % (from 133.43 MB to 1.35 MB) and decreases the estimated cost of garbling by 99.998 % (from 33 minutes to 0.04 seconds per gate) in comparison to Acharya et al. [AHKP22]. Moreover, it improves the cost of rerandomization from 2.23 hours to 0.05 seconds per gate, which is a reduction of 99.9993 %. Therefore our work shows for the first time that RGS and the SCALES protocol (and hence YOSO-like MPC) are practically feasible for simple circuits.

– While we focused on RGS, our KMHE can also be utilized for other applications. Firstly, it can be viewed as a batched version of BHHO, allowing for a more compact encryption of multiple messages. If we encrypt ℓ messages using BHHO with a key of length κ, then we get a ciphertext containing $\mathcal{O}(\ell\kappa)$ group elements, while our scheme only requires $\mathcal{O}(\ell + \kappa)$ group elements, with significantly better concrete efficiency. Furthermore, our scheme inherits the leakage resilience of BHHO, and thus could potentially also be used in constructions where BHHO is used as a leakage-resilient scheme, e.g. leakage-resilient CCA-secure PKE [FKMV12] or tamper-resilient PKE [DFMV17].

1.4 Publication Overview

This thesis builds upon the following peer-reviewed publications:

[FvdHM+23] Marc Fischlin, Jonas von der Heyden, Marian Margraf, Frank Morgner, Andreas Wallner, and Holger Bock. Post-quantum security for the extended access control protocol. In Felix Günther and Julia Hesse, editors, *Security Standardisation Research - 8th International Conference, SSR 2023, Lyon, France,*

April 22-23, 2023, Proceedings, volume 13895 of *Lecture Notes in Computer Science*, pages 22–52. Springer, 2023. doi:10.1007/978-3-031-30731-7_2

[vdHSB+25] Jonas von der Heyden, Nils Schlüter, Philipp Binfet, Martin Asman, Markus Zdrallek, Tibor Jager, and Moritz Schulze Darup. Privacy-preserving power flow analysis via secure multi-party computation. *IEEE Trans. Smart Grid*, 16(1):344–355, 2025. doi:10.1109/TSG.2024.3453491

[HvdHJ25] Raphael Heitjohann, Jonas von der Heyden, and Tibor Jager. Rerandomizable garbling, revisited. In *Advances in Cryptology - CRYPTO 2025, 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025. Proceedings*, Lecture Notes in Computer Science. Springer, 2025. URL: https://eprint.iacr.org/2025/843

Other publications not included in this thesis:

[MvdH21] Frank Morgner and Jonas von der Heyden. Analyzing requirements for post quantum secure machine readable travel documents. In Heiko Roßnagel, Christian H. Schunck, and Sebastian Mödersheim, editors, *Open Identity Summit 2021*, pages 205–210, Bonn, 2021. Gesellschaft für Informatik e.V

1.5 Outline

The remainder of this thesis is structured as follows:

Chapter 2: Preliminaries.
This chapter introduces the basic notions and conventions used in this thesis. Additionally, it defines the basic cryptographic primitives and complexity assumptions used throughout this thesis.

Chapter 3: Post-Quantum Security for the Extended Access Control Protocol.
In this chapter, we present the design and implementation of a post-quantum secure version of the EAC protocol, called *PQ-EAC*.

Chapter 4: Privacy-Preserving Power Flow Analysis via Secure Multi-Party Computation.
This chapter describes our privacy-preserving implementation of power flow analysis based on secure multi-party computation.

Chapter 5: Rerandomizable Garbling, Revisited.
In this chapter, we present a new KMHE scheme which allows for the construction of RGS and improves the efficiency of previous constructions by four to five orders of magnitude.

Chapter 6: Conclusion.

This chapter summarizes the contributions of this thesis and discusses future research directions.

2 Preliminaries

2.1 Notation

We define notation and conventions used throughout this thesis. We write the security parameter as λ. For every $n \in \mathbb{N}$, the symbol 1^n denotes the n-bit unary string, and $|x|$ denotes the bit-length of a bit string x. When we sample uniformly at random from a set S, we write $x \leftarrow\!\!\$\ S$. The shorthand $[n]$ stands for the index set $\{1, \ldots, n\}$. For a collection $\{a_1, \ldots, a_n\}$ we write the sequence as $(a_i)_{i \in [n]}$; we use the same convention for tuples. All algorithms are allowed to use randomness. Given a probabilistic polynomial-time algorithm $\mathcal{A}$, the expression $x \leftarrow \mathcal{A}(1^\lambda, (a_i) i \in [n])$ means that we run $\mathcal{A}$ on inputs $1^\lambda, a_1, \ldots, a_n$ with fresh random coins and assign the resulting output to x. When we want to expose the random coins explicitly, we write $x \leftarrow A(1^\lambda, (a_i) i \in [n]; r)$, where r denotes the randomness. When we execute $\mathcal{A}$ independently on each input a_i with fresh randomness each time, we write $(x_i) i \in [n] \leftarrow A(a_i) i \in [n]$.

The function $\max(a, b)$ returns the larger of the two comparable values a and b. An algorithm is called efficient if it runs in probabilistic polynomial time (PPT). We use $\text{poly}(\cdot)$ for an unspecified polynomial function and $\text{polylog}(\cdot)$ for an unspecified polylogarithmic function.

We consider a function $\mathsf{negl}(\lambda)$ to be *negligible*, if for every positive polynomial function $\mathsf{poly}(\lambda)$, there exists an integer n_0 such that, when $\lambda > n_0$, we will have $\mathsf{negl}(\lambda) < \frac{1}{\mathsf{poly}(\lambda)}$. Adversaries are modeled as non-uniform families $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$, and in security definitions we allow $\mathcal{A}$ to maintain state across invocations.

Let $\mathbb{N}$ be the set of natural numbers, not including 0.

Let $s \in \{0,1\}^\ell$ be a bit-string of length ℓ. The Hamming weight of s then denotes the number of 1's in s. By $HW_{\kappa,\kappa/2}$ denote the set of all bit-strings of length κ with Hamming weight $\kappa/2$. S_κ denotes the set of all permutations mapping $[\kappa] \to [\kappa]$. For $\sigma \in S_\kappa$ and a vector $x = (x_1, \ldots, x_\kappa)$ of κ elements, we overload notation and may occasionally also write $\sigma(x)$ to denote the vector $(x_{\sigma^{-1}(1)}, \ldots, x_{\sigma^{-1}(\kappa)})$. By $\sigma_2 \circ \sigma_1$ we denote standard function composition, so $\sigma_2 \circ \sigma_1 : x \mapsto \sigma_2(\sigma_1(x))$. To convert numbers from binary to decimal we define the function $\mathsf{Bin2Dec} : \{0,1\}^* \mapsto \mathbb{N} \cup \{0\}$. It uses the standard approach for converting binary numbers to non-negative decimal.

For a group $\mathbb{G}$ we then denote the identity element in $\mathbb{G}$ by $1_\mathbb{G}$. For a vector of

group elements $\mathbf{g} = (g_1, \ldots, g_\nu) \in \mathbb{G}^\kappa$ and a bit vector $\mathsf{sk} = (s_1, \ldots, s_\kappa) \in \{0,1\}^\kappa$ we write

$$\mathbf{g}^{\mathsf{sk}} = \prod_{j=1}^{\kappa} g_j^{s_j}$$

Let D_1 and D_2 be two distributions over some finite domain $\mathcal{W}$. Then define the statistical distance of D_1 and D_2 to be

$$\Delta(D_1, D_2) = \frac{1}{2} \sum_{x \in \mathcal{W}} |\Pr[D_1 = x] - \Pr[D_2 = x]|.$$

Let D_1 and D_2 be parameterized by $\lambda \in \mathbb{N}$. We write $D_1 \overset{\text{s}}{\approx} D_2$, read "$D_1$ and D_2 are statistically indistinguishable", if $\Delta(D_1, D_2) \leq \mathsf{negl}(\lambda)$, and $D_1 \overset{\text{c}}{\approx} D_2$ if D_1 and D_2 are computationally indistinguishable. We write $D_1 \equiv D_2$ if D_1 and D_2 are the same distributions, i.e. $\Delta(D_1, D_2) = 0$. For random variables X, Y, we denote the expectation of X, and the conditional expectation of X given Y as $\mathcal{E}[X]$, and $\mathcal{E}_{y \leftarrow Y}[X|y]$, respectively. $\mathbb{S}^\kappa$ is the set of all permutations on κ positions.

We denote the *Diffie-Hellman* (DH) key generation procedure with $\mathsf{DH.KeyGen}$ and the derivation of a shared DH key with $\mathsf{DH}(\cdot, \cdot)$.

2.2 Definitions for Key Exchange Protocols

In this section we introduce the underlying primitives and their security notions for building the key exchange protocols in Chapter 3.

2.2.1 Key encapsulation

Definition 1 (Key Encapsulation Mechanism)**.** *A key encapsulation mechanism* $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ *consists of three efficient algorithms where:*

Key Generation: *Algorithm* KeyGen *on input the security parameter* 1^λ *(in unary) outputs a key pair,* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$*. We assume that* 1^λ *is recoverable from either key.*

Encapsulation: *The encapsulation algorithm takes as input a public key* pk *and returns a ciphertext and a key,* $(c, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk})$*. We assume usually that the key is of bit length* λ*.*

Decapsulation: *The decapsulation algorithm takes as input a secret key* sk *and a ciphertext* c*, and returns a key or an error symbol,* $\mathsf{k} \leftarrow \mathsf{Decaps}(\mathsf{sk}, c)$*, where* k *is either of size* λ *or equals* $\perp$*. Usually decapsulation is deterministic.*

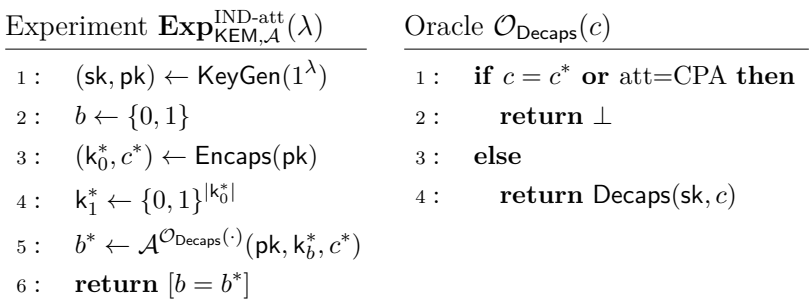

Figure 2.1: IND-CPA/IND-CCA security experiment for key encapsulation mechanism

We require that decapsulation merely has a negligible error. That is, we denote by $\Pr\left[\boldsymbol{Exp}^{decErr}_{\mathsf{KEM}}(\lambda) = 1\right]$ *the probability of an encryption error for* $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$*, where* $\mathsf{Decaps}(\mathsf{sk}, c) \neq \mathsf{k}$ *for* $(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{KeyGen}(\lambda)$ *and* $(c, \mathsf{k}) \leftarrow \mathsf{Encaps}(\mathsf{pk})$*.*

We next define CPA- and CCA-security for key encapsulation mechanism in one go:

Definition 2 (IND-CPA and IND-CCA security of KEM)**.** *For a key encapsulation mechanism* $\mathsf{KEM} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ *and adversary* $\mathcal{A}$ *define experiment* $\boldsymbol{Exp}^{IND\text{-}att}_{\mathsf{KEM},\mathcal{A}}(\lambda)$ *as in Figure 2.1. We say that* KEM *is IND-att secure (for att=CPA or CCA) if for any efficient adversary* $\mathcal{A}$ *we have that*

$$\boldsymbol{Adv}^{IND\text{-}att}_{\mathsf{KEM},\mathcal{A}}(\lambda) := \Pr\left[\boldsymbol{Exp}^{IND\text{-}att}_{\mathsf{KEM},\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2}$$

is negligible.

For our key exchange protocol KemPQEAC we also use a symmetric encryption scheme (for keys k from $\{0,1\}^\lambda$) where $c \leftarrow \mathsf{Enc}(\mathsf{k}, m)$ creates a ciphertext, and $m \leftarrow \mathsf{Dec}(\mathsf{k}, c)$ always recovers the encrypted message m. In the protocol this encryption step provides extra privacy for the chip by encrypting its identity. We do not discuss this privacy property formally here and thus neither the security notions for the encryption scheme.

2.2.2 Message authentication, signature schemes, and certificate schemes

We define message authentication schemes, signature schemes, and certificate schemes with a single definition. All schemes serve the purpose of authenticating

Experiment $\mathbf{Exp}^{\text{EUF-CMA}}_{\mathcal{AS},\mathcal{A}}(\lambda)$	Oracle $\mathcal{O}_{\mathsf{AAuth}}(\mathsf{sk}, m)$
1 : $(\mathsf{sk}, \mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{AKGen}(1^\lambda)$	1 : $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{m\}$
2 : $\mathcal{Q} \leftarrow \emptyset$	2 : $\sigma \leftarrow \mathsf{AAuth}(\mathsf{sk}, m)$
3 : $(m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{AAuth}}(\mathsf{sk},\cdot)}(\mathsf{pk})$	3 : **return** σ
4 : **return** $[\mathsf{SVf}(\mathsf{vk}, m^*, \sigma^*)$ **and** $m^* \notin \mathcal{Q}]$	

Figure 2.2: EUF-CMA security experiment for authentication schemes

data. The only difference between the private-key *message authentication code*s (MACs) and the public-key signature and certificate schemes lies in the verification key vk: MACs use the key $\mathsf{vk} = \mathsf{sk}$ to verify authenticity, whereas signatures and certificates are verified against the public key $\mathsf{vk} = \mathsf{pk}$. In the descriptions and security games we thus set vk accordingly, and the public key for MACs to be empty and for signatures and certificates equal to vk. We call the primitive abstractly an authentication scheme:

Definition 3 (Authentication Scheme)**.** *An authentication scheme* $\mathcal{AS} = (\mathsf{AKGen}, \mathsf{AAuth}, \mathsf{AVf})$ *consists of three efficient algorithms such that:*

Key Generation: *Algorithm* AKGen *on input* 1^λ *returns a key triple* $(\mathsf{sk}, \mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{SKGen}(1^\lambda)$*. We assume that* λ *is recoverable from either key.*

Authentication: *On input the secret key* sk *and a message* $m \in \{0,1\}^*$*, the authentication algorithm outputs an authenticator,* $\sigma \leftarrow \mathsf{AAuth}(\mathsf{sk}, m)$*.*

Verification: *On input a verification key* vk*, a message* m*, an authenticator* σ*, the verification algorithm outputs a decision bit,* $d \leftarrow \mathsf{AVf}(\mathsf{vk}, m, \sigma)$*. Usually,* AVf *is deterministic.*

We require that verification always succeeds. That is, it never happens that $\mathsf{AVf}(\mathsf{sk}, m, \sigma) = 0$ *for any* $(\mathsf{sk}, \mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{SKGen}(1^\lambda)$*, any* $m \in \{0,1\}^*$*, and any* $\sigma \leftarrow \mathsf{AAuth}(\mathsf{sk}, m)$*.*

Unlike key encapsulation, we define authentication schemes with perfect correctness since all known schemes in practice achieve this.

Definition 4 (EUF-CMA of Authentication Schemes)**.** *For an authentication scheme* $\mathcal{AS} = (\mathsf{AKGen}, \mathsf{AAuth}, \mathsf{AVf})$ *and adversary* $\mathcal{A}$ *define experiment* $\boldsymbol{Exp}^{\textit{EUF-CMA}}_{\mathcal{AS},\mathcal{A}}(\lambda)$ *as in Figure 2.2. We say that* $\mathcal{AS}$ *is EUF-CMA if for any efficient adversary* $\mathcal{A}$ *we have that* $\Pr\left[\boldsymbol{Exp}^{\textit{EUF-CMA}}_{\mathcal{AS},\mathcal{A}}(\lambda) = 1\right]$ *is negligible.*

A signature scheme $\mathcal{S} = (\mathsf{SKGen}, \mathsf{Sig}, \mathsf{SVf})$ is an authenticator scheme where $(\mathsf{sk}, \mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{AKGen}(1^\lambda)$ for $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{SKGen}(1^\lambda)$ and $\mathsf{vk} \leftarrow \mathsf{pk}$. A certificate

scheme $\mathcal{C} = (\mathsf{CKGen}, \mathsf{CSign}, \mathsf{CVf})$ is an authenticator scheme where $(\mathsf{sk}, \mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{AKGen}(1^\lambda)$ for $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{CKGen}(1^\lambda)$ and $\mathsf{vk} \leftarrow \mathsf{pk}$. A message authentication code $\mathcal{M} = (\mathsf{MKGen}, \mathsf{MAC}, \mathsf{MVf})$ is an authenticator scheme where $(\mathsf{sk}, \mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{AKGen}(1^\lambda)$ for $\mathsf{sk} \leftarrow \mathsf{MKGen}(1^\lambda)$ and $\mathsf{vk} \leftarrow \mathsf{sk}$ and $\mathsf{pk} \leftarrow \perp$. EUF-CMA security now follows from the general definition. We usually assume that the key generation algorithm simply generates a uniformly distributed key of λ bits.

2.2.3 Key derivation functions

We assume that key derivation functions act as pseudorandom functions, as long as the keying material contains enough (min-)entropy. The latter is captured by considering arbitrary distributions $\mathcal{D}$ which take the security parameter 1^λ as input and output IKM with min-entropy $H_\infty(\mathsf{IKM}) \geq \lambda$. We call such distributions *non-trivial*. We follow here the presentation of Krawczyk [Kra10].

Definition 5 (Key Derivation Function)**.** *A key derivation function* KDF *takes as input keying material* IKM*, context information* ctxt*, and an integer* ℓ*, and outputs a string of length* ℓ*. We assume that the length of* IKM *determines the security parameter* λ*.*

Security now requires that the key derivation function's output for IKM looks random, even if the adversary sees actual derived keys at other inputs (ctxt, ℓ):

Definition 6 (Pseudorandomness of Key Derivation Function)**.** *For a key derivation function* KDF*, adversary* $\mathcal{A}$*, and distribution* $\mathcal{D}$ *define experiment* $\boldsymbol{Exp}^{PRF}_{\mathsf{KDF},\mathcal{A},\mathcal{D}}(\lambda)$ *as in Figure 2.3. We say that* KDF *is pseudorandom if for any efficient adversary* $\mathcal{A}$ *and any non-trivial distribution (with min-entropy* λ*), we have that*

$$\boldsymbol{Adv}^{PRF}_{\mathsf{KDF},\mathcal{A},\mathcal{D}}(\lambda) := \Pr\Big[\boldsymbol{Exp}^{PRF}_{\mathsf{KDF},\mathcal{A},\mathcal{D}}(\lambda) = 1\Big] - \frac{1}{2}$$

is negligible.

2.2.4 Key combiners

For the forward-secure version of the PQEAC protocols we use a static KEM and an ephemeral KEM to share keys k and k^e. In this case the parties need to derive a single key from the two keys. This has been discussed more broadly in the context of KEM combiners in [GHP18] and for quantum adversaries in [BBF+19], but since we have already embedded the KEM mechanism in the key exchange protocol, we focus here on the pure key combining part. We note that we cannot immediately rely on the KEM combiner in [BBF+19], since it assumes faithfully

Experiment $\mathbf{Exp}^{\text{PRF}}_{\mathsf{KDF},\mathcal{A},\mathcal{D}}(\lambda)$

$\mathsf{IKM} \leftarrow \mathcal{D}(1^\lambda)$
$\mathcal{Q} \leftarrow \emptyset$
$b \leftarrow \{0,1\}$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{KDF}}(\mathsf{IKM},\cdot,\cdot),\mathcal{O}_b(\mathsf{IKM},\cdot,\cdot)}(1^\lambda)$
return $[b = b^*]$

Oracle $\mathcal{O}_{\mathsf{KDF}}(\mathsf{IKM}, \mathsf{ctxt}, \ell)$

if $(\mathsf{ctxt}, \ell) \in \mathcal{Q}$ **then**
  return $\perp$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{ctxt}, \ell)\}$
$\mathsf{k} \leftarrow \mathsf{KDF}(\mathsf{IKM}, \mathsf{ctxt}, \ell)$
return k

Oracle $\mathcal{O}_b(\mathsf{IKM}, \mathsf{ctxt}, \ell)$

if $(\mathsf{ctxt}, \ell) \in \mathcal{Q}$ **then**
  return $\perp$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(\mathsf{ctxt}, \ell)\}$
$\mathsf{k}_0 \leftarrow \mathsf{KDF}(\mathsf{IKM}, \mathsf{ctxt}, \ell)$
$\mathsf{k}_1 \leftarrow \{0,1\}^\ell$
return k_b

Figure 2.3: Pseudorandomness experiment for Key Derivation Functions

created but potentially weak encapsulations, whereas in our setting the adversary may choose the encapsulations maliciously. While Bindel et al. [BBF+19] argue that such genuine encapsulations are sufficient to build hybrid authenticated key exchange protocols via Krawczyk's SIGMA compiler [Kra03], the *Extended Access Control* (EAC) protocol does not perfectly comply with the SIGMA standard.

Definition 7 (Key Combiner)**.** *A key combiner* KComb *takes as input keying material* $\mathit{IKM}_0, \mathit{IKM}_1$*, both of length* n*, and outputs a string of length* n*.*

Security demands that KComb is a dual pseudorandom function, meaning that both $\mathsf{KComb}(\mathsf{IKM}_0, \cdot)$ and $\mathsf{KComb}(\cdot, \mathsf{IKM}_1)$ are pseudorandom functions. It follows that we can reasonably assume that HKDF resp. HMAC [Kra10, BL15] or the TLS-based nested key derivation function in [SS17] is an appropriate instantiation for a key combiner.

Definition 8 (Dual Pseudorandomness of Key Combiner)**.** *For a key combiner* KComb *and adversary* $\mathcal{A}$ *define experiment* $\mathbf{Exp}^{dPRF-\beta}_{\mathsf{KComb},\mathcal{A}}(\lambda)$ *as in Figure 2.4. We say that* KComb *is (dual) pseudorandom if for any efficient adversary* $\mathcal{A}$ *we have that*

$$\mathbf{Adv}^{dPRF}_{\mathsf{KComb},\mathcal{A}}(\lambda) := \max_{\beta\in\{0,1\}} \left\{ \Pr\left[\mathbf{Exp}^{dPRF-\beta}_{\mathsf{KComb},\mathcal{A}}(\lambda) = 1\right] - \frac{1}{2} \right\}$$

is negligible.

Experiment $\mathbf{Exp}^{\mathrm{dPRF}-\beta}_{\mathsf{KComb},\mathcal{A}}(\lambda)$

$\mathsf{IKM}_0, \mathsf{IKM}_1 \leftarrow \{0,1\}^\lambda$
$b \leftarrow \{0,1\}$
$\mathcal{Q} \leftarrow \emptyset$
$b^* \leftarrow \mathcal{A}^{\mathcal{O}_{b,\beta}(\mathsf{IKM}_0, \mathsf{IKM}_1, \cdot)}(1^\lambda)$
return $[b = b^*]$

Oracle $\mathcal{O}_{b,\beta}(\mathsf{IKM}_0, \mathsf{IKM}_1, x)$

if $x \in \mathcal{Q}$ **then return** $\perp$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{x\}$
if $\beta = 0$ **then**
  $\mathsf{k}_0 \leftarrow \mathsf{KComb}(\mathsf{IKM}_0, x)$
else
  $\mathsf{k}_0 \leftarrow \mathsf{KComb}(x, \mathsf{IKM}_1)$
$\mathsf{k}_1 \leftarrow \{0,1\}^{|\mathsf{k}_0|}$
return k_b

Figure 2.4: Dual pseudorandomness experiment for Key Combiners

2.3 Secure Multi-party Computation

Below, we introduce secure *multi-party computation* (MPC) which is relevant for Chapter 4 and Chapter 5. We also give a more detailed introduction to secret sharing.

In MPC, n parties (sometimes also called players) $\mathsf{P}_1, \ldots, \mathsf{P}_n$ agree to jointly compute a function $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{z})$, where $\boldsymbol{y} = (y_1, \ldots, y_m)^\top$ and $\boldsymbol{z} = (z_1, \ldots, z_n)^\top$ with $m, n \in \mathbb{N}$. Here, z_i denotes P_i's secret input and $\boldsymbol{y}$ a vector of outputs. The function $\boldsymbol{g}$ is said to be computed *securely* if the following two conditions are satisfied:

– *Correctness:* The correct result $\boldsymbol{y}$ is computed.

– *Privacy:* None of the parties obtains any new information about $\boldsymbol{z}$, other than what can be inferred from $\boldsymbol{y}$.

Up to $n-1$ parties may collude to try to learn about the secrets of other parties. We model this by assuming that an adversary that tries to learn about the secret inputs may corrupt up to t of n parties. In the cases $t \leq \lfloor n/2 \rfloor$ and $t \leq n-1$, we say that we assume an *honest majority* or a *dishonest majority* of parties, respectively. Depending on the threat model, corrupted parties may either be

– *semi-honest* (also called *honest-but-curious*), meaning they try to learn secret inputs using the function output and the information released to them during the regular execution of the MPC protocol, but otherwise follow the protocol faithfully, or

– *malicious*, meaning they may do all of the above and also deviate arbitrarily in their behavior during protocol execution.

Note that the adversary may pool the information gathered by corrupted parties together and (in the malicious model) coordinate the protocol deviations of corrupted parties to learn more information.

In our context, the role of the parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ can be assigned to the prosumers, while the prosumers' secret power data is collected in $\boldsymbol{z}$. However, revealing the node voltages $\mathbf{v}$ would clearly breach the prosumers' privacy. Thus, the output $\boldsymbol{y}$ must represent a less vulnerable result, for instance checking operating conditions securely via (4.3). Then, in case of a violation, the operator receives a stabilization strategy which, of course, must be revealed to it. We omit specifying additional computations and focus on the solution of (4.1) here. Thus, $\boldsymbol{y} = \boldsymbol{g}(\boldsymbol{z})$ can mainly be understood as a mapping of power data to the voltage $\mathbf{v}$ such that (4.1) is satisfied.

2.3.1 Secret sharing

One approach towards MPC is secret sharing. It provides a way for a party P_i to distribute information about a secret z_i across all parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ such that they all together hold full information on z_i, yet no party on its own (except P_i) has any information about z_i. For simplicity, we consider a three-party scheme at first: Let $z_1 \in \mathbb{Z}_p$ represent P_1's secret, with p prime. Note that computations on secret-shared numbers are performed over $\mathbb{Z}_p$, which implies a reduction modulo p. To share the secret, P_1 chooses r_1, r_2 uniformly at random in $\mathbb{Z}_p$ and sets $r_3 = z_1 - r_1 - r_2$. Observe that for each r_1, r_2 and r_3 all values in $\mathbb{Z}_p$ are equally likely, which is why they do not contain any information about z_1. Therefore, we can distribute the *shares* r_1, r_2 and r_3 to $\mathsf{P}_1, \mathsf{P}_2$ and P_3, respectively, such that no party alone learns something new about z_1, but all players together can *reveal* (or "*open*") the secret by computing $z_1 = r_1 + r_2 + r_3$. We denote a secret-shared number z_i as $[z_i]$ and its j'th share as $[z_i]^{(j)}$. Thus, in the case above, we can also write $[z_1] = ([z_1]^{(1)}, [z_1]^{(2)}, [z_1]^{(3)}) = (r_1, r_2, r_3)$.

It is easy to see that $\mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3$ can compute $z_4 = z_1 + z_2 + z_3$ where z_1, z_2, z_3 are arbitrary secret-shared numbers. First, each player P_i (for $i \in \{1, 2, 3\}$) locally computes $[z_4]^{(i)} = [z_1]^{(i)} + [z_2]^{(i)} + [z_3]^{(i)}$. Then, to *reveal* z_4, they can send each other their shares of z_4 and compute $z_4 = [z_4]^{(1)} + [z_4]^{(2)} + [z_4]^{(3)}$. From now on we write $[z_1] + [z_2]$ to mean that each player P_i computes the sum of their local shares $[z_1]^{(i)}, [z_2]^{(i)}$ as described above. Addition, subtraction and multiplication of a secret-shared number with a public constant, as well as subtraction of two secret-shared numbers, can be computed locally in a similar fashion.

In order to compute the product of two secret-shared numbers, one can use *multiplication triples* [Bea92]. Here, we assume that the n players are in possession of secret-shared triples $[a], [b], [c]$ of the form $c = ab$ where a, b are chosen uniformly

at random from $\mathbb{Z}_p$. Now observe that

$$\begin{aligned} z_1 z_2 &= (z_1 + a - a)(z_2 + b - b) \\ &= (z_1 + a)z_2 + (z_1 + a)b - (z_1 + a)b - (z_2 + b)a + ab \\ &= (z_1 + a)z_2 - (z_2 + b)a + c. \end{aligned}$$

Assuming a, b and c are secret, the parties can open $z_1 + a$ and $z_2 + b$ without any loss of privacy and compute

$$[z_1 z_2] = (z_1 + a)[z_2] - (z_2 + b)[a] + [c]$$

based on addition and multiplication by a public constant. From now on we write $[z_1][z_2]$ to mean that $\mathsf{P}_1, \ldots, \mathsf{P}_n$ follow a multiplication procedure such as the one described above to securely compute $[z_1 z_2]$. Remarkably, it is possible to express any function in terms of addition and multiplication gates. We omit giving details and refer the interested reader to [CS10, AS19] for algorithms for standard mathematical functions.

Thus, the approach taken above allows secure computation of arbitrary functions in the *preprocessing model* even with a *dishonest majority* where all but one of the parties are dishonest and colluding. We refer to multiplication triples and other random numbers that are helpful during computation but unrelated to the actual secret inputs or the computed function as *correlated randomness*. When we say that a computation is secure in the preprocessing model, we mean that the computation is secure under the assumption that the correlated randomness required for the computation has been generated securely. This is the case if, for example, the correlated randomness has either been generated and distributed by a trusted third party (a so-called *dealer*), or if the correlated randomness has been generated interactively among the computing parties using secure protocols based on *homomorphic encryption* (HE) [KPR18] or *oblivious transfer* (OT) [KOS16a]. We refer to the generation of correlated randomness as the *offline* or *preprocessing* phase of the computation, and denote the input-dependent computation as the *online* phase. We can avoid the preprocessing model by, for example, using the MPC protocols BGW [BGW88] or CCD [CCD88].

Lastly, let us consider the complexity of several operations as a preparation for the following section. While local computation in secret sharing is typically cheap, communication constitutes a bottleneck. Against this background, the number of *communication rounds* (how often a communication is invoked) is a useful metric. Recall that additions and multiplications of secret-shared numbers with public constants, and $[z_1] + [z_2]$ are communication-free. On the other hand, $[z_1][z_2]$ requires at least one communication round, and (due to truncation [CS10]) at least two for fixed-point numbers. Elementary functions that are useful for the

solution of the power flow problem (4.1) are divisions, trigonometric functions like $\sin(z)$, and square roots of secret-shared numbers. Based on the methods presented in [AS19], these require approximately 40, 40, and 87 rounds, respectively, for a 64-bit number.

3 Post-Quantum Security for the Extended Access Control Protocol

Author's contribution. The results in this chapter are based on joint work with Marc Fischlin, Marian Margraf, Frank Morgner, Andreas Wallner and Holger Bock, published in [FvdHM+23]. The author's contributions include the design of all versions of the PQ-EAC protocol. Marc Fischlin provided the security proofs. Frank Morgner and Andreas Wallner implemented PQ-EAC and conducted benchmarks. The author of this thesis also contributed to the implementation of the protocols in the benchmarking software. Marian Margraf and Holger Bock provided project supervision.

3.1 Motivation and Overview

The *Extended Access Control* (EAC) protocol for authenticated key agreement was proposed by the *German Federal Office for Information Security* (BSI) for German *electronically-enabled passport*s (ePassports) in 2005 and is defined in the technical guideline TR-03110 [Bun16]. EAC is meant to provide authenticated key establishment between a terminal and a smart card. ICAO Doc 9303 [Int21], a standard for *electronic machine-readable travel document*s (eMRTDs) such as ePassports, mandates authentication via EAC before a terminal may request sensitive data such as fingerprints from the chip in an eMRTD. The security of EAC rests on assumptions about the hardness of computing discrete logarithms. Due to the results of Shor [Sho94], these assumptions are now considered to be false when taking into account quantum computers. In this chapter we present PQ-EAC, a new version of the EAC protocol that achieves security against attacks from quantum computers by substituting Diffie-Hellman key exchange with post-quantum *key encapsulation mechanism*s (KEMs).

Post-quantum cryptography. In 1994, Peter Shor presented a quantum algorithm that solves integer factorization and discrete logarithms—two problems for which no efficient algorithm on classical computers is known—in polynomial time [Sho94]. Shor's work thereby falsified the foundational assumption of public-key cryptography that factorization and discrete logarithms are hard problems. While only a quantum

computer of sufficient size could break public-key cryptography as used in modern internet infrastructure, recent developments show that such a machine is a realistic scenario [A+25].

Fortunately, there are alternatives to cryptography based on factorization or discrete logarithms. In 2022, *National Institute of Standards and Technology* (NIST) announced that they will standardize the lattice-based KEM Kyber [SAB+22], two lattice-based signature schemes (Dilithium [LDK+22], Falcon [PFH+22]) and the hash-based signature scheme SPHINCS+ [HBD+22] as quantum-resistant cryptographic mechanisms [AAC+22]. In addition, the hash-based signature schemes LMS [MCF19] and XMSS [HBG+18] have already been recommended in NIST SP 800-208 [Nat20].

Hybrid protocols. Many of these conjecturally *post-quantum secure* schemes have not yet been adequately studied in terms of their security. This concerns not only cryptanalytic attacks using classical computers, but also side-channel and error attacks and the exploitation of implementation errors. Therefore, it is prudent to use hybrid protocols, i.e., a combination of classical and quantum-resistant primitives; see also, for example, the recommendations of the German Federal Office for Information Security [Bun20].

Post-quantum security for the EAC protocol. Even though a quantum computer powerful enough to break modern cryptography might be more than a decade away, in the context of identity documents urgent action is required. This is because identity documents typically have a validity period of ten years and technical changes require lengthy regulatory approval. Therefore, the development of a new quantum-resistant version of the EAC protocol, ideally using hybrid schemes that combine post-quantum and classical algorithms, is imperative. In this chapter, we showcase several versions of a quantum-resistant EAC protocol, called *PQ-EAC*, with varying security properties and trade-offs. We also provide an efficient software implementation of this new protocol for contactless smart cards (in eMRTDs) as well as inspection terminals, in order to create a blueprint for the standardization of the new EAC protocol with *International Civil Aviation Organization* (ICAO). Our benchmarks show that our implementation is efficient enough for use in practice.

3.1.1 Notation

To distinguish between the EAC protocol as defined by BSI [Bun16] and our proposal for post-quantum EAC, we call the former *classic EAC* or *EAC classic* and the latter *PQ-EAC*. We continue to write EAC when referring to the protocol as such. EAC is usually conducted between two parties: One the one side we

have an eMRTD which might be an identity card, a passport or similar, in the following called "chip". On the other side we have a chip reader, which could be an inspection terminal at border control, a device to authenticate identity cards for e-government services, or similar, in the following called "terminal". The terms MRTD and eMRTD are used in this document as a generic reference to all types of machine-readable travel documents based on optical character reading or electronically enabled means. Examples of eMRTDs are ePassports, laissez-passer, identity cards, seafarer cards and refugee travel documents.

We use the notion of *forward secrecy* as given in Boyd and Gellert [BG19]: An *authenticated key exchange* (AKE) protocol provides forward secrecy if compromise of long-term secrets does not lead to compromise of session keys of previously completed sessions.

3.1.2 Related work

There have been multiple works concerned with making Diffie-Hellman-based protocols quantum-resistant: Schwabe et al. [SSW20] present post-quantum versions of TLS, including a version with mutual authentication. Hülsing et al. [HNS+21] show how to achieve post-quantum security for the handshake protocol of the WireGuard VPN. Brendel et al. [BFG+20] attempt to give a post-quantum alternative to Signal's X3DH handshake using split KEMs. Finally, Angel et al. [ADH+22] introduce a post-quantum variant of the Noise framework. Unlike PQ-EAC, the protocols mentioned above cannot serve as drop-in replacements for EAC as standardized in TR-03110. However, our work and the mentioned protocols share the basic idea of replacing a Diffie-Hellman key exchange with KEMs to achieve post-quantum security.

Recent publications in regards to ePassports have been concerned with the security of *Basic Access Control* (BAC) [AKQ08, BFK09, FHMS19], the security of *Password-Authenticated Connection Establishment* (PACE) [BFK09, BDFK12] and the security of classic EAC [CV10, DF11]. Besides a status report [MvdH21], and a master thesis by Kußmaul [Kuß19] to the best of our knowledge there have not been any publications regarding the post-quantum security of the EAC protocol.

3.1.3 Outline

We start out with a discussion of ePassports in Section 3.2 and exhibit the original EAC protocol in Section 3.3. In Section 3.4 we subsequently propose several modifications to EAC, which will replace the Diffie-Hellman-style key exchange with KEMs. Afterwards, we give security proofs for the new protocols in Section 3.5. Finally, we discuss our implementation and give performance benchmarks in Section 3.6.

3.2 ePassports

The ICAO, a specialized agency of the United Nations, introduced standards for electronic eMRTDs and specifically ePassports in 2006 in Volume 2 of the sixth edition of ICAO Doc 9303 [Int21]. Since then, more than 140 countries have adopted ePassports, with 90 countries enrolled in the ICAO Public Key Directory.

Security mechanisms for ePassports. In the following we give an overview of the security goals of ePassports and what mechanisms are used to achieve these goals. Firstly, because they use contactless *radio-frequency identification* (RFID) technology to communicate with terminals, ePassports are subject to *skimming* attacks: In skimming attacks, the attacker retrieves data from the chip by making connection attempts in close physical proximity of the eMRTD. Even if the attacker cannot retrieve any information from the passport, he might be able to link it to previous connection attempts, meaning that he is able to *track* the passport. If a eMRTD is secure against such attacks, it is said to be *unlinkable*. To achieve unlinkability in the ePassport, the *machine-readable zone* (MRZ) is used as trust anchor and read by the terminal via *optical character recognition* (OCR) instead of RFID. BAC, and alternatively PACE, are security mechanisms that derive a shared key between chip and terminal and authenticate the terminal based on the knowledge of the MRZ. This minimal type of authentication allows the terminal access to information that can be read by anyone in physical possession of the ePassport, including primary biometrics (facial image). The underlying assumption here is that when a passport holder allows a terminal to read the MRZ of his passport via OCR, he is also consenting to the transmission of other data visually recognizable on the document. The access to secondary biometrics, meaning either fingerprints or iris data, requires the execution of Terminal Authentication (a subprotocol of EAC).

Secondly, the ePassport needs to be protected against attacks on its integrity: One way to forge a passport would be to modify the data groups (name, nationality, etc.) on a passport's chip. To prevent this, the passport issuer hashes all data groups, signs them and stores them in the Document Security Object (SO_D). To verify data integrity, a terminal must then use *Passive Authentication* (described below). An adversary could overcome these provisions by copying the SO_D to a new chip (also called *cloning*). Assuming that the protections provided for the chip's private key successfully prohibit any access to it, cloning can be detected during EAC when the chip fails to generate a shared key.

BAC vs. PACE. As mentioned above, the terminal establishes a weak kind of authentication based on the MRZ, either using BAC or PACE. BAC uses a

deterministic algorithm to derive a key from the MRZ. Since the MRZ consists of the holder's birth date, the eMRTD's expiry date and a 9-digit serial number[1], the maximum entropy of this key is 73 bits, but in practice the entropy of the key can be assumed to be 40-50 bits [BFK09]. The low entropy of the key permits *sniffing* attacks where the communication between terminal and chip is recorded by an eavesdropper and later decrypted by guessing the shared key [LKLP07, AKQ08]. Moreover, Filimonov et al. showed in 2019 that it is possible to track passports using BAC [FHMS19]. Due to these weaknesses, it is recommended (and mandated for ePassports issued after 2017) to use PACE instead of BAC [BFK09]. During PACE, chip and terminal generate a high-entropy shared key through *password-authenticated key exchange* (PAKE), an asymmetric key exchange procedure based on a shared (possibly low-entropy) password.

Passive Authentication. After the execution of BAC or PACE, the terminal uses a protocol called *Passive Authentication* to authenticate the data that is stored on the chip. Passive Authentication involves the following steps:

– The terminal reads the *Document Security Object* (SO_D) from the chip and retrieves the corresponding Document Signer certificate, the *Country Signing Certificate Authority* (CSCA) certificate and the corresponding certificate revocation list. The SO_D holds hashes of all data groups and a signature over these hashes. Note that the chip's public key is also stored in one of these data groups.

– The terminal verifies the Document Signer certificate.

– The terminal then computes the hash values of all data groups it has access to and compares them to the hash values from the SO_D. Subsequently it verifies the signature over these hash values using the public key from the Document Signer certificate.

Passive Authentication does not protect against cloning attacks, where the SO_D (but not the secret key which is assumed to be secured against copying) is copied from one eMRTD to another, but only verifies the integrity of the data groups on the chip. To prove that the chip is actually in possession of the corresponding secret key, EAC and Chip Authentication come into play.

[1] Depending on the issuing country, the serial number is subject to various constraints: Some digits may be predictable check digits, some blocks may indicate the issuing passport office, etc.

3.3 Classic EAC Protocol

We now proceed to describe the classic EAC protocol. EAC was devised by the Brussels Interoperability Group as a mechanism to protect sensitive data stored in eMRTDs and is published by the BSI in their technical report TR-03110 [Bun16]. EAC as defined in this specification includes two authentication mechanisms, *Chip Authentication* (CA) and *Terminal Authentication* (TA): CA authenticates the chip and TA prevents the reading of biometric data from unauthorized terminals. While EAC originated in the context of travel documents, it is also utilized in different contexts: For example, version 2 of EAC authenticates transactions performed by the German identity card, such as proof of identity or e-government. Even though we focus on the passport context in this work, most of our results apply more generally. EAC works as follows: The terminal uses CA to verify the authenticity of the chip. Vice versa, the chip can verify the inspection system's authorization to read sensitive biometric data such as fingerprints or iris data during TA [Int21]. In EAC Version 1, which is included in ICAO's standard for ePassports, chip and terminal first execute CA and then TA. If EAC is not executed, the terminal is not allowed to read secondary biometric data. To ensure more privacy for the eMRTD holder, in (the otherwise identical) version 2 of EAC chip and terminal first perform TA and then CA, and the terminal is only allowed to read data groups once PACE, TA and CA have completed. EAC Version 2 is, for example, used by the German identity card. In this work we aim for the stronger security properties of EAC Version 2 but maintain compatibility with EAC Version 1.

Public-key infrastructure. In order for chip and terminal to be able to mutually authenticate each other during EAC, each document issuing country has established a *public-key infrastructure* (PKI), which consists of the following entities (among others): a CSCA and a *Country Verifying Certificate Authority* (CVCA), Document Signer certificates, Document Verifier certificates and certificate revocation lists. Document Signer certificates and key pairs are issued by the CSCA to the entity that is manufacturing the eMRTDs. The private key is then used to sign a list of hashes of the data groups of the eMRTD. The signature, hashes and the respective Document Signer certificate are then stored in the SO_D and can be used by the eMRTD to authenticate itself to a terminal.

Similarly, the CVCA issues manufacturers of terminals Document Verifier key pairs and certificates, which can be used by a terminal to authenticate itself towards chips.

To enable international interoperability, all countries share their CSCA and CVCA public keys (via a master list), Document Signer certificates, Document Verifier certificates and certificate revocation lists in the ICAO Public Key Directory.

Terminal Authentication. The goal of TA as shown in Figure 3.1 is to prove to the chip that the inspection terminal is authorized to view sensitive data of the chip. In the context of passports, sensitive data means secondary biometric data such as fingerprints or iris data of the passport holder. To show that the terminal is in possession of a key pair $(\mathsf{pk}_{\mathcal{T}}, \mathsf{sk}_{\mathcal{T}})$ that is validated by a Document Verifier certificate, the terminal initiates a challenge-response procedure.

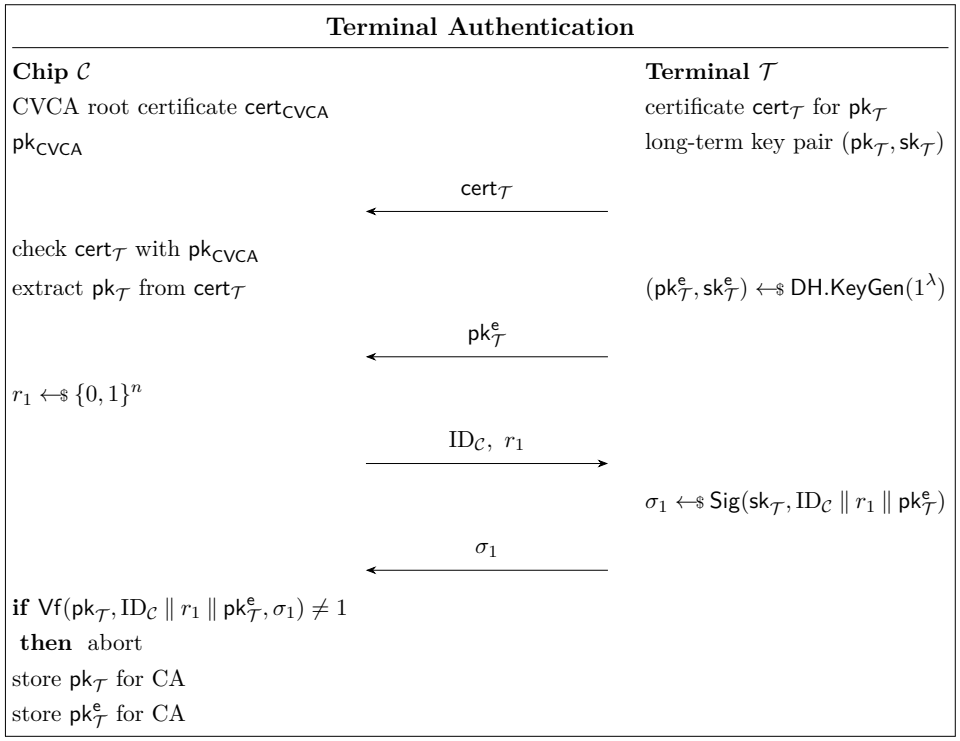

Figure 3.1: Terminal Authentication in the classic EAC protocol.

Firstly, the terminal generates an ephemeral Diffie-Hellman key pair $(\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}, \mathsf{sk}^{\mathsf{e}}_{\mathcal{T}})$ and sends $\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}$ to the chip. It also sends its certificate $\mathsf{cert}_{\mathcal{T}}$, which consists of the Document Verifier certificate and the validation of its public key $\mathsf{pk}_{\mathcal{T}}$ by the Document Verifier. Subsequently, the chip verifies the certificate and the terminal's public key $\mathsf{pk}_{\mathcal{T}}$.

To generate the challenge, the chip chooses uniformly at random a nonce r_1 and sends it to the terminal. In addition, it sends a pre-agreed value $\mathrm{ID}_{\mathcal{C}}$. If PACE is used, $\mathrm{ID}_{\mathcal{C}}$ is a suitable hash of the chip's public key from the PACE key agreement; this serves to bind EAC with the previous PACE execution. If BAC is used, it is the document number read from the MRZ.

Finally, the terminal proves possession of the secret key $\mathsf{sk}_{\mathcal{T}}$ by generating a signature over r_1, $\mathrm{ID}_{\mathcal{C}}$ and its ephemeral public key. The chip concludes the TA if

it can successfully verify the signature using $\mathsf{pk}_{\mathcal{T}}$.

Chip Authentication. In CA (shown in Figure 3.2), the chip $\mathcal{C}$ authenticates itself to the terminal $\mathcal{T}$ with its static *Diffie-Hellman* (DH) public key pair $(\mathsf{sk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{C}})$ that is validated by the Document Signer. At the outset, the chip sends the terminal its Document Signer certificate, its SO_D and its static public key $\mathsf{pk}_{\mathcal{C}}$, all of which we will call $\mathsf{cert}_{\mathcal{C}}$ in Figure 3.2.[2] Afterwards, the terminal sends the ephemeral public Diffie-Hellman key $\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}$ generated during the TA. As the chip already received $\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}$ during TA, one might wonder whether this message could be eliminated. Unfortunately, this is not possible due to the need to maintain compatibility with EAC Version 1 where the TA is optional.

Chip Authentication

Chip $\mathcal{C}$		**Terminal** $\mathcal{T}$
static DH key pair $(\mathsf{sk}_{\mathcal{C}}, \mathsf{pk}_{\mathcal{C}})$		CSCA root certificate $\mathsf{cert}_{\mathsf{CSCA}}$
certificate $\mathsf{cert}_{\mathcal{C}}$ for $\mathsf{pk}_{\mathcal{C}}$		ephemeral key pair $(\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}, \mathsf{sk}^{\mathsf{e}}_{\mathcal{T}})$
$r_2 \leftarrow\!\!\$\ \{0,1\}^n$		
	$\mathsf{cert}_{\mathcal{C}}, \mathsf{cert}_{\mathsf{DS}} \rightarrow$	
		check $\mathsf{cert}_{\mathcal{C}}$ with $\mathsf{cert}_{\mathsf{CSCA}}$
		extract $\mathsf{pk}_{\mathcal{C}}$ from $\mathsf{cert}_{\mathcal{C}}$
	$\leftarrow \mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}$	
verify that $\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}$ matches $\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}$ from TA		$\mathsf{k} = \mathsf{DH}(\mathsf{sk}^{\mathsf{e}}_{\mathcal{T}}, \mathsf{pk}_{\mathcal{C}})$
$\mathsf{k} = \mathsf{DH}(\mathsf{sk}_{\mathcal{C}}, \mathsf{pk}^{\mathsf{e}}_{\mathcal{T}})$		
$\mathsf{k}_{\mathsf{MAC}} = \mathsf{KDF}_{\mathsf{MAC}}(\mathsf{k}, r_2)$		
$\mathsf{k}_{\mathsf{ENC}} = \mathsf{KDF}_{\mathsf{ENC}}(\mathsf{k}, r_2)$		
$\mathsf{t} = \mathsf{MAC}(\mathsf{k}_{\mathsf{MAC}}, \mathsf{pk}^{\mathsf{e}}_{\mathcal{T}})$		
	$r_2, \mathsf{t} \rightarrow$	
		$\mathsf{k}_{\mathsf{MAC}} = \mathsf{KDF}_{\mathsf{MAC}}(\mathsf{k}, r_2)$
		$\mathsf{k}_{\mathsf{ENC}} = \mathsf{KDF}_{\mathsf{ENC}}(\mathsf{k}, r_2)$
		if $\mathsf{t} \neq \mathsf{MAC}(\mathsf{k}_{\mathsf{MAC}}, \mathsf{pk}^{\mathsf{e}}_{\mathcal{T}})$
		then abort
return $\mathsf{k}_{\mathsf{ENC}}$		**return** $\mathsf{k}_{\mathsf{ENC}}$

Figure 3.2: Chip Authentication in the original EAC protocol.

To avoid non-repudiation—which is undesirable from a privacy standpoint—$\mathcal{C}$ does not produce a signature to authenticate itself. Instead, chip and terminal

[2]This is usually performed during Passive Authentication but to simplify the presentation, we show it as part of the CA.

establish a shared key k via a DH key exchange using the chip's long-term DH key $\mathsf{pk}_{\mathcal{C}}$ and the terminal's ephemeral DH key $\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}$. Afterwards, the chip generates a nonce r_2 uniformly at random and derives keys $\mathsf{k}_{\mathsf{MAC}}, \mathsf{k}_{\mathsf{ENC}}$ from the shared key k and r_2. Finally, $\mathcal{C}$ sends sends r_2 and a *message authentication code* (MAC) t over $\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}$ under the key $\mathsf{k}_{\mathsf{MAC}}$ to $\mathcal{T}$. If t matches the MAC computed by $\mathcal{T}$, the Chip Authentication concludes successfully. $\mathsf{k}_{\mathsf{ENC}}$ can be used to encrypt any further communication between terminal and chip after the completion of EAC.

Security properties. The main goals of EAC are the authentication of chip and terminal to each other and the establishment of a shared secret key between chip and terminal. Dagdelen and Fischlin [DF11] have proven authenticated key-exchange (AKE) security for EAC in the random oracle model. However, there are also other security properties that the EAC protocol is supposed to provide: Firstly, the protocol should provide *forward secrecy*, meaning that past session keys are protected from being recovered, even if the long-term secrets of chip or terminal are compromised. While the classic EAC version only provides this for the terminal's long-term secret [DF11], we will present versions of PQ-EAC providing forward secrecy for the long-term keys of both chip and terminal in Section 3.4.

Secondly, since the chip represents a travel document, it is desirable for the chip to have plausible deniability of participation in EAC, meaning we do **not** want *non-repudiation*. Since signatures usually provide non-repudiation, in EAC the authenticity of the chip is established through a key-agreement.

3.4 Quantum-Resistant EAC Protocol Versions

This section is concerned with our efforts towards a quantum-resistant version of EAC (called PQ-EAC). EAC is subject to many, sometimes conflicting constraints. For example, to reduce waiting times for access control, it would be beneficial to use round-reduced EAC. However, for privacy reasons, it may not be desirable for the chip to send information about its public key to an unauthenticated terminal. Similarly, forward secrecy helps to protect earlier messages in the case that the private key of the chip is compromised but increases the number of cryptographic operations. An additional consideration is that some versions of PQ-EAC might be more suitable than others for maintaining backward-compatibility with EAC classic.

We distinguish between solutions with the terminal signing the data (SigPQEAC, see Section Section 3.4.1) and an alternative where the terminal uses a long-term KEM for authentication (KemPQEAC, see Section Section 3.4.2). The common approach in both cases is to replace the Diffie-Hellman key agreement step in the

CA by having the terminal send a secret key protected under the chip's long-term KEM key $\mathsf{pk}_{\mathcal{C}}$. The KEM itself is assumed to be *post-quantum secure*.

For each of the two types we can consider a forward-secure variant where the chip, in addition to the long-term KEM key $\mathsf{pk}_{\mathcal{C}}$, also generates an ephemeral KEM key $\mathsf{pk}_{\mathcal{C}}^{e}$ during protocol execution. The terminal then encapsulates another key k^{e} under $\mathsf{pk}_{\mathcal{C}}^{e}$. The long-term key is then used for authentication, especially of the ephemeral part. Both parties derive the session key from either the encapsulated key k (in the non-forward-secure version) resp. together with the encapsulated key k^{e} (in the forward-secure version with the ephemeral KEM). Another option we discuss below is to combine the TA and CA phases and save on the round-trip time by combining some data into single transmissions.

We thus have eight protocol variants via possible combinations: with and without terminal signatures, with and without forward security, and potentially combining messages. The combined versions allow for further simplifications, e.g., by letting the terminal only create a single signature instead of two signatures. We present the two fundamentally different versions, one with signatures for terminal authentication and one with KEMs for authentication, and discuss the other four sub versions for either one within. We discuss further options for the combined versions afterwards. We leave it to the discretion of the implementing bodies to weigh advantages and disadvantages of the various versions.

Assumptions. For all protocols we assume that the following holds:

- Passive Authentication binds the chip's data groups with its key pair $(\mathsf{pk}_{\mathcal{C}}, \mathsf{sk}_{\mathcal{C}})$.
- The communication partners have agreed on $\mathrm{ID}_{\mathcal{C}}$. If PACE is used, it is a suitable hash of the chip's public key from the previous PACE execution; if BAC is used, it is the document number read from the MRZ.
- The protocol is instantiated with a post-quantum or hybrid *IND-CCA-secure* key encapsulation mechanism, a post-quantum or hybrid *EUF-CMA-secure* signature scheme, a key derivation function KDF with output length λ and a dual pseudorandom key combiner KComb. See Section 2.2 for the respective definitions and security experiments of the mentioned primitives.
- The security parameter λ is of appropriate size and all keys have been generated according to λ.

We note that while chip and terminal send the EAC protocol messages in an encrypted channel that was established through BAC or PACE, PQ-EAC would still provide AKE security if all messages were sent in the open.

3.4.1 Authentication with signatures

We start with the four variants in which the terminal uses a quantum-resistant signature scheme to authenticate. The protocol framework is given in Figure 3.3 and describes the plain variant, as well as the forward-secure variant (with the optional ephemeral KEM steps written as [ ]) and the combined version of TA and CA (with the dashed arrows indicating which protocol messages can be combined).

In the protocol we use all relevant exchanged data ($\mathsf{cert}_{\mathcal{T}}, \mathsf{cert}_{\mathcal{C}}, r_2, c, [\mathsf{pk}^e_{\mathcal{C}}, c^e]$) for authentication purposes, including the certificates and the nonce and ciphertexts sent by the terminal. Since these data coincide with the session identifier sid used in the security proof we conveniently write sid in the protocol, too. We note that the final value $\mathsf{k}_{\mathtt{CNF}}$ transmitted by the chip with the last protocol message also serves as an authentication tag for all values in $\mathsf{sid} = (\mathsf{cert}_{\mathcal{T}}, \mathsf{cert}_{\mathcal{C}}, r_2, c, [\mathsf{pk}^e_{\mathcal{C}}, c^e])$. Instead of using a message authentication code we use a derived key under the KDF directly, inserting sid into the key derivation step instead, and taking advantage of the pseudorandomness of the KDF. This method has been suggested for example in [FGSW16] for key confirmation. In principle, one could also use $\mathsf{k}_{\mathtt{CNF}}$ in a message authentication code, applied to some public input.

