



Title	Isogeny-Based Multi- and Proxy Signatures, and a Private Tracing Protocol Based on Multi-Signatures
Author(s)	De Goyon, Mathieu
Citation	大阪大学, 2026, 博士論文
Version Type	VoR
URL	https://doi.org/10.18910/105281
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

<https://ir.library.osaka-u.ac.jp/>

The University of Osaka

Doctoral Dissertation

**Isogeny-Based Multi- and Proxy Signatures,
and a Private Tracing Protocol Based on
Multi-Signatures**



**Graduate School of Engineering,
The University of Osaka**

DE GOYON Mathieu

October 2025

Supervisor: Professor Atsuko Miyaji

Abstract

The potential development of large-scale quantum computers has raised serious concerns about the security of classical cryptographic systems. In response, the field of post-quantum cryptography has emerged to design cryptographic schemes that remain secure even against quantum adversaries. Isogeny-based cryptography is the most recent addition to post-quantum cryptography and started garnering attention in the 2010s. Digital signature schemes are a fundamental part of public key cryptography, providing authenticity, integrity, and non-repudiation for digital messages. They are essential for secure communication, software updates, electronic transactions, and virtually all systems requiring trust and verification. Although the traditional digital signature scheme consists only of a single prover and verifier, numerous variants have been developed over the years to address specific needs, such as multi-signatures, ring signatures, group signatures, and proxy signatures. These variants extend the functionality of traditional signatures to support more complex security models in real-world applications, such as contact tracing. As isogeny-based cryptography is a relatively young research field, many commonly used cryptographic primitives, such as variants of signature schemes, have not yet been fully developed or widely implemented. This leaves room for exploration and optimization in designing secure and efficient isogeny-based alternatives, while taking advantage of its strength with its small keys and signature compared to other post-quantum based constructions such as lattice.

Multi-signature schemes have been extensively studied for decades and form a well-established area of research in cryptography. These schemes enable a group of users to collaboratively generate a single, compact signature on a shared message, ensuring that all designated parties have approved the content. Their efficiency and security properties make them particularly valuable in applications such as blockchain transactions and distributed systems. Active security in multi-signature schemes requires the signature to be unforgeable even in the worst-case scenario, where malicious signers would collude to generate a multi-signature in a group with a single honest signer. Our first contribution is to propose an actively secure isogeny-based multi-signature scheme.

Proxy signature schemes have been the subject of extensive research over the years, offering cryptographic solutions for secure delegation of signing authority. These schemes allow an original signer to designate a proxy signer to generate signatures on their behalf, within the limitations imposed by the original signer. Such mechanisms are particularly useful in settings like secure communications, delegated access control, and corporate approval systems where it is essential for operations to continue even in the absence of the designated signer. Proxy

signatures are required to be unforgeable even if either the original or the proxy signer is malicious, which is commonly referred to as the existential unforgeability under chosen message attack and chosen warrant attack. Some security models with more stringent requirements, such as existential unforgeability under chosen message attack with proxy key exposition (EUF-CMA-PKE), require the proxy signature to be unforgeable even if the proxy signing key are leaked. Our second contribution is to propose two proxy signatures satisfying both models using isogeny-based cryptography.

Contact tracing applications have become a crucial tool in managing public health during pandemics, relying on digital protocols to identify and notify individuals exposed to infectious diseases. These systems must balance effective tracking with strong privacy guarantees, ensuring that users' identities remain confidential. At the same time, it is essential that they reflect accurately the situation on the ground. To achieve this, advanced cryptographic techniques, such as ring signatures and multi-signatures, can be employed, making contact tracing a promising real-world application for these variant signature schemes. Our third contribution is to propose a contact tracing application which correctly reflects the interactions between patients and their close contacts while preserving the privacy of its users.

The work of this dissertation is divided into studies about isogeny-based multi-signatures, isogeny-based proxy signatures, and efficient multi-party contact tracing protocols.

- A proposal of an actively secure isogeny-based multi-signature that uses zero-knowledge proofs and commitments to guarantee the existential unforgeability of the signature.
- A proposal of two isogeny-based proxy signature protocols guaranteeing existential unforgeability in the case of a corrupted original or proxy signer, one designed to remain secure even in the event of proxy key leakage, and another constructed under the assumption that the proxy key remains secret.
- A proposal of a contact tracing protocol using ID-based ring signatures and multi-signatures to accurately reflect the interactions between users in an efficient way while preserving their privacy.

The first study is composed of the first isogeny-based multi-signature scheme by extending the isogeny-based signature scheme CSI-SharK [ABCP22] to the multi-party setting. Our scheme is proven unforgeable in the random oracle model under the MT-GAIP and C_{k-1} -VPwAI assumptions. To achieve security, a new proof system based on zero-knowledge proof and commitments is proposed by modifying the piecewise verifiable proof system proposed in the Distributed Key Generation protocol CSI-RASHI [BDPV21].