Terminal Authentication. The terminal $\mathcal{T}$ initiates the TA protocol by sending the chip its certificate $\mathsf{cert}_{\mathcal{T}}$, which contains its public key $\mathsf{pk}_{\mathcal{T}}$, a signature over $\mathsf{pk}_{\mathcal{T}}$ and its Document Verifier certificate. The certificates can be validated by the chip with its CVCA certificate. If the verification of certificates was successful, the chip chooses a nonce r_1 uniformly at random and sends it to $\mathcal{T}$, along with $\mathrm{ID}_{\mathcal{C}}$. Subsequently, $\mathcal{T}$ signs r_1 and $\mathrm{ID}_{\mathcal{C}}$, and sends the signature to $\mathcal{C}$. The TA protocol completes successfully if the chip can verify the signature using $\mathsf{pk}_{\mathcal{T}}$. We note that for the authentication of the terminal only, the Terminal Authentication protocol may be run solely, without the subsequent Chip Authentication. The Chip Authentication requires the (valid) public key $\mathsf{pk}_{\mathcal{T}}$ that was obtained during Terminal Authentication.

Chip Authentication. While certificates are usually checked during Passive Authentication before EAC starts, here we move this part of the process into the CA to simplify the presentation. $\mathcal{C}$ sends its Document Signer certificate, its SO_D and its public key (all of which we will call $\mathsf{cert}_{\mathcal{C}}$) to $\mathcal{T}$. The terminal can then verify the certificate chain using the CSCA root certificate. In addition, $\mathcal{C}$ sends a λ-bit number r_2, chosen uniformly at random.

In order to check that $\mathcal{C}$ is in possession of the secret key $\mathsf{sk}_{\mathcal{C}}$ pertaining to $\mathsf{pk}_{\mathcal{C}}$, the terminal uses $\mathsf{pk}_{\mathcal{C}}$ to encapsulate a shared key k in a ciphertext c. It then sends c to $\mathcal{C}$, along with a signature to prove integrity of the encapsulation. Next, $\mathcal{C}$ verifies the signature and decapsulates c to obtain k. k is then used along with the

session id sid to derive a confirmation key $\mathsf{k}_{\mathrm{CNF}}$ and an encryption key $\mathsf{k}_{\mathtt{KEY}}$. The final value $\mathsf{k}_{\mathrm{CNF}}$ serves as an authentication tag for all values in sid.

Forward secrecy. As is, the plain version of SigPQEAC provides forward secrecy for the case that only the terminal's long term-keys are corrupted. However, an adversary would still be able to decrypt past communication after corrupting the chip. To achieve real forward secrecy, we propose several changes to the CA protocol, written as [ ].

In a nutshell, the chip generates an additional, ephemeral key pair $(\mathsf{pk}^{\mathsf{e}}_{\mathcal{C}}, \mathsf{sk}^{\mathsf{e}}_{\mathcal{C}})$ and sends the public key to $\mathcal{T}$. The terminal then uses $\mathsf{pk}^{\mathsf{e}}_{\mathcal{C}}$ to encapsulate a second key k^{e}, and sends the encapsulation c^{e} back to $\mathcal{C}$, along with the encapsulation c of the first key and a signature over both encapsulations. Both keys, k and k^{e}, serve as input to derive $\mathsf{k}_{\mathrm{CNF}}$ and $\mathsf{k}_{\mathtt{KEY}}$. This way, an attacker that compromises the chip after EAC and subsequent communication has terminated will not be able to recover the ephemeral key k^{e} of that session or other previous sessions, and therefore will not be able to generate the session key $\mathsf{k}_{\mathtt{KEY}}$.

One potential attack that is enabled by sending the ephemeral public key to $\mathcal{T}$, is that an active adversary could exchange $\mathsf{pk}^{\mathsf{e}}_{\mathcal{C}}$ for a different public key and bring himself in possession of k^{e}, which is half of the input required to generate $\mathsf{k}_{\mathtt{KEY}}$. However, since both keys $\mathsf{k}, \mathsf{k}^{\mathsf{e}}$ are combined using a dual pseudorandom key combiner, the adversary still only has negligible chance of guessing the correct session key.

Round-reduced combined version. At the cost of sacrificing some privacy it is easy to improve efficiency of the protocol described above. In particular, we can combine two pairs of messages into one message each: Instead of exchanging a challenge and a signature during TA as well as during CA, the terminal can encapsulate a key, sign a concatenation of the session id sid and the encapsulation and send the signature and the encapsulation to the chip in one go. The resulting combined variant of PQ-EAC is shown by the dashed arrows indicating which protocol messages can be combined.

As shown in Section 3.6.1, saving two messages and the signature verification improves performance significantly. On the flip side, in the SigPQEAC variant, the combined version forces the chip to send its public key before the terminal has been authenticated. In the KemPQEAC version, the chip only sends the encrypted $\mathsf{cert}_{\mathcal{C}}$ and will hence remain anonymous to an unauthenticated terminal. Even though the public key is usually considered public information, it is desirable to avoid sending it to unauthorized terminals to protect the eMRTD holder's privacy as much as possible. This could be important in contexts where an execution of PACE does not already authorize the terminal to read data.

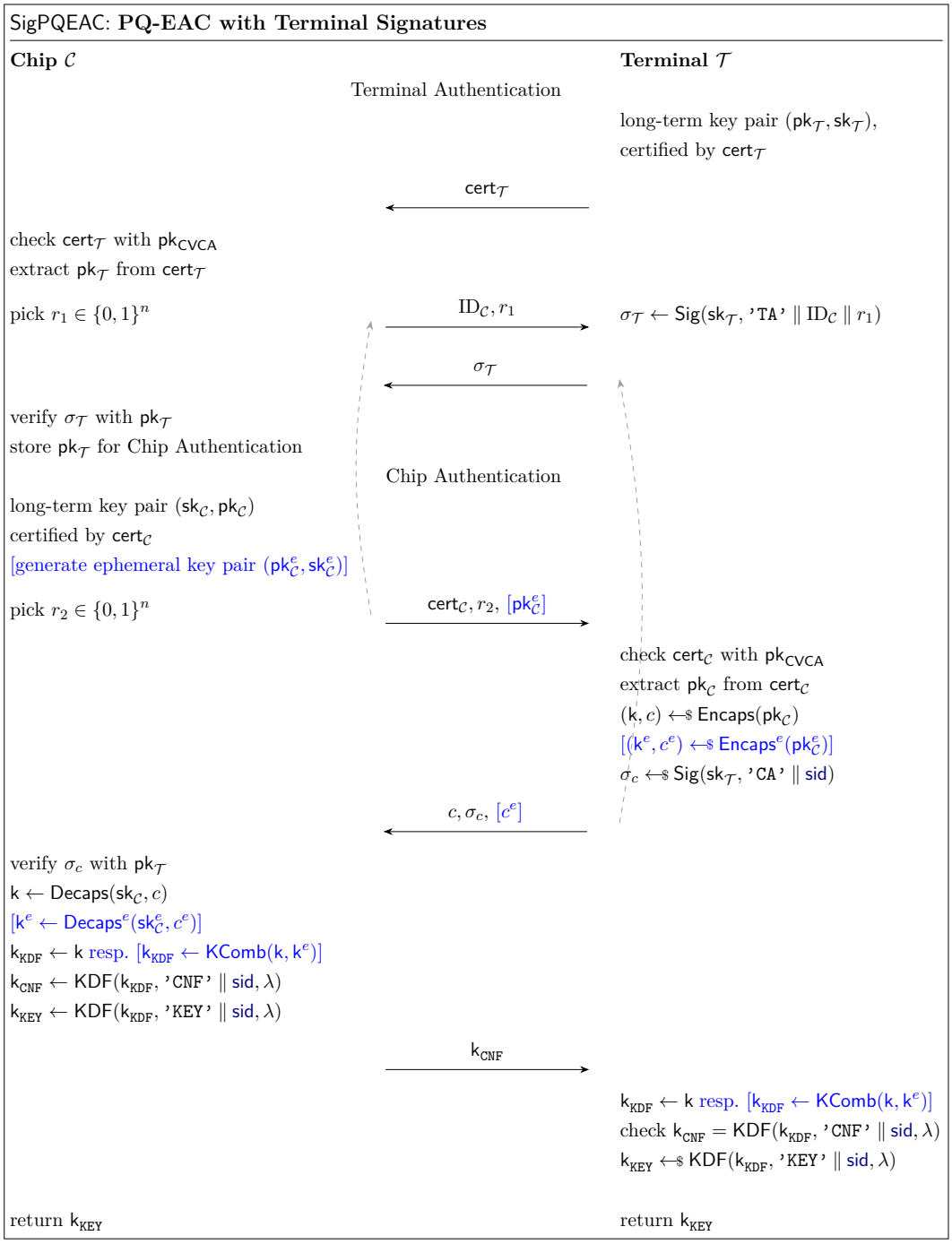

Figure 3.3: PQ-EAC with terminal signatures, divided into TA and CA phase. Optional steps for achieving forward security are given in blue and brackets [ ]. In the combined version we can save a full round trip by combining protocol messages as indicated by the arrows. We note that each party sets the partner identity pid to be the distinguished name in the received certificate. The session identifier consists of all sent data, except for the authentication parts,
$\mathsf{sid} = (\mathsf{cert}_\mathcal{T}, \mathsf{cert}_\mathcal{C}, r_2, c, [\mathsf{pk}_\mathcal{C}^e, c^e])$.

3.4.2 Authentication via long-term KEMs

Next we present in Figure 3.4 the four variants in which the terminal avoids signatures, which are usually expensive, but uses a long-term key of a KEM to authenticate instead. To this end, the chip sends a key $\mathsf{k}_{\mathtt{TA}}$ encapsulated under the terminal's long-term key. While the chip has to perform an additional key encapsulation to do so, this is less expensive than the signature verification it would have to perform normally. The key is used to derive a confirmation value $\mathsf{k}_{\mathtt{TCNF}}$, replacing the signature in terminal authentication, and a MAC key $\mathsf{k}_{\mathtt{TMAC}}$ used instead of the second signature. A noteworthy feature of these versions is that now the certificate $\mathsf{cert}_{\mathcal{C}}$ can be sent encrypted in the CA step to hide the long-term identity of the card. The key $\mathsf{k}_{\mathtt{TENC}}$ for this encryption is also generated in the TA phase with the help of the terminal's long-term KEM.

3.5 Security Proofs

Below, we provide a security proof for PQ-EAC in the real-or-random security model of Bellare and Rogaway. We start with a short overview of the proof: The security properties to show are session matching and key secrecy. Session matching covers fundamental properties such as partnered sessions deriving the same key, pairwise uniqueness of partners, and correct identification of roles and partners. For all protocol versions (SigPQEACand KemPQEAC, with or without forward secrecy, with or without round reduction) this property follows by protocol construction and the uniqueness of nonces. We note that all proofs take into account potential decryption errors of KEMs. Key secrecy ensures that only the intended partner shares the derived session key and that session keys are indistinguishable from random to the adversary, unless the adversary trivially knows the key. The proof for key secrecy for the variants of SigPQEAC is via game-hopping. Here, we only outline the main steps. We first exclude attacks in which a party accepts a different public key pk as identified in the issued certificate cert. By the unforgeability of the certificate scheme this cannot occur, except with negligible probability. But it follows that, if the chip accepts only the certified public key of the terminal, then the unforgeability of the terminal's signature scheme ensures that only the honest terminal is able to create the valid signature σ_{c} over the ciphertext c (and the ephemeral values $\mathsf{pk}^{\mathsf{e}}_{\mathcal{T}}$ and c^{e} in the forward-secure version). We can conclude that the encapsulated key k (resp. keys k and k^{e}) must have been created by an honest terminal, such that the IND-CCA security of the chip's long-term KEM guarantees that they are hidden from the adversary. The security of the key derivation function (together with the key combiner in the forward-secure version) ensure that the session key is indistinguishable from random for the adversary.

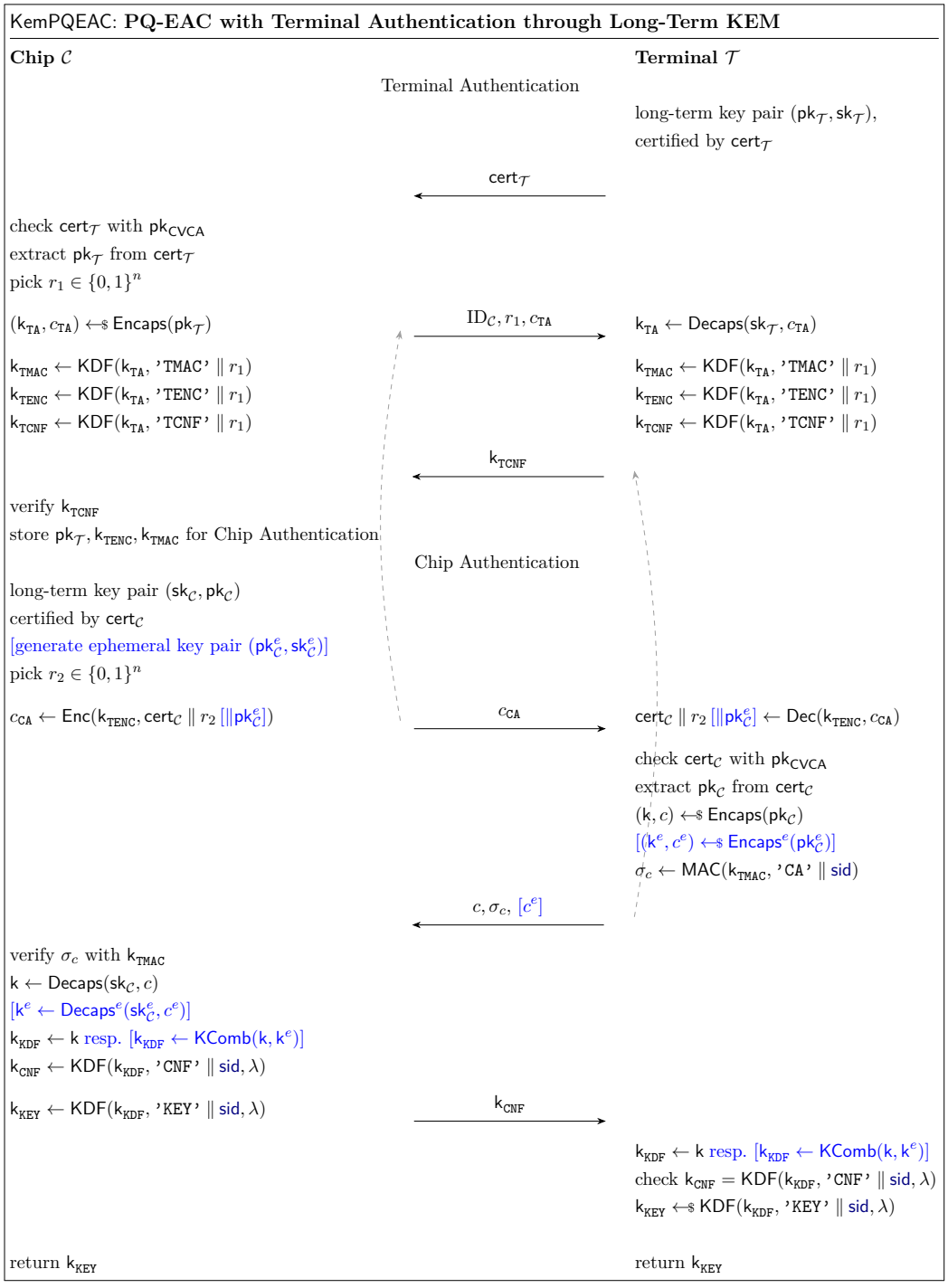

Figure 3.4: PQ-EAC with terminal authentication through long-term KEMs, divided into TA and CA phase. Optional steps for achieving forward security are given in blue and brackets [ ]. In the combined version we can save a full round trip by combining protocol messages as indicated by the arrows.

We note that in the analysis the final confirmation value sent by the chip is only required for the forward-secure case. Here the adversary may impersonate a chip and later learn the long-term secret key of the KEM. But the adversary would have to create a valid confirmation value *before* it later corrupts the chip's long-term secret. This once more is infeasible by the security of the chip's KEM.

The proof of the KEM-based variant KemPQEAC is very similar to the signature-based protocol. Only here we need to argue that only the honest terminal can create the valid MAC σ_{c} over the ciphertext(s). This can be shown via two extra steps in which we argue that only the terminal can decapsulate $\mathsf{k}_{\mathtt{TA}}$ in terminal authentication (by the IND-CCA security of the terminal's long-term KEM) and that the derived keys from $\mathsf{k}_{\mathtt{TA}}$ are random by the security of the key derivation function. The other steps are as in the proof of SigPQEAC.

We finally remark that our proofs show post-quantum security of the protocols. The proof steps consist of (straightline) reductions to the involved standard primitives like signature schemes, KEMs, or key derivation functions, and do not require idealized primitives such as random oracles. Hence, any successful quantum adversary against the key exchange protocol would thus yield a successful quantum adversary against the underlying primitive. Assuming that the primitives are all quantum-resistant, it follows that the overall protocol also withstands such attacks.

We prove the security of the two variants SigPQEAC (with signatures) and KemPQEAC (with KEM-based authentication). We start with the signature-based variant, and simultaneously consider the forward-secure and non-forward-secure version in the proof. We discuss afterwards that the proof also holds for the combined version. The proof of the KEM-based version then follows straightforwardly from the signature case, as we discuss afterwards.

3.5.1 Key exchange security model

In this subsection we define the security model which we will use in the security proof below.

Setup. We assume that there is a set $\mathcal{P}$ of parties. At the beginning, each party $P \in \mathcal{P}$ is assigned a long-term key pair $(\mathsf{sk}_P, \mathsf{pk}_P)$, certified via cert_P under a public key $\mathsf{pk}_{\mathsf{CVCA}}$ of a country verifying certificate authority. We assume that all parties know this public key of the certificate authority. For the attack we assume, to the advantage of the adversary, that all public keys pk_P and certificates cert_P are already known by the adversary. We also presume that each certificate carries the identifier from $\mathcal{P}$ or, to be precise, since we do not make any stipulation on the form of the identifiers in $\mathcal{P}$, we rather set the identifiers according to the distinguished name in the certificate.

It is sometimes convenient to further divide the set of parties $\mathcal{P}$ disjointly into chips and terminals, $\mathcal{P} = \mathcal{C} \cup \mathcal{T}$. We say that a party P is a chip if $P \in \mathcal{C}$, and a terminal of $P \in \mathcal{T}$. Note that the two protocol versions, with signatures and using KEMs instead, differentiate between the type of the public keys for terminals: In the former case it is a certified signing key, in the latter case it is a certified KEM key. Chips use KEM keys in both versions. For the abstract security model, however, we do not distinguish the type of public key.

Sessions. The adversary can now interact with the protocol in different sessions, representing a local protocol execution of a party. This is done via oracles to initiate new sessions (INIT), to send protocol messages to a session (SEND), to reveal a session key (REVEAL), to test a session (TEST), and to corrupt the long-term key of a party (CORRUPT). Besides the protocol state the session can be described by the following entries:

- lbl is a unique administrative identifier lbl, chosen by the game during the execution.
- owner denotes the identity (from $\mathcal{P}$) of the owner of the session.
- role $\in \{\text{chip}, \text{terminal}\}$ describes the role of the party owner.
- pid specifies the identity of the intended communication partner from $\mathcal{P}$.
- st $\in \{\text{running}, \text{accepted}, \text{rejected}\}$ denotes the current state of the session.
- sid denotes the protocol's session identifier, initialized to $\perp$ and set only once upon acceptance.
- revealed $\in \{\text{true}, \text{false}\}$ indicates if the adversary has revealed the session key.
- tested $\in \{\text{true}, \text{false}\}$ indicates if the adversary has tested the session.
- key denotes the session key (or $\perp$).

It is convenient to use the unique identifier lbl to refer to individual entries, i.e., by writing lbl.owner, lbl.role, lbl.pid etc.

Attacks. As mentioned earlier we prove security of the protocols in the real-or-random model of Bellare Rogaway [BR93]. This means that a secret challenge bit $b \leftarrow \{0, 1\}$ is chosen randomly at the outset, and the adversary either receives the actual session key of a tested session, or an independently sampled random string of the same length, the choice depending on the bit b. The task of the adversary is to

predict b, ruling out trivial attacks like cases where the adversary knows the session key of a communication partner. To capture forward security we also assume a set $\mathcal{F} \subseteq \mathcal{P}$ describing the parties resp. roles which provide forward security. In our setting, if the chip picks an ephemeral KEM key, then $\mathcal{F} = \mathcal{C} \cup \mathcal{T}$ covers all parties, else we have forward secrecy only for the terminals, $\mathcal{F} = \mathcal{T}$.

The attacker initially receives all certificates cert_P including the public keys pk_P of all parties $P \in \mathcal{P}$. We will use the initially empty set $\mathfrak{C}$ to store the identities of all corrupt parties. During the attack, the adversary may issue the following oracle queries (where we cover both forward and non-forward security in a single definition):

Initialization: The $\textsc{Init}(P, r)$ command first checks that party $P \in \mathcal{P} \setminus \mathfrak{C}$ is not corrupt, and that $P \in \mathcal{C}$ belongs to the chips for role $r =$ chip resp. $P \in \mathcal{T}$ for $r =$ terminal. If all properties hold then it picks a new label lbl and initializes a new session for this label for owner $\mathsf{owner} \leftarrow P$ with role $\mathsf{role} \leftarrow r$. It sets $\mathsf{st} \leftarrow$ running, $\mathsf{revealed} \leftarrow \mathsf{false}$, $\mathsf{tested} \leftarrow \mathsf{true}$, $\mathsf{sid} \leftarrow \perp$, and $\mathsf{key} \leftarrow \perp$.

Sending protocol messages: Upon calling $\textsc{Send}(\mathsf{lbl}, m)$ for an initialized session with identifier lbl, check that the session owner $\mathsf{lbl.owner} \notin \mathfrak{C}$ has not been corrupted yet. If so, then deliver the protocol message m to the party and hand the possible response back to the adversary. This may affect the status of the session and switch it to rejected or accepted (which we assume is known by the adversary). If it turns to accepted then the key key and the session identifier sid are set.

Upon acceptance, if there exists a partnered session $\mathsf{lbl}' \neq \mathsf{lbl}$ with $\mathsf{lbl.sid} = \mathsf{lbl}'.\mathsf{sid}$ as well as $\mathsf{lbl}'.\mathsf{revealed} = \mathsf{true}$, or if $\mathsf{lbl.pid} \in \mathfrak{C}$, then set $\mathsf{lbl.revealed} \leftarrow \mathsf{true}$; else set $\mathsf{lbl.revealed} \leftarrow \mathsf{false}$. If there exists a partnered session $\mathsf{lbl}' \neq \mathsf{lbl}$ with $\mathsf{lbl.sid} = \mathsf{lbl}'.\mathsf{sid}$ and $\mathsf{lbl}'.\mathsf{tested} = \mathsf{true}$, then set $\mathsf{lbl.tested} \leftarrow \mathsf{true}$; else set $\mathsf{lbl.tested} \leftarrow \mathsf{false}$. This means that the session here inherits the $\mathsf{revealed}$ and tested flag from a partnered session resp. is considered to be revealed if the intended partner is already corrupt upon completion of the session here.

Reveal session key: Upon $\textsc{Reveal}(\mathsf{lbl})$ check that $\mathsf{lbl.st} =$ accepted. If not, return $\perp$. Else, return $\mathsf{lbl.key}$ and set $\mathsf{lbl.revealed} \leftarrow \mathsf{true}$ and also set $\mathsf{lbl}'.\mathsf{revealed} \leftarrow \mathsf{true}$ for any partnered session $\mathsf{lbl}' \neq \mathsf{lbl}$ with $\mathsf{lbl}'.\mathsf{sid} = \mathsf{lbl.sid}$.

Test session: For $\textsc{Test}(\mathsf{lbl})$ check that $\mathsf{lbl.st} =$ accepted and that $\mathsf{lbl.tested} = \mathsf{false}$. If not, return $\perp$. Otherwise, if $b = 0$ then return $\mathsf{lbl.key}$, and if $b = 1$ then return a random value $\mathsf{k} \leftarrow \{0,1\}^{|\mathsf{lbl.key}|}$ of the same length. Set $\mathsf{lbl.tested} \leftarrow \mathsf{true}$ and also $\mathsf{lbl}'.\mathsf{tested} \leftarrow \mathsf{true}$ for any other partnered session $\mathsf{lbl}' \neq \mathsf{lbl}$ with $\mathsf{lbl}'.\mathsf{sid} = \mathsf{lbl.sid}$. The latter spares us from keeping track of consistency, when the adversary tests both partners.

Corruption of long-term secrets: When calling CORRUPT(P) for $P \in \mathcal{P} \setminus \mathfrak{C}$, return sk_P to the adversary and add P to $\mathfrak{C} \leftarrow \mathfrak{C} \cup \{P\}$. Note that any session for this party cannot be stepped anymore via SEND queries.

In the non-forward-secure setting, when $P \notin \mathcal{F}$, mark all sessions lbl with $\mathsf{lbl.owner} = P$ or $\mathsf{lbl.pid} = P$ as revealed: $\mathsf{lbl.revealed} \leftarrow \mathsf{true}$. This means that subsequent corruption of the long-term secret endangers the secrecy of the session key (on either side).

We assume that the adversary eventually outputs a guess b^* for b. We write $\mathbf{Exp}^{\mathrm{ke}}_{\Pi,\mathcal{A},\mathcal{C},\mathcal{T},\mathcal{F}}(\lambda) = 1$ if $b^* = b$ in the above experiment for adversary $\mathcal{A}$.

3.5.2 Security requirements

With the above attack model we can now state the security requirements. We note that we also require functional correctness in the sense that if two parties engage in an interaction and the adversary does not interfere with the executions, then the two parties obtain the same session identifier sid. This can be easily seen to hold for the protocols. Session matching now enforces that, even in presence of the adversary, two partnered sessions also derive the same key and correctly identify the intended partner and its role:

Definition 9 (Session Matching)**.** *A key exchange protocol* Π *(for party sets* $\mathcal{P} = \mathcal{C} \cup \mathcal{T}$ *and* $\mathcal{F}$*) provides session matching if for any efficient adversary* $\mathcal{A}$ *the following holds:*

Matching Keys: *For any distinct completed partnered sessions* $\mathsf{lbl}, \mathsf{lbl}'$ *with* $\mathsf{lbl.sid} = \mathsf{lbl}'\mathsf{.sid} \neq \perp$ *we have* $\mathsf{lbl.key} = \mathsf{lbl}'\mathsf{.key}$*.*

Exclusive Session Identifiers: *There do not exist three distinct sessions* lbl*,* lbl'*,* lbl'' *with* $\mathsf{lbl.sid} = \mathsf{lbl}'\mathsf{.sid} = \mathsf{lbl}''\mathsf{.sid} \neq \perp$*.*

Matching Roles: *For any distinct partnered sessions* $\mathsf{lbl} \neq \mathsf{lbl}'$ *with* $\mathsf{lbl.sid} = \mathsf{lbl}'\mathsf{.sid} \neq \perp$ *we have* $\{\mathsf{lbl.role}, \mathsf{lbl}'\mathsf{.role}\} = \{\mathit{chip}, \mathit{terminal}\}$*.*

Authentication: *For any distinct partnered sessions* $\mathsf{lbl} \neq \mathsf{lbl}'$ *with* $\mathsf{lbl.sid} = \mathsf{lbl}'\mathsf{.sid} \neq \perp$ *we have* $\mathsf{lbl.pid} = \mathsf{lbl}'\mathsf{.owner}$*.*

We write $\boldsymbol{Exp}^{match}_{\Pi,\mathcal{A},\mathcal{C},\mathcal{T},\mathcal{F}}(\lambda) = 1$ *if the adversary in the attack described above violates any of the four properties.*

Key secrecy now demands that the adversary cannot predict the secret challenge bit b better than by guessing, guaranteeing that session keys look random. This requires to exclude some trivial attacks, e.g., where a party interacts with an already corrupt partner or where the session key (of the party or of the partner) has also been revealed:

Definition 10 (Key Secrecy)**.** *A key exchange protocol* Π *(for party sets* $\mathcal{P} = \mathcal{C} \cup \mathcal{T}$ *and* $\mathcal{F}$*) provides key secrecy if for any efficient adversary* $\mathcal{A}$ *mounting the attack described above, the probability that*

Correct Prediction: $b^* = b$*, and*

Non-trivial attack: *There do not exist (not necessarily distinct) completed sessions* $\mathsf{lbl}, \mathsf{lbl}'$ *with* $\mathsf{lbl}.\mathsf{sid} = \mathsf{lbl}'.\mathsf{sid} \neq \perp$ *with* $\mathsf{lbl}.\mathsf{tested} = \mathsf{lbl}'.\mathsf{revealed} = \mathsf{true}$*.*

is negligibly close to $\frac{1}{2}$*. More concretely we write* $\boldsymbol{Adv}^{ke}_{\Pi,\mathcal{A},\mathcal{C},\mathcal{T},\mathcal{F}}(\lambda)$ *for the probability minus* $\frac{1}{2}$ *that the two properties hold.*

3.5.3 Security proofs for SigPQEAC

We first show the session matching property for SigPQEAC. We note once more that we give the proof of the forward-secure and non-forward-secure version within a single proof:

Proposition 1 (Session Matching of SigPQEAC)**.** *The protocol SigPQEAC provides session matching. Specifically, for any adversary* $\mathcal{A}$ *initiating at most* S *sessions we have*

$$\begin{aligned}&\Pr\Big[\boldsymbol{Exp}^{match}_{\Pi,\mathcal{A},\mathcal{C},\mathcal{T},\mathcal{F}}(\lambda) = 1\Big]\\ &\quad\leq\ 2S^2 \cdot \Big(2^{-|r_2|} + 2^{-|\mathsf{k}|} + \Pr\Big[\boldsymbol{Exp}^{decErr}_{\mathsf{KEM}}(\lambda) = 1\Big] + \Pr\Big[\boldsymbol{Exp}^{decErr}_{\mathsf{KEM}^e}(\lambda) = 1\Big]\Big).\end{aligned}$$

We note that the decryption error term $\Pr\Big[\mathbf{Exp}^{\mathrm{decErr}}_{\mathsf{KEM}^e}(\lambda) = 1\Big]$ is only required in the forward-secure version where the parties use an ephemeral KEM. In the non-forward-secure version this term disappears.

Proof. Recall that we have to show four properties, namely, that (1) identical session identifiers imply identical keys, (2) session identifiers are unique between two (partnered) session, (3) the roles match, and (4) for any distinct partnered sessions they point to the identity of the other party. Session identifiers are given as $\mathsf{sid} = (\mathsf{cert}_{\mathcal{T}}, \mathsf{cert}_{\mathcal{C}}, c, r_2, [\mathsf{pk}^e_{\mathcal{C}}, c^e])$, and the partner identities are the distinguished names appearing in the certificates.

As for (1), the final session key is derived as $\mathsf{KDF}(\mathsf{k}_{\mathtt{KDF}}, \texttt{'KEY'} \,\|\, \mathsf{sid})$, where $\mathsf{k}_{\mathtt{KDF}}$ is the encapsulated key in ciphertext c under long-term key $\mathsf{pk}_{\mathcal{C}}$ or, if present, mixed with the encapsulated key k^e in ciphertext c^e under the ephemeral key $\mathsf{pk}^e_{\mathcal{C}}$. Since all values $c, \mathsf{pk}_{\mathcal{C}}, c^e, \mathsf{pk}^e_{\mathcal{C}}$ and r_2 appear in the session identifier, a key mismatch for identical session identifiers can only happen in case of a decryption error for the encapsulations chosen by an honest terminal. The probability for this to happen for either of the two keys in any session is at most $S \cdot (\Pr\Big[\mathbf{Exp}^{\mathrm{decErr}}_{\mathsf{KEM}}(\lambda) = 1\Big] +$

$\Pr\left[\mathbf{Exp}^{\text{decErr}}_{\mathsf{KEM}^e}(\lambda) = 1\right]$). In the proposition's claim we subsume this under the factor S^2 for property (2).

For (2) note that the client contributes random string r_2 to the session identifiers, and the terminal sends a ciphertext c containing a random key k. Hence, the probability of having a session identifier for a chip-chip combination is at most $2^{-|r_2|}$, and for a terminal-terminal combination at most $2^{-|\mathsf{k}|} + \Pr\left[\mathbf{Exp}^{\text{decErr}}_{\mathsf{KEM}}(\lambda) = 1\right]$, taking into account the possibility of creating the same ciphertext for different keys, resulting in a decryption error. Multiplying this by the at most S^2 number of session pairs, still yields a negligible bound. For (3) we assume that a chip or terminal only starts a communication in its indented role, such that the first certificate $\mathsf{cert}_{\mathcal{T}}$ specifies the terminal, and the second one $\mathsf{cert}_{\mathcal{C}}$ determines the chip. Hence, (4) also follows, because a match on session identifiers means that distinct partnered sessions use the same certificates, pointing to each other mutually. □

Theorem 1 (Key Secrecy of SigPQEAC)**.** *Protocol SigPQEAC provides key secrecy. Specifically, for any adversary $\mathcal{A}$ initiating at most S sessions with U users there exist adversaries $\mathcal{B}_{\mathsf{cert}}, \mathcal{B}_{\mathsf{Sig}}, \mathcal{C}_{\mathsf{KEM}}, \mathcal{C}_{\mathsf{KEM}^e}, \mathcal{C}_{\mathsf{KDF}}, \mathcal{C}_{\mathsf{KComb}}$ such that*

$$
\begin{aligned}
&\boldsymbol{Adv}^{ke}_{\Pi,\mathcal{A},\mathcal{C},\mathcal{T},\mathcal{F}}(\lambda) \\
&\quad\leq\ 3S^3 \cdot \left(2^{-|r_2|} + 2^{-|\mathsf{k}|} + \Pr\left[\boldsymbol{Exp}^{decErr}_{\mathsf{KEM}}(\lambda) = 1\right]\right) \\
&\qquad + U \cdot \left(\Pr\left[\boldsymbol{Exp}^{cert\text{-}unf}_{\mathcal{C},\mathcal{B}_{\mathsf{cert}}}(\lambda) = 1\right] + \Pr\left[\boldsymbol{Exp}^{EUF\text{-}CMA}_{\mathcal{S},\mathcal{B}_{\mathsf{Sig}}}(\lambda) = 1\right]\right) \\
&\qquad + S^2 \cdot \Big(4U \cdot \boldsymbol{Adv}^{IND\text{-}CCA}_{\mathsf{KEM},\mathcal{C}_{\mathsf{KEM}}}(\lambda) + 2 \cdot \boldsymbol{Adv}^{IND\text{-}CPA}_{\mathsf{KEM}^e,\mathcal{C}_{\mathsf{KEM}^e}}(\lambda) \\
&\qquad\qquad + 4 \cdot \boldsymbol{Adv}^{PRF}_{\mathsf{KDF},\mathcal{C}_{\mathsf{KDF}},\mathcal{D}_{KDF}}(\lambda) + 4 \cdot \boldsymbol{Adv}^{dPRF}_{\mathsf{KComb},\mathcal{C}_{\mathsf{KComb}}}(\lambda) + 2^{-|\mathsf{k}_{CNF}|}\Big)
\end{aligned}
$$

Here, the sets for forward security are given as $\mathcal{F} = \mathcal{P} = \mathcal{C} \cup \mathcal{T}$ if the ephemeral KEM^e is used, and $\mathcal{F} = \mathcal{T}$ otherwise. The algorithms $\mathcal{B}_{\mathsf{cert}}, \mathcal{B}_{\mathsf{Sig}}, \mathcal{C}_{\mathsf{KEM}}, \mathcal{C}_{\mathsf{KEM}^e}, \mathcal{C}_{\mathsf{KComb}}$, $\mathcal{C}_{\mathsf{KDF}}$ run in approximately equal time as $\mathcal{A}$, and $\mathcal{D}_{KDF}$ is the uniform distribution on bit strings of length equal to $|\mathsf{k}_{KDF}|$.

Proof. Since the proofs for both versions are almost identical we give them simultaneously, explaining the slight adaptions when using the ephemeral data. We also note that the proof for the combined version where we slot together the chip's and terminal's messages is also identical, such that we do not mention this explicitly here. The proof itself follows by game hopping, starting with $\mathbf{Game}_0$, the original attack of adversary $\mathcal{A}$. In each game hop we make slight changes to the games, gradually taking away attack possibilities. Formally, we will declare the adversary to lose and we stop the game immediately if one of these attack possibilities is triggered. We compensate for this by adding the (usually small) probability of such an attack to happen to the overall advantage of adversary $\mathcal{A}$. In the final game, $\mathcal{A}$'s advantage in predicting the secret bit b is exactly $\frac{1}{2}$.

We denote by $\Pr[\mathbf{Game}_i]$ the probability that $\mathcal{A}$ wins, i.e., correctly predicts the bit b in the i-th game and satisfies the freshness condition, minus the pure guessing probability $\frac{1}{2}$. In particular, $\Pr[\mathbf{Game}_0] = \Pr\left[\mathbf{Exp}^{\mathrm{ke}}_{\Pi,\mathcal{A},\mathcal{C},\mathcal{T},\mathcal{F}}(\lambda) = 1\right] - \frac{1}{2}$. We also say that a party (resp. a key) is honest (at a certain point in time during the attack) if this party (resp. the party holding the key) has not been added to $\mathfrak{C}$ through a CORRUPT request at this point; else it is called malicious or corrupt.

Game$_0$ The original attack of $\mathcal{A}$.

Game$_1$ In this game we declare the adversary to lose if there are two (or more) sessions of honest chips resulting in the same value r_2.

Since the values r_2 are chosen uniformly, the probability of a collision in two of the at most S sessions, is given by $S^2 \cdot 2^{-|r_2|}$ by the birthday bound. Hence,

$$\Pr[\mathbf{Game}_0] \leq \Pr[\mathbf{Game}_1] + S^2 \cdot 2^{-|r_2|}.$$

Game$_2$ In this game we now declare the adversary to lose if there are two (or more) sessions of the honest terminals sending the same ciphertext values c for the same chip identity $\mathsf{cert}_{\mathcal{C}}$.

Note that any such collision would mean that, either the encapsulated keys k must be equal, which happens with probability at most $S^2 \cdot 2^{-|\mathsf{k}|}$ across all sessions, or the keys are distinct but map to the same ciphertext c. The latter, however, would mean that decryption cannot be correct, such that we have a bound of $S^2 \cdot \Pr\left[\mathbf{Exp}^{\mathrm{decErr}}_{\mathsf{KEM}}(\lambda) = 1\right]$ for this case. Therefore,

$$\Pr[\mathbf{Game}_1] \leq \Pr[\mathbf{Game}_2] + S^2 \cdot (2^{-|\mathsf{k}|} + \Pr\left[\mathbf{Exp}^{\mathrm{decErr}}_{\mathsf{KEM}}(\lambda) = 1\right]).$$

Game$_3$ Compared to **Game**$_2$ declare the adversary to lose if an honest party accepts a certificate cert_P of some honest party (with identity P) but for a public key different than the generated one pk_P.

Note that this can happen on the chip side, during Terminal Authentication, as well as on the terminal's side during Chip Authentication. Either way it gives in a straightforward way a reduction $\mathcal{B}_{\mathsf{cert}}$ to the (un)forgeability of certificates: Initially guess (with probability $1/|\mathcal{P}|$) for which party P this will happen. Create all public keys of users, and certify pk_P for the certificate scheme for verification key $\mathsf{pk}_{\mathsf{CVCA}}$. Run $\mathcal{A}$'s attack. If $\mathcal{A}$ eventually sends a valid certificate cert^*_P for $\mathsf{pk}^*_P \neq \mathsf{pk}_P$, then output this certificate with pk^*_P as the certificate forgery.

Since we guess the right party with probability $1/U$ for $U = |\mathcal{P}|$, and the simulation of $\mathcal{A}$'s attack is perfect, we can bound the advantage in the previous

game by the term for certificate unforgeability:

$$\Pr[\mathbf{Game}_2] \leq \Pr[\mathbf{Game}_3] + U \cdot \Pr\Big[\mathbf{Exp}^{\text{cert-unf}}_{\mathcal{C},\mathcal{B}_{\mathsf{cert}}}(\lambda) = 1\Big].$$

At this point any (completed) session of an honest party contains a unique identifying value (r_2 on the chip's side and c on the terminal's side). Furthermore, looking at the opposite received value in that session (c for a chip resp. r_2 for a terminal), it follows that there can exist at most one honest partner session which has sent that value. We can thus, via a hybrid argument, assume that the adversary $\mathcal{A}$ only makes a single TEST oracle call, to a completed honest session, and that we can predict the number i of the session (according to initialization calls) upfront. This comes at a factor S in the security loss.

Game$_4$ We change **Game**$_4$ by letting the adversary lose if an honest chip obtains a valid signature $\sigma_{\mathcal{C}}$ of an honest terminal $\mathcal{T}$ for a message sid which has not been signed by the terminal during chip authentication at this point yet.

Note that, by the previous game hops, the signature can only be for the actual certified public key $\mathsf{pk}_{\mathcal{T}}$ of the terminal. Furthermore, since terminals use different prefixes `'TA'` resp. `'CA'` when signing in the two phases, the terminal cannot have signed these data in terminal authentication either.[3] It follows that such a valid signature constitutes a forgery for the signature scheme $\mathcal{S}$. The formal reduction $\mathcal{B}_{\mathsf{Sig}}$ receives a public key pk of the signature scheme, to guess the terminal's identity upfront with probability $1/|\mathcal{P}|$, and to run the attack in **Game**$_3$, with injecting the given signature key pk as $\mathsf{pk}_{\mathcal{T}}$ for the corresponding terminal. All other steps of the attacks are carried out locally, except for signature requests for our challenge terminal (where our reduction calls the signing oracle). If the reduction eventually encounters a valid signature for previously unsigned data, it outputs this as a valid forgery under $\mathsf{pk} = \mathsf{pk}_{\mathcal{T}}$.

Since the reduction provides a perfect view of $\mathcal{A}$'s attack in **Game**$_3$ up to the point of forgery for a correct guess of the party, we thus have:

$$\Pr[\mathbf{Game}_3] \leq \Pr[\mathbf{Game}_4] + U \cdot \Pr\Big[\mathbf{Exp}^{\text{EUF-CMA}}_{\mathcal{S},\mathcal{B}_{\mathsf{Sig}}}(\lambda) = 1\Big].$$

Note that, since we have at most one honest matching partner according to games **Game**$_1$ and **Game**$_2$, then receiving a valid signature on the chip's side implies that it must come for the partnered terminal session—or the terminal is already corrupt at this point, in which case the chip's session cannot be successfully tested anymore.

For the next game hop we remark that with another factor S we can account for the prediction of the (at most) single honest partner session j, where we assume

[3] This would in particular be also true if we combine the messages and let the terminal only sign once over all values. See Remark Remark 3 after the proof.

$i = j$ if there does not exist such a session. If any of the predictions turns out to be false, then we immediately output a random guess for the secret challenge bit b and stop. We can view this as a game modification, resulting in $\mathbf{Game}_5$:

$\mathbf{Game}_5$ Let the adversary only make a single TEST query and output the session number i and its potential partner session number j at the outset.

With the above discussion it follows that

$$\Pr[\mathbf{Game}_4] \leq S^2 \cdot \Pr[\mathbf{Game}_5].$$

We next branch according to a non-forward secure or forward-secure execution for the (only) test session.

Non-forward secure case. Assume that the i-th session, the one which eventually gets tested, is run in non-forward secure mode. In particular, the session does not involve the ephemeral keys.

$\mathbf{Game}_{6a}$ Consider the ciphertext c for key k sent or received in the i-th session, encrypted under the long-term key $\mathsf{pk}_{\mathcal{C}}$ of a chip with certificate $\mathsf{cert}_{\mathcal{C}}$. Let $\bar{\mathsf{k}} \leftarrow \{0,1\}^{|\mathsf{k}|}$ be a random and independent key of the same length. In all subsequently processed honest sessions of the same chip with $\mathsf{cert}_{\mathcal{C}}$ in which this chip receives c, as well as in the i-th session and in the j-th session, use $\bar{\mathsf{k}}$ instead of k for the internal computations. Note that we do not need to take care of other honest terminal sessions using the same pair $(c, \mathsf{cert}_{\mathcal{C}})$, since there exists none according to $\mathbf{Game}_2$.

Note that if the test session i is a terminal session, then the partner chip session j cannot be for a corrupt chip identity $\mathcal{C}$; else $\mathcal{A}$'s attack would be declared unsuccessful. This, together with the fact that only certified public keys are accepted by the terminal according to $\mathbf{Game}_3$, implies that the public key $\mathsf{pk}_{\mathcal{C}}$ used to generate the ciphertext c must not be compromised. Hence, because the ciphertext c was created by the honest terminal itself, we have a pair $(\mathsf{pk}_{\mathcal{C}}, c)$ generated resp. held by honest parties only.

Analogously, if the test session i is a chip session, then expected partner terminal with certificate $\mathsf{cert}_{\mathcal{T}}$ must be honest at this point, or else the attack is deemed as trivial. But according to $\mathbf{Game}_5$ it must then be the case that the incoming message with the ciphertext c for $\mathsf{pk}_{\mathcal{C}}$ has been signed (and thus created) exactly by the honest partner terminal with $\mathsf{cert}_{\mathcal{T}}$. Once more, it follows that the pair $(\mathsf{pk}_{\mathcal{C}}, c)$ is faithfully generated.

We first note that the probability that the chip with $\mathsf{cert}_{\mathcal{C}}$ "accidentally" receives the ciphertext c before the i-th resp. j-th session is bounded by $S \cdot (2^{-|\mathsf{k}|} + \Pr\left[\mathbf{Exp}^{\text{decErr}}_{keyEM}(\lambda) = 1\right])$. Observe that $\mathbf{Game}_2$ only covers the case of ciphertexts

generated by honest terminals, but now c may have also been created by a malicious terminal. But it also holds here that either the freshly picked key in our ciphertext c equals the previously decapsulated key, or that a decryption error under the honest public key $\mathsf{pk}_{\mathcal{C}}$ occurs. Since there are at most S sessions before, the bound follows. From now on we exclude this case.

Now we can argue that replacing k by $\bar{\mathsf{k}}$ is valid, assuming security of the KEM. The reduction $\mathcal{B}_{\mathsf{KEM}}$ to the IND-CCA security of the long-term KEM now works as follows. The reduction guesses the chip's identity for $\mathsf{pk}_{\mathcal{C}}$ in question at the outset (with probability $1/|\mathcal{P}|$). Knowing $\mathcal{C}, i, j$, it receives $(\mathsf{pk}^*, \mathsf{k}^*, c^*)$ and injects pk^* as $\mathsf{pk}_{\mathcal{C}}$ in the game. This uses the fact that the public key is genuine in the i-th and j-th session. The reduction next executes the entire attack game, but uses k^* whenever it is supposed to use $\bar{\mathsf{k}}$. It also injects c^* in the corresponding terminal session j as c (which is again possible due to the previous considerations). For any other decapsulation request for a ciphertext different from c^* sent to chip $\mathcal{C}$ for $\mathsf{pk}_{\mathcal{C}} = \mathsf{pk}^*$, it calls the decapsulation oracle; all other decapsulation requests for other parties can be done locally. Here we use the fact that no previous session of the chip accidentally receives the same ciphertext c earlier, such that we can indeed process all possible incoming ciphertexts accordingly.

Since the adversary cannot corrupt the chip's long-term KEM key in the non-forward-secure setting, $\mathcal{F} = \mathcal{T}$, the simulation above is sound: We never have to reveal the decapsulation key of the chip later via a CORRUPT request. We also observe that later corruption of the terminal's long-term signing key can be simulated; we only required confidentiality of the key in $\mathbf{Game}_5$ up to the step where the honest terminal signs. In $\mathbf{Game}_{6a}$ the reduction may know the singing key and can reveal it in a subsequent CORRUPT call. It follows that

$$\begin{aligned}\Pr[\mathbf{Game}_5] \;\leq\; & \Pr[\mathbf{Game}_{6a}] \\ & + S \cdot \Big(2^{-|\mathsf{k}|} + \Pr\Big[\mathbf{Exp}^{\text{decErr}}_{\mathsf{KEM}}(\lambda) = 1\Big]\Big) + 2U \cdot \mathbf{Adv}^{\text{IND-CCA}}_{\mathsf{KEM},\mathcal{C}_{\mathsf{KEM}}}(\lambda),\end{aligned}$$

where the factor 2 in term $2U$ compensates for moving from a random challenge bit in the IND-CCA game to one with a fixed challenge bit.

$\mathbf{Game}_{7a}$ Replace in $\mathbf{Game}_{6a}$ the result of applying KDF to $\bar{\mathsf{k}}$ for label `'KEY'` in sessions i and j by an independent and random value of the corresponding length.

To capture the loss for performing this game hop we give a reduction to the pseudorandomness of the KDF for the uniform distribution $\mathcal{D}_{\mathtt{KDF}}(1^\lambda)$ on $\{0,1\}^{|\mathsf{k}_{\mathtt{KDF}}|}$. Our reduction $\mathcal{B}_{\mathsf{KDF}}$ essentially runs $\mathcal{A}$'s attack with help of the oracle $\mathcal{O}_{\mathsf{KDF}}$ (returning true KDF values) and the challenge oracle $\mathcal{O}_b$ (returning actual or random values). The only stipulation is that the two oracles are never called about the same input (ctxt, ℓ). For the simulation note that we have already replaced the actual key by a random value $\bar{\mathsf{k}}$ which now acts as the keying material IKM in the

pseudorandomness game. However, this is done consistently for all chip sessions which receive the same ciphertext c as in the i-th and j-th session. Necessarily, according to the previous game, this can only happen for ciphertexts received after the i-th or j-th session.

In the simulation we thus call oracle $\mathcal{O}_{\mathsf{KDF}}$ for all sessions receiving the same value c about $\mathsf{ctxt} \leftarrow \texttt{'CNF'} \parallel \mathsf{sid}$ resp. $\mathsf{ctxt} \leftarrow \texttt{'KEY'} \parallel \mathsf{sid}$ and $\ell = \lambda$. For the i-th and j-th session, however, for prefix `'KEY'` we call oracle $\mathcal{O}_b$ instead, for the value $\texttt{'KEY'} \parallel \mathsf{ctxt}$. This is admissible since any session of honest parties uses a unique value r_2 resp. c according to $\mathbf{Game}_1$ and $\mathbf{Game}_2$, such that the context information ctxt is two calls to $\mathcal{O}_{\mathsf{KDF}}$ and $\mathcal{O}_b$ are never equal in other sessions (and in the i-th and j-th session the call for label `CNF` is different). Our reduction eventually outputs whatever $\mathcal{A}$ outputs.

It is now easy to conclude that

$$\Pr[\mathbf{Game}_{6a}] \leq \Pr[\mathbf{Game}_{7a}] + 2 \cdot \mathbf{Adv}^{\mathrm{PRF}}_{\mathsf{KDF},\mathcal{B}_{\mathsf{KDF}},\mathcal{D}_{\mathsf{KDF}}}(\lambda),$$

for the uniform input distribution $\mathcal{D}_{\mathsf{KDF}}$ on bit strings of length $|\mathsf{k}_{\mathsf{KDF}}|$.

Remarkably, this already concludes the analysis in the non-forward secure case. We have now replaced $\mathsf{k}_{\mathsf{KEY}}$ by an independent and random value in the test session, such that $\mathcal{A}$ cannot predict b better than by guessing. This step, in contrast to the forward-secure case considered next, does not depend on the security of the confirmation value $\mathsf{k}_{\mathsf{CNF}}$. The reason is that the security of the chip's long-term KEM key, together with the fact that it cannot be legitimately corrupted later, already provides the necessary level of (implicit) authentication.

Forward-secure case. The forward-secure case is slightly more involved but follows a similar line of reasoning. We branch off from $\mathbf{Game}_5$ again. We denote by the terminal session of interest the session i if this is a terminal session resp. the session j if this is the partnered terminal session (and if it exists, i.e., if $i \neq j$). Analogously, we let the chip session of interest be the corresponding other session (if it exists).

A potential attack vector is now that the adversary may impersonate an honest chip $\mathcal{C}$ in a session with the honest terminal $\mathcal{T}$. Then the adversary could send its own (unauthenticated) ephemeral public key $\tilde{\mathsf{pk}}^e$ in the session with $\mathcal{T}$, allowing it to decapsulate the ephemeral key $\tilde{\mathsf{k}}^e$ sent by the terminal. Only the other key part k, protected by the chip's long-term key $\mathsf{pk}_{\mathcal{C}}$, would not be immediately available. But in the forward-secure scenario the adversary could later corrupt the chip's long-term key, revealing also the part k and thus obtain both secrets. As we discuss below, however, the protocol prevents such attacks, since the adversary would have to send the confirmation value $\mathsf{k}_{\mathsf{CNF}}$ in the session with the terminal *before* being able to corrupt the chip's long-term secret and getting hold of both keys.

Game$_{6b}$ Change **Game**$_5$ by declaring the adversary to lose if the terminal session of interest (for certificate $\mathsf{cert}_{\mathcal{T}}$) exists and receives a valid final value $\mathsf{k}_{\texttt{CNF}}$, and where the intended chip partner $\mathsf{cert}_{\mathcal{C}}$ is still honest at this point, but there is no partnered chip session of interest. In particular, the latter means that the test session is the terminal session and we must have $i = j$.

We argue through a sequence of several sub steps that this cannot happen too often. The reason is that, for this to happen, the adversary would still need to learn the key k protected under the chip's (then) honest long-term key $\mathsf{pk}_{\mathcal{C}}$. Otherwise the steps to combine and derive the keys still make $\mathsf{k}_{\texttt{CNF}}$ random. The first step is thus similar to the IND-CCA case in the non-forward case. Since we only consider the case that the chip is honest at this point, and only need to measure if $\mathsf{k}_{\texttt{CNF}}$ is valid before corruption takes place, we can in a first step replace k for any c sent to the intended chip partner by an independent random value $\bar{\mathsf{k}}$. It follows as in **Game**$_{6a}$ that this increases the adversary's success probability (of generating such a value $\mathsf{k}_{\texttt{CNF}}$) by at most

$$S \cdot \left(2^{-|\mathsf{k}|} + \Pr\left[\mathbf{Exp}^{\text{decErr}}_{\mathsf{KEM}}(\lambda) = 1\right]\right) + 2U \cdot \mathbf{Adv}^{\text{IND-CCA}}_{\mathsf{KEM},\mathcal{C}_{\mathsf{KEM}}}(\lambda)$$

for reduction $\mathcal{C}_{\mathsf{KEM}}$.

Next, replace $\mathsf{k}_{\texttt{KDF}} \leftarrow \mathsf{KComb}(\bar{\mathsf{k}}, \mathsf{k}^e)$ in any honest session receiving or sending the ciphertext c for chip $\mathsf{cert}_{\mathcal{C}}$ by a random and independent value $\bar{\mathsf{k}}_{\texttt{KDF}}$. Note that this is possible by the (dual) pseudorandomness of KComb. The reduction $\mathcal{C}_{\mathsf{KComb}}$ controls all steps of **Game**$_{6b}$ and uses the real-or-random oracle to simulate all queries about $\mathsf{KComb}(\bar{\mathsf{k}}, \cdot)$ for sessions with the ciphertext c. Since we have already excluded for the CCA-case the possibility to receive c in some session before it appears in the i-th or j-th session, this simulation is sound and corresponds to have real values $\mathsf{k}_{\texttt{KDF}}$ or random values $\bar{\mathsf{k}}_{\texttt{KDF}}$. Let us stress once more that we let reduction determine its output depending on the adversary sending a valid confirmation value $\mathsf{k}_{\texttt{CNF}}$ in the terminal session of interest. Altogether, we lose another term

$$2 \cdot \mathbf{Adv}^{\text{dPRF}}_{\mathsf{KComb},\mathcal{C}_{\mathsf{KComb}}}(\lambda).$$

The next step is to replace the key $\mathsf{k}_{\texttt{CNF}}$ in the i-th session (and thus j-th session because $i = j$) by an independent and random value $\bar{\mathsf{k}}_{\texttt{CNF}}$. It follows from the pseudorandomness of KDF that this is a valid strategy. More formally, we build a reduction $\mathcal{C}_{\mathsf{KDF}}$ which calls the real-or-random oracle about $\texttt{'CNF'} \,\|\, \mathsf{sid}$ in the i-th session, and the real key derivation oracle for any other honest session (necessarily with a different session identifier). The reduction checks once more if the adversary succeeds in sending a valid confirmation value in the terminal session of interest. This adds another term of

$$2 \cdot \mathbf{Adv}^{\text{PRF}}_{\mathsf{KDF},\mathcal{C}_{\mathsf{KDF}},\mathcal{D}_{\mathsf{KDF}}}(\lambda)$$

for the uniform distribution $\mathcal{D}_{\mathtt{KDF}}$ on strings equal to the length of derivation keys.