The second study is composed of two isogeny-based proxy signature protocols, PKE and CWA, based on CSI-SharK. Both are proven unforgeable under the MT-GAIP and C_{k-1} -VPwAI assumptions in the quantum random oracle model and considers chosen message attacks (corrupted original signer) and chosen warrant attacks (corrupted proxy signer), but only the former considers the risk of proxy key leakage.

The third study is composed of a privacy-preserving contact tracing application that accurately represents the interaction of its users, such as the number of participants. It is achieved by using the contact tracing protocol proposed by Liu et al. [LAY⁺20] and adapt it to the multi-party setting by incorporating a multi-signature [DGNW20, MM00] as a proof of the meeting and a ring signature [CYH05] as a way to alert the close contacts while preserving the patient’s privacy.

To summarize, the first two studies adapt a variant signature scheme, i.e., a multi-signature and a proxy signature, to the isogeny-based setting, while the third study proposes a potential application of such variant schemes. The multi-signature scheme from the first study can be integrated into the third study’s contact tracing protocol by replacing it, although the protocol would need to be adjusted to compensate for the lack of ID-based ring signatures in isogeny-based cryptography.

Acknowledgments

I would like to express my deepest gratitude to everyone who has supported me throughout my Ph.D. journey. First and foremost, I am deeply grateful to my supervisor, Professor Atsuko Miyaji, for giving me a chance to experience the field of research as an exchange student before accepting me once again as a Ph.D student. Her invaluable support and guidance throughout the course of those 3 years as a Ph.D student enabled me to make it this far.

Additionally, I am deeply grateful to the NEC Corporation for its C&C Fellowship, as well as the Jisedai Program. Their generous support eased my financial burdens, allowing to focus completely on my research. I am also deeply grateful to the professors, staff, and colleagues at Miyaji Lab, who were always there whenever I needed it.

I want to especially thank Hide Miyaji, for welcoming me with open arms as soon as I arrived for the first time in Japan and supporting me throughout my Ph.D journey. I am also profoundly grateful to Mohamed Bourefis, my French comrade, who shares a very similar path with me and without whom my Ph.D would have been very different.

My heartfelt thanks to my fellow international students - Nasratullah Ghafoori, Kaiming Chen, Sai Veerya Mahadevan, Po-Chu Hsu, Bingchang He, and Pengxuan Wei- and the brave Japanese students who did their best at communicating with me despite the language barrier - Kodai Hayashida, Akira Fujimoto, Shogo Kuramoto, Yamada Rintaro - who made my stay in Miyaji Lab all the more enjoyable.

Finally, I extend my deepest thanks to my family, and more specifically, my parents, Frederic de Goyon and Beatrice Mevel, my little sister Amelie de Goyon, and my grandparents Nicole Mevel and Gerard Mevel for their love and support throughout my Ph.d despite the distance separating us.

Contents

Abstract	ii
Acknowledgments	v
Contents	vi
List of figures	ix
List of tables	x
1 Introduction	1
1.1 Background	1
1.1.1 Multi-Signature	3
1.1.2 Proxy Signature	4
1.1.3 Contact-Tracing	5
1.2 Contributions	6
1.3 Organization	7
2 Preliminaries	10
2.1 Notations	10
2.2 Signatures	11
2.2.1 Sigma Protocols and Fiat-Shamir Transform	11
2.2.2 Digital Signature	12
2.2.3 Multi-Signature	13
2.2.4 Proxy Signature	14
2.2.5 Threshold Signature	18
2.2.6 Ring Signature	19
2.3 Isogeny-based Cryptography	20
2.3.1 Isogenies	20
2.3.2 SQISign	20
2.3.3 CSIDH	23
2.3.4 Hard-Homogeneous Spaces	23
2.3.5 SeaSign	24
2.3.6 CSI-SharK	26

2.3.7	Signature Comparison	28
2.4	Tracing Components	29
2.4.1	ECDLP and Bilinear Pairing	29
2.4.2	DGNW Multi-Signature	30
2.4.3	MM Multi-Signature	31
2.4.4	CYH ID-based Ring Signature	31
3	Related Works	33
3.1	CSI-RASHI and ThreshER SharK	33
3.1.1	Piecewise Verifiable Proofs	33
3.1.2	Threshold Signature Protocol	35
3.2	DualMS and MuSig-L	37
3.2.1	MuSig-L	37
3.2.2	DualMS	41
3.3	CSI-PS and Generic Construction	44
3.3.1	Generic Construction	44
3.3.2	CSI-PS	45
3.4	Privacy-Preserving Contact Tracing Protocol	48
4	Actively Secure Isogeny-Based Multi-Signature	52
4.1	Group Action Sub-Protocol	53
4.1.1	(Structured) Proof System	53
4.1.2	Round-Robin Group Action Computation	56
4.2	Construction	57
4.3	Analysis	58
4.3.1	Correctness	59
4.3.2	Security	60
4.4	Comparison	68
4.4.1	Multi-Signature Comparison	68
4.4.2	Our Protocol and ThreshER SharK	69
4.5	Implementation	70
4.6	Conclusion	70
5	Isogeny-based Proxy Signature	72
5.1	Properties of Proxy Signature	72
5.1.1	Formal Definition of Security Properties	72
5.1.2	Certificate and Signature Binding	75
5.2	Protocol	76
5.2.1	Choice of Base Signature	76
5.2.2	CWA Proxy Signature	77
5.2.3	PKE Proxy Signature	79
5.3	Security	82