The final step is now to notice that the adversary needs to predict a random and unknown value $\bar{\mathsf{k}}_{\mathtt{CNF}}$. The reason is that we have already replaced the actual value $\mathsf{k}_{\mathtt{CNF}}$ by this random value in the i-th session. Since there is no chip session of interest, it follows that the adversary, when supposed to send the confirmation value, has absolutely no information about $\bar{\mathsf{k}}_{\mathtt{CNF}}$ and can thus only succeed with probability $2^{-|\mathsf{k}_{\mathtt{CNF}}|}$.

We emphasize that we are still not done yet. The previous game hop only covers cases where the chip is honest in the moment when it is supposed to send the confirmation value. But we can conclude that, if the test session is a chip session, then an honest party picks the ephemeral public key $\mathsf{pk}_{\mathcal{C}}^e$ freshly in that execution, and since the terminal cannot be corrupt for a successful attack when sending the signature, it must be the partnered honest terminal session creating the ciphertext c^e. Moreover, by the previous game hop, if the test session is a terminal session and created the ephemeral ciphertext c^e, then there must be a (unique) matching chip session, where the chip was honest when creating $\mathsf{pk}_{\mathcal{C}}^e$ in that session. In both cases we thus have an ephemeral public key $\mathsf{pk}_{\mathcal{C}}^e$ and a ciphertext c^e generated honestly by the partnered sessions.

Game$_{7b}$ Replace the key k^e in the i-th and j-th session by an independent random value $\bar{\mathsf{k}}^e$.

We can now run a reduction to the IND-CPA security of the ephemeral KEM^e. The reduction $\mathcal{C}_{\mathsf{KEM}^e}$ essentially simulates the attack according to **Game**$_{6b}$. We inject a given key pk^* as the ephemeral public key $\mathsf{pk}_{\mathcal{C}}^e$ of the chip $\mathcal{C}$ in the session of interest. If the terminal is supposed to create a ciphertext c^e in its session of interest then we inject the given ciphertext c^* as c^e. We use the corresponding given key value k^*, either random or encapsulated in c^*, as a replacement for k^e in both sessions. Note that this reduction and especially the injection of the given values are possible because, according to the previous game hop, the ephemeral public key and ciphertext are generated by honest parties. Any subsequent corruption of the chip's long-term key can be easily simulated by the reduction, since the ephemeral secret key to $\mathsf{pk}_{\mathcal{C}}^e$ is not revealed for this. We derive:

$$\Pr[\mathbf{Game}_{6b}] \leq \Pr[\mathbf{Game}_{7b}] + 2 \cdot \mathbf{Adv}^{\text{IND-CPA}}_{\mathsf{KEM}^e, \mathcal{C}_{\mathsf{KEM}^e}}(\lambda).$$

Game$_{8b}$ In the i-th and j-th session replace the computed value $\mathsf{k}_{\mathtt{KDF}}$ immediately by an independent and random value $\bar{\mathsf{k}}_{\mathtt{KDF}}$.

By the previous game hop we now use an independent random value $\bar{\mathsf{k}}^e$ in the i-th and j-th session. By the (dual) pseudorandomness of KComb, the output of $\mathsf{KComb}(\mathsf{k}, \bar{\mathsf{k}}^e)$ is thus indistinguishable from random. We can easily wrap this into

a corresponding reduction $\mathcal{C}'_{\mathsf{KComb}}$, and obtain

$$\Pr[\mathbf{Game}_{7b}] \leq \Pr[\mathbf{Game}_{8b}] + 2 \cdot \mathbf{Adv}^{\mathrm{dPRF}}_{\mathsf{KComb},\mathcal{C}'_{\mathsf{KComb}}}(\lambda).$$

$\mathbf{Game}_{9b}$ In the i-th and j-th session replace $\mathsf{k}_{\mathtt{KEY}}$ by an independent and random value $\bar{\mathsf{k}}_{\mathtt{KEY}}$ of the same length.

This can be done by the pseudorandomness of KDF. Formally, we get a reduction $\mathcal{C}'_{\mathsf{KDF}}$ which simulates all other steps of $\mathbf{Game}_{8b}$ locally, but uses the real-or-random oracle for KDF when computing $\mathsf{k}_{\mathtt{KEY}}$ in the sessions of interest. For the confirmation key $\mathsf{k}_{\mathtt{CNF}}$ it uses the actual KDF oracle in the pseudorandomness game. Since the computation of the confirmation key uses a different label `'CNF'` than for the session key, this is admissible according to the pseudorandomness game.

We finally have that the test session (and its partner session) both output an independent and random session key. It follows that the adversary cannot predict the secret challenge bit b in the key exchange protocol better than by guessing.

Conclusion. To bound the original success probability $\Pr[\mathbf{Game}_0]$ we can consider the maximum over the non-forward secure and the forward secure variant:

$$\Pr[\mathbf{Game}_0] \leq \max\left\{\Pr[\mathbf{Game}_{7a}], \Pr[\mathbf{Game}_{9b}]\right\}.$$

We simply sum the other advantages in both cases to get the claimed bound. The stated bound in the theorem simplifies the bound further, e.g., by bunching together adversaries attacking the same primitive and prepending a corresponding factor. □

3.5.4 Variations and remarks

We discuss some properties and options of the protocol and its security proof.

Remark 1 (Post-Quantum Security). We stress that our derived adversaries $\mathcal{B}_{\mathsf{cert}}$, $\mathcal{B}_{\mathsf{Sig}}, \mathcal{C}_{\mathsf{KEM}}, \mathcal{C}_{\mathsf{KEM}^e}, \mathcal{C}_{\mathsf{KComb}}, \mathcal{C}_{\mathsf{KDF}}$ in the reductions of the game hops make straightline (black-box) use of adversary $\mathcal{A}$. It follows that they are quantum if $\mathcal{A}$ is, but the reductions work nonetheless. In particular, if we assume that the underlying primitivies are *post-quantum secure*, then so is the key exchange protocol.

Remark 2 (Terminal Authentication). In the proof we do not consider the terminal's first signature $\sigma_{\mathcal{T}}$ in the Terminal Authentication. The reason is that for the secrecy of the derived key only the second signature σ_c is crucial to prevent the adversary from sending data in the name of the terminal. Still, terminal authentication through $\sigma_{\mathcal{T}}$ provides an extra layer of *entity* authentication via a challenge-response sub protocol, even if Chip Authentication is not executed.

Remark 3 (Combining signatures). In the round-reduced version we can simplify the protocol further. Instead of computing two signatures in the same round on the terminal's side, $\sigma_{\mathcal{T}} \leftarrow \mathsf{Sig}(\mathsf{sk}_{\mathcal{T}}, \texttt{'TA'} \,\|\, \mathrm{ID}_{\mathcal{C}} \,\|\, r_1)$ and $\sigma_c \leftarrow \mathsf{Sig}(\mathsf{sk}_{\mathcal{T}}, \texttt{'CA'} \,\|\, \mathsf{sid})$, one can compute a single signature $\sigma \leftarrow \mathsf{Sig}(\mathsf{sk}_{\mathcal{T}}, \texttt{'TC'} \,\|\, \mathrm{ID}_{\mathcal{C}} \,\|\, r_1 \,\|\, \mathsf{sid})$ instead. The proof remains valid for this case.

Remark 4 (Re-using random challenges). For the combined version, since the random values r_1, r_2 are sent within the same message flow now, one can omit r_1 and use a single random value r_2 for both signatures. Or, if combining with the above signature simplification, omit r_1 and create the (single) signature over the data with r_2 only.

Remark 5 (Key Confimation). We have used a simplification according to [FGSW16] for the confirmation values. Usually, one uses the derived key $\mathsf{k}_{\texttt{CNF}}$ to compute a message authentication code (on quasi unique parts of the session transcripts, such as the r_2-value). This is possible and the proof would indeed hold if the MAC is unforgeable. However, as pointed out in [FGSW16], one can also use some keying material (different from the session key) directly to accomplish key confirmation.

Remark 6 (Transcript Hashing). The signature and later the key derivation steps use the session identifiers $\mathsf{sid} = (\mathsf{cert}_{\mathcal{T}}, \mathsf{cert}_{\mathcal{C}}, r_2, c, [\mathsf{pk}^e_{\mathcal{C}}, c^e])$. To save on storage, one can compute a hash value of sid under a collision-resistant hash function. This will add a term for finding collisions under the hash function to the security bound in the theorem. If the hash function is an iterated hash function, then the party only needs to mix in the current data and merely needs to keep the intermediate hash value. This has been done for example in TLS 1.3 [Res18] for transcript hashes.

3.5.5 Security proofs for KemPQEAC

We next discuss the security of the protocol version KemPQEAC where the terminal uses a long-term KEM key to decapsulate a key $\mathsf{k}_{\texttt{TA}}$ from which it derives an encryption key $\mathsf{k}_{\texttt{TENC}}$, a confirmation key $\mathsf{k}_{\texttt{TCNF}}$ for Terminal Authentication, and a MAC key $\mathsf{k}_{\texttt{TMAC}}$. The encryption key is used by the chip to encrypt its certificate information in Chip Authentication, providing privacy. The confirmation key is used as a replacement for the terminal's signature in Terminal Authentication, and the MAC key is used as a replacement for the signature in Chip Authentication.

We note that the proofs for KemPQEAC are very similar to the one for SigPQEAC, noting that the KEM-based transfer of $\mathsf{k}_{\texttt{TCNF}}$ implicitly ensures that only the honest terminal can recover the key and use it in the message authentication step. We thus only discuss the necessary adaptations. We also note that session matching holds as before.

Theorem 2 (Key Secrecy of KemPQEAC)**.** *Protocol KemPQEAC provides key secrecy. Specifically, for any adversary $\mathcal{A}$ initiating at most S sessions with U*

users there exist adversaries $\mathcal{B}_{\mathsf{cert}}, \mathcal{D}_{\mathsf{MAC}}, \mathcal{D}_{\mathsf{KEM}}, \mathcal{C}_{\mathsf{KEM}^e}, \mathcal{D}_{\mathsf{KDF}}, \mathcal{C}_{\mathsf{KComb}}$ *such that*

$$\begin{aligned}
&\boldsymbol{Adv}^{ke}_{\Pi,\mathcal{A},\mathcal{C},\mathcal{T},\mathcal{F}}(\lambda) \\
&\quad \leq \; 3S^3 \cdot \Big(2^{-|r_1|} + 2^{-|r_2|} + 2^{-|\mathsf{k}|} + \Pr\Big[\boldsymbol{Exp}^{decErr}_{\mathsf{KEM}}(\lambda) = 1\Big]\Big) \\
&\qquad + U \cdot \Pr\Big[\boldsymbol{Exp}^{cert\text{-}unf}_{\mathcal{C},\mathcal{B}_{\mathsf{cert}}}(\lambda) = 1\Big] + SU \cdot \Pr\Big[\boldsymbol{Exp}^{EUF\text{-}CMA}_{\mathcal{M},\mathcal{D}_{\mathsf{Sig}}}(\lambda) = 1\Big] \\
&\qquad + S^2 \cdot \Big(6U \cdot \boldsymbol{Adv}^{IND\text{-}CCA}_{\mathsf{KEM},\mathcal{D}_{\mathsf{KEM}}}(\lambda) + 2 \cdot \boldsymbol{Adv}^{IND\text{-}CPA}_{\mathsf{KEM}^e,\mathcal{C}_{\mathsf{KEM}^e}}(\lambda) \\
&\qquad\qquad + 6U \cdot \boldsymbol{Adv}^{PRF}_{\mathsf{KDF},\mathcal{D}_{\mathsf{KDF}},\mathcal{D}_{KDF}}(\lambda) + 4 \cdot \boldsymbol{Adv}^{dPRF}_{\mathsf{KComb},\mathcal{C}_{\mathsf{KComb}}}(\lambda) + 2^{-|\mathsf{k}_{CNF}|}\Big)
\end{aligned}$$

Here, the sets for forward security are given as $\mathcal{F} = \mathcal{P} = \mathcal{C} \cup \mathcal{T}$ *if the ephemeral* KEM^e *is used, and* $\mathcal{F} = \mathcal{T}$ *otherwise. The algorithms* $\mathcal{B}_{\mathsf{cert}}, \mathcal{B}_{\mathsf{Sig}}, \mathcal{C}_{\mathsf{KEM}}, \mathcal{C}_{\mathsf{KEM}^e}, \mathcal{C}_{\mathsf{KComb}}, \mathcal{C}_{\mathsf{KDF}}$ *run in approximately equal time as* $\mathcal{A}$*, and* $\mathcal{D}_{KDF}$ *is the uniform distribution on bit strings of length equal to* $|\mathsf{k}_{KDF}| = |\mathsf{k}_{TA}|$.

Proof. The proof follows the one for the signature-based variant SigPQEAC almost identically. The only difference lies in $\mathbf{Game}_4$ where the proof for SigPQEAC we relied on the unforgeability of the terminal's signature scheme. Subsequently, however, we merely used the fact that this signature could have only come from the terminal session of interest. We can prove the same property here for the KEM-based solution, but need several game hops to reach that conclusion. We start again in the proof for SigPQEAC after $\mathbf{Game}_4$, and replace the hop to $\mathbf{Game}_5$ by the analogue for MACs:

$\mathbf{Game}'_4$ Modify $\mathbf{Game}_3$ by letting the adversary lose if an honest chip (having sent $c_{\mathtt{TA}}$ and r_1) obtains a valid MAC σ_c of an honest terminal $\mathcal{T}$ for a message sid which has not been authenticated by the terminal during chip authentication at this point yet.

The reduction considers several sub steps: First we need to take into account potential collisions for the r_1 value for honest chips. It follows by the birthday bound that honest chip sessions have colliding values with probability at most $S^2 \cdot 2^{-|r_1|}$. From now on we exclude such cases, meaning that there is at most one honest chip session which uses the same r_1-value as in the forgery.

Next we need to guess the right terminal (with probability $1/U$) as well as the chip session in which this happens for the first time (with probability $1/S$). Then we replace the the key $\mathsf{k}_{\mathtt{TA}}$ encapsulated by the honest chip in ciphertext $c_{\mathtt{TA}}$ by a random key $\bar{\mathsf{k}}_{\mathtt{TA}}$. This is done consistently for any other session of that terminal receiving $c_{\mathtt{TA}}$ (and any other honest chip session creating the same $c_{\mathtt{TA}}$ for $\mathcal{T}$). It follows by the IND-CCA security of the KEM that this is indistinguishable for the adversary. The formal reduction $\mathcal{D}_{\mathsf{KEM}}$ receives $(\mathsf{pk}^*, c^*, \mathsf{k}^*)$ as input and injects pk^* as $\mathsf{pk}_{\mathcal{T}}$ for the predicted terminal, as well as c^* as $c_{\mathtt{TA}}$ in the chip session. It

uses k^* as the decapsulated key in all sessions for this terminal in which it receives $c_{\mathtt{TA}}$ (and, analogously in any honest chip session creating $c_{\mathtt{TA}}$ for $\mathcal{T}$). Note that this is possible since c^* is given to the reduction in advance. Any other ciphertext sent to $\mathcal{T}$ can be decapsulated via the decryption oracle (and honest chips know the encapsulated values anyway). We also remark that we are only interested in the point of time in which $\mathcal{T}$ is still honest, such that CORRUPT queries cannot occur at this point.

Next, replace $\mathsf{k}_{\mathtt{TMAC}}$ in the predicted chip session and in any terminal session receiving $c_{\mathtt{TA}}$ and r_1 by a random value $\bar{\mathsf{k}}_{\mathtt{TMAC}}$. This is admissible since $\mathsf{k}_{\mathtt{TMAC}}$ is derived by the pseudorandom key derivation function KDF, applied to the now random key $\bar{\mathsf{k}}_{\mathtt{TA}}$. By the consideration above, it also holds that there is at most one chip session in which r_1 is used. The reduction $\mathcal{D}_{\mathsf{KDF}}$ is straightforward, and uses the fact that we can compute the other derived keys under $\bar{\mathsf{k}}_{\mathtt{TA}}$ via the KDF oracle, either because they use a different r_1-value or a different prefix `'TENC'` or `'TCNF'`.

Lastly, use the unforgeability of the MAC for the final step. That is, our reduction $\mathcal{D}_{\mathsf{MAC}}$ uses an external MAC oracle for computing the MAC σ_c in chip authentication for any terminal session receiving $c_{\mathtt{TA}}$ and r_1. We remark once more that no other chip (or terminal session) than the predicted chip session needs to call verification (or MAC) for that key. Hence, if the predicted chip session receives a valid σ_c for a previously unauthenticated message `'CA'` $\|$ sid, then the reduction has found a forgery for the MAC scheme.

Altogether it follows:

$$\begin{aligned}\Pr[\mathbf{Game}_3] \leq \Pr[\mathbf{Game}'_4] + S^2 \cdot 2^{-r_1} \\ + SU \cdot \Big(\Pr\Big[\mathbf{Exp}^{\text{EUF-CMA}}_{\mathcal{M},\mathcal{D}_{\mathsf{MAC}}}(\lambda) = 1\Big] + 2 \cdot \mathbf{Adv}^{\text{IND-CCA}}_{\mathsf{KEM},\mathcal{D}_{\mathsf{KEM}}}(\lambda) \\ + 2 \cdot \mathbf{Adv}^{\text{PRF}}_{\mathsf{KDF},\mathcal{D}_{\mathsf{KDF}},\mathcal{D}_{\mathtt{TA}}}(\lambda)\Big)\end{aligned}$$

for the uniform distribution $\mathcal{D}_{\mathtt{TA}}$ on $\{0,1\}^{|\mathsf{k}_{\mathtt{TA}}|}$. The rest of the proof is as in the one for Theorem 1. Subsuming the terms above under the theorem's statement yields the claimed bound. □

Remark 7 (Privacy). Recall that we encrypt the chip's identity $\mathsf{cert}_{\mathcal{C}}$ in the first message of Chip Authentication under the key $\mathsf{k}_{\mathtt{TENC}}$ derived in Terminal Authentication for privacy reasons. In the argument above the security of the encryption schemes does not enter the security bound, such that it follows immediately that the proof also holds if encryption is not used. We note that the chip's identity also enters the value sid which is used at several places in the protocol, but usually in MACs or key derivation steps which are often regarded as pseudorandom function.

3.6 Implementation

To show the feasibility of PQ-EAC for border control systems and to evaluate the chip-side of the protocol proposals in resource-constrained environments, we implemented the combined version of SigPQEAC (without forward secrecy) on a chip similar to those deployed in ePassports and on a terminal of the type that is used for border control checks. The chip is integrated in a proof of concept post-quantum eMRTD and runs an ePassport application compliant with ICAO standards. Our performance tests reveal that the post-quantum version of EAC is practical and only slightly slower than the classic version. We published the benchmarking tool that we used to measure the performance of SigPQEAC on a smart card[4]. Other components utilized for benchmarking such as the VISOCORE® terminal software, the ePassport application and our customized Dilithium and Kyber implementations are part of confidential proprietary software libraries and remain unpublished.

We designed the *application protocol data unit* (APDU) interface for reading files or initiating cryptographic operations on the chip to be compatible with the existing standards of ICAO, BSI, and ISO [Int21, Bun16, Int18, Int20b]. For this reason, our implementation uses more than the minimal number of messages possible. We proceed by stating the results of our runtime experiments with our implementation in Section 3.6.1. Finally, we analyze the impact of data transmission rates on the performance of the protocol in Section 3.6.2. Since transmission speeds vary between platforms, and because incorrect placement of the eMRTD can slow transmission rates significantly, this analysis will help to gauge the real-world performance of PQ-EAC more precisely.

3.6.1 Performance evaluation

In our experimental setup we instantiated Combined PQ-EAC without forward secrecy with the post-quantum signature scheme Dilithium3 [LDK+22] and the post-quantum KEM Kyber1024 [SAB+22, BKS19]. NIST's security level 3 (for the signature scheme) and 5 (for the KEM), respectively, were chosen to support the eMRTD's long lifetime of typically ten years. AES256 and SHA256 were used as supporting cryptographic building blocks for the message authentication codes and key derivation functions. In our implementation we assume that the certificates are based on post-quantum signatures. For the sake of data minimization we stick to a single post-quantum signature and public key per certificate.

[4] `https://github.com/frankmorgner/OpenSC-pqc-SSR2023`

Chip Specification. A proof of concept post-quantum eMRTD was implemented on a a contactless security controller from Infineon Technologies based on an ARM SC300 architecture. The security controller comes with significantly increased RAM-size of 96 kBytes, which is more than double the amount of RAM compared to previous security controller chips in this application domain, to support memory-intensive lattice-based schemes such as Kyber and Dilithium.

Chip software. We implemented Kyber and Dilithium in C according to specifications given in the NIST competition round 3 submissions [SAB+22, LDK+22], taking into account the requirements posed by the hardware architecture of our security controller. In particular, memory requirements had to be carefully managed: Our implementation utilizes almost the full RAM capacity of 96 kBytes. However, not all memory optimization opportunities have been exploited by us: By assigning overlapping memory segments to public-key and symmetric cryptographic operations, it would be possible to reduce memory usage to ca. 60 kBytes. Other than memory optimizations, our versions stay close to the mostly unoptimized reference implementation. Since the focus of our work has been the creation of a functional proof of concept for PQ-EAC, most cryptographic building blocks have not been optimized for performance or hardened against side channel analysis and fault attacks.

Besides a hardware-based random number generator, no acceleration has been used in the project. The chip's ePassport application is running as native code on the hardware without an intermediate operating system. All state management (which needs to carefully consider hardware limitations around the volatile and non-volatile memory) is done in that application. For communication with the terminal we utilize a proprietary ISO/IEC 14443 communication library.

Terminal implementation. For demonstration purposes, we modified terminal hard- and software of the type that is used by German border control with post-quantum algorithms. Specifically, we used a Bundesdruckerei VISOTEC® Expert 800[5] running the verification software VISOTEC® Inspect, which was extended by us with the same post-quantum schemes as the eMRTD.

Benchmarking. We ran 1000 experiment executions with the test chip to benchmark the performance of the combined version of SigPQEAC. We measured the time it took to send and receive the card commands via the PC/SC API on the terminal's operating system. Thus, this duration includes the terminal's overhead of processing the data with the smart card reader's driver, its firmware, transceiving data via ISO 14443, and finally the processing by the test chip. For PQ-EAC, we

[5] https://www.bundesdruckerei-gmbh.de/en/solutions/visocore

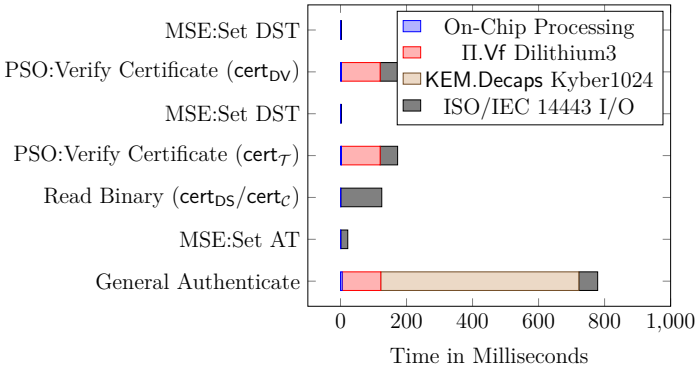

Figure 3.5: Measurements of combined SigPQEAC with 848kbit/s via ISO/IEC 14443 using an emulated chip.

reliably measured a total runtime of 1.28 seconds. The standard deviation for a single command-response pair was between 0.0001 and 0.0003 milliseconds.

In a second experiment we used the same setup with a smart card emulation device (Hitex Tanto3 FPGA) instead of the eMRTD. This device emulates all electrical properties of the eMRTD's chip and allows control and inspection of the card's internal workflow. This way we were able to determine or manually control the connection parameters between reader and card. Specifically, we set the data rate to 848 kBit/s and controlled the frame size for sending 4089 bits of payload data. The emulated chip was constantly run at 100MHz and did not suffer any throttling due to, for example, bad coupling with the smart card reader (a common problem in practice).

The total time for performing all PQ-EAC operations on the emulated chip and the data transfer is 1.28 seconds. In Figure 3.5 we can see that the execution time on the chip is heavily dominated by the cryptographic operations during signature validation and KEM decapsulation. As such, Kyber decapsulation takes 596 ms, Dilithium verification takes 86 ms, but choosing a nonce and reading $\mathsf{cert}_{\mathcal{C}}$ only take 0.01 ms each.

The rest of the transaction time is caused by transferring the data back and forth between terminal and card. Again, most of this can be attributed to the large ciphertexts, public keys, and signatures of Dilithium and Kyber. For example, $\mathsf{cert}_{\mathcal{T}}$ contains a a Dilithium signature (2701 bytes) and the terminal's Dilithium public key (1472 bytes) which add up to 97.6% of the overall size of the certificate. Transferring this data with a data rate R_b of 848 kBit/s takes roughly 50 ms including the ISO/IEC 14443 overhead.

Our analysis shows that the processing time on the chip is almost exclusively caused by the the post-quantum algorithms. Further, we observe that in our proof

of concept implementation each of encapsulation, decapsulation and key generation takes around five times longer than verifying a signature. However, the literature suggests that a better optimized implementation on similar hardware speeds up the Kyber1024 encapsulation by an order of magnitude to half the execution time of the signature verification of Dilithium3 [KPR+]. Such an improvement would most likely allow an overall execution time for PQ-EAC of one second or below.

3.6.2 Impact of the data transfer rate

We observed that the post-quantum algorithms are causing most of the data that needs to be transceived between eMRTD and chip. Using our experimental data we can confirm that the (extended) ISO/IEC 14443 I-Block framing is causing an overhead of roughly 26% of the transmitted protocol data[6]. Also, we observe that the constant overhead for encoding the data into command and response APDUs varies between 9 and 35 bytes. This observation allows to approximate the runtime of other variants of PQ-EAC in relation to data transfer rates.

	Estimated Runtime (in milliseconds)		
Data Rate (R_b)	SigPQEAC	Combined SigPQEAC	KemPQEAC
848 kbit/s	1406	1280	1569
424 kbit/s	1718	1530	1754
106 kbit/s	3586	3204	2865

Table 3.1: Estimation for the protocol variants for typical transmission rates. For details on PQ-EAC Combined at 848 kbit/s see Figure 3.5.

Using the measurements from our experiments with the test chip and the estimated communication load, we present the estimated runtime for the protocols relative to different transmission data rates in Table 3.1. Mostly due to the fact that it transfers less data and uses one less signature verification, the combined version of SigPQEAC achieves the shortest runtime. The advantage of KemPQEAC under low data rates is due to the fact that no signatures, which are typically larger than ciphertexts or keys, need to be sent. The slow key encapsulation of the KEM in our proof-of-concept implementation, however, causes it to be the slowest protocol for higher data transfer rates. Given that benchmarks [KPR+] show that KEM encapsulation is usually much faster than signature verification, runtime of

[6]The ISO/IEC 14443 overhead is between 23% for very big APDUs and 72% for very small APDUs. Since most of the EAC protocols' runtime is spent on big commands, we stick to an approximation near that of bigger commands.

PQ-EAC and especially of KemPQEAC should significantly benefit from further optimizations of the implementation.

Comparison with classical EAC. An execution of classical EAC on eMRTDs and terminals commonly in use today typically takes around 1.5 to 2 seconds. Noting that most deployed electronic passports and verification terminals are already supporting 848 kbit/s data rate, our proof-of-concept implementation for PQ-EAC with well below 2 seconds execution time supports the conclusion that eMRTDs can be migrated to quantum-resistant algorithms. Considering performance gains from hardware acceleration and general implementation improvements we assume that instantiations with hybrid schemes are also feasible.

3.7 Conclusion

In the preceding sections we have presented and implemented quantum-resistant versions of the EAC protocol. Our results show that post-quantum travel documents are practical. Moreover, our implementation of PQ-EAC can still be optimized. It can also easily be instantiated with alternative KEMs and signature schemes. Below, we document a few opportunities for future research.

Post-quantum PKI and PACE. In our protocols we assumed that the authenticity of the chip and terminal are secured by quantum-resistant certificates: A quantum-resistant PKI that will provide such certificates is a challenge to be addressed by future research. Similarly, we assumed that the initial connection between chip and terminal is secured via PACE. As current versions of PACE are based on classical assumptions, there is need for a quantum-resistant construction that can secure the transmission of less sensitive data groups. Another option would be to simply do without PACE and instead mandate an execution of the KEM-only version of PQ-EAC for all requests of chip data, not only for sensitive data groups. This solution would provide the same security guarantees as a sequential execution of PACE and EAC, and additionally rectify a current privacy issue that arises when only PACE is performed: Since Passive Authentication requires the chip to send a list of hashes of all data groups, it allows anyone who knows the chip's MRZ to match the hash of the fingerprint with a collection of precollected fingerprints [CV10]. When EAC is mandatory, we can eliminate this privacy threat by forcing the terminal to authenticate before performing Passive Authentication.

Cryptographic agility. Another problem we leave for future research is cryptographic agility in eMRTDs. In particular, it is desirable to have an update mechanism that provides an authenticated update for eMRTDs in the case that

the deployed quantum-resistant algorithms need to be replaced. This would be the case when flaws in the used algorithms have been discovered. The challenge here is that we have a chicken-egg situation: Ultimately the authentication mechanism of the update would have to inspire more confidence than the post-quantum schemes we have instantiated EAC with. A potential candidate for such an authentication mechanism are hash-based signatures [HBG+18, MCF19]: While they are less practical than lattice-based constructions, their security rests on weaker assumptions, namely on the one-wayness of certain hash functions.

Future directions for eMRTDs. Recently ICAO has established a working group on the so called *Digital Travel Credential* (DTC) and published version 4.4 of "Guiding Core Principles for the Development of Digital Travel Credential DTC" [Int20a]. In this document the DTC is described as consisting of two parts: the DTC Physical Component (DTC-PC) and the DTC Virtual Component (DTC-VC). While a DTC-PC represents a physical eMRTD, the DTC-VC could potentially be stored on a smartphone and then be used in lieu of a physical eMRTD. However, at the moment it is planned to store only primary biometrics (facial image) in the DTC-VC, while secondary biometrics (fingerprint) will be exclusively stored on the DTC-PC. Future research will need to address security of DTCs, especially when coupled with smartphones.

4 Privacy-Preserving Power Flow Analysis

Author's contribution. This chapter describes joint work with Nils Schlüter, Philipp Binfet, Martin Asman, Markus Zdrallek, Tibor Jager, and Moritz Schulze Darup, published in [vdHSB+25]. The author of this thesis, Nils Schlüter, Philipp Binfet and Martin Asman contributed to the design of the privacy-preserving power flow analysis, with the author contributing mainly cryptographic expertise and the others mainly know-how about power flow analysis and numerical optimization. The author also provided the security analysis, wrote the implementation and generated the benchmarks. Martin Asman additionally furthered our understanding of smart grids and suggested flexibility markets as a use case. Markus Zdrallek, Tibor Jager and Moritz Schulze Darup provided project supervision.

4.1 Motivation and Overview

The growing reliance on intermittent renewables, as discussed in Section 1.2, creates grid stabilization challenges that require new management techniques. *power flow analysis* (PFA) is a key tool in this context, as it can anticipate and help mitigate issues like voltage instability and overloads by using power forecasts. Here, usage data is a key enabler because it serves as a basis for such forecasts and more reliable planning. Especially low-voltage networks would benefit from prosumer data since the massive deployment of photovoltaics and batteries has a significant impact on load balances.

Smart meters form an essential building block in smart grids by providing prosumer data in the form of voltage, current, and power. However, the fine-grained data from smart meters, which is essential for these analyses, can reveal sensitive lifestyle patterns. As established by prior work [MSF+10, GGJL12, ADMC17], it is possible to infer detailed personal information, from daily routines to media consumption habits, from power usage data alone. Such information is a valuable resource that, if unprotected, could be used for unwarranted surveillance, intrusive marketing, or even to inform criminal activity [JKD12]. Furthermore, detailed private information can be disclosed when additional background knowledge is available [MSF+10]. Thus, leakage of smart meter data poses a major threat.

At the same time, private companies (with or without regulatory oversight) are partially commissioned to store and process this data. For instance, Itron, Triliant, Landis+Gyr, Oracle, Sensus, and Siemens offer storage and processing solutions for metering data. Besides regulatory measures, technical solutions can help to secure smart metering data against abuse. Hence, institutions such as the European Commission and the US *National Institute of Standards and Technology* (NIST) stress the importance of introducing privacy-preserving measures to smart grids [Eur15] [NIS14].

Against this background, we are interested in a privacy-preserving PFA based on confidential smart metering and forecast data. This way, a reliable grid operation and security are no longer conflicting goals. Our interest in power flow is additionally driven by its fundamental role in power systems and the fact that solution techniques via Newton's method (which we will use here) are applied across various contexts. Specifically, these methods are pivotal in state estimation, optimal power flow [Zhu15, Chapters 2.6 and 8], and advanced grid optimization techniques [FEMH18], underscoring their relevance beyond the scope of this work.

4.1.1 Outline

This chapter is structured as follows. In Section 4.2, we briefly recall the power flow problem and introduce the techniques that form the basis of our privacy-preserving implementation. Section 4.3 then deals with a suitable power flow formulation, while Section 4.4 and Section 4.5 are concerned with its *multi-party computation* (MPC) implementation and security, respectively. Section 4.6 presents the benchmarks for two different smart grid topologies. We conclude with Section 4.7.

4.2 Preliminaries

This section serves as a preparation for the remainder of this chapter and will introduce the power flow problem.

The power flow problem. Power systems are typically modeled by a set of buses (nodes) and branches (transmission lines). Remarkably, assuming stationary operating conditions in a grid with alternating current, the $N \in \mathbb{N}$ buses are fully described by the voltages $\mathbf{v} \in \mathcal{C}^N$ (typically given in terms of magnitude and phase angle) as well as the active and reactive powers $\boldsymbol{p} \in \mathbb{R}^N$ and $\boldsymbol{q} \in \mathbb{R}^N$. In this work, we focus on the privacy of smart grid prosumers. Therefore, only load buses (PQ-buses) are of interest here. In other words, $\mathbf{v}$ is the unknown quantity while forecasts based on historical data or predictions are available for the active and reactive power vectors $\boldsymbol{p}_0$ and $\boldsymbol{q}_0$, respectively.

In order to find $\mathbf{v}$, we first note that it is linearly related to the currents $\boldsymbol{i} \in \mathcal{C}^N$ by the admittance matrix $\boldsymbol{Y} = \boldsymbol{G} + \mathrm{j}\boldsymbol{B}$ via $\boldsymbol{i} = \boldsymbol{Y}\mathbf{v}$. With this relationship, the bus powers can be equivalently and compactly expressed via the apparent power as $\boldsymbol{s} = \mathrm{diag}(\mathbf{v})(\boldsymbol{Y}\mathbf{v})^*$. Finally, with help of the prescribed values $\boldsymbol{s}_0 = \boldsymbol{p}_0 + \mathrm{j}\boldsymbol{q}_0$, we find the relation

$$\boldsymbol{s}(\mathbf{v}) - \boldsymbol{s}_0 = \mathrm{diag}(\mathbf{v})(\boldsymbol{Y}\mathbf{v})^* - \boldsymbol{p}_0 - \mathrm{j}\boldsymbol{q}_0 = \mathbf{0}, \tag{4.1}$$

which is the well-known power flow equation. Note that (4.1) is a non-linear system of equations in $\mathbf{v}$ for which Newton's method is a popular choice. To this end, (4.1) is rephrased in terms of $\boldsymbol{F}(\boldsymbol{x}) = \mathbf{0}$ (details follow in Section 4.3.2). Then, instead of directly solving $\boldsymbol{F}(\boldsymbol{x}) = \mathbf{0}$, a sequence of linear systems in the form

$$\boldsymbol{J}(\boldsymbol{x}_k)\Delta\boldsymbol{x}_k = -\boldsymbol{F}(\boldsymbol{x}_k), \tag{4.2}$$

with $k \in \mathbb{N}$ and the Jacobian $\boldsymbol{J} = \partial\boldsymbol{F}/\partial\boldsymbol{x}$, is solved until the residual $\|\boldsymbol{F}(\boldsymbol{x}_k)\|_2^2$ is sufficiently small. After solving (4.2) for $\Delta\boldsymbol{x}_k$, the state is updated according to $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta\boldsymbol{x}_k$.

Once voltages $\mathbf{v}$ satisfying (4.1) are found, the remaining unknown quantities are readily computed, such as the voltage and branch current magnitudes $\boldsymbol{V}$ and $\boldsymbol{I}_b$. These quantities are of interest for verifying the operating conditions of the grid via

$$\boldsymbol{V}_{\min} \leq \boldsymbol{V} \leq \boldsymbol{V}_{\max} \quad \text{and} \quad \boldsymbol{I}_b \leq \boldsymbol{I}_{\max}. \tag{4.3}$$

In case (4.3) is violated, the grid's safety and stability is endangered and electrical devices or systems may be damaged. Therefore, it is of paramount importance that the grid operator takes adequate measures to ensure (4.3) at all times.

4.3 Tailored Power Flow Analysis

Equipped with the knowledge from Section 4.2, we can now shift our focus towards a privacy-preserving solution of the power flow problem (4.1) by first identifying a suitable solution algorithm. In fact, one of the main ingredients for a practical implementation is a solution algorithm that is adapted to the requirements of MPC protocols. While the techniques we discuss and employ in this section are known, interdisciplinary knowledge is required to select a well-suited tailored approach.

4.3.1 Discussion of different formulations

There are different formulations available for the power flow problem [SVW19]. First, a complex formulation can be considered, where (4.1) is solved directly

with $\mathbf{v}$ as a variable. However, the derivative of a complex conjugate is not well-defined. Thus, [LN97] relies on an approximation based on differentials, which can deteriorate the step direction. In [SVW19], the conjugate currents are kept in the formulation, which leads to linear convergence (opposed to quadratic convergence in other formulations). Thus, although a complex formulation can reduce the solution effort in each iteration, the number of iterations is expected to be higher. Second, a polar formulation for (4.1) can be used, which is a popular choice. There, $\mathbf{v}$ is understood in terms of voltage magnitude and phase angle. In this case, the Jacobian will contain sine and cosine terms, which have to be reevaluated whenever a phase angle is updated. Especially in an MPC formulation, this would become a major source of inefficiency. Although intermediate forms exist, the best suited option seems to be a Cartesian formulation because the Jacobian can be implemented using only additions and multiplications, which are native MPC operations.

4.3.2 Cartesian power flow formulation

Since a Cartesian formulation is well-known we will only briefly specify the remaining quantities in (4.2). First, real and complex components are treated separately, which leads to the definitions

$$\boldsymbol{x} = \begin{pmatrix} \mathbf{v}_R \\ \mathbf{v}_I \end{pmatrix} = \begin{pmatrix} \mathrm{Re}\,(\mathbf{v}) \\ \mathrm{Im}\,(\mathbf{v}) \end{pmatrix} \quad \text{and} \quad \boldsymbol{F}(\boldsymbol{x}) = \begin{pmatrix} \boldsymbol{p}(\boldsymbol{x}) - \boldsymbol{p}_0 \\ \boldsymbol{q}(\boldsymbol{x}) - \boldsymbol{q}_0 \end{pmatrix}.$$

Next, we proceed with $\boldsymbol{s} = \mathrm{diag}(\mathbf{v})(\boldsymbol{Y}\mathbf{v})^*$ and obtain

$$\begin{pmatrix} \boldsymbol{p}(\boldsymbol{x}) \\ \boldsymbol{q}(\boldsymbol{x}) \end{pmatrix} = \begin{pmatrix} \mathrm{diag}(\mathbf{v}_R) & \mathrm{diag}(\mathbf{v}_I) \\ \mathrm{diag}(\mathbf{v}_I) & -\mathrm{diag}(\mathbf{v}_R) \end{pmatrix} \begin{pmatrix} \boldsymbol{G} & -\boldsymbol{B} \\ \boldsymbol{B} & \boldsymbol{G} \end{pmatrix} \begin{pmatrix} \mathbf{v}_R \\ \mathbf{v}_I \end{pmatrix} \tag{4.4}$$

which specifies $\boldsymbol{F}(\boldsymbol{x})$ and

$$\boldsymbol{J}(\boldsymbol{x}) = \begin{pmatrix} \boldsymbol{H}_1 + \boldsymbol{H}_3 & \boldsymbol{H}_2 + \boldsymbol{H}_4 \\ \boldsymbol{H}_2 - \boldsymbol{H}_4 & -\boldsymbol{H}_1 + \boldsymbol{H}_3 \end{pmatrix} \quad \text{with} \tag{4.5}$$

$$\begin{aligned} \boldsymbol{H}_1 &= \mathrm{diag}(\mathbf{v}_R)\boldsymbol{G} + \mathrm{diag}(\mathbf{v}_I)\boldsymbol{B}, \quad \boldsymbol{H}_3 = \mathrm{diag}(\boldsymbol{G}\mathbf{v}_R - \boldsymbol{B}\mathbf{v}_I), \\ \boldsymbol{H}_2 &= \mathrm{diag}(\mathbf{v}_I)\boldsymbol{G} - \mathrm{diag}(\mathbf{v}_R)\boldsymbol{B}, \quad \boldsymbol{H}_4 = \mathrm{diag}(\boldsymbol{B}\mathbf{v}_R + \boldsymbol{G}\mathbf{v}_I). \end{aligned}$$

In order to avoid ambiguous solutions for $\boldsymbol{F}(\boldsymbol{x}) = \mathbf{0}$, the voltage at the so-called slack node must be fixed, which serves as a voltage reference. Here, the slack node is, without loss of generality, assumed to be the first node in the grid, and the corresponding rows in $\boldsymbol{F}$ as well as the rows and columns in $\boldsymbol{J}$ are omitted.

4.3.3 Line search

Instead of using the solution of (4.2) directly as a Newton step, we introduce the step size η such that the update becomes $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \eta\Delta\boldsymbol{x}_k$. By choosing η suitably, the convergence speed and the robustness of Newton's method is enhanced. Finding an appropriate η is usually done by approximating the optimal step size

$$\eta^\star = \arg\min_\eta g(\eta) = \arg\min_\eta \frac{1}{2}\boldsymbol{F}(\eta)^\top \boldsymbol{F}(\eta), \tag{4.6}$$

where we omitted the dependency on $\boldsymbol{x}_k$ as well as $\Delta\boldsymbol{x}_k$, since they are constant in (4.6). Of course, algorithms for (4.6) exist (see for instance [NW99, Chapter 3] and [PTVF07, Chapter 9]). However, they require additional elementary functions and logic, which increases the communication overhead in an MPC solution. Instead, we opted for the following approach.
The optimizer of (4.6) satisfies $\mathrm{d}g(\eta^\star)/\mathrm{d}\eta = 0$ where a change by $\Delta\eta_j = \eta_{j+1} - \eta_j$ with $j \in \mathbb{N}$ is

$$\frac{\mathrm{d}g(\eta_j + \Delta\eta_j)}{\mathrm{d}\eta} \approx \frac{\mathrm{d}g(\eta_j)}{\mathrm{d}\eta} + \frac{\mathrm{d}^2 g(\eta_j)}{\mathrm{d}\eta^2}\Delta\eta_j = 0. \tag{4.7}$$

Then, the finite difference formulas

$$\frac{\mathrm{d}g(\eta_j)}{\mathrm{d}\eta} = \boldsymbol{F}(\eta_j)^\top \boldsymbol{J}(\eta_j)\Delta\boldsymbol{x}_k \approx \frac{1}{\eta_j}\boldsymbol{F}(\eta_j)^\top \left(\boldsymbol{F}(\eta_j) - \boldsymbol{F}(0)\right),$$
$$\frac{\mathrm{d}^2 g(\eta_j)}{\mathrm{d}\eta^2} \approx \frac{\mathrm{d}g(\eta_j)/\mathrm{d}\eta - \mathrm{d}g(\eta_{j-1})/\mathrm{d}\eta}{\eta_j - \eta_{j-1}},$$

and the abbreviation $\xi_j := \boldsymbol{F}(\eta_j)^\top \left(\boldsymbol{F}(\eta_j) - \boldsymbol{F}(0)\right)$ allow rearranging (4.7) into the compact form

$$\Delta\eta_j = \Delta\eta_{j-1}\frac{\eta_{j-1}\xi_j}{\eta_j\xi_{j-1} - \xi_j}. \tag{4.8}$$

Note that in (4.8) computing $\boldsymbol{J}$ is circumvented and allowing $\Delta\eta > 1$ can further reduce the amount of linear equation systems (4.2) to be solved.

4.4 Privacy-Preserving Power Flow Analysis

The following section deals with a privacy-preserving solution of (4.1) using the formulation and tools discussed in Section 2.3.1 and Section 4.3, respectively.
In order to simplify the presentation we use $[x]$ on $x \in \mathbb{R}$ to denote the fixed-point encoding in $\mathbb{Z}_p$ via $z = \lfloor 2^h x \rceil \bmod p$ for some $h \in \mathbb{N}$ and a subsequent secret sharing. Initially, each prosumer P_i computes shares $([p_{0,i}], [q_{0,i}])$ and distributes

them among the other $n-1$ parties. Similarly, shares of the initial guess $[\boldsymbol{x}_0] := ([\mathrm{v}_{R,1}]\ldots,[\mathrm{v}_{R,N}],[\mathrm{v}_{I,1}],\ldots[\mathrm{v}_{I,N}])^\top$ and (depending on their secrecy) shares or the public values of $\boldsymbol{G},\boldsymbol{B}$ are distributed. In case $\boldsymbol{G}$ and $\boldsymbol{B}$ are unavailable at first, one may use the following estimation strategy.

4.4.1 Estimation of the admittance matrix

The admittance matrix $\boldsymbol{Y}$, determined by $\boldsymbol{G}$ and $\boldsymbol{B}$, can be robustly estimated based on the measurements $\{\mathbf{v}_{R,\kappa},\mathbf{v}_{I,\kappa},\boldsymbol{i}_{R,\kappa},\boldsymbol{i}_{I,\kappa}\}_{\kappa=0}^{T}$ where $T\in\mathbb{N}$ is sufficiently large. To this end, we use $\boldsymbol{x}_\kappa^\top=(\mathbf{v}_{R,\kappa}^\top,\mathbf{v}_{I,\kappa}^\top)$ and define $\boldsymbol{w}_\kappa^\top=(\boldsymbol{i}_{R,\kappa}^\top,\boldsymbol{i}_{I,\kappa}^\top)$ such that $\boldsymbol{w}_\kappa=\boldsymbol{W}\boldsymbol{x}_\kappa$ with

$$\boldsymbol{W}=\begin{pmatrix}\boldsymbol{G} & -\boldsymbol{B}\\ \boldsymbol{B} & \boldsymbol{G}\end{pmatrix}.$$

Then, the optimizer $\arg\min_{\boldsymbol{W}}\sum_{\kappa=0}^{T}\|\boldsymbol{W}\boldsymbol{x}_\kappa-\boldsymbol{w}_\kappa\|_2^2$ is

$$\mathrm{vec}\,(\boldsymbol{W}^\star)=\begin{pmatrix}\boldsymbol{x}_0^\top\otimes\mathbb{I}\\ \vdots\\ \boldsymbol{x}_T^\top\otimes\mathbb{I}\end{pmatrix}^{+}\begin{pmatrix}\boldsymbol{w}_0\\ \vdots\\ \boldsymbol{w}_T\end{pmatrix},\tag{4.9}$$

where $(\cdot)^+$ denotes the Moore-Penrose inverse and $\otimes$ is the Kronecker product. In case only measurements $\{\mathbf{v}_{R,\kappa},\mathbf{v}_{I,\kappa},\boldsymbol{p}_\kappa,\boldsymbol{q}_\kappa\}_{\kappa=0}^{T}$ are available, one can build on (4.4). In comparison to before, we define $\tilde{\boldsymbol{w}}_\kappa^\top=(\boldsymbol{p}_\kappa^\top,\boldsymbol{q}_\kappa^\top)$ and

$$\boldsymbol{V}(\boldsymbol{x})=\begin{pmatrix}\mathrm{diag}(\mathbf{v}_R) & \mathrm{diag}(\mathbf{v}_I)\\ \mathrm{diag}(\mathbf{v}_I) & -\mathrm{diag}(\mathbf{v}_R)\end{pmatrix}$$

such that (4.4) becomes $\tilde{\boldsymbol{w}}_\kappa=\boldsymbol{V}(\boldsymbol{x}_\kappa)\boldsymbol{W}\boldsymbol{x}_\kappa$. Then, $\boldsymbol{W}^\star$ is obtained from (4.9) by replacing $\mathbb{I}$ with $\boldsymbol{V}(\boldsymbol{x}_\kappa)$ and $\boldsymbol{w}_\kappa$ with $\tilde{\boldsymbol{w}}_\kappa$. Note that (4.9) can be solved privately (without revealing any of the measurements) using the methods presented above to compute, e.g., a singular value decomposition. Furthermore, there are no hard restrictions on the computation time for $\boldsymbol{W}^\star$.

4.4.2 Gradient vector and Jacobian

Next, $[\boldsymbol{F}(\boldsymbol{x})]$ and $[\boldsymbol{J}(\boldsymbol{x})]$ are computed where we drop the iteration index k and consider $i\in\{1,2,\ldots,N\}$ (for the moment) to simplify the presentation. Given $[\boldsymbol{x}]$, the prosumers first invoke addition and multiplication protocols to compute

$$\begin{aligned}[i_{R,i}]&=\sum_{j=1}^{N}[G_{ij}][\mathrm{v}_{R,j}]-[B_{ij}][\mathrm{v}_{I,j}]\\ [i_{I,i}]&=\sum_{j=1}^{N}[B_{ij}][\mathrm{v}_{R,j}]+[G_{ij}][\mathrm{v}_{I,j}].\end{aligned}\tag{4.10}$$

Then, with $[i_{R,i}], [i_{I,i}]$ at hand, one can compute $[\boldsymbol{F}(\boldsymbol{x})] := ([F_1], \ldots, [F_{2N}])^\top$ via

$$\begin{aligned} [F_i] &= [\mathrm{v}_{R,i}][i_{R,i}] + [\mathrm{v}_{I,i}][i_{I,i}] - [p_{0,i}] \\ [F_j] &= [\mathrm{v}_{I,i}][i_{R,i}] - [\mathrm{v}_{R,i}][i_{I,i}] - [q_{0,i}] \end{aligned} \tag{4.11}$$

where $j = i + N$. Lastly, we focus on the shares in $[\boldsymbol{J}(\boldsymbol{x})]$ which depend on

$$\begin{aligned} [H_{1,ij}] &= [G_{ij}][\mathrm{v}_{R,i}] + [B_{ij}][\mathrm{v}_{I,i}] \\ [H_{2,ij}] &= [G_{ij}][\mathrm{v}_{I,i}] - [B_{ij}][\mathrm{v}_{R,i}] \\ [H_{3,ii}] &= [i_{R,i}] \quad \text{and} \quad H_{3,ij} = 0 \quad \text{for } i \neq j \\ [H_{4,ii}] &= [i_{I,i}] \quad \text{and} \quad H_{4,ij} = 0 \quad \text{for } i \neq j \end{aligned} \tag{4.12}$$

where $[G_{ii}][\mathrm{v}_{R,i}]$, $[G_{ii}][\mathrm{v}_{I,i}]$, $[B_{ii}][\mathrm{v}_{R,i}]$, $[B_{ii}][\mathrm{v}_{I,i}]$ can be reused from (4.10). Finally, we collect the shares and construct $[\boldsymbol{J}(\boldsymbol{x})]$ as in (4.5) via the local addition protocol.
Complexity: The execution of (4.10) requires at most $4N^2 - 4N$ secret multiplications. However, they can be reduced as follows. First, the sparsity structure of $\boldsymbol{G}$ and $\boldsymbol{B}$ may be public information (because it is directly linked to the grid's topology). Then, only multiplications with non-zero factors have to be performed. Second, by assuming $\boldsymbol{G}$ and $\boldsymbol{B}$ are public, all multiplications in (4.10) become local operations. Although this assumption is not strictly necessary, it is reasonable from our perspective since $\boldsymbol{G}$ and $\boldsymbol{B}$ do not contain confidential information (they could be calculated by the prosumers with little effort).
Considering (4.11) and (4.12) note that F_i and J_{ij} for $i, j \in \{1, N+1\}$ can be omitted due to the slack node. Then, (4.11) for $[\boldsymbol{F}(\boldsymbol{x})]$ costs $4N-4$ multiplications. Similarly, (4.12) for $[\boldsymbol{J}(\boldsymbol{x})]$ depends mostly on (4.10). Taking the precomputed results into account, the Jacobian requires at most $4(N-1)^2 - 4N$ additional multiplications. However, for the computation of $[H_{1,ij}], [H_{2,ij}]$ also potential savings due to public $\boldsymbol{G}, \boldsymbol{B}$ apply. Interestingly, if $\boldsymbol{G} = \boldsymbol{G}^\top$ and $\boldsymbol{B} = \boldsymbol{B}^\top$ then the previous results from (4.10) determine (4.12) even if $\boldsymbol{G}$ and $\boldsymbol{B}$ are secret, i.e., (4.12) becomes free.
In total, the computation of $[\boldsymbol{F}(\boldsymbol{x})]$ and $[\boldsymbol{J}(\boldsymbol{x})]$ requires at most $8N^2 - 12N$ multiplications. The fastest case arises if $\boldsymbol{G}, \boldsymbol{B}$ are not kept secret, in which (4.10) and (4.12) become local operations and can be computed with $4N - 4$ multiplications.

4.4.3 Solving the system of linear equations

Here, we consider solving $\boldsymbol{J}(\boldsymbol{x})\Delta\boldsymbol{x} = -\boldsymbol{F}(\boldsymbol{x})$ but using $[\boldsymbol{J}(\boldsymbol{x})]$ and $[\boldsymbol{F}(\boldsymbol{x})]$. Taking into account that $\boldsymbol{J}$ offers no special properties (other than sparsity because a bus in low voltage grids has typically not more than two neighbors), we will experimentally compare a suitable direct with an indirect solver. Namely, an

lower-upper (LU) decomposition [PTVF07, Chapter 2.3] with a subsequent forward-backward substitution, and *generalized minimal residual method* (GMRES) [Saa03, Algorithm 6.9]. Both have been slightly optimized for an MPC execution.

Direct solver

Algorithm 1 illustrates our LU decomposition and the forward-backward substitution based on shares. As a reminder, the LU decomposition computes $[\boldsymbol{L}], [\boldsymbol{U}]$ such that $\boldsymbol{LU} = \boldsymbol{J}(\boldsymbol{x})$. This allows to solve $\boldsymbol{L}\Delta\tilde{\boldsymbol{x}} = -\boldsymbol{F}(\boldsymbol{x})$ first and then $\boldsymbol{U}\Delta\boldsymbol{x} = \Delta\tilde{\boldsymbol{x}}$ where the triangular structure of $\boldsymbol{L}$ and $\boldsymbol{U}$ is beneficial. Note that we refrain from pivoting (swapping rows and/or columns) to save communications (mainly for comparisons). While no pivoting is generally not recommended due to numerical instability, our strategy can be justified a priori as follows. We note that pivoting is necessary to avoid a division by 0, which occurs if an element in $\mathrm{diag}(\boldsymbol{J}(\boldsymbol{x}))$ is 0. There, $\mathrm{diag}(\boldsymbol{J}(\boldsymbol{x}))$ denotes setting all but the main diagonal elements of $\boldsymbol{J}(\boldsymbol{x})$ to zero. In other words, as long as the elements in $\mathrm{diag}(\boldsymbol{J}(\boldsymbol{x}))$ are sufficiently far away from 0, pivoting is not required. Our approach is now to check a priori that this is fulfilled for $\boldsymbol{x} \in \mathbb{R}^{2(N-1)}$ based on $\boldsymbol{G}$ and $\boldsymbol{B}$. By inspection of (4.5) one can see that $\mathrm{diag}(\boldsymbol{J})$ depends on $\mathrm{diag}(\boldsymbol{H}_3 \pm \boldsymbol{H}_1)$ which can be rearranged into

$$\begin{pmatrix} \boldsymbol{G} + \mathrm{diag}(\boldsymbol{G}) & -\boldsymbol{B} + \mathrm{diag}(\boldsymbol{B}) \\ \boldsymbol{G} - \mathrm{diag}(\boldsymbol{G}) & -\boldsymbol{B} - \mathrm{diag}(\boldsymbol{B}) \end{pmatrix} \begin{pmatrix} \mathbf{v}_R \\ \mathbf{v}_I \end{pmatrix} = \boldsymbol{K}\boldsymbol{x}.$$

Then, $\boldsymbol{K}\boldsymbol{x} \neq \boldsymbol{0}$ without $\boldsymbol{x} = \boldsymbol{0}$ requires an empty nullspace of $\boldsymbol{K}$ which is the case if $\det(\boldsymbol{K}) \neq 0$. Furthermore, one can efficiently compute a bound from zero for the elements in $\mathrm{diag}(\boldsymbol{J})$ by, e.g., using a binary search for the smallest positive ε_i for all $i \in \{1, \ldots, 2N\}$ such that $\sigma^\star > 0$ resulting from

$$\min_{\sigma, \boldsymbol{x}} \sigma \;\text{ s.t. }\; -\sigma - \varepsilon_i \leq \boldsymbol{K}_{i,:}\boldsymbol{x} \leq \varepsilon_i + \sigma, \;\; 0 \leq \sigma,$$

is fulfilled. Clearly, $\boldsymbol{K}_{i,:}\boldsymbol{x} \notin [-\varepsilon_i, \varepsilon_i]$ as long as $\sigma^\star > 0$, because otherwise $\sigma = 0$ would have been optimal. Finally, the bound is $\min\{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_{2N}\}$. Still, if the aforementioned conditions fail, one could add a small disturbance δ as in $\boldsymbol{J} + \delta\mathbb{I}$, where $\mathbb{I}$ is a conformal identity matrix. Then, a relative error e for $\|\Delta\boldsymbol{x}\|_2$ is achieved when $\delta = e/\|\boldsymbol{J}\|_2$ (see [Dem97, Chapter 2]).