5.3.1	Correctness	82
5.3.2	Security	83
5.4	Comparison	87
5.5	Conclusion	88
6	Efficient Multi-Party Contact Tracing Protocol	90
6.1	Security Model	90
6.1.1	System Assumptions	91
6.1.2	Threat Model	91
6.2	Proposed Contact Tracing Protocols	92
6.2.1	Design of our Multi-Party Tracing Protocol	92
6.2.2	Registration Phase	93
6.2.3	Meeting Phase	93
6.2.4	Alerting Phase	96
6.2.5	Tracing Phase	96
6.3	Security	97
6.3.1	Threat Model Justification	97
6.3.2	Security Analysis	98
6.4	Comparison	101
6.5	Conclusion	102
7	Conclusion and Future Works	103
7.1	Summary of our Results	103
7.2	Future Works	104
	List of publications	105
	References	106

List of Figures

1.1	Overview of the dissertation	8
2.1	SQISign identification protocol	21
3.1	Prover and Verifier of the PVP schemes	34
3.2	CSI RASHI structured (single) DKG protocol	35
3.3	Proxy signature generic construction	44
4.1	Prover and Verifier of the structured proof system	54
4.2	SP^{Single} and $SP^{Structured}$	56
5.1	Proxy signature setting based on SQI-Sign	76
6.1	Verification Phase with $n = 3$	94
6.2	Proof Generation Phase: OF-MP Tracing and PF-MP Tracing	96

List of Tables

2.1	Notations used in this paper	10
2.2	Comparison between isogeny-based signatures (size in kB)	28
4.1	Comparison between PVP [BDPV21] and PS in the structured/single case	56
4.2	Comparison between Multi-I ($k = 2^4, 2^{12}$), MuSig-L [BTT22] and DualMS [Che23] (size in kB)	68
4.3	Comparison between Multi-I and ThersHER SharK [ABCP23b]	69
4.4	Multi-I benchmark results for $n = 2, 3, 4$ with ($ZK = 0$) and without ($ZK = 1$) proof systems (in seconds)	70
5.1	Comparison between CSI-PS, PKE, CWA (in kB, seconds)	89
6.1	Comparison between OF-MP Tracing, PF-MP Tracing and Primitive [LAY ⁺ 20] .	102

Chapter 1

Introduction

1.1 Background

Post-Quantum Cryptography (PQC), which seeks to design systems secure against quantum computers, has so far been developed along the following main approaches: Lattice-Based Cryptography, Code-Based Cryptography, Multivariate Polynomial Cryptography, Hash-Based Cryptography, and Isogeny-Based Cryptography. Today, it is estimated that PQC represents less than 0.1% of operational cryptosystems currently in use [SHY⁺24]. While current quantum computers lack the qubits necessary to break the security of current implementations, the remaining 99.9% would be vulnerable if a viable quantum computer was developed.

While lattice-based cryptography currently dominates the landscape of post-quantum schemes due to its versatility, efficiency, and well-understood hardness assumptions, isogeny-based cryptography offers distinct advantages. In particular, isogeny schemes achieve much smaller key and signature sizes compared to lattice-based constructions, making them appealing in scenarios where bandwidth and storage are constrained. Research on Isogeny-based Cryptography started in 1997 when Couveignes [Cou06], and later independently Stolbunov and Rostovstev [Sto10], proposed a scheme commonly referred to as Couveignes-Rostovstev-Stolbunov (CRS) scheme. However, CRS suffered from slow computations due to its use of ordinary elliptic curves, causing it to be deemed impractical. Isogeny-based cryptography only started gaining attracting in 2011 and 2014 when Jao et al. [JF11, FJP14] proposed a new isogeny-based key exchange scheme, Supersingular Isogeny Diffie-Hellman (SIDH). An alternative key exchange scheme to SIDH was proposed in 2018 by Castryck et al. [CD22], which was obtained by modifying CRS. Instead of using ordinary elliptic curves, Commutative Supersingular Diffie-Hellman (CSIDH), only considers the restriction of the endomorphism to the base field, therefore achieving Commutativity despite using supersingular elliptic curves. CSIDH relies on the group action between the ideal class group of the endomorphism ring and the set of ideal classes of elliptic curves over $\mathbb{F}_p$. An alternative group action to CSIDH, SCALLOP [FFK⁺23] was later proposed which uses the set of oriented supersingular curves instead. SIDH was later proven to be insecure in 2022 [Rob22], after which the technique used to break its security can be used to embed an isogeny into an isogeny of higher dimension. Such a technique led to an updated version of

SCALLOP, called SCALLOP-HD [CLP24] which is built upon SCALLOP but uses a different orientation taking advantage of isogenies of dimension 2. More recently, a new group action, PEGASIS [DEF+25] which does not require the computation of the structure of the class group and relies on 4-dimensional isogenies.