Complexity: By omitting the slack node, we have $l = 2N - 2$. Here, we picked the l degrees of freedom offered by the LU decomposition to select $L_{ii} = 1$, which spares us l divisions (that would occur in line 10) and $(l^2 - l)/2$ multiplications (due to $i + 1$ in line 5). The initialization allows starting the iteration at $i = 2$ in line 1. Additionally, we separated the for-loops for $[U_{ij}]$ and $[L_{ji}]$ for reasons that become clear in Section 4.4.5. Now, line 10 and 11 require each $(l^2 - l)/2$ sequential multiplications while $1/[U_{ii}]$ can be reused.

Algorithm 1 Direct solver

Input: $[\boldsymbol{J}(\boldsymbol{x})]$ where $\boldsymbol{J} \in \mathbb{R}^{l \times l}$. **Outputs:** $[\boldsymbol{L}], [\boldsymbol{U}]$
Initialize $[\boldsymbol{U}] = ([\boldsymbol{J}_{1,:}] \ [\boldsymbol{0}_{l \times l-1}])$ and
$[\boldsymbol{L}] = ([\boldsymbol{J}_{:,1}]/[J_{1,1}] \ ([\boldsymbol{0}_{l-1 \times 1}] \ [\mathbb{I}_{l-1 \times l-1}])^\top)$

for $i = 2$ to l **do** ▷ LU decomposition
  for $j = i$ to l **do**
    $[U_{ij}] = [J_{ij}] - \sum_{k=1}^{i-1} [L_{ik}][U_{kj}]$
  end for
  for $j = i + 1$ to l **do**
    $[L_{ji}] = ([J_{ji}] - \sum_{k=1}^{i-1} [L_{jk}][U_{ki}])/[U_{ii}]$
  end for
end for
for $i = 1$ to l **do** ▷ forward-backward substitution
  $[\Delta \tilde{x}_i] = [F_i] - \sum_{j=1}^{i-1} [L_{ij}][\Delta \tilde{x}_j]$
end for
for $i = l$ to 1 **do**
  $[\Delta x_i] = ([\Delta \tilde{x}_i] - \sum_{j=i+1}^{l} [U_{ij}][\Delta x_j])/[U_{ii}]$
end for

Thus, Algorithm 1 requires l multiplicative inverses and at most $l^3/3 - l^2/2 + l/6$ multiplications. Further optimizations are possible if the sparsity structure of $\boldsymbol{J}$ is public. This enables to calculate which L_{ij}, U_{ij} will be zero beforehand such that corresponding multiplications can be omitted. Finally, it can sometimes be effective to reuse $\boldsymbol{LU} = \boldsymbol{J}(\boldsymbol{x}_{k-1})$ as a factorization for $\boldsymbol{J}(\boldsymbol{x}_k)$. Unfortunately, this turned out to be not expedient in our numerical experiments.

Indirect solver

In comparison to a direct solver, an iterative approach is utilized by indirect solvers that refine the solution. Here, we opted for GMRES which requires additions, multiplications, divisions, and square roots. However, we omit to present the pseudocode of our GMRES implementation here because the algorithm is significantly more tedious while an analysis similar to before can be applied. Nonetheless, the interested reader can find details in our code[2].
Importantly, the convergence speed of GMRES crucially depends on the condition of $\boldsymbol{J}$. Thus, a preconditioner $\boldsymbol{P}$ is typically used such that $\boldsymbol{PJ}\Delta\boldsymbol{x} = -\boldsymbol{PF}$ is solved instead of (4.2). However, if $\boldsymbol{P}$ is not specifically constructed, then $\boldsymbol{PJ}$ loses a lot of $\boldsymbol{J}$'s sparsity. In this context, we investigated different preconditioners during our numerical experiments in Section 4.6. Surprisingly, precomputing $\boldsymbol{P} = \boldsymbol{J}(\boldsymbol{x}_0)^{-1}$,

where $\boldsymbol{x}_0$ is selected close to the nominal operating conditions of the grid, and simply reusing $\boldsymbol{P}$ throughout all iterations led to the best results. Furthermore, warm-starting the GMRES with $\Delta\boldsymbol{x}_{k-1}$ in the k-th step was not advantageous in our experiments.

4.4.4 Line search

Lastly, we focus on the MPC realization of the line search procedure from Section 4.3.3, which is presented in Algorithm 2. Here, ℓ_{ls} is a predetermined number

Algorithm 2 Line search

Inputs: $[\boldsymbol{x}], [\Delta\boldsymbol{x}], [\boldsymbol{F}(0)]$ where $\boldsymbol{x}, \boldsymbol{F} \in \mathbb{R}^l$. **Output:** $[\boldsymbol{x}^\star_{k+1}]$
Initialize $j = 1$, $[\eta_1] = [0.95]$, $[\eta_0] = [1]$

1: $[x'_i] = [x_i] + [\eta_{j-1}][\Delta x_i]$ for all i
2: $[\boldsymbol{F}(\eta_{j-1})]$ by executing (4.10) and (4.11)
3: $[\xi_{j-1}] = \sum_{i=1}^{l} [F_i(\eta_{j-1})]\,([F_i(\eta_{j-1})] - [F_i(0)])$
4: **while** $j \leq \ell_{\text{ls}}$ **do**
5: $\quad [x'_i] = [x_i] + [\eta_j][\Delta x_i]$ for all i
6: $\quad [\boldsymbol{F}(\eta_j)]$ by executing (4.10) and (4.11)
7: $\quad [\xi_j] = \sum_{i=1}^{l} [F_i(\eta_j)]\,([F_i(\eta_j)] - [F_i(0)])$
8: $\quad [\Delta\eta_j] = [\Delta\eta_{j-1}][\eta_{j-1}][\xi_j]/([\eta_j][\xi_{j-1}] - [\xi_j])$
9: $\quad [\eta_{j+1}] = [\eta_j] + [\Delta\eta_j]$
10: $\quad j \leftarrow j + 1$
11: **end while**
12: $[x_{k+1,i}] = [x_i] + [\eta_{j+1}][\Delta x_i]$ for all i

of iterations. Note that large $\boldsymbol{F}(\boldsymbol{x})$ can lead to numerical issues in line 7 due to the fixed-point encoding. This issue is, however, easily addressed by replacing $\boldsymbol{F}(\boldsymbol{x})$ with $\epsilon\boldsymbol{F}(\boldsymbol{x})$, where ϵ is small and positive, which has no effect on line 8.

Complexity: Again, by omitting the slack node, we have $l = 2N - 2$. Observe that $\boldsymbol{F}(0)$ and for $j > 1$ also old $\boldsymbol{F}(\eta_j)$ can be reused. Then, the multiplication cost of one iteration is given by the cost of (4.10), (4.11) (detailed above) and $2l + 4$ (again with $l = 2N - 2$) multiplications. Additionally, one division in line 8 is needed.

4.4.5 Overview and communication

Finally, our privacy-preserving PFA denoted by Π_{PFA} is recapitulated in Algorithm 3 where ℓ is a predetermined bound on the number of Newton iterations required for convergence. This prevents correlating the number of iterations with the residuum $\|\boldsymbol{F}(\boldsymbol{x}_k)\|_2^2$.

Algorithm 3 Power flow analysis Π_{PFA}

Inputs: $[\boldsymbol{x}_0], [\boldsymbol{p}_0], [\boldsymbol{q}_0], [\boldsymbol{G}], [\boldsymbol{B}]$. **Output:** $[\boldsymbol{x}_{k+1}]$

Initialize $k = 1$

1: **while** $k \leq \ell$ **do**
2: $[\boldsymbol{J}(\boldsymbol{x}_k)]$, $[\boldsymbol{F}(\boldsymbol{x}_k)]$ by (4.10),(4.11), (4.12)
3: $[\Delta\boldsymbol{x}_k]$ via Algorithm 1 or GMRES
4: $[\boldsymbol{x}_{k+1}]$ by Algorithm 2
5: $k \leftarrow k + 1$
6: **end while**

Since a communication round takes at least 1 ms under realistic conditions, just naively computing operations sequentially will lead to inefficient implementations with high communication complexity. Thus, it is imperative to batch communication rounds as much as possible. Concretely, that means that we will delay communication until a point is reached where all local operations that do not depend on communication have been processed. Considering the aforementioned algorithms, the multiplications for (4.10) can be batched entirely. Then, $[\boldsymbol{F}(\boldsymbol{x})]$ and $[\boldsymbol{J}(\boldsymbol{x})]$ can be batched in two rounds after the execution of (4.10) which reduces to one round for public $\boldsymbol{G}, \boldsymbol{B}$. Furthermore, the multiplications in lines 3 and 6 of Algorithm 1 and the multiplications in Algorithm 2 can be batched into one and three rounds, respectively.

In addition, we need to optimize the latency of our connections. In our observation, the runtime of secure PFA grows in proportion to the *round-trip time* (RTT) of the connection between the participating parties. Thus, prosumers should be connected to each other via a LAN which makes RTTs of around 1 ms possible.

4.5 Security

Below, we describe threat models and provide a security analysis in the *universal composability* (UC) framework.

4.5.1 Threat model and assumptions

Since our algorithm can be used with various MPC protocols that provide differing security guarantees concerning the number of corrupted parties and adversarial behavior (as explained in Section 2.3), our algorithm can be flexibly employed for virtually all threat models. In Section 4.6.3 we provide benchmarks for example grids segmented according to security guarantees.

We assume that the computation is run between all prosumers $\mathsf{P}_1, \ldots, \mathsf{P}_n$ in a

given smart grid, and the inputs $p_{0,i}, q_{0,i}$ of each party P_i are considered secret. The grid operator is only involved in the computation insofar as it may provide $\boldsymbol{G}$ and $\boldsymbol{B}$. Moreover, we assume that the prosumers communicate exclusively via end-to-end encrypted and point-to-point authenticated channels during the computation. Some MPC protocols require correlated randomness, which can be provided by a trusted third party, or be generated interactively between the computing parties. Again, revealing the output of the PFA will also reveal the secret inputs. Hence, the output should be treated as an intermediary result to realize other privacy-preserving applications, e.g., in the area of flexibility markets, demand response, optimal power flow, or grid management.

4.5.2 Security analysis

Since we use MPC protocols that are already proven secure and are able to compute arbitrary functions, we inherit the security of these MPC protocols. We refer to Table 4.1 to give an overview of the protocols used, their concrete security guarantees, and references to their security proofs. All security proofs are in the UC

Table 4.1: Security guarantees of the MPC protocols used by us.

Protocol	Majority	Adversary	Security Type	Security Proof
Shamir	Honest	Semi-honest	Perfect	[AL17, Thm. 4.2]
Atlas	Honest	Semi-honest	Perfect	[GLO+21, Lem. 1]
Semi	Dishonest	Semi-honest	Computational	[KOS16b, Thm. 5]
Temi	Dishonest	Semi-honest	Computational	[DN03, Thm. 3]
Sy-Shamir	Honest	Malicious	Statistical	[CHI+23, Thm. 4.3]
Mascot	Dishonest	Malicious	Computational	[KOS16b, Thm. 5]
SPD$\mathbb{Z}_{2^k}$	Dishonest	Malicious	Computational	[CDE+18, Thm. 2]

framework [Can01]. In the following, we will explain how it applies to Algorithm 3. UC makes a distinction between the real world where a protocol Π is executed between the parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ (of which a subset is controlled by the adversary), and an ideal world, where a trusted party provides Π's *functionality* $\mathcal{F}$ on the inputs of $\mathsf{P}_1, \ldots, \mathsf{P}_n$. Π is considered secure, if an environment $\mathcal{Z}$ controlling the adversary, and providing the inputs to the computing parties, cannot distinguish whether it is interacting with the real world, or with the ideal world where a simulator $\mathcal{S}$ with access to $\mathcal{F}$ emulates an execution of Π. Formally:

Definition 11 (UC-security, Def. 4.19 in [CDN15])**.** *We say that a protocol* Π *securely realizes a functionality* $\mathcal{F}$ *if there exists an efficient simulator* $\mathcal{S}$ *such that:*

Perfect security: *No environment $\mathcal{Z}$ can distinguish whether it is interacting with Π in the real world, or with $\mathcal{S}$ in the ideal world.*

Statistical security: *No environment $\mathcal{Z}$ can distinguish whether it is interacting with Π in the real world, or with $\mathcal{S}$ in the ideal world, except with probability negligible in λ_1.*

Computational security: *No efficient environment $\mathcal{Z}$ can distinguish whether it is interacting with Π in the real world, or with $\mathcal{S}$ in the ideal world, except with probability negligible in λ_2.*

The advantage of protocols shown to be secure in UC is that they remain secure when combined with other UC-secure protocols, and under repeated interaction with multiple users. This is provided by the following theorem.

Theorem 3 (Composition Theorem, Thm. 4.20 in [CDN15])**.** *Let the composition of $\Pi_{\mathcal{F}}$ and $\mathcal{R}$ be a protocol instantiating a functionality $\mathcal{F}$ with perfect / statistical / computational security, and let $\Pi_{\mathcal{R}}$ be a protocol instantiating the functionality $\mathcal{R}$ with the same type of security. Then the composition of $\Pi_{\mathcal{F}}$ and $\Pi_{\mathcal{R}}$ securely realizes the functionality $\mathcal{F}$ with the same type of security.*

All MPC protocols we use to benchmark our privacy-preserving PFA provide a set of UC-secure arithmetic operations (called the *arithmetic black box* (ABB), originally introduced in [DN03]), that comprises secret sharing a number, addition, multiplication, and revealing of secret-shared numbers, as follows. Its ideal functionality $\mathcal{F}_{\mathsf{ABB}}$ is defined as follows.

Definition 12 ($\mathcal{F}_{\mathsf{ABB}}$, [KOS16b, Figure 7])**.**

Input: *On input $(\mathsf{Input}, \mathsf{P}_i, \mathsf{id}, x)$ from P_i and $(\mathsf{Input}, \mathsf{P}_i, \mathsf{id})$ from all other parties, with id a fresh identifier and $x \in \mathbb{Z}_p$, store (id, x).*

Add: *On command $(\mathsf{Add}, \mathsf{id}_1, \mathsf{id}_2, \mathsf{id}_3)$ from all parties (where $\mathsf{id}_1, \mathsf{id}_2$ are present in memory), retrieve (id_1, x), (id_2, y) and store $(\mathsf{id}_3, x + y)$.*

Multiply: *On command $(\mathsf{Mult}, \mathsf{id}_1, \mathsf{id}_2, \mathsf{id}_3)$ from all parties (where $\mathsf{id}_1, \mathsf{id}_2$ are present in memory), retrieve (id_1, x), (id_2, y) and store $(\mathsf{id}_3, x \cdot y)$.*

Output: *On input $(\mathsf{Output}, \mathsf{id})$ from all honest parties (where id is present in memory), retrieve (id, y) and output it to the adversary.*

Note that the ids in above definition are an artifact of the UC security proof. Since even complex operations such as comparisons, divisions, and square roots can be derived from the basic operations in the ABB [AS19], it is sufficient for implementing Algorithm 3. Hence, our privacy-preserving PFA inherits the UC-security of the MPC protocols we are using.

Definition 13 (Ideal functionality of power flow analysis $\mathcal{F}_{\mathsf{PFA}}$)**.** *Given a set of inputs $p_{0,i}, q_{0,i}$ by each party P_i, the ideal functionality $\mathcal{F}_{\mathsf{PFA}}$ outputs the state of the grid in terms of its voltages* $\mathbf{v}$*.*

Note that, as pointed out above, the output of $\mathcal{F}_{\mathsf{PFA}}$ should not be revealed, but rather be used by another secure algorithm as an intermediary result.

Theorem 4. *Let Π_{PFA} be the privacy-preserving PFA defined in Algorithm 3. Then Π_{PFA} realizes $\mathcal{F}_{\mathsf{PFA}}$ with {perfect/statistical/computational} security against a {semi-honest/malicious} adversary and a {honest/dishonest} majority of parties depending on the employed MPC protocol as specified in Table 4.1.*

Theorem 4 follows immediately from Theorem 3 since each operation from the ABB is UC-secure, and Π_{PFA} only uses operations from the ABB.

4.6 Performance Analysis

Below, we present our secure implementation and provide runtime benchmarks.

4.6.1 Example smart grids

For comparability, we selected two low-voltage grids provided by SimBench [M+20][1], namely a semi-urban and a rural network which consists of 44 and 18 buses with 39 and 13 prosumers, respectively. The former is depicted in Figure 4.1. We deliberately investigated the worst-case deviation from desired operating conditions. Consequently, the complexity of our computation may be viewed as an upper bound for PFA in the particular grids. For both grids, we initialize the solution of (4.1) via Newton iterations (4.2) with $\boldsymbol{x}_0 = (\mathbf{v}_R^\top, \mathbf{v}_I^\top)^\top = (230, \ldots, 230, 0, \ldots, 0)^\top$. The step size η_j is initialized via $\eta_0 = 1$ and $\eta_1 = 0.99$. During the computation, it is typically larger than 1 for the first iterations and then decays to 0. Compared to constant step sizes, our line search strategy reduced the required Newton iterations (4.2) by $20 - 30\%$ in the experiments. As already noted, the grids mainly differ in their size. The Jacobian $\boldsymbol{J}$ of the semi-urban network has the size 86×86, with 7% non-zero elements. The worst-case condition number of $\boldsymbol{J}(\boldsymbol{x}_k)$ is approximately 8000, while $\boldsymbol{P}(\boldsymbol{x}_0)\boldsymbol{J}(\boldsymbol{x}_k)$ has 40% non-zero elements and a condition of at most 30. Meanwhile, for the rural network, one finds that the Jacobian has the size 34×34, 17% non-zero elements, and a worst-case condition number of (again) 8000 while $\boldsymbol{P}(\boldsymbol{x}_0)\boldsymbol{J}(\boldsymbol{x}_k)$ has a density of 90% and a condition of at most 12.

[1] Datasets `1_LV_semiurb4_0_no_sw` and `1_LV_rural1_0_no_sw` at time steps 2740 and 13871 from https://simbench.de/de/datensaetze/.

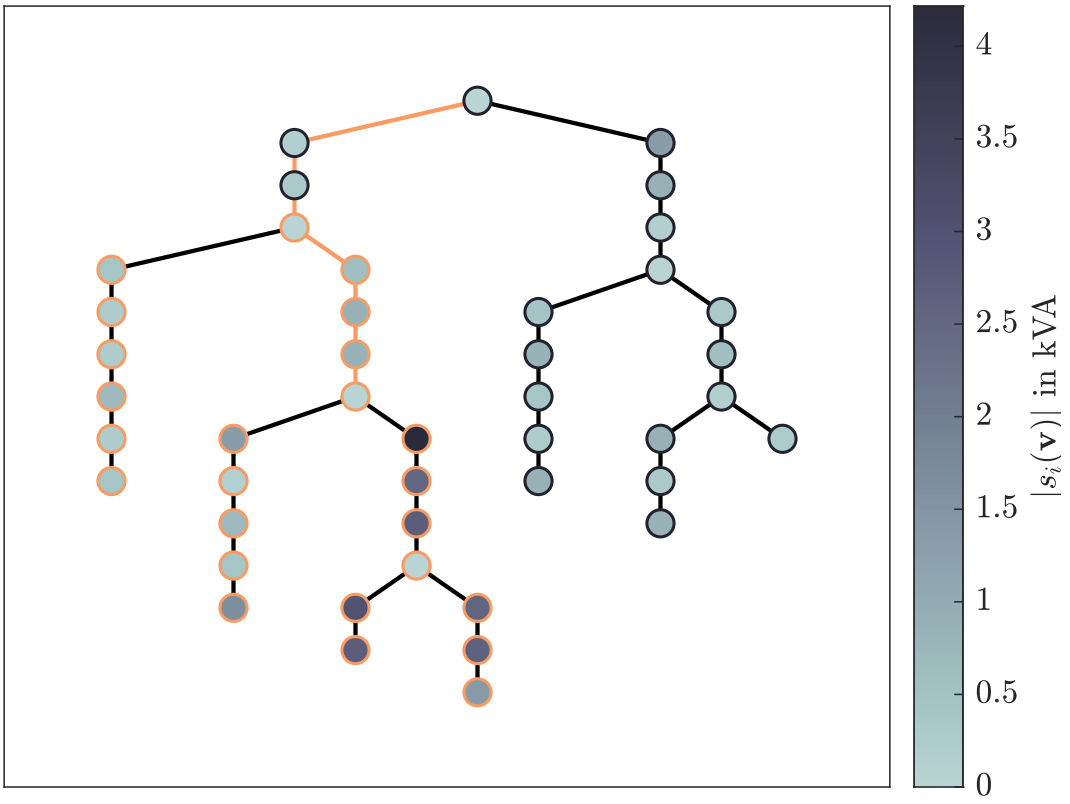

Figure 4.1: Semi-urban network topology with 44 nodes, of which 39 represent prosumers. The face color of the i-th node corresponds to the magnitude $|s_i(\mathbf{v})|$, whereas orange node outlines and edges indicate voltage and current constraint violations (4.3), respectively.

4.6.2 Implementation details

The code for our algorithms, including compilation instructions, can be found on GitHub[2] and in Chapter A. We make use of the framework *Multi-Protocol SPDZ* (MP-SPDZ) [Kel20] which allows benchmarking secure programs using a variety of MPC protocols based on secret sharing. MP-SPDZ also takes care of switching from arithmetic to binary representation when it is advantageous [EGK+20] and represents real numbers (as used in PFA) as members of a finite set (e.g., $\mathbb{Z}_p$) over which most secret sharing schemes are defined. Furthermore, it helps to batch the communication to reduce the total number of communication rounds. We set our parameters for computational and statistical security to commonly accepted standard values [AS19, Esc22]: If the security of an MPC protocol

[2]https://github.com/jvdheyden/privacy-preserving-power-flow

relies on computational assumptions, our computational security parameter is $\lambda_2 = 128$, meaning that an efficient adversary has a success probability smaller than 2^{-128}. Moreover, our statistical security parameter is $\lambda_1 = 40$, meaning that with probability less or equal to 2^{-40}, any adversary may learn something about the secret inputs. To allow for better performance, we limit the fixed-point precision to 64 bits, of which $h = 32$ bits are allocated to the fractional part. This implies $p > 2^{64}$, in order to avoid overflows. Usually higher precision would be necessary for the fault-free execution of our algorithms, but we enable lower precision by scaling numbers up or down in critical stages. We let each prosumer run in one virtual machine. To simulate real-world conditions, the virtual machines communicate with each other via point-to-point TCP connections that are encrypted and authenticated by TLS. Since RTTs higher than 10 ms render our solution inefficient for most use cases, we provide benchmarks for $\{1, 5, 10\}$ ms and simulate this latency between the virtual machines using Linux's traffic control (*tc*).

4.6.3 Benchmarks

This section provides benchmarks for our privacy-preserving implementation of PFA via LU decomposition and GMRES. Out of the multitude of possibilities for privacy-preserving computation in the smart grid context, we propose a few setups that promise to be both realistic in terms of grid architecture and efficient in terms of runtime. We benchmark runtime for the following parameters of interest:

- Offline vs. online phase: We give benchmarks for the full computation (meaning offline and online phase together) and also break out the online phase.
- Round Trip Time.
- Security model, meaning whether we assume semi-honest or malicious behavior of prosumers and whether we require an honest majority or allow a dishonest majority of prosumers.
- Number of computing parties in the grid.

Table 4.2 presents benchmarks for the online phase of a secure PFA. Here, we assume that the correlated randomness has been generated between the parties beforehand, or has been distributed by a trusted dealer. Table 4.3 presents benchmarks of securely running a PFA, including the secure interactive generation of correlated randomness.

We can see from the benchmark data that grid size and network latency have a significant impact on the runtime of our privacy-preserving PFA. If we assume that the prosumers are connected in a LAN via Ethernet, where RTTs of 1 ms are

Table 4.2: Runtime in seconds, online phase only (* fastest and † slowest)

	Semi-honest, honest majority[I]		Semi-honest, dishonest majority[II]		Malicious, dishonest majority[III]	
RTT (ms)	LU	GMRES	LU	GMRES	LU	GMRES
semi-urban grid with $n = 39$						
1	72	111	62*	114	450	199
5	2,227	3,549	280	535	2,701	1,073
10	2,227	3,624	566	1,075	3,993†	1,672
rural grid with $n = 13$						
1	27*	41	56	92	52	85
5	102	163	300	498	218	364
10	197	316	566	935†	420	700

[I]Based on BGW [AL17] ("Shamir"). [II]Semi-honest version of [KOS16b] ("Semi"). [III]Based on [CDE+18] ("SPD$\mathbb{Z}_{2^k}$").

realistic, runtimes of under 30 seconds (for the online phase) and four minutes (for the entire computation, i.e., preprocessing and online phase) are possible in small grids with 13 prosumers. If prosumers are expected to deviate from the protocol, security against malicious behavior is required. Here, runtimes of one minute and under three hours are possible in the online phase and for the entire computation, respectively. As we increase latency and grid size, our measured runtime increases so that in a grid with 39 prosumers online computation is finished in approximately two minutes, but preprocessing and online phase together take at least 70 minutes. As described in more detail below, the expensive preprocessing phase can, however, be omitted under certain assumptions.

4.6.4 Discussion

Many smart meters today communicate via unoptimized Ethernet, *broadband over power lines* (BPL) or LTE connections characterized by relatively high RTTs in the range of 20 ms to 150 ms, which would render our solution impractical. Assuming an optimized setup, however, RTTs of 1 ms for LAN-based connections, and 5 ms for WAN-based connections between smart meters would be possible. For a holistic assessment of practicality, one should also consider hardware constraints: We used a high-performance server for our benchmarks, on which 13 resp. 39 virtual machines (each representing one prosumer) shared 16 3.5 GHz *Ryzen 9* cores. Nevertheless, the use of more resource-constrained processors usually integrated into smart meters should not be a problem, at least in the online phase of protocols secure against semi-

Table 4.3: Runtime in seconds, offline and online phase combined

	Semi-honest, honest majority[I]		Semi-honest, dishonest majority[II]		Malicious, honest majority[III]	
RTT (ms)	LU	GMRES	LU	GMRES	LU	GMRES
semi-urban grid with $n = 39$						
1	2,584*	5,598		>24 hours		
5	54,880	>24 hours		>24 hours		
rural grid with $n = 13$						
1	234	232*	1,994	1,570	11,774	10,445
5	387	424	6,023	4,655	18,358	16,388
10	584	672	10,800	8,291	24,943†	22,356

[I]Based on [GLO+21] ("Atlas") in the semi-urban case and on BGW [AL17] ("Shamir") in the rural case. [II] Based on [DN03] and [KS22, Appendix B.1] using LWE ("Temi"). [III] Based on [CHI+23] ("Sy-Shamir").

honest adversaries. This is because in these cases, even with a short RTT of 1 ms, time spent on communication dominates CPU time by a factor of ten. With longer RTTs such as 10 ms, time spent on communication can dominate CPU time by a factor of 100. Moreover, RAM load was roughly 0.5 GB to 0.75 GB per prosumer during the online phase. While this is slightly more than typically available in smart meters, it should be noted that our protocols can still be optimized for memory usage. Moreover, the 0.5 GB includes system overhead, which is likely going to be much lower in the operating systems deployed in smart meters. To allow on-device preprocessing and protocols secure in the malicious model, however, hardware improvements to smart meters might be necessary. In addition, even given sufficiently powerful hardware, generation of correlated randomness might take too long to complete offline, especially in use cases with only a small idle period between computations.

For these use cases, it may make sense to outsource the generation and distribution of correlated randomness to a trusted dealer. As long as this dealer is oblivious to the communication transmitted during the online phase, it will learn nothing about the secret inputs. In extension, this means that we also have to assume that none of the prosumers is colluding with the dealer. We can reduce the trust required in the dealer by using a setup where the correlated randomness is generated interactively by two non-colluding dealers (e.g., the grid operator and a governmental regulator) such that both dealers only ever see partial shares ('shares of shares') of the correlated randomness assigned to a certain prosumer. Both dealers distribute their partial shares to the respective prosumers, who can then reconstruct their

full shares of the correlated randomness from them. This would allow outsourcing of the expensive preprocessing phase.

4.7 Conclusion and Outlook

In this work, we investigated the practicality of MPC techniques for privacy-preserving PFA in the context of smart grids. To this end, we devised a power flow solution algorithm and implemented a privacy-preserving PFA, and provided runtime benchmarks for various MPC protocols, grid parameters, and threat models. Further, we optimized the efficiency of our implementation through various means such as batching of communication rounds and enabling lower precision by dynamic rescaling.

Our method is particularly suited for market-based and preventive smart grid applications where prosumer data is used. In these use cases, PFA is performed as part of a grid state forecast to detect grid congestions at an early stage. To solve these grid congestions, market-based solutions like local flexibility markets [ATB17][EUM18], preventive control concepts, or dynamic grid charges are possible. Since the lead times of these solutions allow between fifteen minutes and one day for running PFA, our privacy-preserving solution is well-suited here. On the other hand, we have shorter lead times of ten seconds to one minute in real-time smart grid control systems, and currently, our implementation can only accommodate these runtimes for small grids with a short RTT.

Real-time smart grid applications might hence be addressed in future research. Due to the availability of better tooling, we focus on MPC via secret sharing in this work. However, it would also be possible to achieve MPC by means of *threshold fully homomorphic encryption* where the secret key is distributed among many parties such that the input of all of those parties is required for decryption. In this setting, all parties would encrypt their secret inputs and send the encryptions to a semi-honest server that will perform the computation. Once the function has been evaluated, all parties get the output ciphertext and partake in the decryption. The advantages of this approach are that the communication load is low and that the computational load can be securely outsourced to a high-performance server, potentially allowing for shorter runtimes.

Another direction for future research is to explore alternative methods for solving the power flow problem. In this context, the holomorphic embedding load flow method [Tri12] stands out due to its guaranteed convergence to the operable correct solution. Essentially, this is realized by replacing complex voltage variables with a holomorphic function, which results in a solution algorithm that may be well aligned with MPC protocols.

5 Rerandomizable Garbling Schemes, Revisited

Author's contribution. This chapter is based on joint work with Raphael Heitjohann and Tibor Jager published in [HvdHJ25]. The idea to utilize bilinear pairings to improve efficiency of KMHE was conceived by Raphael Heitjohann. The author of this thesis and Raphael Heitjohann discovered gaps in the proofs of the initial version of SCALES [AHKP22] and jointly devised new definitions and security notions for RGS. Raphael Heitjohann wrote the formal proofs for this work with some input from the author. The author of this thesis provided the concrete efficiency estimates for the new KMHE scheme and RGS.

5.1 Motivation and Overview

The BHHO encryption scheme [BHHO08] provides two very remarkable properties:

1. It achieves *circular security* in the standard model. Hence, it provides security even when a "key cycle" is encrypted, *i.e.*, secret key sk_i is encrypted under public key $\mathsf{pk}_{i \bmod n+1}$ for $i = 1, \ldots, n$.

2. Another distinguishing feature is that it is a *key-and-message homomorphic encryption* (KMHE) scheme. Thus, it enables efficiently computable homomorphisms in the key and the message space, such that in both spaces it is the *same* homomorphism. Furthermore, it provides *KMH rerandomizability*, which essentially guarantees that a rerandomized ciphertext is indistinguishable from a fresh one, even for the party that created the original ciphertext, and *key privacy under leakage*, which guarantees that this even holds when information about the rerandomization function is leaked.

The original work [BHHO08] mainly focused on circular security, but the unique combination of a key-and-message homomorphism with the aforementioned additional properties of the BHHO scheme has enabled many further, advanced applications, where circular security is not required. This includes *rerandomizable garbling schemes* (RGS) [GHV10], a rerandomizable version of Yao's garbled circuits[Yao86]. RGS enable applications such as i-hop homomorphic encryption

[GHV10], secure computation without simultaneous interaction [HLP11], combiners for indistinguishability obfuscation schemes [AJN+16], reusable garbled circuits [ZLXL23], and (outsourced) *multi-party computation* (MPC) [AHKP22, AHKP24] in the *you only speak once* (YOSO) paradigm [GHK+21].

Protocols in the YOSO paradigm are particularly interesting, as they allow for secure outsourced computation in the presence of a mobile adversary. Concretely, this means that lightweight input providers can outsource expensive MPC computations to a small committee of servers while maintaining security against a mobile adversary with corruption budget much larger than the size of the committee. This is due to the fact that servers are unknown until they speak once and then disappear again. Unfortunately, most known approaches to YOSO-MPC[GHK+21, KRY22, CDGK22] are still theoretical due to the use of expensive primitives, with the YOSO-like protocol *MPC with small clients and larger ephemeral servers* (SCALES) [AHKP22] being the closest to practical feasibility. Remarkably, SCALES also allows for dishonest majority MPC with round complexity independent of function depth, a feature previously only achieved by BMR- and FHE-based constructions [BMR90, LPSY15, HSS20, Sma23]. We observe that the main bottleneck preventing an efficient instantiation of SCALES is the BHHO encryption scheme. Hence, it is worthwhile to consider improvements and alternatives to BHHO.

Alternatives to and limitations of BHHO. Even though BHHO was published already about 17 years ago, it is currently still the *only* known encryption scheme that provides the required functional and security properties for RGS. The main limitation of BHHO is that encrypting κ bits in a way that allows for the necessary homomorphisms yields a ciphertext of length $\mathcal{O}(\kappa^2)$ group elements. Both encryption and decryption require $\mathcal{O}(\kappa^2)$ exponentiations. Since garbling a circuit C using RGS requires $\mathcal{O}(|C|)$ ciphertexts, it seems currently practically infeasible to instantiate and use RGS based on BHHO in applications, even for simple circuits. In this chapter, we describe a KHME scheme that improves space and time complexity in comparison to BHHO by almost 4 orders of magnitude, while preserving properties essential for RGS (homomorphism, rerandomizability) and losing non-essential ones (circular security). We therefore claim to enable the first practical YOSO-like MPC construction for small circuits, for example computing the maximum. See Table 5.1 for detailed performance numbers.

Our contributions. We summarize our contributions as follows:

Basic KMHE. In Section 5.3, we describe our new "basic" KMHE scheme, which can be seen as a pairing-based analogue of [BHHO08]. It reduces the ciphertext size and the computational costs both asymptotically and concretely

very significantly. A ciphertext of our scheme consists of a *linear* (in the security parameter) number of group elements, as opposed to a *quadratic* number in BHHO. Concretely, when instantiated with the BLS12-381 pairing, a ciphertext of our scheme has size 360 kB, while (expanded) BHHO instantiated with the high-performance Curve25519 [Ber06] requires 16.68 MB. This is an improvement by 97.83 %. Using the RELIC toolkit [AGM+] to estimate the number of clock cycles, an encryption takes about 6.10×10^8 clock cycles (about 0.01 seconds with a parallelized implementation on a processor with 16 cores running at 4 Ghz), while BHHO takes 1.6×10^{13} clock cycles (4 minutes), which is an improvement by 99.96 %.

This scheme not only serves to explain the main idea and new techniques underlying our subsequent, more complex constructions, but we consider it also of independent interest. It might prove useful for applications that require a practical encryption scheme that provides the same homomorphism in the key- and the message space, but not a full-fledged RGS scheme.

Gate KMHE. In Section 5.4 we introduce the new notion of a *gate KMHE* scheme. Essentially, a gate KMHE scheme adopts the above "basic" KMHE scheme to the use in a (rerandomizable) garbled circuit, following Yao's seminal construction [Yao86]. In addition to the trivial syntactic changes (e.g., the encryption now takes four keys and messages as input, as required to garble a gate of a given circuit), we also define and prove several new, non-trivial functional and security properties that are required to build *rerandomizable* garbled circuits from gate KMHE. The main advantage of using gate KMHE instead of the *sharable KMHE* from [AHKP22] is improved performance. While RGS from sharable KMHE requires eight ciphertexts per garbled gate, gate KMHE uses double-key encryption to reduce this to four ciphertexts, and enables randomness reuse to decrease the number of group elements used in each ciphertext even further.

Comparing the concrete space and time improvements of Gate KMHE to [AHKP22], for the same setting as considered above we achieve a garbled gate size of 1.35 MB, where [AHKP22] requires 133.43 MB (98.99 % improvement), and garbling a gate is estimated to take 0.04 seconds (33 minutes for [AHKP22], 99.996 % improvement).

RGS from Gate KMHE. In Section 5.5 we show how to build RGS from gate KMHE. The construction follows the approaches from [GHV10, AHKP22], which in turn are based on [Yao86].

We have identified some small theoretical issues with the RGS construction in [AHKP22]: In particular, their Rerand privacy was not sufficient to achieve

incremental Decomposable Randomized Encoding (iDRE, a construction leading directly towards *SCALES*) privacy (for details see Chapter B). Acharya et al. acknowledged the existence of this gap and accordingly updated their paper with strengthened KMH privacy and Rerand privacy properties. In addition, they switched their KMHE from the plain version of BHHO to the expanded version to exhibit perfect rerandomizability. This allows their KMHE and RGS schemes to fulfill the stronger versions of KMH privacy and Rerand privacy, but comes at the cost of decreased efficiency: In comparison to their old scheme, the new KMHE scheme's ciphertext is κ group elements larger and encryption is slower by a multiplicative factor of κ. As our pairing-based KMHE scheme is only computationally rerandomizable, we take a different approach to fixing the original gap by defining an extended variant of Acharya et al.'s original computational Rerand privacy notion. We additionally introduce new notions of *KMH rerandomizability* and *key privacy under leakage*. Overall, these changes allow for the use of our pairing-based KMHE, facilitating significant reductions in space and runtime complexity. We remark that our variant of RGS can be also used for the maliciously-secure version of SCALES [AHKP24].

Improvements in space and runtime complexity As shown in Table 5.1, our work significantly reduces space and runtime complexity of RGS by four orders of magnitude. Garbling a 966 gate circuit computing the maximum of eight 8-bit numbers yields a RGS circuit size of 1.27 GB (125.87 GB in [AHKP22]) and is estimated to take 37 seconds (22 days in [AHKP22]). Garbling an AES-128 circuit with 34,576 gates requires an RGS circuit size of 45.60 GB (4.4 TB in [AHKP22]) and is estimated to take 20 minutes (2 years in [AHKP22]). Thus, the space requirements are improved by 98.99 % and the computational requirements even by 99.998 %. Two key aspects further highlight the significance of these optimizations:

1. Our improvements are particularly impactful in protocols requiring multiple rerandomizations of a garbling, especially when these garblings must be exchanged among multiple parties, as in the SCALES protocol from [AHKP22].
2. While our performance analysis assumes computation on 16 4-GHz CPUs, both garbling and rerandomization are parallelizable on up to $\kappa = 526$ cores. Hence, on high-performance servers with $16 \leq n \leq \kappa$ cores, runtimes are even lower than those shown in Table 5.1.

Hence, while RGS and the SCALES protocol remain relatively expensive primitives, we demonstrate for the first time that this approach is within reach of practical feasibility, particularly for simple circuits.

Table 5.1: Detailed comparison of the RGS from SCALES [AHKP22] and our work in bytes and clock cycles (cc). We set security parameter $\lambda = 128$ and $\kappa \geq \log q + 2\lambda + 3/2 \log \kappa$, where q refers either to the group order of $\mathbb{H}$ or $\mathbb{G}_{25519}$. c, Encryption (Enc.) and Evaluation (Eval.) refer to sharable KMHE in [AHKP22, Construction 3] and our basic KMHE (Construction 1), respectively.

			[AHKP22]	Our work	Improvement
Curve			Curve25519	BLS12-381	
κ			522	526	
$\lvert\mathsf{c}\rvert$	# Elems.	in $\mathbb{G}_{25519}$	$2\kappa^2 + 3\kappa + 1$	0	
		in $\mathbb{G}$	0	κ	
		in $\mathbb{H}$	0	κ	
		in $\mathbb{G}_T$	0	$\kappa + 1$	
	Size		16.68 MB	0.36 MB	97.83 %
Size of one garbled gate			133.43 MB	1.35 MB	98.99 %
Size of one garbled Max circuit			125.87 GB	1.27 GB	98.99 %
Size of one garbled Mult circuit			1.74 TB	18.04 GB	98.99 %
Size of one garbled AES circuit			4.4 TB	45.60 GB	98.99 %
Enc.	# Expon.	in $\mathbb{G}_{25519}$	$\kappa^3 + \kappa^2$	0	
		in $\mathbb{G}$	0	1	
		in $\mathbb{H}$	0	κ	
		in $\mathbb{G}_T$	0	κ	
	Clock cycles		1.6×2^{13}(4 min)	6.10×2^{8}(0.01 s)	99.996 %
Garbling one gate (cc)			1.28×2^{14}(33 min)	2.44×2^{9}(0.04 s)	99.998 %
Garbling a Max circuit (cc)			1.24×2^{17}(22 d)	2.36×2^{12}(37 s)	99.998 %
Garbling a Mult circuit (cc)			1.75×2^{18}(317 d)	3.33×2^{13}(9 min)	99.998 %
Garbling an AES circuit (cc)			4.43×2^{18}(2 yr)	8.43×2^{13}(20 min)	99.998 %
Eval.	# Expon.	in $\mathbb{G}_{25519}$	$4\kappa^3 + 7\kappa^2 + 4\kappa + 1$	0	
		in $\mathbb{G}$	0	κ	
		in $\mathbb{H}$	0	2κ	
		in $\mathbb{G}_T$	0	$\kappa + 1$	
	Clock cycles		6.41×2^{13}(17 min)	9.64×2^{8}(0.02 s)	99.998 %
Rerand. a garbled gate (cc)			5.13×2^{14} (2.23 h)	3.35×2^{9}(0.05 s)	99.9993 %
Rerand. a garbl. Max circuit (cc)			4.95×2^{17}(90 d)	3.24×2^{12}(51 s)	99.9993 %
Rerand. a garbl. Mult circuit (cc)			7.01×2^{18}(3.5 yr)	4.58×2^{13}(11 min)	99.9993 %
Rerand. a garbl. AES circuit (cc)			1.77×2^{19}(9 yr)	1.16×2^{14}(30 min)	99.9993 %

Methodology. Table 5.1 shows complexity estimates comparing the RGS from SCALES and our work in terms of size and performance. We instantiate [AHKP22] with the high-performance Curve25519 [Ber06], denoting the group with $\mathbb{G}_{25519}$. Our protocols are instantiated with BLS12-381, denoting the source groups of

the pairing with $\mathbb{G}$ and $\mathbb{H}$, where $\mathbb{H}$ is the group with a smaller representation of elements, and the target group with $\mathbb{G}_T$. Hence, group elements in $\mathbb{G}_{25519}, \mathbb{G}, \mathbb{H}$ and $\mathbb{G}_T$ can be represented by 256, 768, 384 and 4608 bits, respectively. Using the benchmarking toolkit RELIC [AGM+] on a 2019 AMD Ryzen 9 3950X with 16 times 4.7 GHz, the number of clock cycles (cc) we obtained for exponentiations in $\mathbb{G}_{25519}, \mathbb{G}, \mathbb{H}$ and $\mathbb{G}_T$ are 112,269, 481,170, 196,432 and 957,284, respectively. Additionally, we measured 2,313,901 clock cycles for a BLS12-381 pairing. We then estimate concrete runtimes in seconds (s), minutes (min) and hours (h) by extrapolating the clock cycles, assuming a processor with 16 cores each running at 4 GHz. However, garbling and rerandomization are actually parallelizable to up to κ cores. Hence, runtime could be even lower than shown in Table 5.1. The Max, Mult and AES circuits refers to Boolean circuits comprising 966 gates computing the max of eight 8 bit numbers, 13,675 gates computing a 64 bit multiplication, and 34,576 gates computing an AES-128 encryption, respectively.

Related work. Gentry et al. [GHV10] were the first to show that [BHHO08] can be used to build RGS. Acharya et al. [AHKP22] built the SCALES protocol from RGS, and also pointed out a gap in [GHV10] by demonstrating that rerandomizability cannot be reduced to semantic security and key leakage resilience alone, introduced KMH privacy as a stronger property, and proved it for BHHO. In a recent follow-up work, Acharya et al. [AHKP24] then extend SCALES to a dishonest majority of input providers and servers in the malicious model, respectively.

Technical overview. In the following, we give a technical overview about how our novel KMHE and RGS constructions allow for the aforementioned efficiency improvements.

There are three aspects to consider when designing a KMHE scheme suitable for constructing an RGS: *Firstly*, for the scheme to be able to encrypt its own secret keys, the key space must be a subset of the message space. *Secondly*, we must be able to apply the *same* homomorphism to both the secret key and the message. *Finally*, the homomorphism must be able to transform the key in such a way that a transformed key is indistinguishable from a fresh one, even given some information about the homomorphism. We call this property *key privacy under leakage*.

Key-and-message homomorphism. We will now take the example of the ElGamal scheme and explore how it could be adapted to these requirements. A secret key is $x \leftarrow\!\!\$\, \mathbb{Z}_q$, the corresponding public key is g^x where $g \in \mathbb{G}$ is a generator in a finite, cyclic group $\mathbb{G}$ of prime order q. To encrypt a message $m \in \mathbb{G}$, encryption samples $a \leftarrow\!\!\$\, \mathbb{Z}_q$, and outputs the ciphertext $(g^a, g^{xa} \cdot m)$. However, the message m is now an element of the group $\mathbb{G}$ while the key x is an element in the exponent field $\mathbb{Z}_q$,

violating our first KMHE requirement. A common solution is to instead encode the message as an exponent in $\mathbb{Z}_q$. Encryption of such a $m \in \mathbb{Z}_q$ is then done by sampling $a \leftarrow\!\!\$\, \mathbb{Z}_q$, and outputting the ciphertext $(c_1, c_2) = (g^a, g^{xa} \cdot g^m)$. Now the key space and message space are both $\mathbb{Z}_q$. This also addresses the homomorphism and rerandomization requirements: We can add an offset $\sigma \in \mathbb{Z}_q$ to the secret key by computing $c_2 \cdot c_1^{\sigma} = g^{(x+\sigma)a} \cdot g^m$. Then $x + \sigma$ looks like a fresh uniform secret key. The message homomorphism works via $c_2 \cdot g^{\tau} = g^{x\tau} \cdot g^{m+\tau}$ for any $\tau \in \mathbb{Z}_q$.

However, note that now decryption requires computing the discrete log of $g^{m+\tau}$; this only works if the message is fairly small. Since we want to encrypt and rerandomize secret keys, which are uniform over $\mathbb{Z}_q$, the above approach is not feasible.

Key privacy under leakage. There is a second reason why the construction with the $(\mathbb{Z}_q, +)$ homomorphism will not work: For RGS, our KMHE also needs to provide key privacy under leakage (previously part of the *KMH privacy* property from [AHKP22]), which informally means the following:

We sample two random secret keys sk^1 and sk^2 and a random key transformation function σ (in the previous ElGamal example this was the offset $\sigma \in \mathbb{Z}_q$, implying a function $m \mapsto m + \sigma$). Given sk^1, sk^2, and $\sigma(\mathsf{sk}^2)$, the value $\sigma(\mathsf{sk}^1)$ must still have high average-min entropy, i.e. look random to a distinguisher. This can only work if the values sk^2 and $\sigma(\mathsf{sk}^2)$ leak very little information about the function σ. In our ElGamal example, sk^2 would correspond to $x_2 \leftarrow\!\!\$\, \mathbb{Z}_q$ and $\sigma(\mathsf{sk}_2) = x_2 + \sigma$. Clearly, given both these values, an adversary can recover σ and then the value $\sigma(\mathsf{sk}^1) = x_1 + \sigma$ has no entropy at all.

The solution to this, introduced by [GHV10], is to use a permutation homomorphism over the key space $HW_{\kappa,\kappa/2}$, consisting of all bit-strings of length κ with Hamming weight $\kappa/2$ (meaning exactly half the bits are zeroes and half are ones). As shown in [GHV10, Lemma 9], obtaining the input $\mathsf{sk} \in HW_{\kappa,\kappa/2}$ and output $\sigma(\mathsf{sk})$ for some permutation σ leaks very little information about the permutation itself. Intuitively, this is because for each input sk and output sk', there are many permutations that could map sk to sk'.

A permutation homomorphism, however, decreases efficiency of the scheme significantly: A message $m \in HW_{\kappa,\kappa/2} \subset \{0,1\}^{\kappa}$ must be encrypted bit-wise, leading to a $O(\kappa) = O(\lambda)$ multiplicative blowup in ciphertext size compared to regular ElGamal.

We also need the ability to support the same permutation homomorphism for the secret key. Here, the BHHO scheme [BHHO08] comes into play. A BHHO public key consists of bases $g_1, \ldots, g_\kappa \in \mathbb{G}^\kappa$ and the "hash" $\prod_{j \in \kappa} g_j^{\mathsf{sk}_j}$. Thanks to the left-over hash lemma [HILL99], as long as sk has sufficient min-entropy, $\prod_{j \in [\kappa]} g_j^{\mathsf{sk}_j}$ looks like a random group element g^x, so security of the scheme follows

from *decisional Diffie-Hellman* (DDH). To encrypt a message $m \in \{0,1\}^\kappa$, each message bit m_i is encrypted as $(g_1^{a_i}, \ldots, g_\kappa^{a_i}, \prod_{j\in\kappa} g_j^{a_i \mathsf{sk}_j} \cdot g^{m_i})$, where $a_i \leftarrow\!\!\$\, \mathbb{Z}_q$. The key permutation itself can then be applied by first permuting $g_1, \ldots, g_\kappa$, and then rerandomizing the public key and the ciphertext. BHHO public keys can be easily rerandomized by exponentiating all elements in the public key with a random $r \leftarrow\!\!\$\, \mathbb{Z}_q$. The ciphertext elements can be rerandomized analoguosly.

Improving efficiency using pairings. We will now examine the efficiency of the BHHO scheme as presented in [GHV10, AHKP22] and then outline our idea to improve space and computation complexity quadratically. Since each message bit adds $\kappa + 1$ group elements to the ciphertext, for κ message bits, we require $\kappa^2 + \kappa$ many group elements. The quadratic runtime and space requirement is due to the fact that the bases $g_1, \ldots, g_\kappa$ have to be combined with the randomness a_i of each message bit m_i, resulting in elements $g_1^{a_i}, \ldots, g_\kappa^{a_i}$ for $i \in [\kappa]$. To avoid this blowup one could separate the $g_1, \ldots, g_\kappa$ and $a_1, \ldots, a_\kappa$ and give them out separately so that the combinations can later be computed dynamically during decryption. While revealing $a_1, \ldots, a_\kappa$ in plain is clearly not secure, a bilinear pairing $\mathrm{e} : \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$ allows us to compute g_T^{ab} given just g^a and h^b, where $g \in \mathbb{G}$, $h \in \mathbb{H}$, and $a, b \in \mathbb{Z}_q$. We can therefore encode the randomness $a_1, \ldots, a_\kappa$ as group elements $h^{a_1}, \ldots, h^{a_\kappa}$ and use the pairing to compute $\mathrm{e}(g_j, h^{a_i}) = \mathrm{e}(g_j^{a_i}, h)$ for any combination $i, j \in [\kappa]$. This allows us to compute a ciphertext as $c_i = (h^{a_i}, g_T^{m_i} \cdot y^{a_i}) \quad \forall i \in [\kappa]$ for $y = \mathrm{e}\left(\prod_{j\in[\kappa]} g_j^{\mathsf{sk}_j}, h\right)$, resulting in the aforementioned quadratic reductions in space and time complexity.

Further efficiency improvements through gate KMHE. In order to utilize a KMHE scheme as part of a garbling scheme, one needs a variant of KMHE that requires *two* keys to decrypt any ciphertext. To this end, Acharya et al. [AHKP22] introduced *sharable KMHE* which solves this problem by 2-out-of-2-secret-sharing the message, and encrypting each share using a single-key KMHE scheme. This approach results in two full ciphertexts per message; for a Boolean gate with two input wires, we thus need eight ciphertexts per garbled gate. We introduce *gate KMHE* as a way to improve upon the performance and space requirements of sharable KMHE, making use of the specific algebraic properties of our KMHE scheme. We will now take a closer look at the techniques used to achieve these improvements. Recall that we previously encrypted a message $m \in \{0,1\}^\kappa$ as $c_i = (h^{a_i}, g_T^{m_i} \cdot y^{a_i}) \quad \forall i \in [\kappa]$, where $y = \mathrm{e}\left(\prod_{j\in[\kappa]} g_j^{\mathsf{sk}_j}, h\right)$. To halve the number of ciphertexts per gate, we do not secret-share the message, but instead use "double encryption" and encrypt a single ciphertext using both secret keys sk^0 and sk^1. The functionality remains the same: Both keys are necessary to obtain the full

message m. The resulting ciphertext is given by $c_i = (h^{a_i}, g_T^{m_i} \cdot y^{a_i}) \quad \forall i \in [\kappa]$, where $y = \mathrm{e}\Big(\prod_{j\in[\kappa]} (g_{0,j}^{\mathsf{sk}_j^0} \cdot g_{1,j}^{\mathsf{sk}_j^1}), h\Big)$.

We further halve the number of elements in $\mathbb{G}$ required per garbled gate by using only two sets of bases $g_1, \ldots, g_\kappa$ for four ciphertexts. This works as follows: Recall that a garbled gate uses four secret keys $\mathsf{sk}^{\alpha\beta}$, for $\alpha, \beta \in \{0,1\}$, two per input wire. Using double encryption, a garbled gate can then be visualized as

$$\begin{pmatrix} \mathsf{Enc}(\mathsf{sk}^{00}, \mathsf{sk}^{10}, m^1) \\ \mathsf{Enc}(\mathsf{sk}^{00}, \mathsf{sk}^{11}, m^2) \\ \mathsf{Enc}(\mathsf{sk}^{01}, \mathsf{sk}^{10}, m^3) \\ \mathsf{Enc}(\mathsf{sk}^{01}, \mathsf{sk}^{11}, m^4) \end{pmatrix}$$

To construct an RGS, we only need to be able to apply the same key homomorphism σ_0 to the "left" keys sk^{00} and sk^{01}, and similarly for σ_1 and the "right" keys sk^{10} and sk^{11}. Hence, we only need two sets of bases, a set $g_{0,1}, \ldots, g_{0,\kappa}$ to apply σ_0, and a set $g_{1,1}, \ldots, g_{1,\kappa}$ to apply σ_1. To make it possible to actually share these random bases across the different ciphertexts, we introduce the gate KMHE primitive, which internally creates these four double encryptions while enabling a reuse of these left and right bases.

5.2 Bilinear Groups

Let $\mathcal{G}(1^\lambda)$ be a probabilistic algorithm that, on input the security parameter λ, produces a bilinear group description $(q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathrm{e})$. This description satisfies the following requirements:

– The structures $\mathbb{G}$, $\mathbb{H}$, and $\mathbb{G}_T$ form groups of prime order q, with respective generators g, h, and g_T.

– The mapping $\mathrm{e} : \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$ provides efficient computation while satisfying bilinearity—that is, $\mathrm{e}(X^a, Y^b) = \mathrm{e}(X, Y)^{ab}$ for all $X \in \mathbb{G}$, $Y \in \mathbb{H}$ and $a, b \in \mathbb{Z}_q$—along with non-degeneracy, meaning $\mathrm{e}(g, h) \neq 1_{\mathbb{G}_T}$. We employ multiplicative notation throughout for all groups.

We establish several computational assumptions for these groups along with supporting lemmas.

Definition 14 (Decisional Diffie-Hellman)**.** *For a group $\mathbb{G}$ of prime order q, the decisional Diffie-Hellman assumption asserts that*

$$\{(g_1, g_2, g_1^r, g_2^r) : g_1, g_2 \leftarrow\!\$\, \mathbb{G}, r \leftarrow\!\$\, \mathbb{Z}_q^*\} \overset{c}{\approx} \{(g_1, g_2, v_1, v_2) : g_1, g_2, v_1, v_2 \leftarrow\!\$\, \mathbb{G}\}.$$

The subsequent lemma extends the standard DDH assumption to multiple generators, following the work of Naor and Reingold [NR97]. This extension follows from DDH with tight security reduction, as demonstrated in Lemma 4.2 of [NR97].

Lemma 5 (Generalized DDH)**.** *Let $\mathbb{G}$ be a group with prime order q. Assuming the decisional Diffie-Hellman assumption holds for $\mathbb{G}$, we have that*

$$\left\{(g_1,\ldots,g_\rho,g_1^r,\ldots,g_\rho^r) : g_1,\ldots,g_\rho \leftarrow\!\!\$\ \mathbb{G}, r \leftarrow\!\!\$\ \mathbb{Z}_q^*\right\} \\ \overset{c}{\approx} \left\{(g_1,\ldots,g_\rho,v_1,\ldots,v_\rho,) :\ g_1,\ldots,g_\rho,v_1,\ldots,v_\rho \leftarrow\!\!\$\ \mathbb{G}\ \right\},$$

where $\rho = \rho(\lambda)$ is a polynomial of constant degree.