Isogeny-based signature schemes can be divided into two categories: CSIDH-based signatures such as Seasign [FG19], CSI-FiSh [BKV19], CSI-SharK [ABCP22], and signatures based on the Deuring Correspondence, such as SQISign [FKL+20]. The possibility of using isogenies of higher dimension led to several variants of SQISign being proposed, notably SQISignHD [DLRW24] which uses isogenies of degree 4 and 8, and SQISign2D-East [NOC+24] SQISign2D-West [BDF+24], and SQIPrime [DF24] which rely on isogenies of degree 2 and achieve better performances than SQISign. More recently, a new isogeny-based signature, PRISM [BBC+25], was introduced based on the Deuring Correspondence but taking a different direction from SQISign by relying on the hash-and-sign paradigm. Commutative Supersingular Isogeny based Fiat-Shamir (CSI-FiSh) and its variant CSI-FiSh with Sharing-friendly Keys (CSI-SharK) in particular improve the signature generation from SeaSign by computing the underlying ideal class group, allowing elements to be sampled uniformly at random. Moreover, the commutativity of CSI-FiSh and CSI-SharK, inherited from being based on the CSIDH group action, make them highly flexible.

Digital signature schemes are foundational tools in modern cryptography, providing authentication, integrity, and non-repudiation for digital communications and transactions. However, as digital systems become more interconnected and complex, basic signature schemes may no longer suffice for the diverse requirements of emerging applications. This has led to the development of various signature scheme variants, each tailored to address specific practical or structural needs. Among these, *multi-signature*, *proxy signature* and *ring signature* schemes, introduce new functionalities while preserving the core security properties of standard digital signatures. Multi-signature schemes are especially useful in scenarios where a group of signers must collectively authorize a single message. Instead of requiring each signer to produce an individual signature, a single multi-signature can serve as proof of agreement from all participants. This approach significantly reduces both the size of the signature and the cost of verification. They are primarily used in contexts such as blockchains for multi-signature wallets and federated learning [XZL21, DGNW20, BDN18], but can also be applied to contact tracing applications to serve as proof of contact of a group with a covid-19 patient. Proxy-signature schemes are used in contexts such as distributed systems [BPW03] or E-commerce [ZSL03], where operations must continue even in the absence of the designated signer, as they allow the delegation of signing rights to a proxy signer. Such a proxy signer is then able to sign in the stead of the original signer, within the allotted boundaries set by the authorization certificate, such as a limited message space and a limited amount of time. Although several variant signature schemes, such as a ring signature scheme, have been proposed in isogeny-based cryptography, the development of variant signature schemes remains underdeveloped. Notably, multi-signature schemes and proxy-signature schemes constructions are either missing entirely or exist only in early, non-practical forms.

1.1.1 Multi-Signature

Multi-signatures were originally proposed by Itakura and Nakamura in 1983 [IN83] to enable a group of signers to collectively produce a single signature on the same message, thereby lowering the computational cost associated with generating and verifying multiple individual signatures. It is primordial for a multi-signature to remain unforgeable even if corrupted signers collude, which we refer to as a rogue-key attack. Today, desirable properties that multi-signature schemes must achieve include the following:

- Security in the Plain Public Key (PPK) model: Introduced by Belare and Neven in 2006 [BN06], the PPK model permits each signer to make their public key publicly available without relying on a dedicated Key Generation protocol or a Trusted Third Party (TTP).
- Key Aggregation: The concept of Key Aggregation was first introduced by Maxwell et al. in MuSig [MPSW18]. It allows the signer to aggregate their public keys into an aggregated public key. The multi-signature can then be verified using the base signature's verification algorithm by replacing the signer's public key with the aggregated public key.
- Signature size and verification time independent of the number of signers.

Threshold signatures, like multi-signatures, belong to the family of multi-party signature schemes. They are specifically intended for situations where a signature can be generated collaboratively by any subset of signers meeting a predefined threshold. In a group of n participants, if at least k signers agree, a valid threshold signature can be produced. Full threshold threshold signatures (with t set to n) are especially similar to multi-signatures, although there exist several key differences between those two schemes:

- In a multi-signature scheme, each participant first generates an individual key pair using the **Key Generation** algorithm. The corresponding public keys are then combined into a single aggregate public key during the **Key Aggregation** phase. In contrast, threshold signature schemes adopt a different strategy: the secret key is divided into multiple shares and distributed among the signers. This distribution can be performed by a **Trusted Third Party (TTP)** or via non-interactive mechanisms such as *Replicated Secret Sharing* [CS20] or *Verifiable Secret Sharing (VSS)* [ABCP23b].
- In threshold signatures, the t signers participating in the signing process reconstruct the secret shares of the remaining $n - t$ signers, typically using *Lagrange Interpolation*. Consequently, the secret sharing process must be executed each time a subset of signers produces a threshold signature. In contrast, multi-signature schemes allow each signer to reuse the same key pair across multiple signing sessions.