The following lemma, a special case of Lemma 1 in [BHHO08], will also be needed. It is implied by DDH with logarithmic reduction loss [Vil12].

Lemma 6 (Matrix DDH)**.** *Let $\eta, \nu \in \mathbb{N}$ and let $\mathbb{G}$ be a group with prime order q. Then, assuming decisional Diffie-Hellman assumption holds for $\mathbb{G}$, it holds that*

$$\mathbf{G} \overset{c}{\approx} \mathbf{G}',$$

where $\mathbf{G} \leftarrow\!\!\$\ \mathbb{G}^{\eta\times\nu}$ is uniform and $\mathbf{G}' \leftarrow\!\!\$\ \mathrm{rank}_1(\mathbb{G}^{\eta\times\nu})$ is a uniform rank-1 matrix.

The *symmetric external Diffie-Hellman* (SXDH) assumption extends DDH to both source groups $\mathbb{G}$ and $\mathbb{H}$ within bilinear group constructions.

Definition 15 (Symmetric External Diffie-Hellman (SXDH))**.** *For any security parameter $\lambda \in \mathbb{N}$ and bilinear group description $(p, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathrm{e}) \leftarrow \mathcal{G}(1^\lambda)$, the symmetric external Diffie-Hellman assumption requires that decisional Diffie-Hellman holds in both groups $\mathbb{G}$ and $\mathbb{H}$.*

We also formulate a SXDH-analogue of Theorem 5 for convenience, it is straightforwardly implied by the hardness of DDH in the source groups of the bilinear group setting.

Corollary 7 (Generalized SXDH)**.** Assuming the symmetric external Diffie-Hellman assumption holds for $\mathcal{G}$, then generalized DDH (Theorem 5) holds for groups $\mathbb{G}$ and $\mathbb{H}$, respectively.

5.3 Basic KMHE Scheme

In the following section we describe our basic KMHE scheme. This scheme alone will not yet be sufficient to construct an RGS, and we will describe an extension to a *gate KMHE* variant in Section 5.4, which is tailored to the use in garbled circuits.

However, the basic scheme illustrates the main idea of the construction, and helps to explain the proof techniques used for the gate KMHE in a simpler setting. We also rely on it for our performance comparison (Table 5.1) with the BHHO-based KMHE scheme from [AHKP22].

A noteworthy aspect of our basic KMHE scheme is that it can be viewed as a batched version of BHHO, allowing for a more compact encryption of multiple messages. If we encrypt ℓ messages using BHHO with a key of length κ, then we get a ciphertext containing $\mathcal{O}(\ell\kappa)$ group elements, while our scheme only requires $\mathcal{O}(\ell + \kappa)$ group elements, with significantly better concrete efficiency.

Furthermore, our scheme inherits the leakage resilience of BHHO, and thus could potentially also be used in constructions where BHHO is used as a leakage-resilient scheme, e.g. leakage-resilient CCA-secure PKE [FKMV12] or tamper-resilient PKE [DFMV17].

However, in our opinion the unique feature of BHHO and our scheme is that it provides the same homomorphism in the key and message space. RGS are a prime application that use this feature, and we are currently not aware of others. But we believe that our work might also pave the way towards more practical applications that exploit this unique feature.

We proceed by defining the syntax of KMHE, adopted from [AHKP22]. We then continue with our construction.

Definition 16 (Key-and-Message-Homomorphic Encryption)**.** *A KMHE scheme is a tuple of PPT algorithms* $\mathsf{KMHE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *with the following syntax.*

$\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$**:** *This algorithm takes as input the security parameter* λ *and returns public parameters* p*. The public parameters define a key space* $\mathcal{K}$*, a message space* $\mathcal{M}$*, and a ciphertext space* $\mathcal{C}$*.*

$\mathsf{sk} \leftarrow \mathsf{KGen}(\mathsf{p})$**:** *This algorithm takes* p *and returns a secret key* $\mathsf{sk} \in \mathcal{K}$*.*

$\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{p}, \mathsf{sk}, m)$**:** *Given a message* $m \in \mathcal{M}$ *and a secret key* sk*, this algorithm returns a ciphertext* $\mathsf{c} \in \mathcal{C}$*.*

$\mathsf{c}' \leftarrow \mathsf{Eval}(\mathsf{p}, \sigma, \tau, \mathsf{c})$**:** *Given a ciphertext* $\mathsf{c} \in \mathcal{C}$ *and two functions* $\sigma, \tau \in \mathcal{F}$*, where the set* $\mathcal{F}$ *is defined by* λ*, this algorithm returns a ciphertext* c'*.*

$m' \leftarrow \mathsf{Dec}(\mathsf{p}, \mathsf{sk}, \mathsf{c})$**:** *This algorithm takes as input* p *and* sk*, and returns a message* $m' \in \mathcal{M}$*.*

By R_{Enc} and R_{Eval} we denote the randomness spaces for Enc and Eval, respectively. Unless specified otherwise, randomness is always assumed to be sampled uniformly at random.

Construction 1 (Basic KMHE scheme)**.** Our basic KMHE scheme consists of the following algorithms:

$\mathsf{Setup}(1^\lambda)$**:** Generate and output the description of a bilinear group

$$\mathsf{p} = (q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda).$$

We define κ as the smallest positive and even integer such that $\kappa - \frac{1}{2}\log\kappa - 1/2 \geq \log q + 2\lambda$. These parameters define the following sets:

- the message space as $\mathcal{M} = \{0,1\}^\kappa$.
- the ciphertext space as $\mathcal{C} = (\mathbb{H} \times \mathbb{G})^\kappa \times \mathbb{G}^\kappa \times \mathbb{G}_T$.
- the key space as $\mathcal{K} = HW_{\kappa,\kappa/2}$, the set of bit strings of length κ with Hamming weight $\kappa/2$. Note that we have $\mathcal{K} \subset \mathcal{M}$, i.e. all keys are in the message space and thus the scheme is able to encrypt its own keys.
- the set of key- and message-homomorphisms as $\mathcal{F} = S_\kappa$, the set of permutations over κ elements,
- and randomness spaces $R_{\mathsf{Enc}} = \mathbb{G}^\kappa \times \mathbb{Z}_q^\kappa$ and $R_{\mathsf{Eval}} = \mathbb{Z}_q^\kappa \times \mathbb{Z}_q^*$.

$\mathsf{KGen}(\mathsf{p})$**:** Sample and output $\mathsf{sk} \leftarrow\!\!\$\ \mathcal{K}$.

$\mathsf{Enc}(\mathsf{p}, \mathsf{sk}, m; r)$**:** Parse $m = (m_1, \ldots, m_\kappa) \in \{0,1\}^\kappa$, $\mathsf{sk} = (s_1, \ldots, s_\kappa) \in \{0,1\}^\kappa$, and $r = (g_1, \ldots, g_\kappa, a_1, \ldots, a_\kappa) \in R_{\mathsf{Enc}}$. Then compute

$$y = \mathrm{e}\left(\prod_{j\in[\kappa]} g_j^{s_j}, h\right),$$

and

$$c_i = (h^{a_i}, g_T^{m_i} \cdot y^{a_i}) \quad \forall i \in [\kappa].$$

Finally output ciphertext $\mathsf{c} = (c_1, \ldots, c_\kappa, g_1, \ldots, g_\kappa, y)$.

$\mathsf{Eval}(\mathsf{p}, \sigma, \tau, \mathsf{c}; r)$**:** Parse $\mathsf{c} = (c_1, \ldots, c_\kappa, g_1, \ldots, g_\kappa, y)$ and $r = (a'_1, \ldots, a'_\kappa, d) \in R_{\mathsf{Eval}}$. Apply the key permutation σ to obtain $\hat{g}_1, \ldots, \hat{g}_\kappa$ and the message permutation τ to obtain $\hat{c}_1, \ldots, \hat{c}_\kappa$ such that

$$\hat{g}_{\sigma(i)} = g_i \qquad \text{and} \qquad \hat{c}_{\tau(i)} = c_i$$

for all $i \in [\kappa]$. To computationally hide the permutations and make the ciphertext indistinguishable from a ciphertext produced by $\mathsf{Enc}(\mathsf{p}, \sigma(\mathsf{sk}), \tau(m))$, rerandomize the ciphertext by computing

$$\begin{aligned} c'_i &= \left(\left(\hat{c}_{i,1} \cdot h^{a'_i}\right)^{1/d}, \hat{c}_{i,2} \cdot y^{a'_i}\right) \quad \forall i \in [\kappa], \\ g'_i &= \hat{g}_i^d \quad \forall i \in [\kappa], \text{ and} \\ y' &= y^d. \end{aligned}$$

Finally, output $\mathsf{c}' = (c'_1, \ldots, c'_\kappa, g'_1, \ldots, g'_\kappa, y')$.

$\mathsf{Dec}(\mathsf{p}, \mathsf{sk}, \mathsf{c})$**:** Parse $\mathsf{c} = (c_1, \ldots, c_\kappa, g_1, \ldots, g_\kappa, \cdot)$ and $\mathsf{sk} = (s_1, \ldots, s_\kappa) \in \{0,1\}^\kappa$. For all $i \in [\kappa]$, compute

$$g_T^{m'_i} = c_{i,2} \cdot \mathrm{e}\left(\prod_{i \in [\kappa]} g_i^{s_j}, c_{i,1}\right)^{-1}$$

and output $m' = (m'_1, \ldots, m'_\kappa) \in \{0,1\}^\kappa$.

Verifying correctness of the scheme is a straightforward calculation.

Definition 17 (Correctness)**.** *We say that* $\mathsf{KMHE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is* correct *if for all* $\lambda \in \mathbb{N}$*,* $\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $\mathsf{sk} \leftarrow \mathsf{KGen}(\mathsf{p})$*, and* $\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{p}, \mathsf{sk}, m)$ *with* $m \in \mathcal{M}$ *it holds that*

$$\Pr\left[\mathsf{Dec}(\mathsf{p}, \mathsf{sk}, \mathsf{c}) = m\right] = 1.$$

Theorem 8. *Construction 1 satisfies* correctness*.*

The proof is a straightforward calculation.

Proof. Let $\mathsf{sk} = (s_1, \ldots, s_\kappa)$, $m = (m_1, \ldots, m_\kappa)$ and $\mathsf{c} = (c_1, \ldots, c_\kappa, g_1, \ldots, g_\kappa, y) \leftarrow \mathsf{Enc}(\mathsf{p}, \mathsf{sk}, m)$. For every $i \in [\kappa]$, decryption $\mathsf{Dec}(\mathsf{p}, \mathsf{sk}, \mathsf{c})$ computes

$$\begin{aligned}
g_T^{m'_i} &= c_{i,2} \cdot \mathrm{e}\left(\prod_{j \in [\kappa]} g_j^{s_j}, c_{i,1}\right)^{-1} \\
&= g_T^{m_i} \cdot y^{a_i} \cdot \mathrm{e}\left(\prod_{j \in [\kappa]} g_j^{s_j}, h^{a_i}\right)^{-1} \\
&= g_T^{m_i} \cdot \mathrm{e}\left(\prod_{j \in [\kappa]} g_j^{s_j}, h^{a_i}\right) \cdot \mathrm{e}\left(\prod_{j \in [\kappa]} g_j^{s_j}, h^{a_i}\right)^{-1} \\
&= g_T^{m_i}
\end{aligned}$$

□

5.3.1 KMH correctness

KMH correctness ensures that a ciphertext created by Eval is well-formed and that the homomorphisms are applied correctly. More precisely, given a ciphertext produced by $\mathsf{Enc}(\mathsf{p}, \mathsf{sk}, m)$, which encrypts message m under secret key sk, we can run $\mathsf{Eval}(\mathsf{p}, \sigma, \tau, \mathsf{Enc}(\mathsf{p}, \mathsf{sk}, m))$ to obtain an encryption of message $\tau(m)$ under secret key $\sigma(\mathsf{sk})$.

Definition 18 (KMH Correctness)**.** $\mathsf{KMHE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is KMH-correct, if for all* $\lambda \in \mathbb{N}$, $\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$, $\mathsf{sk} \leftarrow \mathsf{KGen}(\mathsf{p})$, $m \in \mathcal{M}$, $\sigma, \tau \in \mathcal{F}$, $r_1 \in R_{\mathsf{Enc}}$, $r_2 \in R_{\mathsf{Eval}}$ *there exists randomness* $r' \in R_{\mathsf{Enc}}$ *such that*

$$\mathsf{Eval}(\mathsf{p}, \sigma, \tau, \mathsf{Enc}(\mathsf{p}, \mathsf{sk}, m; r_1); r_2) = \mathsf{Enc}(\mathsf{p}, \sigma(\mathsf{sk}), \tau(m); r').$$

The following lemma will be useful to prove KMH correctness.

Lemma 9. *Let* $\mathsf{c} = (c_1, \ldots, c_\kappa, g_1, \ldots, g_\kappa, y)$ *be such that*

$$\begin{aligned} y &= \mathrm{e}\left(\prod_{j\in[\kappa]} g_j^{s_j}, h\right), \\ c_i &= (h^{a_i}, g_T^{m_i} \cdot y^{a_i}) \quad \forall i \in [\kappa], \end{aligned}$$

where $a_1, \ldots, a_\kappa \in \mathbb{Z}_q$, $g_1, \ldots, g_\kappa \in \mathbb{G}$, $m_1, \ldots, m_\kappa \in \{0,1\}$, *and* $s_1, \ldots, s_\kappa \in \{0,1\}$. *Then for all* $\lambda \in \mathbb{N}$, $\sigma, \tau \in \mathcal{F}$, $\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$, $r = (a'_1, \ldots, a'_\kappa, d) \in R_{\mathsf{Eval}}$, *and* $\mathsf{c}' = (c'_1, \ldots, c'_\kappa, g'_1, \ldots, g'_\kappa, y') \leftarrow \mathsf{Eval}(\mathsf{p}, \sigma, \tau, \mathsf{c}; r)$, *it holds that*

$$\begin{aligned} y' &= y^d, \\ g'_{\sigma(i)} &= g_i^d \quad \forall i \in [\kappa], \\ c'_{\tau(i)} &= (h^{u_i}, g_T^{m_i} \cdot y'^{u_i}) \quad \forall i \in [\kappa], \end{aligned} \tag{5.1}$$

where $u_1, \ldots, u_\kappa \in \mathbb{Z}_q$ *and* $d \in \mathbb{Z}_q^*$.

Proof. By definition of our KMHE scheme, c' is given by

$$\begin{aligned} y' &= y^d, \\ g'_{\sigma(i)} &= g_i^d \quad \forall i \in [\kappa], \\ c'_{\tau(i)} &= (h^{(a_i+a'_i)/d}, g_T^{m_i} \cdot y^{a_i+a'_i}) \quad \forall i \in [\kappa]. \end{aligned}$$

Now rewrite

$$y^{a_i+a'_i} = y^{b(a_i+a'_i)/d} = y'^{(a_i+a'_i)/d},$$

and set $u_i = (a_i + a'_i)/d$ for all $i \in [\kappa]$, fulfilling Equation (5.1). □

Theorem 10. *Construction 1 satisfies* KMH correctness.

Proof. Let $\mathsf{c}_1 = (\hat{c}_1, \ldots, \hat{c}_\kappa, \hat{g}_1, \ldots, \hat{g}_\kappa, \hat{y}) \leftarrow$
$acKMHEnc(\mathsf{p}, \mathsf{sk}, m; r_1)$.
Consider now $\mathsf{c}_2 = (c'_1, \ldots, c'_\kappa, g'_1, \ldots, g'_\kappa, y') \leftarrow \mathsf{Eval}(\mathsf{p}, \sigma, \tau, \mathsf{c}_1; r_2)$. By definition of the KMHE scheme, c_1 satisfies the conditions of Lemma 9.

Therefore, by Lemma 9, c_2 fulfills

$$\begin{aligned} y' = \hat{y}^b &= \mathrm{e}\left(\prod_{j\in[\kappa]} \hat{g}_j^{ds_j}, h\right), \\ g'_{\sigma(i)} &= \hat{g}_i^d \quad \forall i \in [\kappa], \\ c'_{\tau(i)} &= (h^{u_i}, g_T^{m_i} \cdot y'^{u_i}) \quad \forall i \in [\kappa], \end{aligned}$$

where $u_1, \ldots, u_\kappa \in \mathbb{Z}_q$ and $d \in \mathbb{Z}_q^*$.

We now prove that there exists $r' = (g_1, \ldots, g_\kappa, a_1, \ldots, a_\kappa) \in R_{\mathsf{Enc}}$ such that $\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{p}, \sigma(\mathsf{sk}), \tau(m); r')$ satisfies $\mathsf{c} = \mathsf{c}_2$. Specifically, we choose $r' = (g'_1, \ldots, g'_\kappa, u_1, \ldots, u_\kappa)$. Then c is given by

$$\begin{aligned} y = \mathrm{e}\left(\prod_{j\in[\kappa]} g'^{s_j}_{\sigma(j)}, h\right) &= \mathrm{e}\left(\prod_{j\in[\kappa]} \hat{g}_j^{bs_j}, h\right) = \hat{y}^b = y', \\ g'_{\sigma(j)} &= g_i \quad \forall i \in [\kappa], \\ c_{\tau(i)} &= (h^{a_i} = h^{u_i}, g_T^{m_i} \cdot y^{a_i} = g_T^{m_i} \cdot y'^{u_i}) \quad \forall i \in [\kappa], \end{aligned}$$

which is exactly c_2. □

5.3.2 CPA security

Definition 19 (IND-CPA Security)**.** $\mathsf{KMHE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is IND-CPA-secure, if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|2 \cdot \Pr\left[\mathsf{KMHE\text{-}IND\text{-}CPA}_{\mathcal{A},\mathsf{KMHE}}(1^\lambda) = 1\right] - 1\right| \leq \mathsf{negl}(\lambda),$$

where the KMHE-IND-CPA *experiment is defined in Figure 5.1.*

We now formally prove our basic scheme to be IND-CPA-secure. Our gate KMHE scheme will follow the same proof structure later, albeit with increased complexity. Therefore this proof serves as a simpler illustration of the used techniques.

Theorem 11. *Assuming hardness of the SXDH-problem (Definition 15) in the bilinear groups defined by* p*, Construction 1 is* IND-CPA-secure*.*

Building blocks of the security proof. The security analysis will use that secret keys have sufficient entropy, and apply the left-over hash lemma [HILL99, Lemma 4.8]. The *min-entropy* of a random variable X is defined as

$$H_\infty(X) = -\log\left(\max_{x\in\mathrm{supp}(X)} \Pr[X = x]\right)$$

$\underline{\mathsf{KMHE\text{-}IND\text{-}CPA}(1^\lambda)}$

$\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$

$(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{p}, 1^\lambda)$

$\mathsf{sk} \leftarrow \mathsf{KGen}(\mathsf{p})$

$b \leftarrow\!\!\$\ \{0,1\}$

$\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{p}, \mathsf{sk}, m_b)$

$b' \leftarrow \mathcal{A}^{\mathsf{Enc}(\mathsf{p},\mathsf{sk},\cdot)}(\mathsf{c})$

return $(b' = b)$

Figure 5.1: Definition of Game KMHE-IND-CPA.

Lemma 12 (Left-over Hash Lemma (LHL))**.** *Let q be a prime, $\kappa \in \mathbb{N}$, $\epsilon > 0$, and let $X = (X_1, \ldots, X_\kappa \in \{0,1\}^\kappa$ be a random variable such that $H_\infty(X) \geq \log q + 2\log(1/\epsilon)$. Then for $a \leftarrow\!\!\$\ \mathbb{Z}_q^\kappa$ it holds that*

$$\Delta((a, \sum_{i\in[\kappa]} a_i X_i), (a, U(\mathbb{Z}_q))) \leq \epsilon.$$

where $U(\mathbb{Z}_q)$ is the uniform distribution over $\mathbb{Z}_q$.

To use the LHL we will need a bound on the min-entropy of secret keys, given by the following lemma.

Lemma 13. *Let $\kappa \geq 2$ be an even integer and let $\mathsf{sk} \leftarrow\!\!\$\ HW_{\kappa,\kappa/2}$. Then*

$$H_\infty(\mathsf{sk}) \geq \kappa - \frac{1}{2}\log\kappa - 1/2.$$

Proof. We have $|HW_{\kappa,\kappa/2}| = \binom{\kappa}{\kappa/2}$. Stanica [Sta01] gives the following bound for the binomial coefficient:

$$\binom{\kappa}{\kappa/2} \geq 2^{\kappa-1} \cdot \frac{1}{\sqrt{\kappa/2}}$$

Combining this bound and the fact that sk is sampled uniformly from $HW_{\kappa,\kappa/2}$ we get that

$$H_\infty(\mathsf{sk}) = \log\binom{\kappa}{\kappa/2} \geq \kappa - \frac{1}{2}\log\kappa - 1/2.$$

□

Lastly, the following lemma will be useful to simplify the proof of CPA security.

Lemma 14. *Let $\eta, \nu \in \mathbb{N}$, $\mathbf{g}_1, \ldots, \mathbf{g}_\eta \leftarrow\!\!\$\, \mathbb{G}^\nu$, where $\mathbb{G}$ is a group of prime order q where the DDH assumption (Definition 14) holds, and $\mathsf{sk} = (s_1, \ldots, s_\nu) \in \{0,1\}^\nu$ such that $H_\infty(\mathsf{sk}) \geq \log q + 2\lambda$. Then it holds that*

$$\left\{\mathbf{g}_1, \ldots, \mathbf{g}_\eta, \mathbf{g}_1^{\mathsf{sk}}, \ldots, \mathbf{g}_\eta^{\mathsf{sk}}\right\} \overset{c}{\approx} \left\{\mathbf{g}_1, \ldots, \mathbf{g}_\eta, g_1', \ldots, g_\eta'\right\},$$

where $g_1', \ldots, g_\eta' \leftarrow\!\!\$\, \mathbb{G}$.

Proof. Define $\mathbf{G} \in \mathbb{G}^{\eta \times \nu}$ to be the matrix whose rows are given by $\mathbf{g}_1, \ldots, \mathbf{g}_\eta$. Since we are assuming hardness of the DDH problem in $\mathbb{G}$, by Theorem 6 it holds that $\mathbf{G} \overset{\text{c}}{\approx} \hat{\mathbf{G}}$, where $\hat{\mathbf{G}}$ is a uniformly random rank-1 matrix of the same dimension. Let now $\hat{\mathbf{g}}_1, \ldots, \hat{\mathbf{g}}_\eta \in \mathbb{G}^\nu$ be the rows of $\hat{\mathbf{G}}$. Since $\hat{\mathbf{G}}$ is a uniformly random rank-1 matrix, with $\alpha_1 = 1$ there exist uniformly distributed $\alpha_2, \ldots, \alpha_\eta$ such that

$$\hat{\mathbf{g}}_i = \hat{\mathbf{g}}_1^{\alpha_i} \quad \forall i \in [\eta],$$

which yields that

$$\left\{\mathbf{g}_1, \ldots, \mathbf{g}_\eta, \mathbf{g}_1^{\mathsf{sk}}, \ldots, \mathbf{g}_\eta^{\mathsf{sk}}\right\} \overset{\text{c}}{\approx} \left\{\hat{\mathbf{g}}_1, \ldots, \hat{\mathbf{g}}_\eta, \left(\hat{\mathbf{g}}_1^{\mathsf{sk}}\right)^{\alpha_1}, \ldots, \left(\hat{\mathbf{g}}_1^{\mathsf{sk}}\right)^{\alpha_\eta}\right\}, \tag{5.2}$$

Parsing $\hat{\mathbf{g}}_1 = (\hat{g}_{1,1}, \ldots, \hat{g}_{1,\nu})$, since $\hat{\mathbf{g}}_1$ is uniformly distributed, there exist uniformly distributed $x_1, \ldots x_\nu \in \mathbb{Z}_q$ such that $\hat{g}_{1,j} = g^{x_j}$ for all $j \in [\nu]$ and some generator $g \in \mathbb{G}$. Then we can write

$$\hat{\mathbf{g}}_1^{\mathsf{sk}} = \prod_{j \in [\nu]} \hat{g}_{1,j}^{s_j} = \prod_{j \in [\nu]} g^{x_j s_j} = g^{\sum_{j \in [\nu]} x_j s_j}.$$

Since $H_\infty(\mathsf{sk}) \geq \log q + 2\lambda$ and $x_1, \ldots, x_\nu \leftarrow\!\!\$\, \mathbb{Z}_q$, we can apply the left-over hash lemma (Theorem 12), resulting in

$$\Delta\left(\hat{\mathbf{g}}_1^{\mathsf{sk}}, \hat{g}\right) = \Delta\left(\prod_{j \in [\nu]} g^{x_j s_j}, \hat{g}\right) \leq 2^{-\lambda} = \mathsf{negl}(\lambda), \tag{5.3}$$

where $\hat{g} \leftarrow\!\!\$\, \mathbb{G}$. Since the $\alpha_2, \ldots, \alpha_\eta$ are uniformly distributed, it holds that

$$\hat{g}^{\alpha_i} \equiv g_i' \quad \forall i \in [\eta], \tag{5.4}$$

where $g_1', \ldots, g_\eta' \leftarrow\!\!\$\, \mathbb{G}$. Combining Equations (5.2) to (5.4) plus another invocation of DDH assumption to replace $\hat{\mathbf{G}}$ with $\mathbf{G}$ again we then obtain the desired statement. □

Proof of Theorem 11. We describe a sequence of games that moves from the regular IND-CPA security game to a game where the message is perfectly hidden. In the latter game the adversary has no chance of winning, indistinguishability of the sequence of games then ensures that the adversary also cannot win the original IND-CPA security game except with negligible probability.

Game$_1$. This is the KMHE-IND-CPA game.

Game$_2$. Let $\mathsf{c}_1, \ldots, \mathsf{c}_Q$, where

$$\mathsf{c}_i = (c_{i,1}, \ldots, c_{i,\kappa}, g_{i,1}, \ldots, g_{i,\kappa}, y_i) \leftarrow \mathsf{Enc}(\mathsf{p}, \mathsf{sk}, m_i')$$

for $i \in [Q], Q \in \mathsf{poly}(\lambda)$, be the ciphertexts given to the adversary via the encryption queries and let $\mathsf{c}_{Q+1} = (c_{Q+1,1}, \ldots, c_{Q+1,\kappa}, g_{Q+1,1}, \ldots, g_{Q+1,\kappa}, y_{Q+1})$ be the challenge ciphertext encrypting the challenge message m_b. We alter the behaviour of Enc in this hybrid, specifically how it computes the y_i in each c_i for all $i \in [Q+1]$. Enc samples an additional $x_i \leftarrow\!\!\$\ \mathbb{Z}_q$ and then sets

$$y_i = \mathrm{e}\left(g^{x_i}, h\right),$$

where $g \in \mathbb{G}$ is from p. Enc then uses these y_i for the remaining computation of c_i, specifically the $c_{i,1}, \ldots, c_{i,\kappa}$.

Game$_3$. Consider now the challenge ciphertext c_{Q+1}. In **Game$_2$**, the message-dependent part of each $c_{Q+1,j}$, $j \in [\kappa]$, is given by

$$c_{Q+1,j,2} = g_T^{m_j} \cdot y_{Q+1}^{a_j} = g_T^{m_j} \cdot (\mathrm{e}(g^{x_{Q+1}}, h))^{a_j} = g_T^{m_j} \cdot \mathrm{e}\left(g, h^{x_{Q+1} a_j}\right).$$

For **Game$_3$** we change this to

$$c_{Q+1,j,2} = g_T^{m_j} \cdot \mathrm{e}\left(g, h_j\right),$$

where $h_1, \ldots, h_\kappa \leftarrow\!\!\$\ \mathbb{G}$. Now m_j is perfectly hidden by the uniform value $\mathrm{e}\left(g, h_j\right)$.

We proceed to prove indistinguishability of these hybrids:

Game$_1$ $\overset{\mathsf{c}}{\approx}$ Game$_2$: Define $\mathbf{g}_i = (g_{i,1}, \ldots, g_{i,\kappa})$ for all $i \in [Q+1]$. By definition of our KMHE scheme it then holds that in **Game$_1$**, each y_i in c_i is computed as

$$y_i = \mathrm{e}\left(\prod_{j \in [\kappa]} g_{i,j}^{s_j}, h\right) = \mathrm{e}\left(\mathbf{g}_i^{\mathsf{sk}}, h\right) \quad \forall i \in [Q+1].$$

Our choice of κ and Theorem 13 ensures that

$$H_\infty(\mathsf{sk}) \geq \kappa - \frac{1}{2}\log\kappa - 1/2 \geq \log q + 2\lambda.$$

Since an adversary distinguishing $\mathbf{Game}_1$ and $\mathbf{Game}_2$ obtains no other information on sk except for the y_i, we can apply Theorem 14, resulting in

$$\left\{\mathbf{g}_1, \ldots, \mathbf{g}_{Q+1}, \mathbf{g}_1^{\mathsf{sk}}, \ldots, \mathbf{g}_{Q+1}^{\mathsf{sk}}\right\} \overset{\mathrm{c}}{\approx} \left\{\mathbf{g}_1, \ldots, \mathbf{g}_{Q+1}, g'_1, \ldots, g'_{Q+1}\right\}, \tag{5.5}$$

where $g'_1, \ldots, g'_{Q+1} \leftarrow\!\!\$\ \mathbb{G}$. Since $\mathbb{G}$ is cyclic, for all $i \in [Q+1]$ there exists a uniformly distributed $x_i \in \mathbb{Z}_q$ such that $g'_i = g^{x_i}$. The right side of Equation (5.5) thus corresponds to $\mathbf{Game}_2$.

$\mathbf{Game}_2 \overset{\mathsf{c}}{\approx} \mathbf{Game}_3$: From $\mathbf{Game}_2$ to $\mathbf{Game}_3$ we are only changing c_{Q+1}. In $\mathbf{Game}_2$ it is given by

$$\begin{aligned}
y_{Q+1} &= \mathrm{e}\left(g^{x_{Q+1}}, h\right) = \mathrm{e}\left(g, h^{x_{Q+1}}\right), \\
g_{Q+1,j} &\leftarrow\!\!\$\ \mathbb{G} \quad \forall j \in [\kappa], \\
c_{Q+1,j} &= \left(h^{a_{Q+1,j}}, g_T^{m_j} \cdot y_{Q+1}^{a_{Q+1,j}} = g_T^{m_j} \cdot \mathrm{e}\left(g, h^{x_{Q+1} a_{Q+1,j}}\right)\right) \quad \forall j \in [\kappa].
\end{aligned}$$

Since the $a_{Q+1,1}, \ldots, a_{Q+1,\kappa}$ are uniformly distributed and not given to the adversary, the elements $h^{a_{Q+1,1}}, \ldots, h^{a_{Q+1,\kappa}}$ look uniform over $\mathbb{G}$. Also, h was uniformly sampled as well. Therefore, we can apply Generalized SXDH (Theorem 7), resulting in

$$\begin{aligned}
h, h^{a_{Q+1,1}}, \ldots, h^{a_{Q+1,\kappa}}, h^{x_{Q+1}}, h^{x_{Q+1} a_{Q+1,1}}, \ldots, h^{x_{Q+1} a_{Q+1,\kappa}} \\
\overset{\mathrm{c}}{\approx} h, h^{a_{Q+1,1}}, \ldots, h^{a_{Q+1,\kappa}}, h', h_1, \ldots, h_\kappa,
\end{aligned}$$

where $h', h_1, \ldots, h_\kappa \leftarrow\!\!\$\ \mathbb{G}$. For c_{Q+1} this means that

$$c_{Q+1,j} \overset{\mathrm{c}}{\approx} \left(h^{a_{Q+1,j}}, g_T^{m_j} \cdot \mathrm{e}\left(g, h_j\right)\right) \quad \forall j \in [\kappa].$$

The right side corresponds to $\mathbf{Game}_3$, concluding the proof.

5.3.3 KMH rerandomizability

As indicated in the technical overview, RGS requires a KMHE scheme with rerandomizability. Intuitively, it should hold that a ciphertext produced by $\mathsf{Eval}(\mathsf{p}, \sigma, \tau, \mathsf{Enc}(\mathsf{p}, \mathsf{sk}, m))$ is computationally indistinguishable from a *fresh* ciphertext produced by $\mathsf{Enc}(\mathsf{p}, \sigma(\mathsf{sk}), \tau(m))$. We will later formally prove this property for the gate KMHE extension of the basic KHME scheme in Section 5.4. Note that our basic KMHE scheme is also rerandomizable. However, we do not provide a formal definition of rerandomizability since we will not need it. The basic idea of the proof is to use Theorem 5 to show that $g'_i = \hat{g}_i^d \ \forall i \in [\kappa]$ look like fresh group elements, and that further rerandomizing the remaining randomness a_i by adding $a'_i \ \forall i \in [\kappa]$ results in a fresh-looking ciphertext.

5.4 Gate KMHE

We now introduce and construct a novel component for RGS termed *gate KMHE*. This gate KMHE represents an augmentation of conventional KMHE, specialized for encrypting gates within garbled circuits, serving as RGS's primary building block. Such specialization enables optimizations like randomness reuse, which cannot be achieved when treating the basic KMHE scheme as a black box, thereby facilitating more efficient rerandomizable garbling constructions.

5.4.1 Definition of gate KMHE

Conceptually, a gate KMHE scheme delivers the capabilities needed to garble gates in Yao's garbled circuit framework [Yao86]. Recall that in garbled circuits each garbled gate is represented by four ciphertexts encrypting respective messages. Depending on the gate type being garbled, some messages may be identical. Double encryption protects these messages under four distinct secret keys, structured so that possessing any two of the four keys enables decryption of one message.

Our gate KMHE concept mirrors this structure while incorporating enhanced functional demands (particularly, applying identical homomorphisms to both keys and messages) and strengthened security guarantees (including computational indistinguishability between homomorphically-evaluated ciphertexts and freshly generated encryptions).

Definition 20 (Gate KMHE)**.** *A* gate *KMHE scheme is a tuple of PPT algorithms* $\mathsf{GKMHE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *with the following syntax.*

$\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$**:** *This algorithm takes as input the security parameter* λ *and returns public parameters* p*. The public parameters define a key space* $\mathcal{K}$*, a message space* $\mathcal{M}$*, and a ciphertext space* $\mathcal{C}$*.*

$\mathsf{sk} \leftarrow \mathsf{KGen}(\mathsf{p})$**:** *This algorithm takes* p *and returns a secret key* $\mathsf{sk} \in \mathcal{K}$*.*

$\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{p}, \mathsf{sk}^{00}, \mathsf{sk}^{01}, \mathsf{sk}^{10}, \mathsf{sk}^{11}, m^1, m^2, m^3, m^4)$**:** *On input four messages* m^1*,* m^2*,* m^3*,* $m^4 \in \mathcal{M}$ *and four secret keys* $\mathsf{sk}^{\beta_0\beta_1}$*,* $\beta_0, \beta_1 \in \{0,1\}$*, this algorithm returns a ciphertext* $\mathsf{c} \in \mathcal{C}$*.*

$\mathsf{c}' \leftarrow \mathsf{Eval}(\mathsf{p}, \sigma_0, \sigma_1, \tau, \mathsf{c})$**:** *On input a ciphertext* $\mathsf{c} \in \mathcal{C}$ *and three functions* $\sigma_0, \sigma_1, \tau \in \mathcal{F}$*, where the set* $\mathcal{F}$ *is defined by* λ*, this algorithm returns a ciphertext* c'*.*

$m' \leftarrow \mathsf{Dec}(\mathsf{p}, \mathsf{sk}^0, \mathsf{sk}^1, \mathsf{c})$**:** *This algorithm takes as input* p *and two secret keys* $\mathsf{sk}^0, \mathsf{sk}^1$*, and returns a message* $m' \in \mathcal{M}$*.*

By R_{KGen}, R_{Enc}, and R_{Eval} we denote the randomness spaces for KGen, Enc, and Eval, respectively. Unless specified otherwise, randomness is always assumed to be sampled uniformly at random.

5.4.2 Construction of gate KMHE

The subsequent gate KMHE construction augments the basic KMHE scheme detailed in Section 5.3.

Construction 2 (Gate KMHE)**.** Our gate KMHE scheme consists of the following algorithms.

$\mathsf{Setup}(1^\lambda)$**:** Generate and output the description of a bilinear group

$$\mathsf{p} = (q, g, h, g_T, \mathbb{G}, \mathbb{H}, \mathbb{G}_T, \mathrm{e}) \leftarrow \mathcal{G}(1^\lambda).$$

We define κ as the smallest positive and even integer such that $\kappa - \frac{3}{2}\log\kappa \geq \log q + 2\lambda$.[1] These parameters define the following sets:

- the message space as $\mathcal{M} = \{0,1\}^\kappa$.
- the key space as $\mathcal{K} = HW_{\kappa,\kappa/2}$, the set of bit strings of length κ with Hamming weight $\kappa/2$. Note that we have $\mathcal{K} \subset \mathcal{M}$.
- the ciphertext space as $\mathcal{C} = ((\mathbb{H} \times \mathbb{G})^\kappa \times \mathbb{G}_T)^4 \times \mathbb{G}^{2\kappa}$,
- the set of key- and message-homomorphisms as $\mathcal{F} = S_\kappa$, the set of permutations over κ elements,
- and randomness spaces $R_{\mathsf{KGen}} = \{0,1\}^\lambda$, $R_{\mathsf{Enc}} = \mathbb{G}^{2\kappa} \times \mathbb{Z}_q^{4\kappa} \times S_4$, and $R_{\mathsf{Eval}} = \mathbb{Z}_q^{4\kappa} \times \mathbb{Z}_q^* \times S_4$.

$\mathsf{KGen}(\mathsf{p}; r)$**:** Use r to sample and output $\mathsf{sk} \leftarrow\!\!\$\ HW_{\kappa,\kappa/2}$.

$\mathsf{Enc}(\mathsf{p}, \mathsf{sk}^{00}, \mathsf{sk}^{01}, \mathsf{sk}^{10}, \mathsf{sk}^{11}, m^1, m^2, m^3, m^4; r)$**:** Parse

$$\begin{aligned}
&(g_{0,1}, \ldots, g_{0,\kappa}, g_{1,1}, \ldots, g_{1,\kappa},\\
&\ r_1^{(1)}, \ldots, r_\kappa^{(1)}, r_1^{(2)}, \ldots, r_\kappa^{(2)},\\
&\ r_1^{(3)}, \ldots, r_\kappa^{(3)}, r_1^{(4)}, \ldots, r_\kappa^{(4)}, \pi) = r \in R_{\mathsf{Enc}},\\
&(s_1^{0\alpha}, \ldots, s_\kappa^{0\alpha}) = \mathsf{sk}^{0\alpha} \in \{0,1\}^\kappa \quad \forall \alpha \in \{0,1\},\\
&(s_1^{1\beta}, \ldots, s_\kappa^{1\beta}) = \mathsf{sk}^{1\beta} \in \{0,1\}^\kappa \quad \forall \beta \in \{0,1\},\\
&(m_1^i, \ldots, m_\kappa^i) = m^i \in \{0,1\}^\kappa \quad \forall i \in [4].
\end{aligned}$$

Then compute

$$z_i = \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{0\alpha}} \cdot g_{1,j}^{s_j^{1\beta}}\right), h\right) \quad \forall \alpha, \beta \in \{0,1\},$$

[1] Note that the gate KMHE construction requires a slightly larger lower bound on κ than the basic scheme. This is because the gate KMHE construction needs to tolerate additional leakage of information about the secret key. However, for reasonable concrete choices of q and λ, the size of κ is almost the same.

where $i = \mathsf{Bin2Dec}(\alpha\beta) + 1$, and encrypt each message m^i by computing

$$c_{i,j} = \left(h^{r_j^{(i)}}, g_T^{\mu_j^i} \cdot z_i^{r_j^{(i)}} \right) \quad \forall j \in [\kappa].$$

Finally, set

$$\mathsf{c}^i = (c_{i,1}, \ldots, c_{i,\kappa}, z_i) \quad \forall i \in [4],$$

and output ciphertext

$$\mathsf{c} = \left(\pi \left(\mathsf{c}^1, \mathsf{c}^2, \mathsf{c}^3, \mathsf{c}^4 \right), g_{0,1}, \ldots, g_{0,\kappa}, g_{1,1}, \ldots, g_{1,\kappa} \right).$$

$\mathsf{Eval}(\mathsf{p}, \sigma_0, \sigma_1, \tau, \mathsf{c}; r)$**:** Parse

$$\begin{aligned}
&(\hat{r}_1^{(1)}, \ldots, \hat{r}_\kappa^{(1)}, \hat{r}_1^{(2)}, \ldots, \hat{r}_\kappa^{(2)}, \\
&\hat{r}_1^{(3)}, \ldots, \hat{r}_\kappa^{(3)}, \hat{r}_1^{(4)}, \ldots, \hat{r}_\kappa^{(4)}, \delta, \pi') = r \in R_{\mathsf{Eval}}, \\
&(\mathsf{c}^1, \mathsf{c}^2, \mathsf{c}^3, \mathsf{c}^4, g_{0,1}, \ldots, g_{0,\kappa}, g_{1,1}, \ldots, g_{1,\kappa}) = \mathsf{c}, \\
&(c_{i,1}, \ldots, c_{i,\kappa}, z_i) = \mathsf{c}^i \quad \forall i \in [4].
\end{aligned}$$

Apply key permutations σ_0 and σ_1 by computing

$$\left(\hat{g}_{\beta,\sigma_\beta(1)}, \ldots, \hat{g}_{\beta,\sigma_\beta(\kappa)} \right) = (g_{\beta,1}, \ldots, g_{\beta,\kappa}) \quad \forall \beta \in \{0,1\},$$

and then apply message permutation τ by computing

$$\left(\hat{c}_{i,\tau(1)}, \ldots, \hat{c}_{i,\tau(\kappa)} \right) = (c_{i,1}, \ldots, c_{i,\kappa}) \quad \forall i \in [4].$$

Next, we rerandomize the result by computing

$$\begin{aligned}
z_i' &= z_i^\delta \quad \forall i \in [4], \\
g_{\beta,j}' &= \hat{g}_{\beta,j}^\delta \quad \forall j \in [\kappa], \beta \in \{0,1\}, \\
c_{i,j}' &= \left(\left(\hat{c}_{i,j,1} \cdot h^{\hat{r}_j^{(i)}} \right)^{1/\delta}, \hat{c}_{i,j,2} \cdot z_i^{\hat{r}_j^{(i)}} \right) \forall j \in [\kappa], i \in [4].
\end{aligned}$$

Finally, set

$$\mathsf{c}'^i = (c_{i,1}', \ldots, c_{i,\kappa}', z_i') \quad \forall i \in [4],$$

and define the output ciphertext as

$$\mathsf{c}' = \left(\pi' \left(\mathsf{c}'^1, \mathsf{c}'^2, \mathsf{c}'^3, \mathsf{c}'^4 \right), g_{0,1}', \ldots, g_{0,\kappa}', g_{1,1}', \ldots, g_{1,\kappa}' \right).$$

$\mathsf{Dec}(\mathsf{p}, \mathsf{sk}^0, \mathsf{sk}^1, \mathsf{c})$**:** Parse

$$(s_1^\beta, \ldots, s_\kappa^\beta) = \mathsf{sk}^\beta \in \{0,1\}^\kappa \quad \forall \beta \in \{0,1\},$$
$$(\mathsf{c}^1, \mathsf{c}^2, \mathsf{c}^3, \mathsf{c}^4, g_{0,1}, \ldots, g_{0,\kappa}, g_{1,1}, \ldots, g_{1,\kappa}) = \mathsf{c},$$
$$(c_{i,1}, \ldots, c_{i,\kappa}, z_i) = \mathsf{c}^i \quad \forall i \in [4],$$

First we need to identify which of the four sub-ciphertexts c^i can be correctly decrypted with the given keys. Compute

$$z = \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^0} \cdot g_{1,j}^{s_j^1}\right), h\right)$$

and identify the z_i, $i \in [4]$, such that $z = z_i$. If there is more than one, then abort by outputting $\perp$. Otherwise, proceed by computing and outputting $\mu' = (\mu'_1, \ldots, \mu'_\kappa)$, where

$$g_T^{\mu'_j} = c_{i,j,2} \cdot \mathrm{e}\left(\prod_{n\in[\kappa]} \left(g_{0,n}^{s_n^0} \cdot g_{1,n}^{s_n^1}\right), c_{i,j,1}\right)^{-1} \quad \forall j \in [\kappa].$$

5.4.3 Correctness

Since there are four possible messages to decrypt, Dec must be able to identify, given two keys, which of the messages to decrypt. Correctness ensures that this is done correctly, up to some negligible error probability.

Definition 21 (Correctness)**.** $\mathsf{GKMHE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is* correct*, if for all* $\lambda \in \mathbb{N}$*,* $\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $\mathsf{sk}^{00}, \mathsf{sk}^{01}, \mathsf{sk}^{10}, \mathsf{sk}^{11} \leftarrow \mathsf{KGen}(\mathsf{p})$*, and*

$$\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{p}, \mathsf{sk}^{00}, \mathsf{sk}^{01}, \mathsf{sk}^{10}, \mathsf{sk}^{11}, \mu^{00}, \mu^{01}, \mu^{10}, \mu^{11})$$

with $\mu^{00}, \mu^{01}, \mu^{10}, \mu^{11} \in \mathcal{M}$ *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr\left[\mathsf{Dec}(\mathsf{p}, \mathsf{sk}^{0\alpha}, \mathsf{sk}^{1\beta}, \mathsf{c}) = \mu^{\alpha\beta}\right] \geq 1 - \mathsf{negl}(\lambda) \quad \forall \alpha, \beta \in \{0,1\}.$$

Theorem 15. *Construction 2 is correct (Definition 21).*

Proof sketch. Verifying correctness is a straightforward calculation, almost exactly as for the basic scheme (cf. Theorem 8). The only difference is that the decryption algorithm additionally computes z and identifies the z_i, $i \in [4]$, such that $z = z_i$. To show correctness, we additionally have to argue that $z_i \neq z_j$ for $i \neq j$, except for negligible probability. For any $z' \in \{z_1, z_2, z_3, z_4\}$ in a ciphertext it holds that $z' = \mathrm{e}\left(\vec{g_0}^{\mathsf{sk}^0} \vec{g_1}^{\mathsf{sk}^1}, h\right)$ for *uniform* $\vec{g_\nu}$ and secret keys sk^ν, $\nu \in \{0,1\}$. Our choice of κ and Theorem 13 then ensures that we can apply the LHL (Theorem 12) to show that every z' is indeed statistically close to uniform, and thus the probability that $z_i = z_j$ for $i \neq j$ is negligible. □

5.4.4 KMH Correctness

KMH correctness for gate KMHE is almost identical to the corresponding definition for the basic KMHE scheme (Definition 18), but extended to match the gate KMHE syntax.

Definition 22 (KMH Correctness)**.** $\mathsf{GKMHE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is* KMH-correct*, if for all* $\lambda \in \mathbb{N}$*,* $\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$*,* $\mathsf{sk}^{00}, \mathsf{sk}^{01}, \mathsf{sk}^{10}, \mathsf{sk}^{11} \leftarrow \mathsf{KGen}(\mathsf{p})$*,* $\mu^1, \mu^2, \mu^3, \mu^4 \in \mathcal{M}$*,* $\sigma_0, \sigma_1, \tau \in \mathcal{F}$*,* $r_1 \in R_{\mathsf{Enc}}$*,* $r_2 \in R_{\mathsf{Eval}}$ *there exists randomness* $r' \in R_{\mathsf{Enc}}$ *such that*

$$\begin{aligned}&\mathsf{Eval}(\mathsf{p}, \sigma_0, \sigma_1, \tau, \mathsf{Enc}(\mathsf{p}, \mathsf{sk}^{00}, \mathsf{sk}^{01}, \mathsf{sk}^{10}, \mathsf{sk}^{11}, \mu^1, \mu^2, \mu^3, \mu^4; r_1); r_2)\\ &\quad = \mathsf{Enc}(\mathsf{p}, \sigma_0(\mathsf{sk}^{00}), \sigma_0(\mathsf{sk}^{01}), \sigma_1(\mathsf{sk}^{10}), \sigma_1(\mathsf{sk}^{11}), \tau(\mu^1), \tau(\mu^2), \tau(\mu^3), \tau(\mu^4); r').\end{aligned}$$

Theorem 16. *Construction 2 is KMH-correct (Definition 22).*

Proof. Parse $(s_1^{\beta_0\beta_1}, \ldots, s_\kappa^{\beta_0\beta_1}) = \mathsf{sk}^{\beta_0\beta_1}$ for all $\beta_0, \beta_1 \in \{0,1\}$. Let r_1 be given as in Definition 22 and set

$$\begin{aligned}&(g_{0,1}, \ldots, g_{0,\kappa}, g_{1,1}, \ldots, g_{1,\kappa},\\ &\ r_1^{(1)}, \ldots, r_\kappa^{(1)}, r_1^{(2)}, \ldots, r_\kappa^{(2)},\\ &\ r_1^{(3)}, \ldots, r_\kappa^{(3)}, r_1^{(4)}, \ldots, r_\kappa^{(4)}, \pi_1) = r_1 \in \mathbb{G}_T^{2\kappa} \times \mathbb{Z}_q^{4\kappa} \times S_4,\end{aligned}$$

and

$$\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{p}, \mathsf{sk}^{00}, \mathsf{sk}^{01}, \mathsf{sk}^{10}, \mathsf{sk}^{11}, m^1, m^2, m^3, m^4; r_1),$$

where

$$\begin{aligned}\left(\mathsf{c}^1, \mathsf{c}^2, \mathsf{c}^3, \mathsf{c}^4, g_{0,1}, \ldots, g_{0,\kappa}, g_{1,1}, \ldots, g_{1,\kappa}\right) &= \mathsf{c},\\ (c_{i,1}, \ldots, c_{i,\kappa}, y_i) &= \mathsf{c}^i \quad \forall i \in [4].\end{aligned}$$

By definition of Enc, c is computed as

$$\begin{aligned}\begin{pmatrix} y_{\pi_1(1)} \\ y_{\pi_1(2)} \\ y_{\pi_1(3)} \\ y_{\pi_1(4)} \end{pmatrix} &= \begin{pmatrix} \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{00}} \cdot g_{1,j}^{s_j^{10}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{00}} \cdot g_{1,j}^{s_j^{11}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{10}} \cdot g_{1,j}^{s_j^{10}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{10}} \cdot g_{1,j}^{s_j^{11}}\right), h\right) \end{pmatrix}, \\ c_{\pi_1(i),j} &= \left(h^{a_j^{(i)}}, g_T^{m_j^i} \cdot y_{\pi_1(i)}^{a_j^{(i)}}\right) \quad \forall i \in [4], j \in [\kappa].\end{aligned} \tag{5.6}$$

Consider now

$$\left(\hat{\mathsf{c}}^1, \hat{\mathsf{c}}^2, \hat{\mathsf{c}}^3, \hat{\mathsf{c}}^4, \hat{g}_{0,1}, \ldots, \hat{g}_{0,\kappa}, \hat{g}_{1,1}, \ldots, \hat{g}_{1,\kappa}\right) = \hat{\mathsf{c}} \leftarrow \mathsf{Eval}(\mathsf{p}, \sigma_0, \sigma_1, \tau, \mathsf{c}; r_2)$$

where

$$\begin{aligned}
&(\hat{a}_1^{(1)}, \ldots, \hat{a}_\kappa^{(1)}, \hat{a}_1^{(2)}, \ldots, \hat{a}_\kappa^{(2)},\\
&\hat{a}_1^{(3)}, \ldots, \hat{a}_\kappa^{(3)}, \hat{a}_1^{(4)}, \ldots, \hat{a}_\kappa^{(4)}, d, \pi_2) = r_2 \in R_{\mathsf{Eval}} = \mathbb{Z}_q^{4\kappa} \times \mathbb{Z}_q^* \times S_4,\\
&(\hat{c}_{i,1}, \ldots, \hat{c}_{i,\kappa}, \hat{y}_i) = \hat{\mathsf{c}}^i \quad \forall i \in [4].
\end{aligned}$$

By definition of Eval in Construction 2, $\hat{\mathsf{c}}$ is then computed as

$$\begin{aligned}
\begin{pmatrix} \hat{y}_{\pi_2(\pi_1(1))} \\ \hat{y}_{\pi_2(\pi_1(2))} \\ \hat{y}_{\pi_2(\pi_1(3))} \\ \hat{y}_{\pi_2(\pi_1(4))} \end{pmatrix} = \begin{pmatrix} y^d_{\pi_1(1)} \\ y^d_{\pi_1(2)} \\ y^d_{\pi_1(3)} \\ y^d_{\pi_1(4)} \end{pmatrix} &= \begin{pmatrix} \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{00}} \cdot g_{1,j}^{s_j^{10}}\right)^d, h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{00}} \cdot g_{1,j}^{s_j^{11}}\right)^d, h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{01}} \cdot g_{1,j}^{s_j^{10}}\right)^d, h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{01}} \cdot g_{1,j}^{s_j^{11}}\right)^d, h\right) \end{pmatrix}, \\
\hat{g}_{0,\sigma_0(j)} &= g_{0,j}^d \quad \forall j \in [\kappa], \\
\hat{g}_{1,\sigma_1(j)} &= g_{1,j}^d \quad \forall j \in [\kappa], \\
\hat{c}_{\pi_2(\pi_1(i)),\tau(j)} &= \left(h^{\left(a_j^{(i)} + \hat{a}^{(\pi_1(i))}_{\tau(j)}\right)/d}, g_T^{m_j^i} \cdot y_{\pi_1(i)}^{a_j^{(i)} + \hat{a}^{(\pi_1(i))}_{\tau(j)}}\right) \quad \forall j \in [\kappa], i \in [4].
\end{aligned} \tag{5.7}$$

We now prove that there exists $r' \in R_{\mathsf{Enc}}$ such that

$$\mathsf{c}' \leftarrow \mathsf{Enc}(\mathsf{p}, \sigma_0(\mathsf{sk}^{00}), \sigma_0(\mathsf{sk}^{01}), \sigma_1(\mathsf{sk}^{10}), \sigma_1(\mathsf{sk}^{11}), \tau(m^1), \tau(m^2), \tau(m^3), \tau(m^4); r')$$

satisfies $\mathsf{c}' = \hat{\mathsf{c}}$. Specifically, we choose

$$\begin{aligned}
r' = (&\hat{g}_{0,1}, \ldots, \hat{g}_{0,\kappa}, \hat{g}_{1,1}, \ldots, \hat{g}_{1,\kappa},\\
&u_1^{(1)}, \ldots, u_\kappa^{(1)}, u_1^{(2)}, \ldots, u_\kappa^{(2)},\\
&u_1^{(3)}, \ldots, u_\kappa^{(3)}, u_1^{(4)}, \ldots, u_\kappa^{(4)}, \pi_2 \circ \pi_1) \in \mathbb{G}^{2\kappa} \times \mathbb{Z}_q^{4\kappa} \times S_4,
\end{aligned}$$

where we define

$$u^{(i)}_{\tau(j)} = \left(a_j^{(i)} + \hat{a}^{(\pi_1(i))}_{\tau(j)}\right)/d \quad \forall j \in [\kappa], i \in [4].$$

Then, parsing

$$\begin{aligned}
\left(\mathsf{c}'^1, \mathsf{c}'^2, \mathsf{c}'^3, \mathsf{c}'^4, g'_{0,1}, \ldots, g'_{0,\kappa}, g'_{1,1}, \ldots, g'_{1,\kappa}\right) &= \mathsf{c},\\
(c'_{i,1}, \ldots, c'_{i,\kappa}, y'_{i_1}) &= \mathsf{c}'^i \quad \forall i \in [4],
\end{aligned}$$

it holds by the definition of Enc in Construction 2 that

$$\begin{aligned}
\begin{pmatrix} y'_{\pi_2(\pi_1(1))} \\ y'_{\pi_2(\pi_1(2))} \\ y'_{\pi_2(\pi_1(3))} \\ y'_{\pi_2(\pi_1(4))} \end{pmatrix} &= \begin{pmatrix} \mathrm{e}\left(\prod_{j\in[\kappa]} \left(\hat{g}_{0,j}^{\sigma_0(\mathsf{sk}^{00})_j} \cdot \hat{g}_{1,j}^{\sigma_1(\mathsf{sk}^{10})_j}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(\hat{g}_{0,j}^{\sigma_0(\mathsf{sk}^{00})_j} \cdot \hat{g}_{1,j}^{\sigma_1(\mathsf{sk}^{10})_j}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(\hat{g}_{0,j}^{\sigma_0(\mathsf{sk}^{01})_j} \cdot \hat{g}_{1,j}^{\sigma_1(\mathsf{sk}^{11})_j}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(\hat{g}_{0,j}^{\sigma_0(\mathsf{sk}^{01})_j} \cdot \hat{g}_{1,j}^{\sigma_1(\mathsf{sk}^{11})_j}\right), h\right) \end{pmatrix} \\
&= \begin{pmatrix} \mathrm{e}\left(\prod_{j\in[\kappa]} \left(\hat{g}_{0,\sigma_0(j)}^{s_j^{00}} \cdot \hat{g}_{1,\sigma_1(j)}^{s_j^{10}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(\hat{g}_{0,\sigma_0(j)}^{s_j^{00}} \cdot \hat{g}_{1,\sigma_1(j)}^{s_j^{10}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(\hat{g}_{0,\sigma_0(j)}^{s_j^{01}} \cdot \hat{g}_{1,\sigma_1(j)}^{s_j^{11}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(\hat{g}_{0,\sigma_0(j)}^{s_j^{01}} \cdot \hat{g}_{1,\sigma_1(j)}^{s_j^{11}}\right), h\right) \end{pmatrix} \\
&= \begin{pmatrix} \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{00}} \cdot g_{1,j}^{s_j^{10}}\right)^d, h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{00}} \cdot g_{1,j}^{s_j^{10}}\right)^d, h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{01}} \cdot g_{1,j}^{s_j^{11}}\right)^d, h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{01}} \cdot g_{1,j}^{s_j^{11}}\right)^d, h\right) \end{pmatrix} = \begin{pmatrix} \hat{y}_{\pi_2(\pi_1(1))} \\ \hat{y}_{\pi_2(\pi_1(2))} \\ \hat{y}_{\pi_2(\pi_1(3))} \\ \hat{y}_{\pi_2(\pi_1(4))} \end{pmatrix},
\end{aligned}$$

and

$$\begin{aligned}
c'_{\pi_2(\pi_1(i)),\tau(j)} &= \left(h^{u_{\tau(j)}^{(i)}}, g_T^{m_j^i} \cdot y_{\pi_2(\pi_1(i)}^{\prime u_{\tau(j)}^{(i)}}\right) \\
&= \left(h^{\left(a_j^{(i)} + \hat{a}_{\tau(j)}^{(\pi_1(i))}\right)/d}, g_T^{m_j^i} \cdot \hat{y}_{\pi_2(\pi_1(i))}^{\left(a_j^{(i)} + \hat{a}_{\tau(j)}^{(\pi_1(i))}\right)/d}\right) \\
&= \left(h^{\left(a_j^{(i)} + \hat{a}_{\tau(j)}^{(\pi_1(i))}\right)/d}, g_T^{m_j^i} \cdot y_{\pi_1(i)}^{d\left(a_j^{(i)} + \hat{a}_{\tau(j)}^{(\pi_1(i))}\right)/d}\right) \\
&= \left(h^{\left(a_j^{(i)} + \hat{a}_{\tau(j)}^{(\pi_1(i))}\right)/d}, g_T^{m_j^i} \cdot y_{\pi_1(i)}^{a_j^{(i)} + \hat{a}_{\tau(j)}^{(\pi_1(i))}}\right) \\
&= \hat{c}_{\pi_2(\pi_1(i)),\tau(j)} \quad \forall j \in [\kappa], i \in [4].
\end{aligned}$$

Since we also chose $g'_{\beta,j} = \hat{g}_{\beta,j}$ for all $\beta \in \{0,1\}, j \in [\kappa]$, we have proven that $\mathsf{c}' = \hat{\mathsf{c}}$, completing the proof. □

5.4.5 CPA-Security

We require IND-CPA security against an adversary which knows two out of the four secret keys used to encrypt a ciphertext, and therefore is able to decrypt one out of the four messages encrypted in the gate KMHE ciphertext. The other three messages should remain indistinguishable.

More precisely, we define the IND-CPA-security experiment for a gate KMHE in Figure 5.2. The experiment is parametrized by $(\beta_0, \beta_1) \in \{0,1\}^2$, which specify two of the four keys known to the adversary, and thereby also the message m^{β_0,β_1} that can be decrypted with these two keys.

1. The experiment generates parameters $\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$ and samples a random bit $b \xleftarrow{\$} \{0,1\}$

2. The adversary receives p and outputs seven messages:
   - One "known" message $m^{\beta_0\beta_1} \in \mathcal{M}$, for which the adversary will receive the corresponding secret keys
   - Three "challenge" message pairs $(m_0^{\beta_0,1-\beta_1}, m_1^{\beta_0,1-\beta_1})$, $(m_0^{1-\beta_0,\beta_1}, m_1^{1-\beta_0,\beta_1})$, and $(m_0^{1-\beta_0,1-\beta_1}, m_1^{1-\beta_0,1-\beta_1})$, each of which corresponds to at least one secret key that will remain unknown to the adversary.

3. The experiment generates four keys $\mathsf{sk}^{0,0}, \mathsf{sk}^{0,1}, \mathsf{sk}^{1,0}, \mathsf{sk}^{1,1} \leftarrow \mathsf{KGen}(\mathsf{p})$ and uses them to encrypt the "known" message $m^{\beta_0\beta_1}$ along with the three "challenge" messages $m_b^{\beta_0,1-\beta_1}$, $m_b^{1-\beta_0,\beta_1}$, $m_b^{1-\beta_0,1-\beta_1}$ in the challenge ciphertext.

4. The adversary receives the challenge ciphertext and the two secret keys $\mathsf{sk}^{0,\beta_0}, \mathsf{sk}^{1,\beta_1}$, and outputs a bit b'. During this phase, the adversary also has access to three oracles which it can use to obtain encryptions under the unknown keys $\mathsf{sk}^{0,1-\beta_0}$, and $\mathsf{sk}^{0,1-\beta_1}$.

5. The adversary wins the game, if $b' = b$. We say that it breaks the IND-CPA security of the gate KMHE scheme, if it wins with significantly better probability than guessing.

Definition 23 (IND-CPA Security)**.** $\mathsf{GKMHE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is IND-CPA-secure, if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all* $\beta_0, \beta_1 \in \{0,1\}$ *it holds that*

$$\left|2 \cdot \Pr\left[\mathsf{GKMHE\text{-}IND\text{-}CPA}^{\beta_0,\beta_1}_{\mathcal{A},\mathsf{GKMHE}}(1^\lambda) = 1\right] - 1\right| \leq \mathsf{negl}(\lambda),$$

where the GKMHE-IND-CPA *experiment is defined in Figure 5.2.*

$\mathsf{GKMHE\text{-}IND\text{-}CPA}^{\beta_0,\beta_1}_{\mathcal{A},\mathsf{GKMHE}}(1^\lambda)$

$\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda); b \leftarrow\!\!\$\ \{0,1\}$

$\left(m^{\beta_0,\beta_1}, m_0^{\beta_0,1-\beta_1}, m_1^{\beta_0,1-\beta_1}, m_0^{1-\beta_0,\beta_1}, m_1^{1-\beta_0,\beta_1}, m_0^{1-\beta_0,1-\beta_1}, m_1^{1-\beta_0,1-\beta_1}\right) \leftarrow \mathcal{A}(1^\lambda, \mathsf{p})$

$m_0^{\beta_0,\beta_1}, m_1^{\beta_0,\beta_1} \leftarrow m^{\beta_0,\beta_1}$

$\mathsf{sk}^{0,0}, \mathsf{sk}^{0,1}, \mathsf{sk}^{1,0}, \mathsf{sk}^{1,1} \leftarrow \mathsf{KGen}(\mathsf{p})$

$\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{p}, \mathsf{sk}^{0,0}, \mathsf{sk}^{0,1}, \mathsf{sk}^{1,0}, \mathsf{sk}^{1,1}, m_b^{0,0}, m_b^{0,1}, m_b^{1,0}, m_b^{1,1})$

$b' \leftarrow \mathcal{A}^{O(\cdot,\mathsf{sk}^{0,1-\beta_0},\cdot,\mathsf{sk}^{1,1-\beta_1},\cdot,\cdot,\cdot,\cdot),O(\cdot,\mathsf{sk}^{0,1-\beta_0},\cdot,\cdot,\cdot,\cdot,\cdot,\cdot),O(\cdot,\cdot,\cdot,\mathsf{sk}^{1,1-\beta_1},\cdot,\cdot,\cdot,\cdot)}(\mathsf{c}, \mathsf{sk}^{0,\beta_0}, \mathsf{sk}^{1,\beta_1})$

return $(b' = b)$

Figure 5.2: Definition of Game GKMHE-IND-CPA. The oracle $O(\mathsf{sk}^{0,\beta_0}, \mathsf{sk}^{0,1-\beta_0}, \mathsf{sk}^{1,\beta_1}, \mathsf{sk}^{1,1-\beta_1}, m^1, m^2, m^3, m^4)$, given secret keys $\mathsf{sk}^{0,\beta_0}, \mathsf{sk}^{0,1-\beta_0}, \mathsf{sk}^{1,\beta_1}, \mathsf{sk}^{1,1-\beta_1}$, and messages m^1, m^2, m^3, m^4, returns the output of $\mathsf{Enc}(\mathsf{p}, \mathsf{sk}^{0,0}, \mathsf{sk}^{0,1}, \mathsf{sk}^{1,0}, \mathsf{sk}^{1,1}, m^1, m^2, m^3, m^4)$.

Theorem 17. *Construction 2 is IND-CPA-secure, assuming hardness of the SXDH-problem (Definition 15) in the bilinear groups defined by* p.

Proof. The game GKMHE-IND-CPA is parameterized by two bits $\beta_0, \beta_1 \in \{0,1\}$ which determine the keys the adversary obtains. To simplify our notation, without loss of generality we will assume that $\beta_0 = \beta_1 = 0$. The other three cases follow analogously.

We define a sequence of games, starting from $\mathsf{GKMHE\text{-}IND\text{-}CPA}^{0,0}_{\mathcal{A},\mathsf{GKMHE}}$ and ending with a game where the challenge messages are perfectly hidden. Since the adversary $\mathcal{A}$ cannot win in the later game, computational indistinguishability of the sequence of games then ensures that $\mathcal{A}$ also cannot win in game $\mathsf{GKMHE\text{-}IND\text{-}CPA}^{0,0}_{\mathcal{A},\mathsf{GKMHE}}$, except with negligible probability.

Game$_1$. This is the $\mathsf{GKMHE\text{-}IND\text{-}CPA}^{0,0}_{\mathcal{A},\mathsf{GKMHE}}$ game. Let

$$\mathsf{c}_{1,i_1} = \left(\mathsf{c}^{00}_{1,i_1}, \mathsf{c}^{01}_{1,i_1}, \mathsf{c}^{10}_{1,i_1}, \mathsf{c}^{11}_{1,i_1}, g_{1,i_1,0,1}, \ldots, g_{1,i_1,0,\kappa}, g_{1,i_1,1,1}, \ldots, g_{1,i_1,1,\kappa}\right)$$
$$\mathsf{c}_{2,i_2} = \left(\mathsf{c}^{00}_{2,i_2}, \mathsf{c}^{01}_{2,i_2}, \mathsf{c}^{10}_{2,i_2}, \mathsf{c}^{11}_{2,i_2}, g_{2,i_2,0,1}, \ldots, g_{2,i_2,0,\kappa}, g_{2,i_2,1,1}, \ldots, g_{2,i_2,1,\kappa}\right)$$
$$\mathsf{c}_{3,i_3} = \left(\mathsf{c}^{00}_{3,i_3}, \mathsf{c}^{01}_{3,i_3}, \mathsf{c}^{10}_{3,i_3}, \mathsf{c}^{11}_{3,i_3}, g_{3,i_3,0,1}, \ldots, g_{3,i_3,0,\kappa}, g_{3,i_3,1,1}, \ldots, g_{3,i_3,1,\kappa}\right)$$

for $i_1 \in [Q_1], i_2 \in [Q_2], i_3 \in [Q_3]$, be the ciphertexts given to the adversary via the oracles

$$O(\cdot, \mathsf{sk}^{0,1-\beta_0}, \cdot, \mathsf{sk}^{1,1-\beta_1}, \cdot, \cdot, \cdot, \cdot),$$
$$O(\cdot, \mathsf{sk}^{0,1-\beta_0}, \cdot, \cdot, \cdot, \cdot, \cdot, \cdot), \text{ and}$$
$$O(\cdot, \cdot, \cdot, \mathsf{sk}^{1,1-\beta_1}, \cdot, \cdot, \cdot, \cdot),$$

respectively. Let also

$$\mathsf{c} = (\mathsf{c}^{00}, \mathsf{c}^{01}, \mathsf{c}^{10}, \mathsf{c}^{11}, g_{0,1}, \ldots, g_{0,\kappa}, g_{1,1}, \ldots, g_{1,\kappa})$$

be the challenge ciphertext. Furthermore, parse

$$\begin{aligned}\mathsf{c}_{\gamma,i_\gamma}^{\delta_0\delta_1} &= (c_{\gamma,i_\gamma,\delta_0\delta_1,1}, \ldots, c_{\gamma,i_\gamma,\delta_0\delta_1,\kappa}, y_{\gamma,i_\gamma,\delta_0\delta_1}),\\ \mathsf{c}^{\delta_0\delta_1} &= (c_{\delta_0\delta_1,1}, \ldots, c_{\delta_0\delta_1,\kappa}, y_{\delta_0\delta_1})\end{aligned}$$

for all $\gamma \in [3]$, $i_\gamma \in [Q_\gamma]$, $\delta_0, \delta_1 \in \{0,1\}$.

Game$_2$. We alter how each query response ciphertext, including the challenge ciphertext, is created by Enc. Specifically, we replace the terms involving the keys sk^{01} and sk^{11} unknown to the adversary with uniform exponents. More precisely, for each $\mathsf{c}_{1,i}$, $i \in [Q_1]$, Enc, given secret keys $\mathsf{sk}_{1,i}^{00}$, sk^{01}, $\mathsf{sk}_{1,i}^{10}$, and sk^{11}, samples additional $x_{1,i}^{(01)}, x_{1,i}^{(11)} \leftarrow\!\!\$\ \mathbb{Z}_q^*$ and then computes

$$\begin{aligned}y_{1,i,01} &= \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{1,i,0,j}^{\mathsf{sk}_{1,i,j}^{00}}\right) \cdot g^{x_{1,i}^{(11)}}, h\right),\\ y_{1,i,10} &= \mathrm{e}\left(g^{x_{1,i}^{(01)}} \cdot \prod_{j\in[\kappa]} g_{1,i,1,j}^{\mathsf{sk}_{1,i,j}^{10}}, h\right),\\ y_{1,i,11} &= \mathrm{e}\left(g^{x_{1,i}^{(01)}} \cdot g^{x_{1,i}^{(11)}}, h\right),\end{aligned}$$

The rest of the computation of $\mathsf{c}_{1,i}$ is exactly as in **Game$_1$**. The challenge ciphertext c is adjusted in the exact same manner using elements $x^{(01)}, x^{(11)} \leftarrow\!\!\$\ \mathbb{Z}_q^*$ since it also is computed using the keys sk^{01} and sk^{11}.

The remaining ciphertexts now follow a similar process albeit with some small differences due to the different keys used. For each $\mathsf{c}_{2,i}$, $i \in [Q_2]$, Enc, given secret keys $\mathsf{sk}_{2,i}^{00}$, sk^{01}, $\mathsf{sk}_{2,i}^{10}$, and $\mathsf{sk}_{2,i}^{11}$, samples an additional $x_{2,i}^{(01)} \leftarrow\!\!\$\ \mathbb{Z}_q^*$ and then computes

$$\begin{aligned}y_{2,i,10} &= \mathrm{e}\left(g^{x_{2,i}^{(01)}} \cdot \prod_{j\in[\kappa]} g_{2,i,1,j}^{\mathsf{sk}_{2,i,j}^{10}}, h\right),\\ y_{2,i,11} &= \mathrm{e}\left(g^{x_{2,i}^{(01)}} \cdot \prod_{j\in[\kappa]} g_{2,i,1,j}^{\mathsf{sk}_{2,i,j}^{11}}, h\right).\end{aligned}$$

For each $\mathsf{c}_{3,i}$, $i \in [Q_3]$, Enc, given secret keys $\mathsf{sk}_{3,i}^{00}$, $\mathsf{sk}_{3,i}^{01}$, $\mathsf{sk}_{3,i}^{10}$, and sk^{11}, samples

an additional $x_{3,i}^{(11)} \leftarrow\!\!\$\ \mathbb{Z}_q^*$ and then computes

$$y_{3,i,01} = \mathrm{e}\left(\prod_{j\in[\kappa]}\left(g_{3,i,0,j}^{\mathsf{sk}_{3,i,j}^{00}}\right)\cdot g^{x_{3,i}^{(11)}}, h\right),$$
$$y_{3,i,11} = \mathrm{e}\left(\prod_{j\in[\kappa]}\left(g_{3,i,0,j}^{\mathsf{sk}_{3,i,j}^{01}}\right)\cdot g^{x_{3,i}^{(11)}}, h\right).$$

Game$_3$. In this game we alter the challenge ciphertext to perfectly hide the challenge messages m^{01}, m^{10} and m^{11}. In **Game$_2$**, by definition of Enc, each $c_{\delta_0\delta_1,i}$, $i \in [\kappa], \delta_0, \delta_1 \in \{0,1\}$, is computed as

$$c_{\delta_0\delta_1,i} = \left(h^{a_i^{(\delta_0\delta_1)}}, g_T^{m_i^{\delta_0\delta_1}} \cdot y_{\delta_0\delta_1}^{a_i^{(\delta_0\delta_1)}}\right)$$

with

$$y_{01}^{a_i^{(01)}} = \mathrm{e}\left(\prod_{j\in[\kappa]}\left(g_{0,j}^{\mathsf{sk}_j^{00}}\right), h^{a_i^{(01)}}\right)\cdot \mathrm{e}\left(g^{x^{(11)}}, h^{a_i^{(01)}}\right),$$
$$y_{10}^{a_i^{(10)}} = \mathrm{e}\left(g^{x^{(01)}}, h^{a_i^{(10)}}\right)\cdot \mathrm{e}\left(\prod_{j\in[\kappa]} g_{1,j}^{\mathsf{sk}_j^{10}}, h^{a_i^{(10)}}\right),$$
$$y_{11}^{a_i^{(11)}} = \mathrm{e}\left(g^{x^{(01)}}, h^{a_i^{(11)}}\right)\cdot \mathrm{e}\left(g^{x^{(11)}}, h^{a_i^{(11)}}\right),$$

for all $i \in [\kappa]$. For **Game$_3$**, we change each $c_{\delta_0\delta_1,i}$, $i \in [\kappa], \delta_0, \delta_1 \in \{0,1\}$ except for $\delta_0 = \delta_1 = 0$ to be computed as

$$c_{\delta_0\delta_1,i} = \left(h^{a_i^{(\delta_0\delta_1)}}, g_T^{m_i^{\delta_0\delta_1}} \cdot y_{\delta_0\delta_1,i}\right),$$

where $y_{\delta_0\delta_1,1}, \ldots, y_{\delta_0\delta_1,\kappa} \leftarrow\!\!\$\ \mathbb{G}$. Now each $m_i^{\delta_0\delta_1}$ is perfectly hidden by the uniform $y_{\delta_0\delta_1,i}$.

We now prove that the hybrid games are computationally indistinguishable.

Game$_1$ $\overset{\mathsf{c}}{\approx}$ Game$_2$: Define vectors

$$\vec{g}_{\gamma,i_\gamma,\delta} = (g_{\gamma,i_\gamma,\delta,1}, \ldots, g_{\gamma,i_\gamma,\delta,\kappa}) \quad \forall \gamma \in [3], i \in [Q_\gamma], \delta \in \{0,1\},$$
$$\vec{g}_\delta = (g_{\delta,1}, \ldots, g_{\delta,1}) \quad \forall \delta \in \{0,1\}.$$

The elements inside the ciphertexts that depend on sk^{01} and sk^{11} can then be written as

$$\prod_{j\in[\kappa]} g_{\gamma,i_\gamma,\delta,j}^{\mathsf{sk}_j^{\delta 1}} = \vec{g}_{\gamma,i_\gamma,\delta}^{\mathsf{sk}^{\delta 1}} \quad \forall \gamma \in [3], i \in [Q_\gamma], \delta \in \{0,1\},$$
$$\prod_{j\in[\kappa]} g_{\delta,j}^{\mathsf{sk}_j^{\delta 1}} = \vec{g}_{\delta}^{\mathsf{sk}^{\delta 1}} \quad \forall \delta \in \{0,1\}.$$

Our choice of κ and Theorem 13 ensures that

$$H_\infty(\mathsf{sk}^{01}) = H_\infty(\mathsf{sk}^{11}) \geq \kappa - \frac{3}{2}\log\kappa \geq \kappa - \frac{1}{2}\log\kappa - 1/2 \geq \log q + 2\lambda,$$

where the second inequality holds due to our choice of κ as a positive, even integer. Hence, we have $\kappa \geq 2$ (and actually $\kappa \gg 2$ much larger for reasonable choices of λ).

We can now apply Theorem 14, resulting in

$$\begin{aligned}
\left\{\vec{g}_{\gamma,i_\gamma,0}, \vec{g}_0, \vec{g}_{\gamma,i_\gamma,0}^{\mathsf{sk}^{01}}, \vec{g}_0^{\mathsf{sk}^{01}}\right\}_{\gamma\in[3],i_\gamma\in[Q_\gamma]} &\overset{\mathrm{c}}{\approx} \left\{\vec{g}_{\gamma,i_\gamma,0}, \vec{g}_0, g'_{\gamma,i_\gamma,0}, g'_0\right\}_{\gamma\in[3],i_\gamma\in[Q_\gamma]}, \\
\left\{\vec{g}_{\gamma,i_\gamma,1}, \vec{g}_1, \vec{g}_{\gamma,i_\gamma,1}^{\mathsf{sk}^{11}}, \vec{g}_1^{\mathsf{sk}^{11}}\right\}_{\gamma\in[3],i_\gamma\in[Q_\gamma]} &\overset{\mathrm{c}}{\approx} \left\{\vec{g}_{\gamma,i_\gamma,1}, \vec{g}_1, g'_{\gamma,i_\gamma,1}, g'_1\right\}_{\gamma\in[3],i_\gamma\in[Q_\gamma]},
\end{aligned} \tag{5.8}$$

where $g'_{\gamma,i_\gamma,0}, g'_{\gamma,i_\gamma,1}, g'_0, g'_1 \leftarrow\!\!\$\ \mathbb{G}$ look like fresh uniform elements. Since $\mathbb{G}$ is cyclic, these elements can also be written with respect to basis g. Hence, the right side of Equation (5.8) corresponds to $\mathbf{Game}_2$.

$\mathbf{Game}_2 \overset{\mathsf{c}}{\approx} \mathbf{Game}_3$: We apply Theorem 7 twice, resulting in

$$\begin{aligned}
&\{h, h^{a_1^{(01)}}, \ldots, h^{a_\kappa^{(01)}}, h^{a_1^{(11)}}, \ldots, h^{a_\kappa^{(11)}}, \\
&\quad h^{x^{(11)}}, h^{x^{(11)}a_1^{(01)}}, \ldots, h^{x^{(11)}a_\kappa^{(01)}}, h^{x^{(11)}a_1^{(11)}}, \ldots, h^{x^{(11)}a_\kappa^{(11)}}\} \\
\overset{\mathrm{c}}{\approx}\ &\{h, h^{a_1^{(01)}}, \ldots, h^{a_\kappa^{(01)}}, h^{a_1^{(11)}}, \ldots, h^{a_\kappa^{(11)}}, \\
&\quad h^{(11)}, h_1^{(01)}, \ldots, h_\kappa^{(01)}, h_1^{(1111)}, \ldots, h_\kappa^{(1111)}\},
\end{aligned}$$

where $h^{(11)}, h_1^{(01)}, \ldots, h_\kappa^{(01)}, h_1^{(1111)}, \ldots, h_\kappa^{(1111)} \leftarrow\!\!\$\ \mathbb{H}$, and

$$\begin{aligned}
&\{h, h^{a_1^{(10)}}, \ldots, h^{a_\kappa^{(10)}}, h^{a_1^{(11)}}, \ldots, h^{a_\kappa^{(11)}}, \\
&\quad h^{x^{(01)}}, h^{x^{(01)}a_1^{(10)}}, \ldots, h^{x^{(01)}a_\kappa^{(10)}}, h^{x^{(01)}a_1^{(11)}}, \ldots, h^{x^{(01)}a_\kappa^{(11)}}\} \\
\overset{\mathrm{c}}{\approx}\ &\{h, h^{a_1^{(10)}}, \ldots, h^{a_\kappa^{(10)}}, h^{a_1^{(11)}}, \ldots, h^{a_\kappa^{(11)}}, \\
&\quad h^{(01)}, h_1^{(10)}, \ldots, h_\kappa^{(10)}, h_1^{(0111)}, \ldots, h_\kappa^{(0111)}\},
\end{aligned}$$

where $h^{(01)}, h_1^{(10)}, \ldots, h_\kappa^{(10)}, h_1^{(0111)}, \ldots, h_\kappa^{(0111)} \leftarrow\!\!\$\ \mathbb{H}$. Applying this to the $c_{\delta_0\delta_1,i}$, $i \in [\kappa]$, $\delta_0, \delta_1 \in \{0,1\}$ except $\delta_0 = \delta_1 = 0$, in $\mathbf{Game}_2$, we obtain

$$
\begin{aligned}
c_{01,i} &= \left(h^{a_i^{(01)}}, g_T^{m_i^{01}} \cdot y_{01}^{a_i^{(01)}}\right) \\
&= \left(h^{a_i^{(01)}}, g_T^{m_i^{01}} \cdot \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{\mathsf{sk}_j^{00}}\right), h^{a_i^{(01)}}\right) \cdot \mathrm{e}\left(g^{x^{(11)}}, h^{a_i^{(01)}}\right)\right) \\
&\overset{\mathrm{c}}{\approx} \left(h^{a_i^{(01)}}, g_T^{m_i^{01}} \cdot \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{\mathsf{sk}_j^{00}}\right), h^{a_i^{(01)}}\right) \cdot \mathrm{e}\left(g, h_i^{(01)}\right)\right), \\
c_{10,i} &= \left(h^{a_i^{(10)}}, g_T^{m_i^{10}} \cdot y_{10}^{a_i^{(10)}}\right) \\
&= \left(h^{a_i^{(10)}}, g_T^{m_i^{10}} \cdot \mathrm{e}\left(g^{x^{(01)}}, h^{a_i^{(10)}}\right) \cdot \mathrm{e}\left(\prod_{j\in[\kappa]} g_{1,j}^{\mathsf{sk}_j^{10}}, h^{a_i^{(10)}}\right)\right) \\
&\overset{\mathrm{c}}{\approx} \left(h^{a_i^{(10)}}, g_T^{m_i^{10}} \cdot \mathrm{e}\left(g, h_i^{(10)}\right) \cdot \mathrm{e}\left(\prod_{j\in[\kappa]} g_{1,j}^{\mathsf{sk}_j^{10}}, h^{a_i^{(10)}}\right)\right), \\
c_{11,i} &= \left(h^{a_i^{(11)}}, g_T^{m_i^{11}} \cdot y_{11}^{a_i^{(11)}}\right) \\
&= \left(h^{a_i^{(11)}}, g_T^{m_i^{11}} \cdot \mathrm{e}\left(g^{x^{(01)}}, h^{a_i^{(11)}}\right) \cdot \mathrm{e}\left(g^{x^{(11)}}, h^{a_i^{(11)}}\right)\right) \\
&\overset{\mathrm{c}}{\approx} \left(h^{a_i^{(11)}}, g_T^{m_i^{11}} \cdot \mathrm{e}\left(g, h_i^{(0111)}\right) \cdot \mathrm{e}\left(g, h_i^{(1111)}\right)\right).
\end{aligned}
\tag{5.9}
$$

Since the $h_i^{(01)}, h_i^{(10)}, h_i^{(0111)}$, and $h_i^{(1111)}$ are uniformly distributed, the message terms are also uniformly distributed. Therefore, the right side of Equation (5.9) corresponds to $\mathbf{Game}_3$.

□

5.4.6 Key Privacy

Key privacy formalizes the notion that a transformed secret key should be statistically close to a fresh secret key.

Definition 24 (Key Privacy)**.** $\mathsf{GKMHE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *has key privacy, if for all* $\lambda \in \mathbb{N}$, $\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$, $\sigma \leftarrow\!\!\$\ \mathcal{F}$, $\mathsf{sk}, \mathsf{sk}' \leftarrow \mathsf{KGen}(\mathsf{p})$ *it holds that*

$$\{\mathsf{p}, \mathsf{sk}, \sigma(\mathsf{sk})\} \overset{s}{\approx} \{\mathsf{p}, \mathsf{sk}, \mathsf{sk}'\},$$

Theorem 18. *Construction 2 has key privacy (Definition 24).*

Proof. This proof is the same as for Claim 7 in [AHKP22]. The key space is given by $HW_{\kappa,\kappa/2}$, consisting of all bit-strings with $\kappa/2$ 0's and $\kappa/2$ 1's. Permutations are bijective over $HW_{\kappa,\kappa/2}$, therefore $\sigma(\mathsf{sk})$ is distributed uniformly over $HW_{\kappa,\kappa/2}$, just like a fresh key sk'. □

5.4.7 KMH Rerandomizability

The next important security property is that of KMH rerandomizability. KMH rerandomizability essentially provides two guarantees:

1. Eval, when applied to a ciphertext, rerandomizes that ciphertext, resulting in a fresh-looking ciphertext. This holds as long as the randomness used to create the initial ciphertext was uniformly sampled, even if it is known to the adversary.

2. Applying a sequence of $t-1$ Eval operations with homomorphisms $(\sigma_{0,i}, \sigma_{1,i}, \tau_i)$, $i \in [t-1]$, to an honestly generated ciphertext yields a ciphertext that is computationally indistinguishable from a fresh ciphertext that uses the same secret keys (but with all the homomorphisms $\sigma_{0,i}, \sigma_{1,i}$ applied before encryption) to encrypt the same messages (but with all homomorphisms τ_i applied to the initial messages).

This must hold even against an adversary that "knows" the homomorphisms applied in Eval, the randomness used to generate the initial ciphertext, and all randomness used in Eval except for the last call of Eval. KMH rerandomizability is strongly related to KMH correctness. However, the latter focuses on the correctness of the ciphertext homomorphisms, and thus does not make any requirements about the distribution of ciphertexts, unlike KMH rerandomizability.

Definition 25 (KMH Rerandomizability)**.** *A gate KMHE scheme* $\mathsf{GKMHE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is* KMH-rerandomizable*, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\left|2 \cdot \Pr\left[\mathsf{GKMHE\text{-}RERAND}_{\mathcal{A},\mathsf{GKMHE}}(1^\lambda) = 1\right] - 1\right| \leq \mathsf{negl}(\lambda)$$

where the GKMHE-RERAND *experiment is defined in Figure 5.3.*

Theorem 19. *Construction 2 is KMH-rerandomizable, assuming hardness of the DDH-problem (Definition 14) over the group* $\mathbb{G}$ *defined by* p*.*

For the proof of Theorem 19 we will use the following helper lemma. It roughly states that Eval in Construction 2

$\mathsf{GKMHE\text{-}RERAND}_{\mathcal{A},\mathsf{GKMHE}}(1^\lambda)$

$\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$

$(t, (\sigma_{0,i}, \sigma_{1,i}, \tau_i)_{i\in[t-1]}, m^1, m^2, m^3, m^4) \leftarrow \mathcal{A}(\mathsf{p}, 1^\lambda)$

$r^{00}, r^{01}, r^{10}, r^{11} \leftarrow\!\$\ R_{\mathsf{KGen}};\ \mathsf{sk}^{\beta_0\beta_1} \leftarrow \mathsf{KGen}(\mathsf{p}; r^{\beta_0\beta_1}) \quad \forall \beta_0, \beta_1 \in \{0,1\}$

$r_1 \leftarrow\!\$\ R_{\mathsf{Enc}};\ r_2, \ldots, r_t, r' \leftarrow\!\$\ R_{\mathsf{Eval}}$

$b \leftarrow\!\$\ \{0,1\}$

if $b = 0$ **then**

  $\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{p}, \mathsf{sk}^{00}, \mathsf{sk}^{01}, \mathsf{sk}^{10}, \mathsf{sk}^{11}, m^1, m^2, m^3, m^4; r_1)$

  $\mathsf{c}' \leftarrow \mathsf{Eval}(\mathsf{p}, \sigma_{0,t-1}, \sigma_{1,t-1}, \tau_{t-1}, \ldots \mathsf{Eval}(\sigma_{0,1}, \sigma_{1,1}, \tau_1, \mathsf{c}; r_2) \ldots; r_t)$

else

  $\mathsf{sk}'^{\beta_0\beta_1} = \sigma_{\beta_0,t-1}(\sigma_{\beta_0,t-2}(\ldots(\mathsf{sk}^{\beta_0\beta_1})\ldots)) \quad \forall \beta_0, \beta_1 \in \{0,1\}$

  $m'^i = \tau_{t-1}(\tau_{t-2}(\ldots(m^i)\ldots)) \quad \forall i \in [4]$

  $\mathsf{c}' \leftarrow \mathsf{Enc}(\mathsf{p}, \mathsf{sk}'^{00}, \mathsf{sk}'^{01}, \mathsf{sk}'^{10}, \mathsf{sk}'^{11}, m'^1, m'^2, m'^3, m'^4; r')$

endif

$b' \leftarrow \mathcal{A}(\mathsf{c}', r^{00}, r^{01}, r^{10}, r^{11}, r_1, \ldots, r_{t-1})$

return $(b' = b)$

Figure 5.3: Definition of Game GKMHE-RERAND.

– applies the given key and message homomorphisms correctly,

– permutes the ciphertexts correctly,

– and satisfies some requirements on the randomness of the resulting ciphertexts.

Essentially, it covers all requirements of the KMH rerandomizability definition except for the rerandomization of the ciphertext. For that we will use a separate argument.

Lemma 20. *Let*

$$\mathsf{c} = \left(\mathsf{c}^1, \mathsf{c}^2, \mathsf{c}^3, \mathsf{c}^4, g_{0,1}, \ldots, g_{0,\kappa}, g_{1,1}, \ldots, g_{1,\kappa}\right)$$

with

$$\mathsf{c}^i = (c_{i,1}, \ldots, c_{i,\kappa}, y_i) \quad \forall i \in [4]$$

be given by

$$\begin{pmatrix} y_{\pi_1(1)} \\ y_{\pi_1(2)} \\ y_{\pi_1(3)} \\ y_{\pi_1(4)} \end{pmatrix} = \begin{pmatrix} \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{00}} \cdot g_{1,j}^{s_j^{10}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{00}} \cdot g_{1,j}^{s_j^{11}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{10}} \cdot g_{1,j}^{s_j^{10}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{10}} \cdot g_{1,j}^{s_j^{11}}\right), h\right) \end{pmatrix},$$

$$c_{\pi_1(i),j} = \left(h^{a_{1,j}^{(i)}}, g_T^{m_j^i} \cdot y_{\pi_1(i)}^{a_{1,j}^{(i)}}\right) \quad \forall j \in [\kappa], i \in [4].$$

where

$$\begin{aligned} &(a_{1,1}^{(i)}, \ldots, a_{1,\kappa}^{(i)}) \in \mathbb{Z}_q^\kappa \quad \forall i \in [4], \\ &(g_{0,1}, \ldots, g_{0,\kappa}), (g_{1,1}, \ldots, g_{1,\kappa}) \in \mathbb{G}^\kappa, \\ &(s_1^{\beta_0\beta_1}, \ldots, s_\kappa^{\beta_0\beta_1}) \in \{0,1\}^\kappa \quad \forall \beta_0, \beta_1 \in \{0,1\}, \\ &\pi_1 \in S_4. \end{aligned}$$

Then for all $\lambda, \nu \in \mathbb{N}$; $\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$, $(\sigma_{0,n}, \sigma_{1,n}, \tau_n)_{n\in[\nu-1]} \in \mathcal{F}^{3\nu-3}$, $r_2, \ldots, r_\nu \in R_{\mathsf{Eval}}$ *with*

$$\begin{aligned} r_n = (&a_{n,1}^{(1)}, \ldots, a_{n,\kappa}^{(1)}, a_{n,1}^{(2)}, \ldots, a_{n,\kappa}^{(2)}, \\ &a_{n,1}^{(3)}, \ldots, a_{n,\kappa}^{(3)}, a_{n,1}^{(4)}, \ldots, a_{n,\kappa}^{(4)}, d_n, \pi_n) \in \mathbb{Z}_q^{4\kappa} \times \mathbb{Z}_q^* \times S_4 \quad \forall n = 2, \ldots, \nu, \end{aligned}$$

and

$$\widehat{\mathsf{c}} \leftarrow \mathsf{Eval}(\mathsf{p}, \sigma_{0,\nu-1}, \sigma_{1,\nu-1}, \tau_{\nu-1}, \ldots \mathsf{Eval}(\mathsf{p}, \sigma_{0,1}, \sigma_{1,1}, \tau_1, \mathsf{c}; r_2) \ldots; r_\nu),$$

where

$$\begin{aligned} \left(\widehat{\mathsf{c}}^1, \widehat{\mathsf{c}}^2, \widehat{\mathsf{c}}^3, \widehat{\mathsf{c}}^4, \hat{g}_{0,1}, \ldots, \hat{g}_{0,\kappa}, \hat{g}_{1,1}, \ldots, \hat{g}_{1,\kappa}\right) &= \widehat{\mathsf{c}}, \\ (\hat{c}_{i,1}, \ldots, \hat{c}_{i,\kappa}, \hat{y}_i) &= \widehat{\mathsf{c}}^i \quad \forall i \in [4] \end{aligned}$$

it holds that

$$\begin{aligned} \hat{y}_{\pi(\nu,i)} &= y_{\pi_1(i)}^d \quad \forall i \in [4], \\ \hat{g}_{\beta,\sigma_\beta(\nu-1,j)} &= g_{\beta,j}^d \quad \forall \beta \in \{0,1\}, j \in [\kappa], \\ \hat{c}_{\pi(\nu,i),\tau(\nu-1,j)} &= \left(h^{u_j^{(i)}}, g_T^{m_j^i} \cdot \hat{y}_{\pi(\nu,i)}^{u_j^{(i)}}\right) \forall i \in [4], j \in [\kappa] \end{aligned}$$

where $u_1^{(i)}, \ldots, u_\kappa^{(i)} \in \mathbb{Z}_q$ *for all* $i \in [4]$, $d \in \mathbb{Z}_q^*$ *and we denote* $\pi(n, \cdot) = \pi_n(\ldots \pi_1(\cdot) \ldots)$ *for all* $n \in [\nu]$, *and* $\sigma_{\beta_0}(n, \cdot) = \sigma_{\beta,n}(\ldots \sigma_{\beta,1}(\cdot) \ldots)$ *and* $\tau(n, \cdot) = \tau_n(\ldots \tau_1(\cdot) \ldots)$ *for all* $\beta \in \{0,1\}, n \in [\nu - 1]$.

Proof. We prove this via induction We first prove the base case $\nu = 1$. To that end, let

$$\widehat{\mathsf{c}} \leftarrow \mathsf{Eval}(\mathsf{p}, \sigma_{0,1}, \sigma_{1,1}, \tau_1, \mathsf{c}; r_2),$$

where

$$\begin{aligned}\left(\widehat{\mathsf{c}}^1, \widehat{\mathsf{c}}^2, \widehat{\mathsf{c}}^3, \widehat{\mathsf{c}}^4, \hat{g}_{0,1}, \ldots, \hat{g}_{0,\kappa}, \hat{g}_{1,1}, \ldots, \hat{g}_{1,\kappa}\right) &= \widehat{\mathsf{c}}, \\ (\hat{c}_{i,1}, \ldots, \hat{c}_{i,\kappa}, \hat{y}_i) &= \widehat{\mathsf{c}}^i \quad \forall i \in [4].\end{aligned}$$

By definition of Eval in Construction 2, $\widehat{\mathsf{c}}$ is computed as

$$\begin{aligned}\hat{y}_{\pi_2(\pi_1(i))} &= y_{\pi_1(i)}^{d_2} \quad \forall i \in [4], \\ \hat{g}_{\beta,\sigma_{\beta,1}(j)} &= g_{\beta,j}^{d_2} \quad \forall \beta \in \{0,1\}, j \in [\kappa], \\ \hat{c}_{\pi_2(\pi_1(i)),\tau(j)} &= \left(h^{\left(a_{1,j}^{(i)} + a_{2,\tau(j)}^{(\pi_1(i))}\right)/d_2}, g_T^{m_j^i} \cdot y_{\pi_1(i)}^{a_{1,j}^{(i)} + a_{2,\tau(j)}^{(\pi_1(i))}}\right) \quad \forall i \in [4], j \in [\kappa].\end{aligned}$$

By writing

$$y_{\pi_1(i)}^{a_{1,j}^{(i)} + a_{2,\tau(j)}^{(\pi_1(i))}} = \hat{y}_{\pi_2(\pi_1(i))}^{\left(a_{1,j}^{(i)} + a_{2,\tau(j)}^{(\pi_1(i))}\right)/d_2},$$

and setting

$$u_j^{(i)} = (a_{1,j}^{(i)} + a_{2,\tau(j)}^{(\pi_1(i))})/d_2 \quad \forall j \in [\kappa], i \in [4]$$

we obtain the desired result for the base case $\nu = 1$.

We now prove that if Theorem 20 holds for some $\nu = z - 1 \in \mathbb{N}$, then it holds also for $\nu = z$. For all $n \in [z]$, define

$$\mathsf{c}_n \leftarrow \mathsf{Eval}(\mathsf{p}, \sigma_{0,n-1}, \sigma_{1,n-1}, \tau_{n-1}, \ldots \mathsf{Eval}(\mathsf{p}, \sigma_{0,1}, \sigma_{1,1}, \tau_1, \mathsf{c}; r_2) \ldots; r_n),$$

where

$$\begin{aligned}\left(\mathsf{c}_n^1, \mathsf{c}_n^2, \mathsf{c}_n^3, \mathsf{c}_n^4, g_{n,0,1}, \ldots, g_{n,0,\kappa}, g_{n,1,1}, \ldots, g_{n,1,\kappa}\right) &= \mathsf{c}_n, \\ (c_{n,i,1}, \ldots, c_{n,i,\kappa}, y_{n,i}) &= \mathsf{c}_n^i \quad \forall i \in [4].\end{aligned}$$

Since we assumed that Theorem 20 holds for $\nu = z - 1$, we can write c_{z-1} as

$$\begin{aligned}y_{z-1,\pi(z-1,i)} &= y_{\pi_1(i)}^{d} \quad \forall i \in [4], \\ g_{z-1,\beta,\sigma_\beta(z-2,j)} &= g_{\beta,j}^{d} \quad \forall \beta \in \{0,1\}, j \in [\kappa], \\ c_{z-1,\pi(z-1,i),\tau(z-2,j)} &= \left(h^{u_j^{(i)}}, g_T^{m_j^i} \cdot y_{z-1,\pi(z-1,i)}^{u_j^{(i)}}\right) \forall i \in [4], j \in [\kappa]\end{aligned}$$

where $u_1^{(i)}, \ldots, u_\kappa^{(i)} \in \mathbb{Z}_q$ for all $i \in [4]$, $d \in \mathbb{Z}_q^*$. By definition of Eval in Construction 2, c_z is then given by

$$\begin{aligned}
y_{z,\pi(z,i)} &= y_{z-1,\pi(z-1,i)}^{d_z} = y_{\pi_1(i)}^{dd_z} \quad \forall i \in [4], \\
g_{z,\beta,\sigma_\beta(z-1,j)} &= g_{\beta,j}^{dd_z} \quad \forall \beta \in \{0,1\}, j \in [\kappa], \\
c_{z,\pi(z,i),\tau(z-1,j)} &= \left(h^{\left(u_j^{(i)} + a_{z,\tau(z-1,j)}^{(\pi(z-1,i))}\right)/d_z}, g_T^{m_j^i} \cdot y_{z-1,\pi(z-1,i)}^{u_j^{(i)} + a_{z,\tau(z-1,j)}^{(\pi(z-1,i))}} \right) \forall i \in [4], j \in [\kappa].
\end{aligned}$$

We rewrite

$$\begin{aligned}
c_{z,\pi(z,i),\tau(z-1,j)} &= \left(h^{\left(u_j^{(i)} + a_{z,\tau(z-1,j)}^{(\pi(z-1,i))}\right)/d_z}, g_T^{m_j^i} \cdot y_{z-1,\pi(z-1,i)}^{u_j^{(i)} + a_{z,\tau(z-1,j)}^{(\pi(z-1,i))}} \right) \\
&= \left(h^{\left(u_j^{(i)} + a_{z,\tau(z-1,j)}^{(\pi(z-1,i))}\right)/d_z}, g_T^{m_j^i} \cdot y_{z-1,\pi(z-1,i)}^{d_z\left(u_j^{(i)} + a_{z,\tau(z-1,j)}^{(\pi(z-1,i))}\right)/d_z} \right) \\
&= \left(h^{\left(u_j^{(i)} + a_{z,\tau(z-1,j)}^{(\pi(z-1,i))}\right)/d_z}, g_T^{m_j^i} \cdot y_{z,\pi(z,i)}^{\left(u_j^{(i)} + a_{z,\tau(z-1,j)}^{(\pi(z-1,i))}\right)/d_z} \right)
\end{aligned}$$

and set

$$w_j^{(i)} = \left(u_j^{(i)} + a_{z,\tau(z-1,j)}^{(\pi(z-1,i))}\right)/d \quad \forall j \in [\kappa], i \in [4].$$

Overall c_z then satisfies

$$\begin{aligned}
y_{z,\pi(z,i)} &= y_{\pi_1(i)}^{dd_z} \quad \forall i \in [4], \\
g_{z,\beta,\sigma_\beta(z-1,j)} &= g_{\beta,j}^{dd_z} \quad \forall \beta \in \{0,1\}, j \in [\kappa], \\
c_{z,\pi(z,i),\tau(z-1,j)} &= \left(h^{w_j^{(i)}}, g_T^{m_j^i} \cdot y_{z,\pi(z,i)}^{w_j^{(i)}} \right) \forall i \in [4], j \in [\kappa]
\end{aligned}$$

which completes the proof, since $dd_z \in \mathbb{Z}_q^*$ is exactly the statement of Theorem 20 for $\nu = z$. □

We can now proceed with the proof of Theorem 19.

Proof of Theorem 19. We start with case $b = 0$ of the $\mathsf{GKMHE\text{-}RERAND}_{\mathcal{A},\mathsf{GKMHE}}$ game and show that it is computationally indistinguishable from case $b = 1$.

Let c be as in case $b = 0$ of the game $\mathsf{GKMHE\text{-}RERAND}_{\mathcal{A},\mathsf{GKMHE}}$, so

$$\mathsf{c} \leftarrow \mathsf{Enc}(\mathsf{p}, \mathsf{sk}^{00}, \mathsf{sk}^{01}, \mathsf{sk}^{10}, \mathsf{sk}^{11}, m^1, m^2, m^3, m^4; r_1)$$

where

$$
\begin{aligned}
&(g_{0,1}, \ldots, g_{0,\kappa}, g_{1,1}, \ldots, g_{1,\kappa},\\
&\ a^{(1)}_{1,1}, \ldots, a^{(1)}_{1,\kappa}, a^{(2)}_{1,1}, \ldots, a^{(2)}_{1,\kappa},\\
&\ a^{(3)}_{1,1}, \ldots, a^{(3)}_{1,\kappa}, a^{(4)}_{1,1}, \ldots, a^{(4)}_{1,\kappa}, \pi_1) = r_1 \in \mathbb{G}^{2\kappa} \times \mathbb{Z}_q^{4\kappa} \times S_4,\\
&(s_1^{\beta_0\beta_1}, \ldots, s_\kappa^{\beta_0\beta_1}) = \mathsf{sk}^{\beta_0\beta_1} \in \{0,1\}^\kappa \quad \forall \beta_0, \beta_1 \in \{0,1\},\\
&(m_1^i, \ldots, m_\kappa^i) = m^i \in \{0,1\}^\kappa \quad \forall i \in [4],
\end{aligned}
$$

and

$$
\begin{aligned}
\left(\mathsf{c}^1, \mathsf{c}^2, \mathsf{c}^3, \mathsf{c}^4, g_{0,1}, \ldots, g_{0,\kappa}, g_{1,1}, \ldots, g_{1,\kappa}\right) &= \mathsf{c},\\
(c_{i,1}, \ldots, c_{i,\kappa}, y_i) &= \mathsf{c}^i \quad \forall i \in [4].
\end{aligned}
$$

Then by definition of Enc in Construction 2, c is computed as

$$
\begin{aligned}
\begin{pmatrix} y_{\pi_1(1)} \\ y_{\pi_1(2)} \\ y_{\pi_1(3)} \\ y_{\pi_1(4)} \end{pmatrix} &= \begin{pmatrix} \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{00}} \cdot g_{1,j}^{s_j^{10}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{00}} \cdot g_{1,j}^{s_j^{11}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{10}} \cdot g_{1,j}^{s_j^{10}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{s_j^{10}} \cdot g_{1,j}^{s_j^{11}}\right), h\right) \end{pmatrix},\\
c_{\pi_1(i),j} &= \left(h^{a^{(i)}_{1,j}}, g_T^{m_j^i} \cdot y_{\pi_1(i)}^{a^{(i)}_{1,j}}\right) \quad \forall i \in [4], j \in [\kappa].
\end{aligned}
$$

Now consider the $(t-2)^{\text{nd}}$ rerandomization of c, given by

$$
\hat{\mathsf{c}} \leftarrow \mathsf{Eval}(\mathsf{p}, \sigma_{0,t-2}, \sigma_{1,t-2}, \tau_{t-2}, \ldots \mathsf{Eval}(\sigma_{0,1}, \sigma_{1,1}, \tau_1, \mathsf{c}; r_2) \ldots; r_{t-1}).
$$

where we parse

$$
\begin{aligned}
&(a^{(1)}_{t-1,1}, \ldots, a^{(1)}_{t-1,\kappa}, a^{(2)}_{t-1,1}, \ldots, a^{(2)}_{t-1,\kappa},\\
&\ a^{(3)}_{t-1,1}, \ldots, a^{(3)}_{t-1,\kappa}, a^{(4)}_{t-1,1}, \ldots, a^{(4)}_{t-1,\kappa}, d_{t-1}, \pi_{t-1}) = r_{t-1} \in \mathbb{Z}_q^{4\kappa} \times \mathbb{Z}_q^* \times S_4,
\end{aligned}
$$

and

$$
\begin{aligned}
\left(\hat{\mathsf{c}}^1, \hat{\mathsf{c}}^2, \hat{\mathsf{c}}^3, \hat{\mathsf{c}}^4, \hat{g}_{0,1}, \ldots, \hat{g}_{0,\kappa}, \hat{g}_{1,1}, \ldots, \hat{g}_{1,\kappa}\right) &= \hat{\mathsf{c}},\\
(\hat{c}_{i,1}, \ldots, \hat{c}_{i,\kappa}, \hat{y}_i) &= \hat{\mathsf{c}}^i \quad \forall i \in [4].
\end{aligned}
$$

Furthermore, define notation $\pi(n,\cdot) = \pi_n(\dots\pi_1(\cdot)\dots)$ for all $n \in [\nu]$, and $\sigma_{\beta_0}(n,\cdot) = \sigma_{\beta,n}(\dots\sigma_{\beta,1}(\cdot)\dots)$ and $\tau(n,\cdot) = \tau_n(\dots\tau_1(\cdot)\dots)$ for all $\beta \in \{0,1\}, n \in [\nu-1]$ as in Theorem 20.

By applying Theorem 20 with $\nu = t-1$, we can write $\widehat{\mathsf{c}}$ as

$$\begin{aligned}
\hat{y}_{\pi(t-1,i)} &= y^d_{\pi_1(i)} \quad \forall i \in [4],\\
\hat{g}_{\beta,\sigma_\beta(t-2,j)} &= g^d_{\beta,j} \quad \forall \beta \in \{0,1\}, j \in [\kappa],\\
\hat{c}_{\pi(t-1,i),\tau(t-2,j)} &= \left(h^{u^{(i)}_j}, g_T^{m^i_j} \cdot \hat{y}^{u^{(i)}_j}_{\pi(t-1,i)}\right) \forall i \in [4], j \in [\kappa],
\end{aligned}$$

where $u^{(i)}_1, \dots, u^{(i)}_\kappa \in \mathbb{Z}_q$ for all $i \in [4]$, and $d \in \mathbb{Z}^*_q$.

We now show that the last call to Eval where the adversary $\mathcal{A}$ does *not* obtain the randomness r_t suffices to rerandomize $\widehat{\mathsf{c}}$. The full challenge ciphertext in case $b = 0$ is given by

$$\mathsf{c}' \leftarrow \mathsf{Eval}(\mathsf{p}, \sigma_{0,t-1}, \sigma_{1,t-1}, \tau_{t-1}, \widehat{\mathsf{c}}; r_t).$$

where we parse

$$\begin{aligned}
&(a^{(1)}_{t,1}, \dots, a^{(1)}_{t,\kappa}, a^{(2)}_{t,1}, \dots, a^{(2)}_{t,\kappa},\\
&\ a^{(3)}_{t,1}, \dots, a^{(3)}_{t,\kappa}, a^{(4)}_{t,1}, \dots, a^{(4)}_{t,\kappa}, d_t, \pi_t) = r_t \in \mathbb{Z}_q^{4\kappa} \times Z_q^* \times S_4,\\
&(\mathsf{c}'^1, \mathsf{c}'^2, \mathsf{c}'^3, \mathsf{c}'^4, g'_{0,1}, \dots, g'_{0,\kappa}, g'_{1,1}, \dots, g'_{1,\kappa}) = \mathsf{c}',\\
&(c'_{i,1}, \dots, c'_{i,\kappa}, y'_i) = \mathsf{c}'^i \quad \forall i \in [4].
\end{aligned}$$

Again, by definition of Eval in Construction 2, c' is computed as

$$\begin{aligned}
y'_{\pi(t,i)} &= \hat{y}^{d_t}_{\pi(t-1,i)} = y^{dd_t}_{\pi_1(i)}\\
g'_{\beta,\sigma_\beta(t-1,j)} &= \hat{g}^{d_t}_{\beta,\sigma_\beta(t-2,j)} = g^{dd_t}_{\beta,j} \quad \forall \beta \in \{0,1\},\\
c'_{\pi(t,i),\tau(t-1,j)} &= \left(h^{\left(u^{(i)}_j + a^{(\pi(t-1,i))}_{t,\tau(t-1,j)}\right)/d_t}, g_T^{m^i_j} \cdot \hat{y}^{u^{(i)}_j + a^{(\pi(t-1,i))}_{t,\tau(z-1,j)}}_{\pi(t-1,i)}\right)\\
&= \left(h^{\left(u^{(i)}_j + a^{(\pi(t-1,i))}_{t,\tau(t-1,j)}\right)/d_t}, g_T^{m^i_j} \cdot y'^{\left(u^{(i)}_j + a^{(\pi(t-1,i))}_{t,\tau(z-1,j)}\right)/d_t}_{\pi(t,i)}\right)
\end{aligned}$$

for all $i \in [4], j \in [\kappa]$. Since the r_t value is sampled uniformly at random from R_{Eval} and never given to the adversary $\mathcal{A}$, each $a^{(\pi(t-1,i))}_{t,\tau(t-1,j)}$, for $j \in [\kappa], i \in [4]$, looks uniformly random to $\mathcal{A}$. Then also the value $(u^{(i)}_j + a^{(\pi(t-1,i))}_{t,\tau(z-1,j)})/d_t$ looks uniformly random to $\mathcal{A}$. Therefore we get that

$$c'_{\pi(t,i),\tau(t-1,j)} \equiv \left(h^{w^{(i)}_j}, g_T^{m^i_j} \cdot y'^{w^{(i)}_j}_{\pi(t,i)}\right) \quad \forall i \in [4], j \in [\kappa],$$

where $w_j^{(i)} \leftarrow\!\!\$\, \mathbb{Z}_q$ for all $j \in [\kappa], i \in [4]$.