Threshold signatures are particularly suitable for scenarios where only a subset of signers, meeting a predefined threshold, need to collaborate, allowing flexibility in the number of participants. Conversely, multi-signatures are better suited for situations where the participation of all designated signers is mandatory. This is particularly advantageous in contexts where a fixed set

of signers must produce multiple multi-signatures, since they can perform Key Aggregation a single time and reuse the resulting aggregated public key. This approach substantially lowers the computational cost compared to using threshold signatures. While no isogeny-based multi-signatures have been proposed, existing threshold signatures include the threshold signature by De Feo and Meyer [FM19], Sashimi by Cozzo and Smart [CS20] and ThreshER SharK by Atapoor et al. [ABCP23b]. However, the differences between the two schemes make constructing a multi-signature scheme from a threshold signature scheme a non-trivial issue. In the context of multi-party signatures in Isogeny-based Cryptography, which requires the use of commitments, to prevent signers from changing chosen isogenies throughout the protocol, and zero-knowledge proof, which are protocols used by a prover to convince verifiers of the existence of a secret without revealing it, proposing a multi-signature unforgeable against rogue-key attacks is a significant challenge. Moreover, while multi-signatures have been proposed in Lattice-based Cryptography, and require only two-rounds [Che23] with the possibility of one of them being performed offline [BTT22], they generally require a very large key and signature size, making the possibility of an isogeny-based multi-signature an interesting alternative.

1.1.2 Proxy Signature

In contexts such as distributed systems [BPW03] or E-commerce [ZSL03], it is essential for operations to continue even if the designated signer is unavailable. Proxy signatures allow a signer, referred to as original signer, to delegate its signing right to a proxy signer to sign in its place. They were first introduced in 1996 by Mambo et al. [MUO96] and included properties such as verifiability and identifiability. However, Mambo’s construction did not prevent the proxy signature from being used for purposes other than its intended one (prevention of misuse) or prevent the proxy signer from repudiating its signature after its creation (strong undeniability). In 1997, Kim et al. [KPW97] introduced the notion of partial designation and threshold designation to proxy signatures. In 2002, Lee et al [LKK01] proposed a proxy signature scheme satisfying those two properties. In 2003, Boldyreva et al. [BPW03] formally defined the existentially unforgeable under chosen message attack and chosen warrant attack (EUF-CMA-CWA) security model, and proposed a generic construction which can be used to obtain a proxy signature scheme from a digital signature scheme.

Herranz and Saez [HS04] then introduced the notion of fully distributed proxy signatures, and Malkin [MOY04] made the link between key-insulated signatures and proxy signatures. In [SMP08], Schuldt et al. formally proposed the existentially unforgeable under chosen message attack and under proxy key exposure (EUF-CMA-PKE) security model for multi-level signature (i.e enabling delegation chains), which Tang et al. [TX14] adapted to single-level signature. In isogeny-based cryptography, a proxy signature, CSI-PS, was proposed by Dey et al. [DKSD24] but is only proven secure in the existentially unforgeable under chosen message attack (EUF-CMA). Additionally, CSI-PS is based on the SeaSign signature scheme, which suffers from large key and signature sizes. While an EUF-CMA-CWA proxy signature can be derived by using the generic construction, the proposed scheme does not adequately limit the rights of the proxy

signer, making it possible to misuse it. Proposing a more compact proxy signature scheme in both EUF-CMA-CWA and EUF-CMA-PKE models while preventing the possibility of misuse would improve the practicality of an isogeny-based proxy signature scheme.

1.1.3 Contact-Tracing

The Covid-19 pandemic highlighted the importance of monitoring viral transmission as an effective way to mitigate new infections. If a person is informed that they were in proximity to a confirmed Covid-19 case, they can self-isolate early and reduce the likelihood of spreading the virus further. Performing this process manually, however, proved extremely challenging. As a result, researchers proposed contact tracing applications to automate this task, and these solutions were adopted by governments in many regions. Such applications enable smartphones to exchange identifiers with nearby devices and log potential encounters. When a user later tests positive, the app leverages this stored data to warn recent contacts before additional transmissions occur. Their effectiveness is strongly tied to adoption rates, yet many individuals have been reluctant to install them due to concerns over privacy [Has20].

The term *close contact* refers to an individual who has remained in the vicinity of an infected person for a sufficient duration to create an infection risk. Contact tracing systems are generally divided into two categories: *decentralized protocols*, which broadcast information from the infected user to their contacts, and *centralized protocols*, which collect and process data about contacts on a central server. Vaudenay [Vau20b] analyzed both approaches, outlining their advantages, drawbacks, and potential improvements. Singapore released the first centralized Bluetooth-based solution, TraceTogether, while DP3T (Decentralized Privacy-Preserving Proximity Tracing) [TPH⁺20] and PACT (Private Automated Contact Tracing, both East and West variants) represented decentralized alternatives. Later, Vaudenay [Vau20a] introduced refinements to DP3T, strengthening it against relay and replay attacks.