It remains to prove that the remaining elements $y'_{\pi(t,i)}$, $i \in [4]$, and $g'_{\beta,\sigma_\beta(t-1,j)}$, $\beta \in \{0,1\}$, $j \in [\kappa]$, look fresh as well. We rewrite

$$\begin{pmatrix} y'_{\pi(t,1)} \\ y'_{\pi(t,2)} \\ y'_{\pi(t,3)} \\ y'_{\pi(t,4)} \end{pmatrix} = \begin{pmatrix} \hat{y}^{d_t}_{\pi(t-1,1)} \\ \hat{y}^{d_t}_{\pi(t-1,2)} \\ \hat{y}^{d_t}_{\pi(t-1,3)} \\ \hat{y}^{d_t}_{\pi(t-1,4)} \end{pmatrix} = \begin{pmatrix} y^{dd_t}_{\pi_1(1)} \\ y^{dd_t}_{\pi_1(2)} \\ y^{dd_t}_{\pi_1(3)} \\ y^{dd_t}_{\pi_1(4)} \end{pmatrix}$$
$$= \begin{pmatrix} \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{dd_t s_j^{00}} \cdot g_{1,j}^{dd_t s_j^{10}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{dd_t s_j^{00}} \cdot g_{1,j}^{dd_t s_j^{11}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{dd_t s_j^{10}} \cdot g_{1,j}^{dd_t s_j^{10}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(g_{0,j}^{dd_t s_j^{10}} \cdot g_{1,j}^{dd_t s_j^{11}}\right), h\right) \end{pmatrix}$$

Since

- d_t stays hidden from $\mathcal{A}$ and is uniformly distributed over $\mathbb{Z}_q^*$ due to $r_t \leftarrow\!\!\$\, R_{\mathsf{Eval}}$,
- and each $g_{\beta,j}$, $\beta \in \{0,1\}, j \in [\kappa]$, is uniformly distributed over $\mathbb{G}$ due to $r_1 \leftarrow\!\!\$\, R_{\mathsf{Enc}}$.

we can now apply generalized DDH (Theorem 7) over the group $\mathbb{G}$ to obtain

$$\begin{aligned} &\left\{g_{0,1}, \ldots, g_{0,\kappa}, g_{1,1}, \ldots g_{1,\kappa}, g_{0,1}^{d_t}, \ldots, g_{0,\kappa}^{d_t}, g_{1,1}^{d_t}, \ldots g_{1,\kappa}^{d_t}\right\} \\ \overset{\mathrm{c}}{\approx}\ &\left\{g_{0,1}, \ldots, g_{0,\kappa}, g_{1,1}, \ldots g_{1,\kappa}, v_{0,1}, \ldots, v_{0,\kappa}, v_{1,1}, \ldots v_{1,\kappa}\right\}, \end{aligned} \tag{5.10}$$

where $v_{\beta,j} \leftarrow\!\!\$\, \mathbb{G}$ for all $\beta \in \{0,1\}, j \in [\kappa]$. Using Equation (5.10) we obtain

$$g'_{\beta,\sigma_\beta(t-1,j)} = g_{\beta,j}^{dd_t} \overset{\mathrm{c}}{\approx} v_{\beta,j}^{d} \quad \forall \beta \in \{0,1\}, j \in [\kappa],$$

and

$$\begin{pmatrix} y'_{\pi(t,1)} \\ y'_{\pi(t,2)} \\ y'_{\pi(t,3)} \\ y'_{\pi(t,4)} \end{pmatrix} \overset{\mathrm{c}}{\approx} \begin{pmatrix} \mathrm{e}\left(\prod_{j\in[\kappa]} \left(v_{0,j}^{ds_j^{00}} \cdot v_{1,j}^{ds_j^{10}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(v_{0,j}^{ds_j^{00}} \cdot v_{1,j}^{ds_j^{11}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(v_{0,j}^{ds_j^{10}} \cdot v_{1,j}^{ds_j^{10}}\right), h\right) \\ \mathrm{e}\left(\prod_{j\in[\kappa]} \left(v_{0,j}^{ds_j^{10}} \cdot v_{1,j}^{ds_j^{11}}\right), h\right) \end{pmatrix}.$$

Now let

$$\begin{aligned} \underline{r} = (&v_{0,1}^d, \ldots, v_{0,\kappa}^d, v_{1,1}^d, \ldots, v_{1,\kappa}^d, \\ &w_1^{(1)}, \ldots, w_\kappa^{(1)}, w_1^{(2)}, \ldots, w_\kappa^{(2)}, \\ &w_1^{(3)}, \ldots, w_\kappa^{(3)}, w_1^{(4)}, \ldots, w_1^{(4)}, \pi_t \circ \cdots \circ \pi_1) \in \mathbb{G}^{2\kappa} \times \mathbb{Z}_q^{4\kappa} \times S_4. \end{aligned}$$

By the previous arguments we obtain that

$$\mathsf{c}' \overset{\mathrm{c}}{\approx} \mathsf{Enc}(\mathsf{p}, \mathsf{sk}'^{00}, \mathsf{sk}'^{01}, \mathsf{sk}'^{10}, \mathsf{sk}'^{11}, m'^{1}, m'^{2}, m'^{3}, m'^{4}; \underline{r}),$$

where

$$\begin{aligned} \mathsf{sk}'^{\beta_0\beta_1} &= \sigma_{\beta_0,t-1}(\sigma_{\beta_0,t-2}(\ldots(\mathsf{sk}^{\beta_0\beta_1})\ldots)) \quad \forall \beta_0, \beta_1 \in \{0,1\} \\ m'^{i} &= \tau_{t-1}(\tau_{t-2}(\ldots(m^i)\ldots)) \quad \forall i \in [4] \end{aligned}$$

as in case $b = 1$ of the game $\mathsf{GKMHE\text{-}RERAND}_{\mathcal{A},\mathsf{GKMHE}}$.

It remains to show that $\underline{r} \equiv r'$, as required for the case $b = 1$ of the game $\mathsf{GKMHE\text{-}RERAND}_{\mathcal{A},\mathsf{GKMHE}}$. Note that the $v^d_{\beta,j}$ are uniformly distributed over $\mathbb{G}$ since exponentiation with d is a bijection. Furthermore, the composition of permutations $\pi_t \circ \cdots \circ \pi_1$ is uniformly distributed over S_4 since $\pi_t \leftarrow\!\!\$\ S_4$ is uniform. Overall we get that $\underline{r}$ is uniformly distributed over R_{Eval}, so $\underline{r} \equiv r'$ as desired. □

5.4.8 Key privacy under leakage

Our construction of RGS from gate KMHE will require another strong security property of the gate KMHE scheme that we call *key privacy under leakage*. Intuitively, key privacy under leakage guarantees that a ciphertext that was evaluated with key homomorphisms σ_0 and σ_1 is indistinguishable from a fresh ciphertext, even given some information about the homomorphisms. Concretely, for our RGS construction we will have to consider a setting where the adversary first generates and outputs a ciphertext (and thus knows all four secret keys and the encrypted messages), and then random key homomorphisms σ_0 and σ_1 are applied, hidden from the adversary, to rerandomize the keys of the original ciphertext. Subsequently, the adversary receives back the resulting ciphertext. We want that the resulting ciphertext is computationally indistinguishable from a ciphertext encrypted with *independent* keys.

There are two complications:

1. We have to consider a setting where the same key homomorphism $\sigma \in \{\sigma_0, \sigma_1\}$ is applied to *multiple* gate KMHE ciphertexts. This is because in the RGS construction secret keys and homomorphisms are chosen for *wires* of the garbled circuit; the same wire might be an input to $Q \geq 1$ gates in a circuit, and thus the same keys and homomorphisms are used in Q gate KMHE ciphertexts. These Q ciphertexts might encrypt different messages and use different secret keys on their other input wires. Therefore we have to consider a setting where the adversary outputs the keys $(\mathsf{sk}^0, \mathsf{sk}^1)$ shared across all Q ciphertexts, along with specifications α_i, $i \in [Q]$, whether these correspond to the "left" or "right" input wire of the i^{th} gate, as well as two additional secret keys and four messages for each of the $Q \geq 1$ ciphertexts.

2. Along with the resulting challenge ciphertexts, the adversary (which evaluates the garbled circuit) will also receive one of two rerandomized secret keys of the wire. This *leaks* some information about the homomorphism σ. Therefore we have to consider a setting where the adversary outputs secret keys sk^0, sk^1 and receives the rerandomized wire key $\sigma(\mathsf{sk}^\beta)$ as additional leakage about σ along with the challenge ciphertexts. In the challenge ciphertexts, either the rerandomization $\sigma(\mathsf{sk}^{1-\beta})$ of the other wire key $\mathsf{sk}^{1-\beta}$ is used, or an independent random key sk^*, and the adversary has to distinguish between these two cases.

Taking these requirements into account yields the key privacy under leakage security experiment in Figure 5.4. Note that the figure actually defines two security experiments, one for each $\beta \in \{0,1\}$.

Definition 26 (Key Privacy under Leakage)**.** *We say that a gate KMHE scheme* $\mathsf{GKMHE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *has* key privacy under leakage*, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all* $\beta \in \{0,1\}$ *it holds that*

$$\left|2 \cdot \Pr\left[\mathsf{GKMHE\text{-}KPUL}^{\beta}_{\mathcal{A},\mathsf{GKMHE}}(1^\lambda) = 1\right] - 1\right| \leq \mathsf{negl}(\lambda)$$

where the GKMHE-KPUL *experiment is defined in Figure 5.4.*

Theorem 21. *Construction 2 has key privacy under leakage (Definition 26), assuming hardness of the DDH-problem (Definition 14) in the group* $\mathbb{G}$ *defined by* p.

Building blocks of the proof. The following definitions are taken from [DRS04] and were used in a similar context by Naor and Segev [NS09] to prove key-leakage resilience of the circular-secure encryption scheme of [BHHO08].

The *conditional min-entropy* of a random variable X, conditioned on some information y, is defined as $H_\infty(X \mid y) = -\log\left(\max_{x \in \mathrm{supp}(X)} \Pr[X = x \mid y]\right)$. The *average min-entropy* of a random variable X conditioned on the value of a random variable Y is then defined as

$$\tilde{H}_\infty(X \mid Y) = -\log \mathcal{E}_{y \leftarrow Y}\left[2^{-H_\infty(X|y)}\right].$$

We use the following instantiation of the generalized left-over hash lemma from [DRS04], which uses that the inner-product over a field $\mathbb{Z}_q$ is a pairwise independent hash function (see also [NS09]).This variant of the left-over hash lemma also takes leakage of the secret into account.

$\mathsf{GKMHE\text{-}KPUL}^{\beta}_{\mathcal{A},\mathsf{GKMHE}}(1^\lambda)$:

$\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$

$(\mathsf{sk}^0, \mathsf{sk}^1, (m_i^{00}, m_i^{01}, m_i^{10}, m_i^{11}, \alpha_i, \mathsf{sk}_i^{1-\alpha_i,0}, \mathsf{sk}_i^{1-\alpha_i,1})_{i\in[Q]}) \leftarrow \mathcal{A}(\mathsf{p}, 1^\lambda)$

$\sigma \leftarrow\!\!\$\ \mathcal{F};\ b \leftarrow\!\!\$\ \{0,1\};\ \mathsf{sk}^* \leftarrow \mathsf{KGen}(\mathsf{p})$

if $b = 0$ **then**

  $\widehat{\mathsf{sk}}_i^{\alpha_i,1-\beta} \leftarrow \sigma(\mathsf{sk}^{1-\beta}) \quad \forall i \in [Q]$

else

  $\widehat{\mathsf{sk}}_i^{\alpha_i,1-\beta} \leftarrow \mathsf{sk}^* \quad \forall i \in [Q]$

endif

$\widehat{\mathsf{sk}}_i^{\alpha_i,\beta} \leftarrow \sigma(\mathsf{sk}^{\beta}) \quad \forall i \in [Q]$

$\widehat{\mathsf{sk}}_i^{1-\alpha_i,j} \leftarrow \mathsf{sk}_i^{1-\alpha_i,j}$ for $\forall i \in [Q]$ and $j \in \{0,1\}$

$\mathsf{c}_i \leftarrow \mathsf{Enc}(\mathsf{p}, \widehat{\mathsf{sk}}_i^{0,0}, \widehat{\mathsf{sk}}_i^{0,1}, \widehat{\mathsf{sk}}_i^{1,0}, \widehat{\mathsf{sk}}_i^{1,1}, m_i^{00}, m_i^{01}, m_i^{10}, m_i^{11}) \quad \forall i \in [Q]$

$b' \leftarrow \mathcal{A}\left(\sigma(\mathsf{sk}^{\beta}), (\mathsf{c}_i)_{i\in[Q]}\right)$

return $(b' = b)$

Figure 5.4: Definition of Game $\mathsf{GKMHE\text{-}KPUL}^{\beta}$.

Lemma 22 (LHL with leakage)**.** *Let q be a prime, $\kappa \in \mathbb{N}$, $\epsilon > 0$, and let $X = (X_1, \ldots, X_\kappa) \in \{0,1\}^\kappa$ and Y be random variables such that $\tilde{H}_\infty(X \mid Y) \geq \log q + 2\log(1/\epsilon)$. Then for $a \leftarrow\!\!\$\ \mathbb{Z}_q^\kappa$ it holds that*

$$\Delta((a, \sum_{i\in[\kappa]} a_i X_i, Y), (a, U(\mathbb{Z}_q), Y)) \leq \epsilon.$$

where $U(\mathbb{Z}_q)$ is the uniform distribution over $\mathbb{Z}_q$.

The following lemma from [GHV10] gives a lower-bound on the average-min entropy of a permuted key, even if some leakage about the key transformation function is present.

Lemma 23 (Lemma 9 in [GHV10])**.** *Let $\kappa \geq 2$ be an even integer, let $\mathsf{sk}_1, \mathsf{sk}_2 \leftarrow\!\!\$\ HW_{\kappa,\kappa/2}$ and $\sigma \leftarrow\!\!\$\ S_\kappa$. Then*

$$H_\infty(\sigma(\mathsf{sk}_1) \mid \mathsf{sk}_1, \mathsf{sk}_2, \sigma(\mathsf{sk}_2)) \geq \kappa - \frac{3}{2}\log\kappa.$$

Finally, we will use a variant of Theorem 14, extended with leakage. The proof of the following lemma follows almost exactly the proof of the non-leakage variant

Theorem 14, except that we replace the use of the LHL (Theorem 12) with the leakage variant (Theorem 22).

Lemma 24. *Let $\eta, \nu \in \mathbb{N}$, $\mathbf{g}_1, \ldots, \mathbf{g}_\eta \leftarrow\!\!\$\ \mathbb{G}^\nu$, where $\mathbb{G}$ is a group of prime order q where the DDH assumption (Definition 14) holds, Y be a random variable, and $\mathsf{sk} \in \{0,1\}^\nu$ such that $\tilde{H}_\infty(\mathsf{sk} \mid Y) \geq \log q + 2\lambda$. Then it holds that*

$$\left\{\mathbf{g}_1, \ldots, \mathbf{g}_\eta, \mathbf{g}_1^{\mathsf{sk}}, \ldots, \mathbf{g}_\eta^{\mathsf{sk}}, Y\right\} \stackrel{c}{\approx} \left\{\mathbf{g}_1, \ldots, \mathbf{g}_\eta, g'_1, \ldots, g'_\eta, Y\right\},$$

where $g'_1, \ldots, g'_\eta \leftarrow\!\!\$\ \mathbb{G}$.

Using these lemmas we can now proceed with proving key privacy under leakage of Construction 2.

Proof of Theorem 21. Similar to the proof of IND-CPA security we will only consider the case $\beta = 1$, so the key $\sigma(\mathsf{sk}^0)$ remains secret and the adversary $\mathcal{A}$ is given $\sigma(\mathsf{sk}^1)$. The case $\beta = 0$ follows analogously. Furthermore, we assume that $\alpha_1 = \cdots = \alpha_Q = 0$ for all the α_i specified by the adversary, in order to simplify our notation. Since the gate KMHE scheme treats the keys assigned to a "left" or "right" input wire identically, this is without loss of generality.

We describe a series of hybrid distributions from case $b = 0$ to case $b = 1$ of the game $\mathsf{GKMHE\text{-}KPUL}^0_{\mathcal{A},\mathsf{GKMHE}}$. We will then prove these hybrids indistinguishable.

Game$_1$: Case $b = 0$ of the game $\mathsf{GKMHE\text{-}KPUL}^0_{\mathcal{A},\mathsf{GKMHE}}$. We write $\mathsf{c}_1, \ldots, \mathsf{c}_Q$, where

$$\mathsf{c}_i = \left(\mathsf{c}_i^{00}, \mathsf{c}_i^{01}, \mathsf{c}_i^{10}, \mathsf{c}_i^{11}, g_{i,0,1}, \ldots, g_{i,0,\kappa}, g_{i,1,1}, \ldots, g_{i,1,\kappa}\right)$$

for $i \in [Q]$, to denote the challenge ciphertexts given to the adversary. Furthermore, we write $\mathsf{c}_i^{\delta_0\delta_1} = (c_{i,\delta_0\delta_1,1}, \ldots, c_{i,\delta_0\delta_1,\kappa}, y_{i,\delta_0\delta_1})$ for all $i \in [Q], \delta_0, \delta_1 \in \{0,1\}$ and define

$$\vec{g}_{i,0} = (g_{i,0,1}, \ldots, g_{i,0,\kappa}) \qquad \text{and} \qquad \vec{g}_{i,1} = (g_{i,1,1}, \ldots, g_{i,1,\kappa})$$

for all $i \in [Q]$.

Game$_2$: For **Game$_2$**, we alter how each c_i, $i \in [Q]$ is created by Enc, specifically how the values $y_{i,00}$ and $y_{i,01}$ are computed. To this end, parse the permuted keys as $\sigma(\mathsf{sk}^{0,0}) = (s_1, \ldots, s_\kappa)$. Then in the computation of

$$\mathsf{c}_i \leftarrow \mathsf{Enc}(\mathsf{p}, \sigma(\mathsf{sk}^0), \widehat{\mathsf{sk}}_i^{0,1}, \widehat{\mathsf{sk}}_i^{1,0}, \widehat{\mathsf{sk}}_i^{1,1}, m_i^{00}, m_i^{01}, m_i^{10}, m_i^{11}),$$

during computation of c_i, samples $g'_i \leftarrow\!\!\$\ \mathbb{G}$ and then computes

$$y_{i,00} = \mathrm{e}\left(g'_i \cdot \vec{g}_{i,1}^{\widehat{\mathsf{sk}}_i^{1,0}}, h\right),$$
$$y_{i,01} = \mathrm{e}\left(g'_i \cdot \vec{g}_{i,1}^{\widehat{\mathsf{sk}}_i^{1,1}}, h\right).$$

It then proceeds with the rest of the computation of c_i.

Game$_3$: Case $b = 1$ of the game $\mathsf{GKMHE\text{-}KPUL}^{0.}_{\mathcal{A},\mathsf{GKMHE}}$.

We now show that the hybrids are indistinguishable.

Game$_1$ $\overset{\mathsf{c}}{\approx}$ Game$_2$: The proof of this step follows largely the proof of indistinguishability of **Game**$_1$ and **Game**$_2$ in the proof of IND-CPA security (Theorem 17), with a slight difference in how we lower-bound the entropy of the secret key $\sigma(\mathsf{sk}^0)$ due to the additional leakage.

In **Game**$_1$ each $y_{i,00}$ and $y_{i,01}$ in c_i is computed as

$$y_{i,00} = \mathrm{e}\left(\vec{g}_{i,0}^{\sigma(\mathsf{sk}^0)} \cdot \vec{g}_{i,1}^{\widehat{\mathsf{sk}}_i^{1,0}}, h\right),$$
$$y_{i,01} = \mathrm{e}\left(\vec{g}_{i,0}^{\sigma(\mathsf{sk}^0)} \cdot \vec{g}_{i,1}^{\widehat{\mathsf{sk}}_i^{1,1}}, h\right).$$

for all $i \in [Q]$. Our choice of κ and Theorem 23 ensures that

$$\tilde{H}_\infty\left(\sigma(\mathsf{sk}^0) \mid \mathsf{sk}^0, \mathsf{sk}^1, \sigma(\mathsf{sk}^1)\right) \geq \kappa - \frac{3}{2}\log\kappa \geq \log q + 2\lambda.$$

Since DDH holds over $\mathbb{G}$, we can then apply Theorem 24, resulting in

$$\left\{\vec{g}_{1,0}, \ldots, \vec{g}_{Q,0}, \vec{g}_{1,0}^{\sigma(\mathsf{sk}^0)}, \ldots, \vec{g}_{Q,0}^{\sigma(\mathsf{sk}^0)}\right\} \overset{\mathrm{c}}{\approx} \left\{\vec{g}_{1,0}, \ldots, \vec{g}_{Q,0}, g'_1, \ldots, g'_Q\right\}. \tag{5.11}$$

where $g'_1, \ldots, g'_Q \leftarrow\!\!\$\, \mathbb{G}$. The right side of Equation (5.11) thus corresponds to **Game**$_2$.

Game$_2$ $\overset{\mathsf{c}}{\approx}$ Game$_3$: To insert a fresh key sk^*, we now apply Theorem 14. By our choice of κ and Theorem 13 we then have

$$H_\infty(\mathsf{sk}^*) \geq \kappa - \frac{3}{2}\log\kappa \geq \kappa - \frac{1}{2}\log\kappa - 1/2 \geq \log q + 2\lambda.$$

We can then apply Theorem 14, resulting in

$$\left\{\vec{g}_{1,0}, \ldots, \vec{g}_{Q,0}, g'_1, \ldots, g'_Q\right\} \overset{\mathrm{c}}{\approx} \left\{\vec{g}_{1,0}, \ldots, \vec{g}_{Q,0}, \vec{g}_{1,0}^{\mathsf{sk}^*}, \ldots, \vec{g}_{Q,0}^{\mathsf{sk}^*}\right\}. \tag{5.12}$$

The right side of Equation (5.12) then corresponds to **Game**$_3$.

This completes the proof. □

5.5 Rerandomizable Garbling Schemes

We define RGS, mostly following [GHV10, AHKP22]. The main difference is a new definition of rerand privacy plus a Setup algorithm to generate public parameters.

Let $(\mathcal{F}^\lambda_{\mathsf{RGS}})_{\lambda\in\mathbb{N}}$ be a family of function sets $\mathcal{F}^\lambda_{\mathsf{RGS}}$ where $\mathcal{F}^\lambda_{\mathsf{RGS}} = \{f : \mathcal{X} \to \mathcal{Y}\}$ is a set of functions f. Additionally, we define leakage function family $(\phi_\lambda)_{\lambda\in\mathbb{N}}$ to be a family of functions $\phi_\lambda : \mathcal{F}^\lambda_{\mathsf{RGS}} \to \{0,1\}^*$ which map a function $f \in \mathcal{F}^\lambda_{\mathsf{RGS}}$ to the allowed leakage of the garbling scheme $\phi_\lambda(f)$.

Definition 27 (Rerandomizable Garbling Scheme)**.** *A RGS for $(\mathcal{F}^\lambda_{\mathsf{RGS}})_{\lambda\in\mathbb{N}}$ and leakage function family $(\phi_\lambda)_{\lambda\in\mathbb{N}}$ is a tuple of PPT algorithms* $\mathsf{RGS} = (\mathsf{Setup}, \mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ *with the following syntax.*

$\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$**:** *This algorithm takes as input the security parameter λ and returns public parameters p. The public parameters define a label space $\mathcal{L}$, function family $\mathcal{F}^\lambda_{\mathsf{RGS}}$ and leakage function ϕ_λ.*

$(\mathcal{C}, L) \leftarrow \mathsf{Gb}(\mathsf{p}, f)$**:** *Given a function $f \in \mathcal{F}^\lambda_{\mathsf{RGS}}$, this algorithm returns a garbled circuit $\mathcal{C}$ and a set of input labels $L \in \mathcal{L}$.*

$(\mathcal{C}', \pi_{\mathsf{En}}) \leftarrow \mathsf{Rerand}(\mathsf{p}, \mathcal{C})$**:** *Given a garbled circuit $\mathcal{C}$, this algorithm returns a garbled circuit $\mathcal{C}'$ and a label encoding function $\pi_{\mathsf{En}} : \mathcal{L} \to \mathcal{L}$.*

$I \leftarrow \mathsf{En}(\mathsf{p}, L, x)$**:** *This algorithm takes a set of input labels L and an input $x \in \mathcal{X}$, and outputs an input encoding I.*

$y \leftarrow \mathsf{Ev}(\mathsf{p}, \mathcal{C}, I)$**:** *Given a garbled circuit $\mathcal{C}$ and an input encoding I, this algorithm outputs a $y \in \mathcal{Y}$.*

An RGS must fulfill correctness, garbling privacy, and rerand privacy as defined below. As pointed out in [AHKP22], correctness and rerand privacy together imply correctness of rerandomization.

Definition 28 (Correctness)**.** *We say that* RGS *is correct, if for all $\lambda \in \mathbb{N}$, $\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$, $f \in \mathcal{F}^\lambda_{\mathsf{RGS}}$, and $x \in \mathcal{X}$ there exists a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\Pr\left[y = f(x) \;\middle|\; \begin{array}{c} (\mathcal{C}, L) \leftarrow \mathsf{Gb}(\mathsf{p}, f) \\ I \leftarrow \mathsf{En}(\mathsf{p}, L, x) \\ y = \mathsf{Ev}(\mathsf{p}, \mathcal{C}, I) \end{array} \right] \geq 1 - \mathsf{negl}(\lambda).$$

Just like for standard garbled circuits, garbling privacy ensures that the garbled function f and circuit input x remain hidden, except for the function output $f(x)$ and the allowed leakage $\phi_\lambda(f)$.

$\mathsf{RER\text{-}PRIV}_{\mathcal{A},\mathsf{RGS}}(1^\lambda)$

$\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda);\ (f, x, 1^t) \leftarrow \mathcal{A}(\mathsf{p}, 1^\lambda)$
$r_1 \leftarrow\!\$\ R_{\mathsf{Gb}};\ (r_2, \ldots, r_{t-1}) \leftarrow\!\$\ R_{\mathsf{Rerand}};\ b \leftarrow\!\$\ \{0,1\}$
if $b = 0$ **then**
  $r_t \leftarrow\!\$\ R_{\mathsf{Rerand}};\ (\mathcal{C}_1, L_1) \leftarrow \mathsf{Gb}(\mathsf{p}, f; r_1)$
  $(\mathcal{C}_{i+1}, \pi^{i+1}_{\mathsf{En}}) \leftarrow \mathsf{Rerand}(\mathsf{p}, \mathcal{C}_i; r_{i+1})$ for $i \in [t-1]$
  $L \leftarrow \pi^t_{\mathsf{En}}(\cdots \pi^2_{\mathsf{En}}(L_1) \cdots);\ I \leftarrow \mathsf{En}(L, x); \mathcal{C} \leftarrow \mathcal{C}_t$
else
  $(\mathcal{C}, L) \leftarrow \mathsf{Gb}(\mathsf{p}, f);\ I \leftarrow \mathsf{En}(L, x)$
endif
$b' \leftarrow \mathcal{A}((r_1, \ldots, r_{t-1}), \mathcal{C}, I)$
return $(b' = b)$

Figure 5.5: Definition of Game RER-PRIV.

Definition 29 (Garbling Privacy)**.** $\mathsf{RGS} = (\mathsf{Setup}, \mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ *has garbling privacy, if there exists an efficient simulator* $\mathsf{Sim}_{\mathsf{Gb}}$ *such that for all* $\mathsf{p} \leftarrow \mathsf{Setup}(1^\lambda)$*, and for every* $f \in \mathcal{F}^\lambda_{\mathsf{RGS}}$*,* $x \in \mathcal{X}$ *it holds that*

$$\{\mathcal{C}, I\}_{\substack{(\mathcal{C},L)\leftarrow\mathsf{Gb}(\mathsf{p},f)\\ I\leftarrow\mathsf{En}(\mathsf{p},L,x)}} \overset{c}{\approx} \{\mathsf{Sim}_{\mathsf{Gb}}(\mathsf{p}, f(x), \phi_\lambda(f))\}\,.$$

Similar to KMH rerandomizability of gate KMHE, rerand privacy of RGS guarantees that rerandomized garbled circuits are indistinguishable from fresh garbled circuits.

Definition 30 (Rerand Privacy)**.** *We say that* $\mathsf{RGS} = (\mathsf{Setup}, \mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ *has rerand privacy, if for any PPT adversary* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\Pr[\mathsf{RER\text{-}PRIV}_{\mathcal{A},\mathsf{RGS}}(1^\lambda) = 1] \leq 1/2 + \mathsf{negl}(\lambda),$$

where the RER-PRIV *experiment is defined in Figure 5.5.*

5.5.1 RGS from gate KMHE

We proceed by constructing the RGS scheme. It largely follows Construction 1 in [AHKP22], which in turn follows standard garbled circuits [Yao86, LP09], but using gate KMHE instead of the sharable KMHE from [AHKP22].

Let $\mathsf{GKMHE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be a gate KMHE scheme with associated key space $\mathcal{K}$. We also define

– the function family $\mathcal{F}^{\lambda}_{\mathsf{RGS}} = \{f : \{0,1\}^{\ell_{\mathsf{in}}(f)} \to \{0,1\}^{\ell_{\mathsf{out}}(f)}\}$ as the set of Boolean circuits of size polynomial in 1^{λ}, and

– label space $\mathcal{L} = \mathcal{K}^{2\ell_{\mathsf{in}}}$.

We generally consider Boolean circuits consisting of binary gates with two inputs and one output. The fan-out is arbitrary, meaning that the output wire can be connected to arbitrarily many gates. Given a circuit $f \in \mathcal{F}^{\lambda}_{\mathsf{RGS}}$, we use $\mathcal{W}$ to denote the list of wires, $\mathcal{W}_{\mathcal{O}}$ the list of output wires, and $\mathcal{W}_I$ the list of input wires. M denotes the number of wires and N the number of gates. A gate g_i is described by a tuple $g_i = (w_\ell, w_r, w_o, op)$, where w_ℓ and w_r are the left and right input wires, respectively, w_o the output wire and op is the gate function, i.e. any function $op : \{0,1\} \times \{0,1\} \mapsto \{0,1\}$. The leakage function ϕ_λ, given a Boolean circuit, outputs its topology, meaning everything but the gate operation op of each gate.

Construction 3 (RGS from Gate KMHE)**.** $\mathsf{Setup}(1^\lambda)$**:** Generate GKMHE public parameters $\mathsf{p}_{\mathsf{GKMHE}} \leftarrow \mathsf{Setup}(1^\lambda)$ and sample output labels $\mathsf{sk}_0, \mathsf{sk}_1 \leftarrow \mathsf{KGen}(\mathsf{p}_{\mathsf{GKMHE}})$. Output

$$\mathsf{p} \leftarrow (\mathsf{p}_{\mathsf{GKMHE}}, \mathsf{sk}_0, \mathsf{sk}_1)\,.$$

$\mathsf{Gb}(\mathsf{p}, f)$**:** Parse f as a series of boolean gates $g_1, \ldots, g_N$, ordered topologically. For every wire $w_i \in \mathcal{W} \setminus \mathcal{W}_{\mathcal{O}}$ sample labels $\mathsf{sk}^0_{w_i}, \mathsf{sk}^1_{w_i} \leftarrow \mathsf{KGen}(\mathsf{p}_{\mathsf{GKMHE}})$. The output wire labels have already been fixed at setup. Then, for every gate $g_i = (w_\ell, w_r, w_o, op)$, compute the garbling of g_i as

$$G_i \leftarrow \mathsf{Enc}(\mathsf{p}_{\mathsf{GKMHE}}, \mathsf{sk}^0_{w_\ell}, \mathsf{sk}^1_{w_\ell}, \mathsf{sk}^0_{w_r}, \mathsf{sk}^1_{w_r}, \mathsf{sk}^{op(0,0)}_{w_o}, \mathsf{sk}^{op(0,1)}_{w_o}, \mathsf{sk}^{op(1,0)}_{w_o}, \mathsf{sk}^{op(1,1)}_{w_o})$$

Finally, output the garbling $\mathcal{C} = (\phi_\lambda(f), G_1, \ldots, G_N)$ and the input labels $L = \{\mathsf{sk}^0_{w_i}, \mathsf{sk}^1_{w_i}\}_{w_i \in \mathcal{W}_I}$.

$\mathsf{Rerand}(\mathsf{p}, \mathcal{C})$**:** This algorithm relies on the circuit topology described in $\phi_\lambda(f)$ to perform a consistent rerandomization. For every wire $w_i \in \mathcal{W} \setminus \mathcal{W}_{\mathcal{O}}$, sample a key transformation function $\sigma_{w_i} \leftarrow\!\!\$\, \mathcal{F}$. For all output wires $w_i \in \mathcal{W}_{\mathcal{O}}$, let $\sigma_{w_i} = \mathsf{id} \in \mathcal{F}$ be the identity function. Now parse $\mathcal{C} = (\phi_\lambda(f), G_1, \ldots, G_N)$.

To rerandomize the garbling G_i of g_i for all $i \in [N]$, compute

$$G'_i \leftarrow \mathsf{Eval}(\mathsf{p}_{\mathsf{GKMHE}}, \sigma_{w_\ell}, \sigma_{w_r}, \sigma_{w_o}, G_i).$$

Finally, output $\mathcal{C}' = (\phi_\lambda(f), G'_1, \ldots, G'_N)$ and π_{En}, where π_{En} is defined by

$$\pi_{\mathsf{En}} : \mathcal{L} \to \mathcal{L}$$
$$\{\mathsf{sk}^0_{w_i}, \mathsf{sk}^1_{w_i}\}_{w_i \in \mathcal{W}_I} \mapsto \{\sigma_{w_i}(\mathsf{sk}^0_{w_i}), \sigma_{w_i}(\mathsf{sk}^1_{w_i})\}_{w_i \in \mathcal{W}_I}.$$

$\mathsf{En}(\mathsf{p}, L, x)$**:** Given input labels $L = \{\mathsf{sk}_{w_i}^0, \mathsf{sk}_{w_i}^1\}_{w_i \in \mathcal{W}_I}$ and an input $x \in \{0,1\}^{\ell_{\mathsf{in}}}$, output input encoding $I = \{\mathsf{sk}_{w_i}^{x_i}\}_{w_i \in \mathcal{W}_I}$.

$\mathsf{Ev}(\mathsf{p}, \mathcal{C}, I)$**:** Parse $I = \{\mathsf{sk}_{w_i}\}_{w_i \in \mathcal{W}_I}$ and $\mathsf{p} = (\mathsf{p}_{\mathsf{GKMHE}}, \mathsf{sk}_0, \mathsf{sk}_1)$. The circuit is evaluated by decrypting gates using Dec in topological order.

1. The input labels in I are used to evaluate the first gates, i.e. the ones that only use input wires as inputs.
2. Decryption of a garbled gate then provides the evaluator with the output label for that gate, which correspond to the input labels of subsequent gates, allowing them to proceed topologically.
3. Eventually, labels $\mathsf{sk}_{w_{o_i}}$ for all the output wires $\mathcal{W}_{\mathcal{O}} = (w_{o_1}, \ldots, w_{o_{\ell_{\mathsf{out}}}})$ are obtained. The output can then be decoded by checking whether each output label is equal to sk_0 or sk_1, resulting in output $y \in \{0,1\}^{\ell_{\mathsf{out}}}$ with $y_i = 0$ if $\mathsf{sk}_{w_{o_i}} = \mathsf{sk}_0$ and 1 otherwise.

We will now argue correctness, garbling privacy, and rerand privacy of Construction 3. The proofs largely follow the standard security proofs for (rerandomizable) garbled circuits, but adapted to gate KMHE.

Theorem 25. *Construction 3 is correct, assuming that the gate KMHE schme* GKMHE *is correct.*

Proof sketch. Correctness of Construction 3 follows trivially from correctness of the garbled circuit construction (see [LP09, Claim 6] for details) as well as correctness of GKMHE.

Theorem 26. *Construction 3 has garbling privacy, assuming that the gate KMHE scheme* GKMHE *is CPA-secure and position-hiding.*

Proof sketch. Proof of garbling privacy follows similarly to the proof of Theorem 7 in [LP09], therefore we will only sketch it here. From $\phi_\lambda(f)$ the simulator $\mathsf{Sim}_{\mathsf{Gb}}$ obtains all information about the circuit f except for the gate operations. $\mathsf{Sim}_{\mathsf{Gb}}$ can then sample labels for each wire as done by Gb. It picks a random set of wire labels as the active set of labels and then programs the garbled circuit to always return $f(x)$ by encrypting only those active labels in all garbled gate ciphertexts. The subset of active labels corresponding to input wires is then returned as the input label set I. Encrypting only the active labels is ensured by the CPA security of Construction 2, while selecting a random set of wire labels as the active set, rather than using the actual active set for input x, is enabled by the position-hiding property of Construction 2.

Theorem 27. *Construction 3 has rerand privacy (Definition 30), assuming that the gate KMHE scheme* GKMHE *satisfies CPA security, KMH rerandomizability, key privacy, and key privacy under leakage.*

Proof. For the proof we will need some concepts related to evaluation of Boolean circuits and their garblings. First is the concept of topological ordering. A topological ordering of a directed acyclic graph, such as a Boolean circuit, is an ordering of the nodes, such that a node G comes before another node G' if there is a path from G to G'. Intuitively, for a Boolean circuit, a topological ordering of the gates is an ordering in which the gates can be evaluated. The topological ordering ensures that, when evaluating some gate, all necessary input values for the gate have been computed already. In the proof we will instead need a topological ordering of the wires. In such an ordering, a wire must be placed before another wire if the former is needed to compute the value on the latter wire when evaluating the circuit.

Specifically for garbled circuits, we will also need the concept of *active* and *inactive* labels [LP09]. Each wire in a garbled circuit has two associated labels. Given some input x as well as corresponding input labels for the circuit, the active labels are those that are obtained by the evaluator when evaluating the circuit on x. This includes the given input labels. The inactive labels are the ones that are not obtained, they remain hidden.

We now prove Theorem 27 using a sequence of hybrids to prove indistinguishability of case $b = 0$ and case $b = 1$ of the game $\mathsf{RER\text{-}PRIV}_{\mathcal{A},\mathsf{RGS}}$.

Game$_1$: This hybrid corresponds to case $b = 0$ of the $\mathsf{RER\text{-}PRIV}_{\mathcal{A},\mathsf{RGS}}$ game.

Game$_2$: In **Game$_1$**, each garbled gate G'_i of gate $g_i = (w_\ell, w_r, w_o, op)$, $i \in [N]$, in $\mathcal{C}$ is computed as

$$G_i \leftarrow \mathsf{Enc}(\mathsf{p}_{\mathsf{GKMHE}}, \mathsf{sk}^0_{w_\ell}, \mathsf{sk}^1_{w_\ell}, \mathsf{sk}^0_{w_r}, \mathsf{sk}^1_{w_r}, \mathsf{sk}^{op(0,0)}_{w_o}, \mathsf{sk}^{op(0,1)}_{w_o}, \mathsf{sk}^{op(1,0)}_{w_o}, \mathsf{sk}^{op(1,1)}_{w_o})$$

and

$$\begin{aligned} G'_i \leftarrow \mathsf{Eval}(\mathsf{p}_{\mathsf{GKMHE}}, \sigma^{(t-1)}_{w_\ell}, \sigma^{(t-1)}_{w_r}, \sigma^{(t-1)}_{w_o}, \\ \ldots \mathsf{Eval}(\mathsf{p}_{\mathsf{GKMHE}}, \sigma^{(1)}_{w_\ell}, \sigma^{(1)}_{w_r}, \sigma^{(1)}_{w_o}, G_i) \ldots). \end{aligned}$$

In **Game$_2$** we instead now directly compute

$$\begin{aligned} G'_i \leftarrow \mathsf{Enc}(\mathsf{p}_{\mathsf{GKMHE}}, \sigma_{w_\ell}(t-1, \mathsf{sk}^0_{w_\ell}), \sigma_{w_\ell}(t-1, \mathsf{sk}^1_{w_\ell}), \sigma_{w_r}(t-1, \mathsf{sk}^0_{w_r}), \\ \sigma_{w_r}(t-1, \mathsf{sk}^1_{w_r}), \sigma_{w_o}(t-1, \mathsf{sk}^{op(0,0)}_{w_o}), \sigma_{w_o}(t-1, \mathsf{sk}^{op(0,1)}_{w_o}), \\ \sigma_{w_o}(t-1, \mathsf{sk}^{op(1,0)}_{w_o}), \sigma_{w_o}(t-1, \mathsf{sk}^{op(1,1)}_{w_o})), \end{aligned}$$

where from now on we define

$$\sigma_{w_\ell}(i,\cdot) = \sigma^{(i)}_{w_\ell}(\dots\sigma^{(1)}_{w_\ell}(\cdot)\dots),$$
$$\sigma_{w_r}(i,\cdot) = \sigma^{(i)}_{w_r}(\dots\sigma^{(1)}_{w_r}(\cdot)\dots),$$
$$\sigma_{w_o}(i,\cdot) = \sigma^{(i)}_{w_o}(\dots\sigma^{(1)}_{w_o}(\cdot)\dots).$$

Each garbled gate is therefore a fresh encryption which contains the correctly transformed keys and messages. Indistinguishability of $\mathbf{Game}_1$ and $\mathbf{Game}_2$ follows straightforwardly by KMH rerandomizability (Definition 25) of the gate KMHE scheme GKMHE.

$\mathbf{Game}''_{3,0}$: To simplify notation we define this hybrid to be the same as hybrid $\mathbf{Game}_2$.

$\mathbf{Game}_{3,i} \forall i \in [M]$: By evaluating $\mathcal{C}$ using the challenge input labels I, corresponding to input x, we can designate *active* and *inactive* labels as defined previously. Without loss of generality let now $\mathsf{sk}^1_{w_j}$ be the active label and $\mathsf{sk}^0_{w_j}$ be the inactive label of each wire w_j. The other cases follow analogously.

Consider now the i-th wire w_i (by topological ordering) of the circuit $\mathcal{C}$ and all the gates $g_1, \dots, g_n$ that have this wire as one of their input wires. For notational simplicity assume without loss of generality that w_i is the left input wire and that w_i comes before w_r in the topological ordering, the other cases follow similarly. More details on these assumptions are given towards the end of the definition of $\mathbf{Game}_{3,i}$. Keeping these assumptions in mind, the garbling G'_j of gate $g_j = (w_i, w_r, w_o, op)$ from the previous hybrid can be written as

$$\begin{aligned} G'_j \leftarrow \mathsf{Enc}(\mathsf{p}_{\mathsf{GKMHE}}, &\sigma_{w_i}(t-1,\mathsf{sk}^0_{w_i}), \sigma_{w_i}(t-1,\mathsf{sk}^1_{w_i}), \sigma_{w_r}(t-1,\mathsf{sk}^0_{w_r}),\\ &\sigma_{w_r}(t-1,\mathsf{sk}^1_{w_r}), \sigma_{w_o}(t-1,\mathsf{sk}^{op(0,0)}_{w_o}), \sigma_{w_o}(t-1,\mathsf{sk}^{op(0,1)}_{w_o}),\\ &\sigma_{w_o}(t-1,\mathsf{sk}^{op(1,0)}_{w_o}), \sigma_{w_o}(t-1,\mathsf{sk}^{op(1,1)}_{w_o})), \end{aligned}$$

For hybrid $\mathbf{Game}_{3,i}$ we now replace the transformed *inactive* label $\sigma_{w_i}(t-1,\mathsf{sk}^0_{w_i})$ with a fresh key $\underline{\mathsf{sk}}^0_{w_i} \leftarrow \mathsf{KGen}(\mathsf{p}_{\mathsf{GKMHE}})$. The new garbling G'_j for all $j \in [n]$ is then given by

$$\begin{aligned} G'_j \leftarrow \mathsf{Enc}(\mathsf{p}_{\mathsf{GKMHE}}, &\underline{\mathsf{sk}}^0_{w_i}, \sigma_{w_i}(t-1,\mathsf{sk}^1_{w_i}), \sigma_{w_r}(t-1,\mathsf{sk}^0_{w_r}),\\ &\sigma_{w_r}(t-1,\mathsf{sk}^1_{w_r}), \sigma_{w_o}(t-1,\mathsf{sk}^{op(0,0)}_{w_o}), \sigma_{w_o}(t-1,\mathsf{sk}^{op(0,1)}_{w_o}),\\ &\sigma_{w_o}(t-1,\mathsf{sk}^{op(1,0)}_{w_o}), \sigma_{w_o}(t-1,\mathsf{sk}^{op(1,1)}_{w_o})), \end{aligned}$$

Note that if w_i comes after w_r in the topological ordering, then the key corresponding to w_r would have been replaced in a previous hybrid already. The case that w_i is the right input wire only affects the ordering of arguments.

We will explain below that indistinguishability of this hybrid from the previous one follows from key privacy under leakage (Definition 26) of the gate KMHE scheme.

$\mathbf{Game}'_{3,i} \forall i \in [M]$: Consider again the i-th wire w_i (topological ordering) of the circuit $\mathcal{C}$ and and all the gates $g_1, \ldots, g_n$, along with corresponding garblings $G'_1, \ldots, G'_n$, that have this wire as one of their input wires. In hybrid $\mathbf{Game}'_{3,i}$ we now replace the transformed *active* label $\sigma_{w_i}(t-1, \mathsf{sk}^1_{w_i})$ with a fresh key $\underline{\mathsf{sk}}^1_{w_i} \leftarrow \mathsf{KGen}(\mathsf{p}_{\mathsf{GKMHE}})$.Each G'_j, $j \in [n]$, is then given by

$$\begin{aligned} G'_j \leftarrow \mathsf{Enc}(\mathsf{p}_{\mathsf{GKMHE}}, \underline{\mathsf{sk}}^0_{w_i}, \underline{\mathsf{sk}}^1_{w_i}, \sigma_{w_r}(t-1, \mathsf{sk}^0_{w_r}), \\ \sigma_{w_r}(t-1, \mathsf{sk}^1_{w_r}), \sigma_{w_o}(t-1, \mathsf{sk}^{op(0,0)}_{w_o}), \sigma_{w_o}(t-1, \mathsf{sk}^{op(0,1)}_{w_o}), \\ \sigma_{w_o}(t-1, \mathsf{sk}^{op(1,0)}_{w_o}), \sigma_{w_o}(t-1, \mathsf{sk}^{op(1,1)}_{w_o})). \end{aligned}$$

Indistinguishability from the previous hybrid follows by key privacy of the gate KMHE scheme (Definition 24).

$\mathbf{Game}''_{3,i} \forall i \in [M]$: Consider again the i-th wire w_i (topological ordering) of the circuit $\mathcal{C}$ and and all the gates $g_1, \ldots, g_n$, along with corresponding garblings $G'_1, \ldots, G'_n$, that have this wire as one of their input wires. For all $g_j = (w_\ell, w_r, w_o, \cdot)$, $j \in [n]$, check if w_i is the second input wire by topological ordering. If not, then define $\mathbf{Game}''_{3,i} = \mathbf{Game}'_{3,i}$. Otherwise, this means that the key replacement in $\mathbf{Game}'_{3,i}$ lead to both input wires of g_j now having fresh inactive and active labels. In $\mathbf{Game}''_{3,i}$, we now change all encrypted messages inside each G'_j to the active output label $\sigma_{w_o}(t-1, \mathsf{sk}^1_{w_o})$, which means that G'_j is computed as

$$\begin{aligned} G'_j \leftarrow \mathsf{Enc}(\mathsf{p}_{\mathsf{GKMHE}}, \underline{\mathsf{sk}}^0_{w_\ell}, \underline{\mathsf{sk}}^1_{w_\ell}, \underline{\mathsf{sk}}^0_{w_r}, \underline{\mathsf{sk}}^0_{w_r}, \\ \sigma_{w_o}(t-1, \mathsf{sk}^1_{w_o}), \sigma_{w_o}(t-1, \mathsf{sk}^1_{w_o}), \\ \sigma_{w_o}(t-1, \mathsf{sk}^1_{w_o}), \sigma_{w_o}(t-1, \mathsf{sk}^1_{w_o})). \end{aligned}$$

Since $\underline{\mathsf{sk}}^0_{w_\ell}$ and $\underline{\mathsf{sk}}^0_{w_r}$ are kept secret from the adversary distinguishing between $\mathbf{Game}'_{3,i}$ and $\mathbf{Game}_{3,i}$, this step follows from CPA security (Definition 23) of the gate KMHE scheme.

Note that in $\mathbf{Game}'_{3,M}$, all ciphertexts inside the garbled circuit $\mathcal{C}$ encrypt only the active output labels. Therefore, $\mathcal{C}$ looks exactly like a simulated garbling as output by $\mathsf{Sim}_{\mathsf{Gb}}$.

$\mathbf{Game}_4$: This hybrid corresponds to case $b = 1$ of the $\mathsf{RER\text{-}PRIV}_{\mathcal{A},\mathsf{RGS}}$ game, i.e. $\mathcal{A}$ is given a fresh garbling.

We now sketch how to prove indistinguishability of these hybrids.

$\mathbf{Game}_1 \overset{\mathsf{c}}{\approx} \mathbf{Game}_2 = \mathbf{Game}''_{3,0}$: As mentioned before, this is just a straightforward application of KMH rerandomizability (Definition 25) of the gate KMHE scheme.

$\mathbf{Game}''_{3,i-1} \overset{\mathsf{c}}{\approx} \mathbf{Game}_{3,i} \forall i \in [M]$: This indistinguishability follows from key privacy under leakage (Definition 26) of the gate KMHE scheme. We will now sketch why this is the case.

Let $g_1, \ldots, g_n$ be all the gates that have wire w_i as one of their input wires (assume it is the left one again for simplicity), and let $G_1, \ldots, G_n$ be the corresponding garbled gates in the form of gate KMHE ciphertexts as in hybrid $\mathbf{Game}'_{3,i-1}$. Each G'_j is then, by definition of $\mathbf{Game}'_{3,i-1}$ given by

$$\begin{aligned} G'_j \leftarrow \mathsf{Enc}(\mathsf{p}_{\mathsf{GKMHE}}, &\sigma_{w_i}(t-1, \mathsf{sk}^0_{w_i}), \sigma_{w_i}(t-1, \mathsf{sk}^1_{w_i}), \sigma_{w_r}(t-1, \mathsf{sk}^0_{w_r}), \\ &\sigma_{w_r}(t-1, \mathsf{sk}^1_{w_r}), \sigma_{w_o}(t-1, \mathsf{sk}^{op(0,0)}_{w_o}), \sigma_{w_o}(t-1, \mathsf{sk}^{op(0,1)}_{w_o}), \\ &\sigma_{w_o}(t-1, \mathsf{sk}^{op(1,0)}_{w_o}), \sigma_{w_o}(t-1, \mathsf{sk}^{op(1,1)}_{w_o})). \end{aligned}$$

Also recall that

$$\begin{aligned} \sigma_{w_i}(t-1, \cdot) &= \sigma^{(t-1)}_{w_i}(\ldots \sigma^{(1)}_{w_i}(\cdot)\ldots), \\ \sigma_{w_r}(t-1, \cdot) &= \sigma^{(t-1)}_{w_r}(\ldots \sigma^{(1)}_{w_r}(\cdot)\ldots), \\ \sigma_{w_o}(t-1, \cdot) &= \sigma^{(t-1)}_{w_o}(\ldots \sigma^{(1)}_{w_o}(\cdot)\ldots). \end{aligned}$$

The function $\sigma^{(t-1)}_{w_i}$ is implied by the randomness r_t which an adversary distinguishing between $\mathbf{Game}'_{3,i-1}$ and $\mathbf{Game}_{3,i}$ does not obtain. The only leakage of $\sigma_{w_i}(t-1, \mathsf{sk}^0_{w_i})$ in each G'_j is via the active label $\sigma_{w_i}(t-1, \mathsf{sk}^1_{w_i})$ which the adversary can obtain via the active input labels I, and the randomness values $r_1, \ldots, r_{t-1}$ which leak the values $\sigma_{w_i}(t-2, \mathsf{sk}^0_{w_i})$ and $\sigma_{w_i}(t-2, \mathsf{sk}^1_{w_i})$. The only other leakage could be via the gates that have the wire w_i as their output wire, since those garbled gates could contain the inactive label $\sigma_{w_i}(t-1, \mathsf{sk}^0_{w_i})$ as one of the messages. However, definition of $\mathbf{Game}'_{3,i-1}$ ensures that all the garbled gates that are topologically before w_i, only encrypt the active labels.

Therefore, there is no further leakage of $\sigma_{w_i}(t-1, \mathsf{sk}^0_{w_i})$ except for $\sigma_{w_i}(t-1, \mathsf{sk}^1_{w_i})$, $\sigma_{w_i}(t-2, \mathsf{sk}^0_{w_i})$ and $\sigma_{w_i}(t-2, \mathsf{sk}^1_{w_i})$, exactly as required by the definition of key privacy under leakage (Definition 26). Therefore, we can use key privacy under leakage to replace the key $\sigma_{w_i}(t-1, \mathsf{sk}^0_{w_i})$ in all G'_j with a fresh key $\underline{\mathsf{sk}}^0_{w_i}$.

$\mathbf{Game}_{3,i} \overset{\mathsf{c}}{\approx} \mathbf{Game}'_{3,i} \forall i \in [M]$**:** From hybrid $\mathbf{Game}_{3,i}$ to $\mathbf{Game}'_{3,i}$ we replace the active label $\sigma_{w_i}(t-1, \mathsf{sk}_{w_i^1})$ with the fresh key $\underline{\mathsf{sk}}^1_{w_i}$. Since in hybrid $\mathbf{Game}_{3,i}$ we already replaced the inactive label $\sigma_{w_i}(t-1, \mathsf{sk}_{w_i^0})$ with a fresh label, there is no longer any leakage of the key transformation function $\sigma_{w_i}^{(t-1)}$. Therefore, since $r_t \leftarrow\!\!\$\ R_{\mathsf{Rerand}}$ it holds that $\sigma_{w_i}^{(t-1)}$ is uniformly distributed over $\mathcal{F}$ and

$$\left\{\sigma_{w_i}(t-2, \mathsf{sk}^1_{w_i}), \sigma_{w_i}(t-1, \mathsf{sk}_{w_i^1})\right\} \overset{\mathrm{c}}{\approx} \left\{\sigma_{w_i}(t-2, \mathsf{sk}^1_{w_i}), \underline{\mathsf{sk}}^1_{w_i}\right\}$$

by key privacy of the gate KMHE scheme (Definition 24).

$\mathbf{Game}'_{3,i} \overset{\mathsf{c}}{\approx} \mathbf{Game}''_{3,i} \forall i \in [M]$**:** In hybrid $\mathbf{Game}'_{3,i}$ all secret keys used to create the garbled gate ciphertexts G'_j are freshly sampled keys. The adversary distinguishing between $\mathbf{Game}'_{3,i}$ and $\mathbf{Game}''_{3,i}$ obtains only the active keys. Therefore, by CPA security of the gate KMHE scheme (Definition 23), the messages not decryptable by those active keys remaining computationally hidden. Those are exactly the messages we replace in hybrid $\mathbf{Game}''_{3,i}$. Therefore, this step follows from CPA security of the gate KMHE scheme.

$\mathbf{Game}''_{3,M} \overset{\mathsf{c}}{\approx} \mathbf{Game}_4$**:** Indistinguishability of these hybrids is an immediate consequence of garbling privacy (Definition 29).

□

6 Conclusion

This thesis sets out to narrow the gap between advanced cryptographic techniques and their deployment in practice. It delivers two application-level systems—post-quantum authentication for *electronic machine-readable travel document*s (eMRTDs) and privacy-preserving *power flow analysis* (PFA) for smart grids—and one foundational construction, a pairing-based *key-and-message homomorphic encryption* (KMHE) scheme that enables practical constant-round outsourced *multi-party computation* (MPC) via *rerandomizable garbling schemes* (RGS).

For eMRTDs, we introduce *PQ-EAC*, a family of eight protocol variants that replace Diffie–Hellman with post-quantum *key encapsulation mechanism*s (KEMs) and allow different trade-offs regarding signatures, round complexity, and forward secrecy. Our implementation shows that assuming a data rate of 848 kbit/s, one execution of PQ-EAC takes 1.28 s. Computation is dominated by Kyber decapsulation (596 ms) with Dilithium verification around 86 ms. These results demonstrate that our post-quantum protocols for travel documents can meet the demands of practical deployment.

For smart grids, we give the first privacy-preserving realization of PFA. To optimize performance, we utilize a Cartesian formulation of Newton's method, and LU/GMRES solvers with line search. On two Simbench low-voltage grids, the *online* phase completes in ≈ 27 s for 13 prosumers at RTT = 1 ms (semi-honest), and ≈ 62 s for 39 prosumers. Full runs including preprocessing are ≈ 232 s (13 prosumers) and ≥ 70 min (39 prosumers). Since performance degrades very quickly with increasing latency, future research should focus on implementations utilizing constant-round MPC.

Finally, we introduce a pairing-based KMHE with the *same homomorphism* in the key- and message-space together with new security notions—*KMH rerandomizability* and *key privacy under leakage*. This unlocks practical RGS: garbled-gate size drops from 133.43 MB to 1.35 MB (−98.99%), per-gate garbling time from 33 min to 0.04 s (−99.998%), and rerandomization from 2.23 h to 0.05 s (−99.9993%). These improvements make SCALES/YOSO-style constant-round outsourced MPC viable for simple circuits.

In conclusion, the journey from theoretical concept to practical reality is a crucial one for cryptography. This thesis has taken several steps on that path, demonstrating that advanced cryptographic solutions are not merely academic exercises but essential tools for building a more secure and private digital future.

Bibliography

[A+25] Rajeev Acharya et al. Quantum error correction below the surface code threshold. 638(8052):920–926, 2025. doi:10.1038/s41586-024-08449-y.

[AAC+22] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. Status report on the third round of the nist post-quantum cryptography standardization process, nist ir 8413. Technical report, National Institute for Standards and Technology (NIST), 2022. doi:10.6028/NIST.IR.8413-upd1.

[AACM16] Aysajan Abidin, Abdelrahaman Aly, Sara Cleemput, and Mustafa A. Mustafa. An MPC-based privacy-preserving protocol for a local electricity trading market. In *CANS '16*, 2016. doi:10.1007/978-3-319-48965-0_40.

[AC92] Venkataramana Ajjarapu and Colin Christy. The continuation power flow: A tool for steady state voltage stability analysis. *IEEE Transactions on Power Systems*, 7(1):416–423, 1992.

[ÁC11] Gergely Ács and Claude Castelluccia. I Have a DREAM!(DiffeRentially privatE smArt Metering). In *International Workshop on Information Hiding*, pages 118–132. Springer, 2011.

[ADH+22] Yawning Angel, Benjamin Dowling, Andreas Hülsing, Peter Schwabe, and Fiona Johanna Weber. Post quantum noise. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 97–109, Los Angeles, CA, USA, November 7–11, 2022. ACM Press. doi:10.1145/3548606.3560577.

[ADMC17] Muhammad Rizwan Asghar, György Dán, Daniele Miorandi, and Imrich Chlamtac. Smart meter data privacy: A survey. *IEEE Communications Surveys & Tutorials*, 19(4):2820–2835, 2017.

[AGM+] D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[AHKP22] Anasuya Acharya, Carmit Hazay, Vladimir Kolesnikov, and Manoj Prabhakaran. SCALES - MPC with small clients and larger ephemeral servers. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022: 20th Theory of Cryptography Conference, Part II*, volume 13748 of *Lecture Notes in Computer Science*, pages 502–531, Chicago, IL, USA, November 7–10, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-031-22365-5_18.

[AHKP24] Anasuya Acharya, Carmit Hazay, Vladimir Kolesnikov, and Manoj Prabhakaran. Malicious security for SCALES - outsourced computation with ephemeral servers. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part IX*, volume 14928 of *Lecture Notes in Computer Science*, pages 3–38, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland. doi:10.1007/978-3-031-68400-5_1.

[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, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 491–520, Santa Barbara, CA, USA, August 14–18, 2016. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-662-53008-5_17.

[AKQ08] Gildas Avoine, Kassem Kalach, and Jean-Jacques Quisquater. epassport: Securing international contacts with contactless chips. In Gene Tsudik, editor, *FC 2008: 12th International Conference on Financial Cryptography and Data Security*, volume 5143 of *Lecture Notes in Computer Science*, pages 141–155, Cozumel, Mexico, January 28–31, 2008. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-540-85230-8_11.

[AL17] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly secure multiparty computation. *J. Cryptol.*, 30(1):58–151, 2017. doi:10.1007/S00145-015-9214-4.

[AS19] Abdelrahaman Aly and Nigel P. Smart. Benchmarking privacy preserving scientific operations. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 2019: 17th*

International Conference on Applied Cryptography and Network Security, volume 11464 of *Lecture Notes in Computer Science*, pages 509–529, Bogota, Colombia, June 5–7, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-21568-2_25.

[ATB17] Elvira Amicarelli, Tuan Quoc Tran, and Seddik Bacha. Flexibility service market for active congestion management of distribution networks using flexible energy resources of microgrids. In *ISGT-Europe '17*, pages 1–6, 2017. doi:10.1109/ISGTEurope.2017.8260198.

[BBF+19] Nina Bindel, Jacqueline Brendel, Marc Fischlin, Brian Goncalves, and Douglas Stebila. Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography*, Lecture Notes in Computer Science, pages 206–226, Cham, 2019. Springer International Publishing. doi:10.1007/978-3-030-25510-7_12.

[BCIV17] Joppe W. Bos, Wouter Castryck, Ilia Iliashenko, and Frederik Vercauteren. Privacy-friendly forecasting for the smart grid using homomorphic encryption and the group method of data handling. In Marc Joye and Abderrahmane Nitaj, editors, *AFRICACRYPT 17: 9th International Conference on Cryptology in Africa*, volume 10239 of *Lecture Notes in Computer Science*, pages 184–201, Dakar, Senegal, May 24–26, 2017. Springer, Cham, Switzerland. doi:10.1007/978-3-319-57339-7_11.

[BDFK12] Jens Bender, Özgür Dagdelen, Marc Fischlin, and Dennis Kügler. The PACE|AA protocol for machine readable travel documents, and its security. In Angelos D. Keromytis, editor, *FC 2012: 16th International Conference on Financial Cryptography and Data Security*, volume 7397 of *Lecture Notes in Computer Science*, pages 344–358, Kralendijk, Bonaire, February 27 – March 2, 2012. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-32946-3_25.

[Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432, Santa Barbara, CA, USA, August 11–15, 1992. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-46766-1_34.

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and*

Practice of Public Key Cryptography, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228, New York, NY, USA, April 24–26, 2006. Springer Berlin Heidelberg, Germany. doi:10.1007/11745853_14.

[BFG+20] Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, and Douglas Stebila. Towards post-quantum security for Signal's X3DH handshake. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *SAC 2020: 27th Annual International Workshop on Selected Areas in Cryptography*, volume 12804 of *Lecture Notes in Computer Science*, pages 404–430, Halifax, NS, Canada (Virtual Event), October 21-23, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-81652-0_16.

[BFK09] Jens Bender, Marc Fischlin, and Dennis Kügler. Security analysis of the PACE key-agreement protocol. In Pierangela Samarati, Moti Yung, Fabio Martinelli, and Claudio Agostino Ardagna, editors, *ISC 2009: 12th International Conference on Information Security*, volume 5735 of *Lecture Notes in Computer Science*, pages 33–48, Pisa, Italy, September 7–9, 2009. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-04474-8_3.

[BG19] Colin Boyd and Kai Gellert. A modern view on forward security. *The Computer Journal*, 64(1):639–652, 2019. doi:10.1093/comjnl/bxaa104.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In STOC 1988 [STO88], pages 1–10. doi:10.1145/62212.62213.

[BHHO08] Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision Diffie-Hellman. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 108–125, Santa Barbara, CA, USA, August 17–21, 2008. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-540-85174-5_7.

[BKS19] Leon Botros, Matthias J. Kannwischer, and Peter Schwabe. Memory-efficient high-speed implementation of kyber on cortex-m4. In Johannes Buchmann, Abderrahmane Nitaj, and Tajje-eddine Rachidi, editors, *Progress in Cryptology - AFRICACRYPT 2019 - 11th International Conference on Cryptology in Africa, Rabat, Morocco, July*

9-11, 2019, Proceedings, volume 11627 of *Lecture Notes in Computer Science*, pages 209–228. Springer, 2019. doi:10.1007/978-3-030-23696-0_11.

[BL15] Mihir Bellare and Anna Lysyanskaya. Symmetric and dual prfs from standard assumptions: A generic validation of an HMAC assumption. *IACR Cryptol. ePrint Arch.*, page 1198, 2015. URL: http://eprint.iacr.org/2015/1198.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd Annual ACM Symposium on Theory of Computing*, pages 503–513, Baltimore, MD, USA, May 14–16, 1990. ACM Press. doi:10.1145/100216.100287.

[BR93] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer, 1993. doi:10.1007/3-540-48329-2_21.

[Bun16] Bundesamt für Sicherheit in der Informationstechnik. Bsi tr-03110. Standard, 2016.

[Bun20] Bundesamt für Sicherheit in der Informationstechnik. *Migration to Post Quantum Cryptography: Recommendations for action by the BSI.* 2020. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Crypto/Migration_to_Post_Quantum_Cryptography.pdf?__blob=publicationFile&v=2.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press. doi:10.1109/SFCS.2001.959888.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In STOC 1988 [STO88], pages 11–19. doi:10.1145/62212.62214.

[CDE+18] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Major-

ity. In *CRYPTO '18*, volume 10992, pages 769–798. Springer, 2018. doi:10.1007/978-3-319-96881-0_26.

[CDGK22] Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. YOLO YOSO: Fast and simple encryption and secret sharing in the YOSO model. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022, Part I*, volume 13791 of *Lecture Notes in Computer Science*, pages 651–680, Taipei, Taiwan, December 5–9, 2022. Springer, Cham, Switzerland. doi:10.1007/978-3-031-22963-3_22.

[CDN15] R. Cramer, I. B. Damgård, and J. B. Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[CHI+23] Koji Chida, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Daniel Genkin, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. *J. Cryptol.*, 36(3):15, 2023. doi:10.1007/s00145-023-09453-7.

[CLL22] Wei Chen, Lu Liu, and Guo-Ping Liu. Privacy-preserving distributed economic dispatch of microgrids: A dynamic quantization-based consensus scheme with homomorphic encryption. *IEEE Transactions on Smart Grid*, 14(1):701–713, 2022.

[CS10] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In Radu Sion, editor, *FC 2010: 14th International Conference on Financial Cryptography and Data Security*, volume 6052 of *Lecture Notes in Computer Science*, pages 35–50, Tenerife, Canary Islands, Spain, January 25–28, 2010. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-14577-3_6.

[CV10] Rafik Chaabouni and Serge Vaudenay. The extended access control for machine readable travel documents. Cryptology ePrint Archive, Report 2010/103, 2010. URL: https://eprint.iacr.org/2010/103.

[Dem97] James W Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.

[DF11] Özgür Dagdelen and Marc Fischlin. Security analysis of the extended access control protocol for machine readable travel documents. In Mike Burmester, Gene Tsudik, Spyros S. Magliveras, and Ivana Ilic, editors, *ISC 2010: 13th International Conference on Information Security*, volume 6531 of *Lecture Notes in Computer Science*, pages

54–68, Boca Raton, FL, USA, October 25–28, 2011. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-18178-8_6.

[DFKZB13] George Danezis, Cédric Fournet, Markulf Kohlweiss, and Santiago Zanella-Béguelin. Smart meter aggregation via secret-sharing. In *Proceedings of the first ACM workshop on Smart energy grid security*, pages 75–80, 2013.

[DFMV17] Ivan Damgård, Sebastian Faust, Pratyay Mukherjee, and Daniele Venturi. Bounded tamper resilience: How to go beyond the algebraic barrier. *Journal of Cryptology*, 30(1):152–190, January 2017. doi:10.1007/s00145-015-9218-0.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 247–264, Santa Barbara, CA, USA, August 17–21, 2003. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-540-45146-4_15.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 523–540, Interlaken, Switzerland, May 2–6, 2004. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-540-24676-3_31.

[EGK+20] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for MPC over mixed arithmetic-binary circuits. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 823–852, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-56880-1_29.

[EK10] Costas Efthymiou and Georgios Kalogridis. Smart Grid Privacy via Anonymization of Smart Metering Data. In *Proceedings of the 1st International Conference on Smart Grid Communications*, pages 238–243. IEEE, 2010.

[Esc22] Daniel Escudero. An introduction to secret-sharing-based secure multiparty computation. Lecture Notes, 2022. URL: https://eprint.iacr.org/2022/062.

[EUM18] Ayman Esmat, Julio Usaola, and Maria Angeles Moreno. Distribution-level flexibility market for congestion management. *Energies*, 11(5), 2018. doi:10.3390/en11051056.

[Eur15] European Commission, Directorate-General for Energy. Essential regulatory requirements and recommendations for data handling, data safety, and consumer protection, 2015. https://energy.ec.europa.eu/publications/essential-regulatory-requirements-and-recommendations-data-handling-data-safety-and-consumer_en.

[FEMH18] Timm Faulwasser, Alexander Engelmann, Tillmann Mühlpfordt, and Veit Hagenmeyer. Optimal power flow: An introduction to predictive, distributed and stochastic control challenges. *at-Automatisierungstechnik*, 66(7):573–589, 2018.

[FGSW16] Marc Fischlin, Felix Günther, Benedikt Schmidt, and Bogdan Warinschi. Key confirmation in key exchange: A formal treatment and implications for TLS 1.3. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 452–469. IEEE Computer Society, 2016. doi:10.1109/SP.2016.34.

[FHMS19] Ihor Filimonov, Ross Horne, Sjouke Mauw, and Zach Smith. Breaking unlinkability of the ICAO 9303 standard for e-passports using bisimilarity. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019: 24th European Symposium on Research in Computer Security, Part I*, volume 11735 of *Lecture Notes in Computer Science*, pages 577–594, Luxembourg, September 23–27, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-29959-0_28.

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. In Steven D. Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012: 13th International Conference in Cryptology in India*, volume 7668 of *Lecture Notes in Computer Science*, pages 60–79, Kolkata, India, December 9–12, 2012. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-34931-7_5.

[FMH20] Ferdinando Fioretto, Terrence W. K. Mak, and Pascal Van Hentenryck. Differential privacy for power grid obfuscation. *IEEE Transactions on Smart Grid*, 11(2):1356–1366, 2020. doi:10.1109/TSG.2019.2936712.

[FvdHM+23] Marc Fischlin, Jonas von der Heyden, Marian Margraf, Frank Morgner, Andreas Wallner, and Holger Bock. Post-quantum security for the extended access control protocol. In Felix Günther and Julia Hesse, editors, *Security Standardisation Research - 8th International Conference, SSR 2023, Lyon, France, April 22-23, 2023, Proceedings*, volume 13895 of *Lecture Notes in Computer Science*, pages 22–52. Springer, 2023. doi:10.1007/978-3-031-30731-7_2.

[GGJL12] U. Greveler, P. Glösekötterz, B. Justus, and D. Loehr. Multimedia content identification through smart meter power usage profiles. In *Proc. of the Intl. Conference on Information and Knowledge Engineering*, page 1, 2012.

[GHK+21] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: You only speak once - secure MPC with stateless ephemeral roles. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 64–93, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland. doi:10.1007/978-3-030-84245-1_3.

[GHP18] Federico Giacon, Felix Heuer, and Bertram Poettering. KEM Combiners. In Michel Abdalla and Ricardo Dahab, editors, *Public-Key Cryptography – PKC 2018*, Lecture Notes in Computer Science, pages 190–218, Cham, 2018. Springer International Publishing. doi:10.1007/978-3-319-76578-5_7.

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-Hop homomorphic encryption and rerandomizable Yao circuits. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 155–172, Santa Barbara, CA, USA, August 15–19, 2010. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-14623-7_9.

[GJ10] Flavio D. Garcia and Bart Jacobs. Privacy-Friendly Energy-Metering via Homomorphic Encryption. In *Proceedings of the 6th International Workshop on Security and Trust Management*, pages 226–238, 2010.

[GLO+21] Vipul Goyal, Hanjun Li, Rafail Ostrovsky, Antigoni Polychroniadou, and Yifan Song. ATLAS: efficient and scalable MPC in the honest majority setting. In *CRYPTO '21*, volume 12826, pages 244–274. Springer, 2021. doi:10.1007/978-3-030-84245-1_9.

[GSA+21] Matthew B Gough, Sérgio F Santos, Tarek AlSkaif, Mohammad S Javadi, Rui Castro, and João PS Catalão. Preserving privacy of smart meter data in a smart grid environment. *IEEE Transactions on Industrial Informatics*, 18(1):707–718, 2021.

[HBD+22] Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS+. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`.

[HBG+18] A. Huelsing, D. Butin, S. Gazdag, J. Rijneveld, and A. Mohaisen. *XMSS: eXtended Merkle Signature Scheme*, 2018. `doi:10.17487/RFC8391`.

[HHMW22] Jiaqi Huang, Qiushi Huang, Gaoyang Mou, and Chenye Wu. DPW-GAN: High-Quality Load Profiles Synthesis with Differential Privacy Guarantees. *IEEE Transactions on Smart Grid*, 2022.

[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.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 132–150, Santa Barbara, CA, USA, August 14–18, 2011. Springer Berlin Heidelberg, Germany. `doi:10.1007/978-3-642-22792-9_8`.

[HNS+21] Andreas Hülsing, Kai-Chun Ning, Peter Schwabe, Fiona Johanna Weber, and Philip R. Zimmermann. Post-quantum WireGuard. In *2021 IEEE Symposium on Security and Privacy*, pages 304–321, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press. `doi:10.1109/SP40001.2021.00030`.

[HSS20] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. *Journal*

of Cryptology, 33(4):1732–1786, October 2020. doi:10.1007/s00145-020-09355-y.

[HvdHJ25] Raphael Heitjohann, Jonas von der Heyden, and Tibor Jager. Rerandomizable garbling, revisited. In *Advances in Cryptology - CRYPTO 2025, 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025. Proceedings*, Lecture Notes in Computer Science. Springer, 2025. URL: https://eprint.iacr.org/2025/843.

[Int18] International Organization for Standardization / International Electrotechnical Commission. Iso/iec 14443-4: Identification cards – contactless integrated circuit cards – proximity cards. Standard, 6 2018.

[Int20a] International Civil Aviation Organization. *Guiding Core Principles for the Development of Digital Travel Credential (DTC)*. 10 2020. URL: https://www.icao.int/Security/FAL/TRIP/PublishingImages/Pages/Publications/Guiding%20core%20principles%20for%20the%20development%20of%20a%20Digital%20Travel%20Credential%20%20%28DTC%29.PDF.

[Int20b] International Organization for Standardization / International Electrotechnical Commission. Iso/iec 7816-4: Identification cards – integrated circuit cards. Technical report, 5 2020.

[Int21] International Civil Aviation Organization. Icao doc 9303. Standard, 2021. 8th Edition. URL: https://www.icao.int/publications/pages/publication.aspx?docnum=9303.

[JKD12] Marek Jawurek, Florian Kerschbaum, and George Danezis. SOK: Privacy technologies for smart grids – A survey of options. *Microsoft Research, Cambridge, UK*, 1:1–16, 2012.

[JKL16] Hyo Jin Jo, In-Seok Kim, and Dong Hoon Lee. Efficient and Privacy-Preserving Metering Protocols for Smart Grid Systems. *IEEE Transactions on Smart Grid*, 7(3):1732–1742, 2016.

[KDK11] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. Privacy-friendly aggregation for the smart-grid. In Simone Fischer-Hübner and Nicholas Hopper, editors, *PETS 2011: 11th International Symposium on Privacy Enhancing Technologies*, volume 6794 of *Lecture Notes in Computer Science*, pages 175–191, Waterloo, ON, Canada, July 27–29,

2011. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-22263-4_10.

[Kel20] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In Ligatti et al. [LOKV20], pages 1575–1590. doi:10.1145/3372297.3417872.

[KOS16a] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *CCS '16*, pages 830–842, 2016.

[KOS16b] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 830–842, Vienna, Austria, October 24–28, 2016. ACM Press. doi:10.1145/2976749.2978357.

[KPR+] Matthias J. Kannwischer, Richard Petri, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 158–189, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland. doi:10.1007/978-3-319-78372-7_6.

[Kra03] Hugo Krawczyk. SIGMA: the 'sign-and-mac' approach to authenticated diffie-hellman and its use in the ike-protocols. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 400–425. Springer, 2003. doi:10.1007/978-3-540-45146-4_24.

[Kra10] Hugo Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture*

Notes in Computer Science, pages 631–648. Springer, 2010. doi:10.1007/978-3-642-14623-7_34.

[KRY22] Sebastian Kolby, Divya Ravi, and Sophia Yakoubov. Towards efficient YOSO MPC without setup. Cryptology ePrint Archive, Report 2022/187, 2022. URL: https://eprint.iacr.org/2022/187.

[KS22] Marcel Keller and Ke Sun. Secure quantized training for deep learning. In *ICML 2022*, volume 162 of *Proceedings of Machine Learning Research*, pages 10912–10938. PMLR, 2022.

[Kuß19] Felix Kußmaul. Armed for the quantum revolution: Implementing post-quantum-cryptography on the goid card. In Bundesamt für Sicherheit in der Informationstechnik, editor, *IT-Sicherheit als Voraussetzung für eine erfolgreiche Digitalisierung*, pages 275–284. SecuMedia Verlags-GmbH, 2019.

[LDK+22] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[LKLP07] Yifei Liu, Timo Kasper, Kerstin Lemke-Rust, and Christof Paar. E-Passport: Cracking Basic Access Control Keys. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, Lecture Notes in Computer Science, pages 1531–1547, Berlin, Heidelberg, 2007. Springer. doi:10.1007/978-3-540-76843-2_30.

[LLL10] Fengjun Li, Bo Luo, and Peng Liu. Secure Information Aggregation for Smart Grids using Homomorphic Encryption. In *Proceedings of the 1st International Conference on Smart Grid Communications*, pages 327–332. IEEE, 2010.

[LN97] Hieu Le Nguyen. Newton-raphson Method in Complex Form. *IEEE Transactions on Power Systems*, 12(3):1355–1359, 1997.

[LOKV20] Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors. *ACM CCS 2020: 27th Conference on Computer and Communications Security*, Virtual Event, USA, November 9–13, 2020. ACM Press.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009. doi:10.1007/s00145-008-9036-8.

[LPSY15] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 319–338, Santa Barbara, CA, USA, August 16–20, 2015. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-662-48000-7_16.

[LPSY19] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant-round multi-party computation combining BMR and SPDZ. *Journal of Cryptology*, 32(3):1026–1069, July 2019. doi:10.1007/s00145-019-09322-2.

[M+20] Steffen Meinecke et al. Simbench – A Benchmark Dataset of Electric Power Systems to Compare Innovative Solutions Based on Power Flow Analysis. *Energies*, 13(12):3290, 2020.

[MCAA19] Mustafa A Mustafa, Sara Cleemput, Abdelrahaman Aly, and Aysajan Abidin. A secure and privacy-preserving protocol for smart metering operational data collection. *IEEE Transactions on Smart Grid*, 10(6):6481–6490, 2019.

[MCF19] D. McGrew, M. Curcio, and S. Fluhrer. *Leighton-Micali Hash-Based Signatures*, 2019. doi:10.17487/RFC8554.

[MDS+22] Chenggang Mu, Tao Ding, Yuge Sun, Yuhan Huang, Fangxing Li, and Pierluigi Siano. Energy block-based peer-to-peer contract trading with secure multi-party computation in nanogrid. *IEEE Transactions on Smart Grid*, 13(6):4759–4772, 2022. doi:10.1109/TSG.2022.3176624.

[MSF+10] Andres Molina-Markham, Prashant J. Shenoy, Kevin Fu, Emmanuel Cecchet, and David E. Irwin. Private Memoirs of a Smart Meter. In *Proceedings of the 2nd Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, pages 61–66. ACM, 2010.

[MvdH21] Frank Morgner and Jonas von der Heyden. Analyzing requirements for post quantum secure machine readable travel documents. In Heiko Roßnagel, Christian H. Schunck, and Sebastian Mödersheim, editors,

Open Identity Summit 2021, pages 205–210, Bonn, 2021. Gesellschaft für Informatik e.V.

[Nat20] National Institute of Standards and Technology (NIST). Recommendation for stateful hash-based signature schemes, sp 800-208. Standard, 2020. doi:10.6028/NIST.SP.800-208.

[NIS14] NIST. Nistir 7628 rev. 1: Guidelines for smart grid cybersecurity, 2014. doi:10.6028/NIST.IR.7628r1.

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th Annual Symposium on Foundations of Computer Science*, pages 458–467, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press. doi:10.1109/SFCS.1997.646134.

[NS09] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 18–35, Santa Barbara, CA, USA, August 16–20, 2009. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-03356-8_2.

[NW99] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[PFH+22] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[PTVF07] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007.

[Res18] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018. URL: https://www.rfc-editor.org/info/rfc8446, doi:10.17487/RFC8446.

[Saa03] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.

[SAB+22] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022`.

[SDT15] Henrik Sandberg, György Dán, and Ragnar Thobaben. Differentially private state estimation in distribution networks with smart meters. In *Proceedings of the 54th conference on decision and control*, pages 4492–4498. IEEE, 2015.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, NM, USA, November 20–22, 1994. IEEE Computer Society Press. `doi:10.1109/SFCS.1994.365700`.

[Sma23] Nigel P. Smart. Practical and efficient FHE-based MPC. In Elizabeth A. Quaglia, editor, *19th IMA International Conference on Cryptography and Coding*, volume 14421 of *Lecture Notes in Computer Science*, pages 263–283, London, UK, December 12–14, 2023. Springer, Cham, Switzerland. `doi:10.1007/978-3-031-47818-5_14`.

[SS17] John M. Schanck and Douglas Stebila. A Transport Layer Security (TLS) Extension For Establishing An Additional Shared Secret. Internet-Draft draft-schanck-tls-additional-keyshare-00, Internet Engineering Task Force, April 2017. Work in Progress. URL: `https://datatracker.ietf.org/doc/draft-schanck-tls-additional-keyshare/00/`.

[SSW20] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum TLS without handshake signatures. In Ligatti et al. [LOKV20], pages 1461–1480. `doi:10.1145/3372297.3423350`.

[Sta01] Pantelimon Stanica. Good lower and upper bounds on binomial coefficients. *Journal of Inequalities in Pure and Applied Mathematics*, 2(3):30, 2001.

[STO88] *20th Annual ACM Symposium on Theory of Computing*, Chicago, IL, USA, May 2–4, 1988. ACM Press.

[SVW19] Baljinnyam Sereeter, Cornelis Vuik, and Cees Witteveen. On a comparison of Newton-Raphson solvers for power flow problems.

Journal of Computational and Applied Mathematics, 360:157–169, 2019.

[SZW+23] Fangyuan Si, Ning Zhang, Yi Wang, Peng-Yong Kong, and Wenjie Qiao. Distributed optimization for integrated energy systems with secure multiparty computation. *IEEE Internet of Things Journal*, 10(9):7655–7666, 2023. doi:10.1109/JIOT.2022.3209017.

[TGSZ22] Nianfeng Tian, Qinglai Guo, Hongbin Sun, and Xin Zhou. Fully privacy-preserving distributed optimization in power systems based on secret sharing. *iEnergy*, 1(3):351–362, 2022.

[Tri12] Antonio Trias. The holomorphic embedding load flow method. In *2012 IEEE Power and Energy Society General Meeting*, pages 1–8, 2012. doi:10.1109/PESGM.2012.6344759.

[TTM+23] Jean-François Toubeau, Fei Teng, Thomas Morstyn, Leandro Von Krannichfeldt, and Yi Wang. Privacy-Preserving Probabilistic Voltage Forecasting in Local Energy Communities. *IEEE Transactions on Smart Grid*, 14(1):798–809, 2023. doi:10.1109/TSG.2022.3187557.

[vdHSB+25] Jonas von der Heyden, Nils Schlüter, Philipp Binfet, Martin Asman, Markus Zdrallek, Tibor Jager, and Moritz Schulze Darup. Privacy-preserving power flow analysis via secure multi-party computation. *IEEE Trans. Smart Grid*, 16(1):344–355, 2025. doi:10.1109/TSG.2024.3453491.

[Vil12] Jorge Luis Villar. Optimal reductions of some decisional problems to the rank problem. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 80–97, Beijing, China, December 2–6, 2012. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-34961-4_7.

[WZZ21] Tong Wu, Changhong Zhao, and Ying-Jun Angela Zhang. Privacy-Preserving Distributed Optimal Power Flow with Partially Homomorphic Encryption. *IEEE Transactions on Smart Grid*, 12(5):4506–4521, 2021.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press. doi:10.1109/SFCS.1986.25.

[Zhu15] Jizhong Zhu. *Optimization of Power System Operation*. John Wiley & Sons, 2015.

[ZLXL23] Qingsong Zhao, Ximeng Liu, Huanliang Xu, and Yanbin Li. Practical reusable garbled circuits with parallel updates. *Comput. Stand. Interfaces*, 86:103721, 2023. doi:10.1016/J.CSI.2023.103721.

[ZWG+23] Xin Zhou, Bin Wang, Qinglai Guo, Hongbin Sun, Zhaoguang Pan, and Nianfeng Tian. Bidirectional privacy-preserving network-constrained peer-to-peer energy trading based on secure multiparty computation and blockchain. *IEEE Transactions on Power Systems*, 2023.

A Source Code for PFA

lib.py

```
from Compiler.types import *
from Compiler.util import if_else
from Compiler.library import *
from Compiler.instructions import *

def multvec(spA, ijA, x, n):
    # fast multiplication of sparse A with vector x
    b = sfix.Array(n)
    @for_range_opt(n)
    def _(i):
        b[i] = spA[i]*x[i]
    @for_range(n)
    def _(i):
        @for_range(ijA[i]-1, ijA[i+1] - 1)  # index arithmetic
        def _(k):
            b[i] += spA[k] * x[ijA[k] - 1]
    return b

def evalF(x, spG, ijG, spB, ijB, pK, qK, n, slack_voltage):
    uKreal = sfix.Array(n)
    uKreal[0] = slack_voltage
    uKreal[1:] = x[0:n-1]
    uKimag = sfix.Array(n)
    uKimag[0] = 0
    uKimag[1:] = x[n-1:]

    # Compute complex-conjugate node currents
    b = sfix.Array(n)
    b.assign_all(0)
    c = sfix.Array(n)
    c.assign_all(0)
    d = sfix.Array(n)
    d.assign_all(0)
    e = sfix.Array(n)
    e.assign_all(0)

    b = multvec(spG, ijG, uKreal, n)
    c = multvec(spB, ijB, uKimag, n)
    d = multvec(spB, ijB, uKreal, n)
    e = multvec(spG, ijG, uKimag, n)

```

```
    iKreal_conj = sfix.Array(n)
    iKreal_conj.assign_all(0)
    iKimag_conj = sfix.Array(n)
    iKimag_conj.assign_all(0)
    iKreal_conj[:] += b[:] - c[:]
    iKimag_conj[:] += (-1 * d[:]) - e[:]

    # Compute node powers while ingnoring slack node 1
    pN = sfix.Array(n-1)
    pN.assign_all(0)
    qN = sfix.Array(n-1)
    qN.assign_all(0)

    @for_range_opt(n-1)
    def _(i):
        pN[i] = (uKreal[i+1] * iKreal_conj[i+1]) - \
            (uKimag[i+1] * iKimag_conj[i+1])
        qN[i] = (uKreal[i+1] * iKimag_conj[i+1]) + \
            (uKimag[i+1] * iKreal_conj[i+1])

    fval = sfix.Array((n-1) * 2)
    fval.assign_part_vector(pN[:] - pK[:])
    fval.assign_part_vector(qN[:] - qK[:], base=n-1)

    return (fval, iKreal_conj, iKimag_conj)

def compute_alpha(x,dx,Fx,spG,ijG,spB,ijB,pK,qK,num_nodes,slack_voltage):
    # Compute an (approximately) optimal step length for Newton's method...
    # using a secant method.
    n = (num_nodes - 1) * 2
    alpha_old = MemValue(sfix(1)) # MemValues so variables can be used in
    ↪  while-loop
    alpha = MemValue(sfix(0.95))
    dalpha = MemValue(sfix(-0.05))
    xnew = sfix.Array(n)
    xnew.assign_all(0)
    Fj = sfix.Array(n)
    Fj.assign_all(0)
    Fjm1 = sfix.Array(n)
    Fjm1.assign_all(0)
    P = sfix(0)
    denum = sfix(0)
    y = sint(0)
    # Below steps are necessary to replace if-condition in while loop
    xnew[:] = x[:] + dx[:]
    Fjm1 = evalF(xnew,spG,ijG,spB,ijB,pK,qK,num_nodes,slack_voltage)[0]

    # To avoid overflow
    Fjm1[:] *= 10**-5
```

```
90
91      dx_alpha = sfix.Array(n)
92      dx_alpha.assign(dx)
93      dx_alpha[:] *= alpha
94      xnew[:] = x[:] + dx_alpha[:]
95      Fj = evalF(xnew,spG,ijG,spB,ijB,pK,qK,num_nodes,slack_voltage)[0]
96
97      # To avoid overflow
98      Fj[:] *= 10**-5
99      Fx[:] *= 10**-5
100
101     Fj_m = sfix.Matrix(n,1)
102     Fj_m.assign(Fj)
103     P = Fj_m.transpose().dot(Fj[:] - Fx[:])[0]
104     Fjm1_m = sfix.Matrix(n,1)
105     Fjm1_m.assign(Fjm1)
106     @for_range_opt(n)
107     def _(i):
108         Fjm1_m[i][0] *= alpha
109     denum = Fjm1_m.transpose().dot(Fjm1[:] - Fx[:])[0] - P
110     dalpha.write(dalpha * P/denum)
111     alpha_old.write(alpha)
112     alpha.write(alpha + dalpha)
113     Fj = evalF(xnew, spG, ijG, spB, ijB, pK,
114                qK, num_nodes, slack_voltage)[0] # we have to compute this
                   ↪  again for some reason
115
116     # To avoid overflow
117     Fj[:] *= 10**-5
118
119     Fjm1.assign(Fj)  # reuse value from before
120     dx_alpha.assign_all(0)
121     dx_alpha.assign(dx)
122     dx_alpha[:] *= alpha
123     xnew[:] = x[:] + dx_alpha[:]
124     Fj=evalF(xnew,spG,ijG,spB,ijB,pK,qK,num_nodes,slack_voltage)[0]
125
126     # To avoid overflow
127     Fj[:] *= 10**-5
128
129     Fj_m.assign_all(0)
130     Fj_m.assign(Fj)
131     @for_range_opt(n)
132     def _(j):
133         Fj_m[j][0] *= alpha_old
134     P.update(Fj_m.transpose().dot(Fj[:] - Fx[:])[0])
135     @for_range_opt(n)
136     def _(j):
137         Fjm1_m[j][0] = alpha * Fjm1[j]
```

```
    denum.update(Fjm1_m.transpose().dot(Fjm1[:] - Fx[:])[0] - P)
    dalpha.write(dalpha * P/denum)

    #update
    alpha_old.write(alpha)
    alpha.write(alpha + dalpha)

    return alpha
```

newton_raphson_GMRES.mpc

```
from data.rural import *
from lib import *
from Compiler.util import if_else
from Compiler import library
from Compiler import mpc_math

# optimizations
program.use_edabit(True)

# when using semi-honest dishonest majority protocols with not more than 2
↪  parties:
# program.use_trunc_pr = True
# when using a ring-based protocol:
# program.use_split(2)
# set precision
"""
Generally, 2*k must be at most the integer length
for rings and at most m-s-1 for computation modulo
an m-bit prime and statistical security s (default 40).
"""
sfix.set_precision(32,64)
cfix.set_precision(32,64)
print_float_precision(16)

def evalJ_forGMRES(x, iKreal_conj, iKimag_conj, n, spvecG,
                   rowG, fromtoG, nG, spvecB, diagpos):
    # Parse real and imaginary parts of node voltages without slack
    uKreal = sfix.Array(n - 1)
    uKimag = sfix.Array(n - 1)
    uKreal.assign(x[0:n-1])
    uKimag.assign(x[n-1:])
    H1 = sfix.Array(len(spvecG_data))
    H2 = sfix.Array(len(spvecG_data))
    max_posG = max([y-x for x, y in fromtoG_data]) + 1

    rows = []

    for i in range(nG):
        posG = fromtoG[i][1] - fromtoG[i][0]
        row = []
```

```
        for j in range(posG + 1):
            k = j + fromtoG[i][0] - 1
            row.append(rowG[k] - 1)
        rows.append(row)

    for i in range(nG):
        # changes due to index arithmetic
        posG = fromtoG[i][1] + 1 - fromtoG[i][0]
        for j in range(posG):
            k = j + fromtoG[i][0] - 1
            H1[k] = uKreal[rows[i][j]]*spvecG[k] + \
                uKimag[rows[i][j]]*spvecB[k]
            H2[k] = -1*uKreal[rows[i][j]]*spvecB[k] + \
                uKimag[rows[i][j]]*spvecG[k]
    J11 = sfix.Array(len(spvecG_data))
    J12 = sfix.Array(len(spvecG_data))
    J21 = sfix.Array(len(spvecG_data))
    J22 = sfix.Array(len(spvecG_data))

    J11.assign(H1)
    J12.assign(H2)
    J21.assign(H2)
    J22.assign( -1 * H1[:])

    for i in range(len(diagpos)):
        J11[diagpos[i]-1] += iKreal_conj[i+1]
        J12[diagpos[i]-1] -= iKimag_conj[i+1]
        J21[diagpos[i]-1] += iKimag_conj[i+1]
        J22[diagpos[i]-1] += iKreal_conj[i+1]
    return (J11,J12,J21,J22)

def evalAb(P, F, J11, J12, J21, J22, rowG, fromtoG, n, n2):
    F_m = sfix.Matrix(n,1)
    F_m.assign(F)
    b = sfix.Matrix(n,1)
    b.assign(P.dot(F_m))

    A = sfix.Matrix(n,n)
    A.assign_all(0) # otherwise MP-SPDZ reuses old values of A

    max_posG = max([y-x for x, y in fromtoG_data]) + 1
    cols = []

    for i in range(n2):
        posG = fromtoG[i][1] - fromtoG[i][0]
        col = []
        for j in range(posG + 1):
            k = j + fromtoG[i][0] - 1
            col.append(rowG[k] - 1)
```

```
89          cols.append(col)
90
91      for i in range(n):
92          for j in range(n2):
93              posG = fromtoG[j][1] - fromtoG[j][0]
94              for k in range(posG+1):
95                  l = k + fromtoG[j][0] - 1
96                  A[i][j] += P[i][cols[j][k]] * J11[l] + P[i][cols[j][k] + n2] *
                    ↪  J21[l]
97
98      for i in range(n):
99          for j in range(n2):
100             posG = fromtoG[j][1] - fromtoG[j][0]
101             for k in range(posG+1):
102                 l = k + fromtoG[j][0] - 1
103                 A[i][j + n2] += P[i][cols[j][k]] * J12[l] + \
104                     P[i][cols[j][k] + n2] * J22[l]
105
106     return (A,b)
107
108 def GMRESsolve(A,b,x0,n,num_iter,maxiter):
109     if num_iter == 4:
110         maxiter = 7
111     elif num_iter == 5:
112         maxiter = 4
113     elif num_iter > 0:
114         maxiter = 1
115     V = sfix.Matrix(n, maxiter + 1)  # Arnoldi (orthonormal) basis vectors
116     V.assign_all(0)
117     H = sfix.Matrix(maxiter + 1,maxiter + 1) # Hessenberg Matrix
118     H.assign_all(0)
119     c = sfix.Array(maxiter+1)  # "cosine" terms of the Givens rotation
        ↪  matrices
120     c.assign_all(0)
121     s = sfix.Array(maxiter+1)  # "sine" terms of the Givnes rotation matrices
122     s.assign_all(0)
123     x = sfix.Array(n)
124     x.assign_all(0)
125     beta_vec = sfix.Array(maxiter+1)
126     beta_vec.assign_all(0)
127     x0_m = sfix.Matrix(n,1)
128     x0_m.assign(x0)
129     r0 = sfix.Matrix(n, 1)
130     r0.assign_all(0)
131     r_1 = sfix.Matrix(n, 1)
132     r_1.assign_all(0)
133     r0.assign(b[:] - A.dot(x0)[:])
134     tmp1 = r0.transpose().dot(r0)[0][0]
135
```

```
    tmp2 = 1/mpc_math.sqrt(tmp1)
    beta = tmp1 * tmp2
    r_1[:] = r0[:] * tmp2
    for i in range(n):
        V[i][0] = r_1[i][0]

    beta_vec[0] = beta

    for j in range(maxiter):
        # Arnoldi iteration / orthogonalization
        w = sfix.Matrix(n,1)
        w.assign_all(0)
        wj = sfix.Matrix(n, 1)
        wj.assign_all(0)
        for i in range(n):
            w[i][0] = V[i][j]
        wj = A.dot(w)
        for i in range(j+1):
            V_i = sfix.Matrix(n,1)
            V_i.assign_all(0)
            for k in range(n):
                V_i[k][0] = V[k][i]
            H[i][j] = wj.transpose().dot(V_i)[0][0]
            for k in range(n):
                wj[k][0] -= (V_i[k][0] * H[i][j])
        wjwj=wj.transpose().dot(wj)[0][0]
        norm_wj = 1/mpc_math.sqrt(wjwj)
        H[j+1][j] = norm_wj * wjwj
        # Givens rotations
        for i in range(j):
            tmp = c[i] * H[i][j] + s[i] * H[i+1][j]
            H[i+1][j] = -s[i] * H[i][j] + c[i] * H[i+1][j]
            H[i][j] = tmp

        # compute coefficients of the next rotation matrix
        tmp = H[j][j]**2 + H[j+1][j]**2
        den = 1/mpc_math.sqrt(tmp)
        c[j] = H[j][j] * den
        s[j] = H[j+1][j] * den
        # eliminate subdiagonal element in current column
        H[j][j] = c[j] * H[j][j] + s[j] * H[j+1][j]
        H[j+1][j] = 0

        # update residual
        dummy = -1 * s[j] * beta_vec[j]
        beta_vec[j] *= c[j]
        res_norm = 10**9 * dummy**2
        beta_vec[j+1] = dummy

```

```
185             for i in range(n):
186                 wj[i][0] *= norm_wj
187             for i in range(n):
188                 V[i][j+1] = wj[i][0]
189 
190         #  Compute LS solution via back substitution, i.e. solve H * y = beta_vec
191         j = maxiter
192         y = sfix.Array(maxiter+1)
193         y.assign_all(0)
194         y_m = sfix.Matrix(maxiter+1,1)
195         y_m.assign_all(0)
196         H_m = sfix.Matrix(maxiter+1,1)
197         H_m.assign_all(0)
198         for k in range(j,-1,-1):
199             #@if_e(k >= j)
200             #def _():
201             if(k >= j):
202                 y[k] = beta_vec[k]/H[k][k]
203             else:
204                 y_m.assign_all(0)
205                 H_m.assign_all(0)
206                 #@for_range(k+1,j)
207                 #def _(l):
208                 for l in range(k+1,j):
209                     y_m[l][0] = y[l]
210                     H_m[l][0] = H[k][l]
211                 y[k] = (beta_vec[k] - H_m.transpose().dot(y_m)[0][0])/H[k][k]
212 
213         y_m.assign_all(0)
214         y_m.assign(y)
215         x[:] += x0[:]
216         x[:] += V.dot(y_m)[:]
217 
218         return x
219 
220 def main():
221     max_iter = newton_max_iter  # public
222     tol_iter = cfix(10**-6)  # public
223     gmres_tol = cfix(10**-9)  # public
224     gmres_maxiter = gmres_solve_max_iter  # public
225 
226     dim_spvec = len(spvecG_data)  # public
227     dim_spG = len(spG_data)  # public
228     nG = num_nodes - 1  # public
229     n2 = nG # public
230     n = nG * 2  # public
231 
232     dx0 = sfix.Array(n)  # public
233     dx0.assign_all(0)
```

```
234
235     slack_voltage = cfix(230)  # public
236
237     x = sfix.Array((num_nodes-1) * 2)
238
239     for i in range(num_nodes-1):
240         x[i] = sfix(230)  # starting guess
241
242     for i in range(num_nodes-1,num_nodes-1 * 2):
243         x[i] = sfix(0)
244
245     spG = cfix.Array(dim_spG) # public
246     spG.assign(spG_data)
247
248     ijG = cint.Array(dim_spG)  # public
249     ijG.assign(ijG_data)
250
251     spB = cfix.Array(dim_spG) # public
252     spB.assign(spB_data)
253
254     ijB = cint.Array(dim_spG)  # public
255     ijB.assign(ijB_data)
256
257     pK = sfix.Array(num_nodes-1)
258     pK.assign(pK_data)
259
260     qK = sfix.Array(num_nodes-1)
261     qK.assign(qK_data)
262
263     rowG = cint.Array(dim_spvec)  # public
264     rowG.assign(rowG_data)
265
266     spvecG = sfix.Array(dim_spvec)
267     spvecG.assign(spvecG_data)
268
269     spvecB = sfix.Array(dim_spvec)
270     spvecB.assign(spvecB_data)
271
272     J = sfix.Matrix(2*(num_nodes-1), 2*(num_nodes-1))
273
274     diagpos = diagpos_data
275
276     P = sfix.Matrix(n,n)
277     for i in range(n):
278         for j in range(n):
279             P[i][j] = P_data[i][j]
280
281     alpha_ = MemValue(sfix(1))
282
```

```
    num_iterations = 0

    for i in range(max_iter):
        (F, iKreal_conj, iKimag_conj) = evalF(x, spG, ijG, spB,
                            ijB,pK, qK, num_nodes, slack_voltage)
        F_alpha = sfix.Array(n)
        F_alpha.assign(F)

        (J11, J12, J21, J22) = evalJ_forGMRES(x, iKreal_conj,
                            iKimag_conj, num_nodes, spvecG_data, rowG,
                            fromtoG_data, nG, spvecB_data, diagpos)

        (A, b) = evalAb(P, F, J11, J12, J21, J22, rowG, fromtoG_data, n, n2)

        dx = GMRESsolve(A, b, dx0, n, num_iterations, gmres_maxiter)
        dx[:] *= -1
        alpha_.write(compute_alpha(x, dx, F_alpha, spG, ijG, spB, ijB, pK, qK,
                        num_nodes,slack_voltage))
        dx_alpha = sfix.Array(n)
        dx_alpha.assign(dx)
        dx_alpha[:] *= alpha_
        x[:] += dx_alpha[:]
        num_iterations += 1

    print_ln('x=%s',x.reveal())

main()
```

newton_raphson_lu.mpc

```
from Compiler.util import if_else
from data.rural import *
from lib import *

# optimizations
program.use_edabit(True)

# when using semi-honest dishonest majority protocols:
# program.use_trunc_pr = True
# when using a ring-based protocol:
# program.use_split(2)
# set precision
"""
Generally, 2*k must be at most the integer length
for rings and at most m-s-1 for computation modulo
an m-bit prime and statistical security s (default 40).
"""
sfix.set_precision(32,64)
cfix.set_precision(32,64)
print_float_precision(16)

```

```
def evalJ_forLU(x, iKreal_conj, iKimag_conj,n, spvecG,
    rowG, fromtoG, nG, spvecB):
    # Parse real and imaginary parts of node voltages without slack
    uKreal = sint.Array(n - 1)
    uKimag = sint.Array(n - 1)
    uKreal.assign(x[0:n-1])
    uKimag.assign(x[n-1:])
    # assuming same sparsity pattern of G,B
    H1 = sfix.Matrix(n-1,n-1)
    H1.assign_all(0)
    H2 = sfix.Matrix(n-1,n-1)
    H2.assign_all(0)
    max_posG = max([y-x for x, y in fromtoG_data]) + 1

    rows = []
    for i in range(nG):
        posG = fromtoG[i][1] - fromtoG[i][0]
        row = []
        for j in range(posG + 1):
            k = j + fromtoG[i][0] - 1
            row.append(rowG[k] - 1)
        rows.append(row)

    for i in range(nG):
        posG = fromtoG[i][1] + 1 - fromtoG[i][0]
        for j in range(posG):
            k = j + fromtoG[i][0] - 1
            H1[rows[i][j]][i] = uKreal[rows[i][j]]*spvecG[k] + \
                uKimag[rows[i][j]]*spvecB[k]
            H2[rows[i][j]][i] = -1*uKreal[rows[i][j]]*spvecB[k] + \
                uKimag[rows[i][j]]*spvecG[k]

    # Construct Jacobian
    # Wolter, eq. (4.37), (4.39) without factor "3"
    J = sfix.Matrix(2*(n-1),2*(n-1))
    J.assign_all(0)

    # J11
    for i in range(n-1):
        J[i][i] += iKreal_conj[i+1] # ignore slack node
        for j in range(n-1):
            J[i][j] += H1[i][j]
    # J21
    for i in range(n-1,2*(n-1)):
        J[i][i - (n - 1)] += iKimag_conj[i - (n - 1) + 1] # ignore slack node
        for j in range(n-1):
            J[i][j] += H2[i - (n - 1)][j]
    # J12
    for i in range(n-1):
```

```
71          J[i][i + n -1] -= iKimag_conj[i+1] # ignore slack node
72          for j in range(n-1,2*(n-1)):
73              J[i][j] += H2[i][j - (n - 1)]
74
75      # J22
76      for i in range(n-1,2 * (n-1)):
77          J[i][i] += iKreal_conj[i - (n -1) + 1] # ignore slack node
78          for j in range(n-1,2*(n-1)):
79              J[i][j] += -1 * H1[i - (n-1)][j - (n-1)]
80      return J
81
82  # This takes very long, not sure how to optimize
83
84  def LUdecomposition(A,n):
85      # LU decomposition of Matrix A using Dolittle's algorithm
86      # sparsity of A not worth to be exploited with secret sharing since
87      # A only occurs in sums
88      U = sfix.Matrix(n,n)
89      U.assign_all(0)
90      L = sfix.Matrix(n,n)
91      L.assign_all(0)
92      x = sfix(0)
93
94      for i in range(n):
95          L[i][i] = sfix(1)
96
97      for i in range(n):
98          for j in range(i,n):
99              x=sfix(0)
100             for k in range(i):
101                 x +=(L[i][k] * U[k][j])
102             U[i][j] = A[i][j] - x
103             x=sfix(0)
104             for k in range(i):
105                 x +=(L[j][k] * U[k][i])
106             L[j][i] = (A[j][i] - x)/U[i][i]
107     return (L,U)
108
109 def LUsolve(L,U,b,n):
110     # solves linear equation system LUx=b by solving Ly=b and Ux=y
111     y = sfix.Array(n)
112     y.assign_all(0)
113     tmp = sfix(0)
114
115     for i in range(n):
116         tmp = b[i]
117         for j in range(i):
118             tmp -= L[i][j] * y[j]
119         y[i] = tmp / L[i][i]
```

```
120
121     x = sfix.Array(n)
122     x.assign_all(0)
123
124     for i in range(n-1,-1,-1):
125         tmp = y[i]
126         for j in range(i+1,n):
127             tmp -= U[i][j] * x[j]
128         x[i] = tmp / U[i][i]
129
130     return x
131
132 def main():
133     max_iter = 4  # public
134     tol_iter = cfix(10**-6)  # public
135     dim_spvec = len(spvecG_data) #public
136     dim_spG = len(spG_data) #public
137     nG = num_nodes - 1  # public
138     n = nG * 2  # public
139
140     slack_voltage = cfix(230)  # public
141
142     x = sfix.Array((num_nodes-1) * 2)
143
144     @for_range(0, num_nodes-1)
145     def _(i):
146         x[i] = sfix(207) # starting guess
147
148     @for_range(num_nodes-1, (num_nodes-1) * 2)
149     def _(i):
150         x[i] = sfix(0)
151
152     spG = cfix.Array(dim_spG)
153     spG.assign(spG_data)
154
155     ijG = cint.Array(dim_spG)  # public
156     ijG.assign(ijG_data)
157
158     spB = cfix.Array(dim_spG)
159     spB.assign(spB_data)
160
161     ijB = cint.Array(dim_spG)  # public
162     ijB.assign(ijB_data)
163
164     pK = sfix.Array(num_nodes-1)
165     pK.assign(pK_data)
166
167     qK = sfix.Array(num_nodes-1)
168     qK.assign(qK_data)
```

```
169
170     rowG = cint.Array(dim_spvec)  # public
171     rowG.assign(rowG_data)
172
173     spvecG = sfix.Array(dim_spvec)
174     spvecG.assign(spvecG_data)
175
176     spvecB = sfix.Array(dim_spvec)
177     spvecB.assign(spvecB_data)
178
179     J = sfix.Matrix(2*(num_nodes-1), 2*(num_nodes-1))
180     alpha_ = sfix(1)
181
182     for i in range(max_iter):
183         (F, iKreal_conj, iKimag_conj) = evalF(x, spG, ijG, spB, ijB, pK,
184                         qK, num_nodes, slack_voltage)
185         F_alpha = sfix.Array(n)
186         F_alpha.assign(F)
187
188         J = evalJ_forLU(x,iKreal_conj,iKimag_conj,num_nodes,
189                 spvecG,rowG,fromtoG_data,nG,spvecB)
190         (L,U) = LUdecomposition(J,n)
191
192         dx = LUsolve(L,U,F,n)
193         dx[:] *= -1
194         alpha_=compute_alpha(x, dx, F_alpha, spG, ijG, spB, ijB, pK, qK,
195                 num_nodes,slack_voltage)
196         dx_alpha = sfix.Array(n)
197         dx_alpha.assign(dx)
198         dx_alpha[:] *= alpha_
199         x[:] += dx_alpha[:]
200
201     print_ln('x=%s',x.reveal())
202
203 main()
```

B Gaps in the Proofs of [AHKP22]

We have discovered some small gaps in the proofs in [AHKP22]. These gaps motivate the changes we have made to the RGS and KMHE definitions. Note that we do believe that the KMHE and RGS constructions in [AHKP22] also fulfill our new definitions. Therefore, this is mainly an issue of formal correctness.

The gaps occur in their construction of incremental decomposable randomized encodings (iDRE) [AHKP22, Figure 2]. An iDRE is a multi-party protocol for ℓ parties designed to evaluate a function f on some input $x \in \{0,1\}^m$.The construction in [AHKP22] relies on a (weak) KMHE scheme and an RGS scheme. Each party starts off by sampling KMHE keys $\{e^0_{i,j}, e^1_{i,j}\}_{i\in[m]}$ for each input bit x_i in the case of the first party, or KMHE key transformation functions $\{e^0_{i,j}, e^1_{i,j}\}_{i\in[m]}$ for the remaining parties. This is done offline and ahead of the protocol. Once the protocol starts, the first party starts by garbling f, leading to a garbling $\mathcal{C}$ and input labels $L = (L^0_1, L^1_1, \ldots, L^0_m, L^1_m)$ for all possible inputs $x \in \{0,1\}^m$. Each input label is then separately encrypted using the KMHE scheme. The garbling $\mathcal{C}$ and the ciphertexts are then passed on to the next party. Each following party now rerandomizes the garbling via the Rerand algorithm of the RGS scheme. Rerand also outputs the permutations applied to the input labels via π_{En}. These are also applied to the encrypted labels via the KMHE message homomorphism. Lastly, the keys in each label ciphertext are updated by applying the functions given by $\{e^0_{i,j}, e^1_{i,j}\}_{i\in[m]}$ via the KMHE key homomorphism. This last procedure ensures that the label ciphertexts can only be decrypted if all parties collaborate to recompute the keys. Once all parties have finished this process the garbling can be evaluated on some given input x. To do this, the parties together reveal the keys for the input labels $L^{x_1}_1, \ldots, L^{x_m}_m$. These can then be used to evaluate the garbling and obtain the output $f(x)$.

The relevant security notion is called privacy. It ensures that a semi-honest adversary that corrupts all but one party j^*, learns nothing but the function f, the output $f(x)$, and the corrupted parties' inputs. Importantly, it should not be able to learn the honest parties input bit even given the input labels for the whole input x. We argue that the security properties of the weak KMHE scheme and the RGS scheme do not suffice to prove privacy of the iDRE construction.

The proof is given in Claim 3 in [AHKP22]. It works roughly as follows: First CPA security of the KMHE scheme is used to argue that the KMHE ciphertexts

only reveal the labels for the input x, the inactive labels are hidden. Then, rerand privacy of the RGS is used to argue that the garbling output by the $(j^* + 1)^{\text{th}}$ party looks like a fresh garbling. By garbling privacy of the RGS, this fresh garbling then ensures that given the input labels for x, only the output $f(x)$ is leaked, as desired by privacy of the iDRE scheme.

The first gap in this proof occurs in the use of rerand privacy. The rerand privacy in [AHKP22] is the same as our definition but with $t = 1$ fixed. This means it guarantees that a honest rerandomization of a semi-honest garbling looks like a fresh honest garbling. However, the $(j^* + 1)^{\text{th}}$ party does not obtain a semi-honest garbling, it obtains a garbling that has been semi-honestly rerandomized several times before by the previous parties. This case is addressed for our definition with arbitrary t. It is also not clear whether the $t = 1$ definition implies the definition with arbitrary t. Using a standard hybrid argument seems to not work since the honest rerandomization is only guaranteed to rerandomize a fresh garbling, not a semi-honestly rerandomized one. However, we were not able to formally separate the two definitions, e.g. by showing a construction that fulfills the $t = 1$ variant but not the arbitrary t variant. This would require an RGS construction where somehow a semi-honest rerandomization breaks the following honest rerandomization. This is difficult as the only difference between a semi-honest and an honest rerandomization is that the adversary knows the randomness for the former. Therefore it is hard to design the rerandomization such that in one case it works but in the other it breaks. We resolve this issue by changing the definition of rerand privacy (Definition 30) to consider an arbitrary number t of intermediate semi-honest rerandomizations.

The second gap is with the KMHE scheme itself. Acharya et al. [AHKP22] only require this to be a *weak* KMHE scheme, meaning it does not fulfill their notion of KMH privacy. KMH privacy, however, is the only notion in [AHKP22] that guarantees that an output of Eval looks like a fresh encryption, i.e. that Eval rerandomizes the ciphertext. KMH correctness does not suffice since it does not say anything about the distribution of the output of Eval, just that it is a valid ciphertext. Without the ability to replace an output of Eval with a fresh encryption, it is not possible to apply CPA security. To address this we have split KMH privacy up into KMH rerandomizability (Definition 25) and key privacy under leakage (Definition 26) where now KMH rerandomizability is part of the (weak) KMHE definition and key privacy under leakage is required for a strong KMHE scheme.

We now discuss where exactly these issues occur in the proof's reductions and how they can be fixed using our definitions. Mainly of interest is the proof of indistinguishability of hybrids H_2 and H_3 in the proof of their Claim 3, which they prove via a reduction to CPA security of the KMHE scheme. The hybrid H_3 there is defined as the adversary's view in a real-world execution of the iDRE protocol. Acharya et al. [AHKP22] then assume a distinguisher D that can distinguish

hybrid H_2 and H_3 and then construct a distinguisher D' that uses D to break CPA security of the KMHE scheme. However, we argue that D' does not correctly simulate the hybrid H_3.

The first issue is that D' creates the value F_{j^*} as a fresh semi-honest garbling, even though in a real-world execution of the iDRE protocol this would be a semi-honestly rerandomized garbling since j^* is not necessarily the first party in the chain of parties. This then allows Acharya et al. to later use rerand privacy with $t = 1$. If we fix the definition of F_{j^*} to be a rerandomization as in the real-world, then it is possible to use our new rerand privacy definition with arbitrary t to argue indistinguishability of the hybrids H_2 and H_1.

The second issue with the distinguisher D' is that the encrypted labels are created as fresh encryptions instead of being outputs of Eval as they would be in the real world. This can be fixed by defining H_3 to use fresh encryptions and adding another hybrid H_4 which then actually corresponds to the real-world execution. Indistinguishability of H_3 and H_4 then follows from KMH rerandomizability of the KMHE scheme while H_3 and H_2 uses CPA security as before.

Note that Acharya et al. have since fixed these gaps by switching from the standard BHHO scheme to the expanded version, which enables perfect rerandomization and by proxy also perfect rerand privacy.