Any practical protocol must reduce the risk of *false positives* (misidentifying either the infected user or their contacts) while ensuring traceability completeness, i.e., every relevant contact receives a notification. At the same time, the system must respect *patient privacy*. Another open issue is that most existing proposals model interactions strictly as pairwise encounters. For example, if Alice, Bob, and Charlie are all standing together for a prolonged period, Alice's phone would separately record Alice–Bob and Alice–Charlie, instead of capturing the group interaction [Tan20, Gvi20]. This representation does not accurately reflect multi-party meetings. Moreover, verifying each participant individually imposes computational and energy overhead on the device, leading to faster battery drain.

One possible remedy is to integrate multi-signature schemes (for instance, an isogeny-based multi-signature) or ring signatures into the design. With a multi-signature, a group of users can produce a single compact signature for their joint encounter, instead of multiple pairwise ones, thus reducing both storage and computation while capturing the group setting more faithfully. Alternatively, ring signatures provide a mechanism for sending alerts without revealing which group member is the infected patient. This leverages their *anonymity* property, ensuring that

the signer's identity remains hidden even within the group. Using an ID-based ring signature with the government as the trusted authority additionally ensures the impossibility of sending alerts without being recognized as a patient.

1.2 Contributions

The contributions of this study can be divided into three parts. In the first part, we propose the first isogeny-based multi-signature scheme. We build our multi-signature scheme based on the CSI-SharK [ABCP22] signature, using the Multi-Target Group Action Inverse Problem (MT-GAIP) and C_{k-1} -Vectorization Problem with Auxiliary Inputs (C_{k-1} -VPwAI) assumptions. To achieve active Existential Unforgeability under Chosen Message Attack (EUF-CMA), we adapt the Piecewise Verifiable Proof system (PVP) proposed in the Distributed Key Generation (DKG) protocol CSI-RASHI [BDPV21] and the threshold signature ThreshER SharK [ABCP23b] to the multi-signature setting to create a new zero-knowledge proof system. We then construct a sub-protocol allowing the signers to perform the required group action computations in a round-robin way (i.e collaboratively by performing the computation on the result of the previous signer's computations), calling the new proof system at each step. This sub-protocol comes in two versions, a structured and a non-structured which we use in the Key Aggregation and the computation of the multi-signature respectively. We prove the correctness and EUF-CMA of our scheme in the Random Oracle Model (ROM), meaning the Oracle is simulated as a random function, by using a new ad-hoc approach to prove the behavior of malicious signers is restricted during the group action computation together with the Double Forking Lemma [MPSW18] technique. Finally, we compare our scheme to ThreshER SharK and two state-of-the-art lattice-based signatures Dual-MS [Che23] and MuSig-L [BTT22]. We conclude that our scheme is preferable to ThreshER SharK in the case of repeated generations of multi-signatures by a fixed group of signers, as the possibility to skip Key Aggregation after the first multi-signature generation significantly reduces the computation amount, at the cost of losing the robustness (possibility of continuing the protocol in case of disqualifications) inherent to threshold signature schemes. When compared with Dual-MS and MuSig-L, we choose a set of parameters which maintains the balance between key generation and signature generation, resulting in significantly smaller signature size. Additionally, the signature size of our scheme is independent from the number of signers which is not the case in the lattice schemes. Those improvements come at the cost of an increased number of rounds and the impossibility to perform one online.

In the second part, we propose the first formal definitions of the properties of proxy signatures and identify an issue in the case where a certificate and a proxy-signature are not bound together. To prevent this issue, we propose an additional property which we name certificate-signature binding. We build the first isogeny-based proxy signature schemes in the EUF-CMA-CWA and EUF-CMA-PKE models. We choose to use the CSI-SharK signature scheme, using the MT-GAIP and C_{k-1} -VPwAI assumptions for both our schemes instead of SeaSign [FG19] as was done in CSI-PS due to its greater flexibility and ability to sample uniformly at random without using rejection sampling. We then prove the correctness and EUF-CMA-CWA, respectively

EUFCMA-PKE, of our schemes in the Quantum Random Oracle Model, meaning the oracle is simulated as a random function and the attacker can access the oracle in quantum superposition, by reducing the security of our scheme to the security of CSI-SharK. We conclude that our EUFCMA-CWA proposal achieves similar computation and memory cost as the proposal derived from the generic construction [BPW03] but extends it to satisfy the new property of certificate-signature binding, while being significantly faster and more secure than CSI-PS. Our EUFCMA-PKE proposal achieves unforgeability despite the more stringent security requirements at the cost of an increased computation time during key generation and certificate generation, making it comparable to CSI-PS's key generation but with faster proxy signing and verification.

In the third part, we propose a privacy-preserving multi-party contact tracing protocol that accurately represents the interaction of its users, such as the number of participants. We do so by taking a novel approach and shift perspective: instead of considering pairwise interactions between users as is done in other contact tracing apps, we consider the interactions globally. We base our protocol on the contact tracing protocol proposed by Liu et al. [LAY⁺20], which we refer to as primitive multi-party tracing when considered globally (i.e multi-party setting). Our protocol is divided into 4 phases, a registration phase, a meeting phase, an alerting phase and a tracing phase, and prevents the occurrence of false positives and satisfies traceability completeness. Moreover, a proof is generated during the meeting phase by all participants in the meeting, which differentiates it from multiple pairwise interactions. We refer to this property as Multi-Party Proof, which we achieve by incorporating a multi-signature [DGNW20, MM00] as a proof of the meeting. Moreover, we use an ID-based ring signature [CYH05] as a way to alert the close contacts, which prevents the close contacts from finding out which member of the group is the patient due to the anonymity property, and therefore achieves patient privacy. At the same time, the ring signature being ID-based guarantees that only a patient tested positive for Covid-19 is able to generate a valid ID-based ring signature with the authorization of the government. Finally, we compare our protocol to primitive multi-party tracing and conclude that our protocol reduces the communication cost by making it constant instead of linear in the number of people in the group, while reducing the memory and computation cost by using a multi-signature scheme.

1.3 Organization

The remainder of this study is organized as follows: Preliminaries are introduced in **Chapter 2**. Related Works related to multi-signatures, proxy signatures and contact tracing are introduced in **Chapter 3**. The study of an actively secure isogeny-based multi-signature based on CSI-SharK is presented in **Chapter 4**. The study of an isogeny-based proxy signature scheme in the EUFCMA-CWA and EUFCMA-PKE models, based on CSI-SharK is presented in **Chapter 5**. The study of an efficient multi-party contact tracing protocol is presented in **Chapter 6**. **Chapter 7** summarizes this dissertation and illustrates the future works. The overview of the thesis is given in Fig. 1.1.

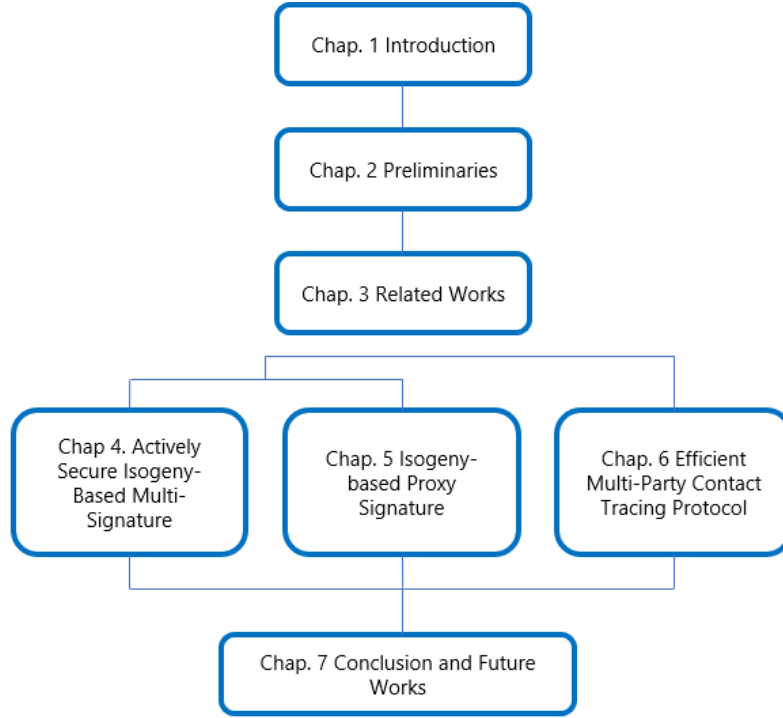


Figure 1.1: Overview of the dissertation

Chapter 4: Isogeny-based Multi-Signature

Publications of this contribution include the following local and international conference papers and a journal:

- Mathieu de Goyon and Atsuko Miyaji. Isogeny-based Multi-Signature Scheme. *Information Security Practice and Experience: 18th International Conference, (ISPEC 2023)*, pages 477-491, 2023 (Ref. [dGM23])
- Mathieu de Goyon and Atsuko Miyaji. Actively Secure Isogeny-based Multi-Signature Scheme. *Journal of Information Processing*, pages 594-607, 2025. (Ref. [dGM25]),
- Mathieu de Goyon and Atsuko Miyaji. Proposed Isogeny Multi-signature. *The 41st Symposium on Cryptography and Information Security (SCIS2024)* (Ref. [dGM24b], No Review)

Chapter 5: Isogeny-based Proxy Signature

Publications of this contribution include the following international conference:

- Mathieu de Goyon and Atsuko Miyaji Isogeny-based Proxy Signature. *Twelfth International Symposium on Computing and Networking Workshops (CANDARW2024)*, pages 252-258, 2024 (Ref. [dGM24a])

Chapter 6: Efficient Multi-Party Contact Tracing

Publications of this contribution include the following local and international conference papers and a journal:

- Mathieu de Goyon, Atsuko Miyaji and Tian Yangguang. Efficient Multi-Party Contact Tracing. *Ninth International Symposium on Computing and Networking (CANDAR2021)*, pages 10-18, 2021 (**Ref.** [dGMT21a])
- Mathieu de Goyon, Atsuko Miyaji and Tian Yangguang. Efficient Multi-Party Contact Tracing. *Journal of Information Processing*, pages 878-887, 2022. (**Ref.** [dGMT22])
- Mathieu de Goyon, Atsuko Miyaji and Tian Yangguang. Generalization of Contact Tracing. *186th Multimedia Communication and Distributed Processing/92nd Computer Security Joint Research Symposium, 2021* (**Ref.** [dGMT21b], **No Review**)

Chapter 2

Preliminaries

In this chapter, we introduce the preliminaries involved in this dissertation. In Section 2.1, we list the notation used in this thesis. In Section 2.2, we define the various signature schemes we use in this thesis. In Section 2.3, we introduce isogeny-based cryptography. Finally, in Section 2.4, we introduce the components we use in our contact tracing protocol.

2.1 Notations

Table 2.1: Notations used in this paper

Notation	Meaning
λ	Security parameter
1^λ	Unary encoding of λ , i.e. string of λ ones
$\mathbb{F}_p$	Finite field of order p
$\mathbb{Z}_N$	Ring of integers modulo N
$\mathbb{G}$	Group
$\mathbb{Z}_N[X]_t$	Ring of polynomials with coefficients in $\mathbb{Z}_N$, of maximal order t
O_E	Point at infinity of elliptic curve E , serving as identity for group law
$\text{End}(E)$	Endomorphism ring of elliptic curve E
$\mathcal{O}$	Order
$\text{Cl}(\mathcal{O})$	Ideal class group
$\mathfrak{a} \star E, [a]E$	Action of the ideal $\mathfrak{a}$ on the elliptic curve E , respectively of the element $a \in \mathbb{Z}_N$ on E
$\mathcal{E}$	Set of $\mathbb{F}_p$ isomorphism classes
$\mathcal{F}$	Forger
$\mathcal{A}$	Adversary
$\mathcal{M}$	Message space
$\mathcal{V}$	Validity period
L_{pk}	List of public keys
L_{ID}	List of identifiers
Rep	Expected number of restarts until a single party can proceed
n	Number of members in a group
$D_{\sigma_y}^{\ell+k}$	Gaussian distribution of dimension $\ell + k$ and standard deviation σ_y
$D_{\sqrt{\hat{\Sigma}}}^{\ell+k}(z)$	Gaussian distribution of dimension $\ell + k$, covariance matrix $\sqrt{\hat{\Sigma}}$ and center z

We list the notation of this thesis in Table 2.1. We assume all adversaries $\mathcal{A}$ in this paper are probabilistic polynomial time (PPT).

2.2 Signatures

In this section, we define the various signature schemes we use throughout this paper.

2.2.1 Sigma Protocols and Fiat-Shamir Transform

Identification protocols are three-step protocols between a Prover and a Verifier for a relation $R \subseteq X \times W$ where X is the statement space and W the witness space. The algorithms $P = (P_1, P_2)$ and $V = (V_1, V_2)$ are oracle-calling Probabilistic Polynomial-Time (PPT) algorithms, with V_2 being deterministic.

Prover is in possession of a valid pair $(x, w) \leftarrow \text{Gen}(1^\lambda)$ with Gen the generation algorithm, returning $x \in X$ and $w \in W$, and publishes x . The interactive protocol goes as follows:

- Prover generates a commitment $com \leftarrow P_1(x, w)$ which the prover sends to V .
- Verifier runs $ch \leftarrow V_1(1^\lambda)$ and sends the challenge ch to P .
- Prover will compute the response $rsp \leftarrow P_2(x, w, ch)$ and send rsp to V .

At the end, the verifier will run $b \leftarrow V_2(x, w, com, rsp)$, with $b = 1$ (accept) or 0 (reject). The protocol transcript (com, ch, rsp) is valid if $b = 1$. An identification protocol is a Sigma protocol if it satisfies the following properties:

Definition 2.1 ([Sta01]). ***Correctness:** An identification protocol Π_Σ is said to be correct if for all $(x, w) \in R$, and if both the prover and the verifier follow the protocol specifications, V always outputs 1.*

Definition 2.2 ([Sta01]). ***Special Soundness:** Π_Σ has special soundness if there exists a PPT algorithm Extract that, given a statement x and two valid transcripts $(com, ch, rsp), (com, ch', rsp')$ such that $ch \neq ch'$, Extract outputs a witness w such that $(x, w) \in R$.*

Definition 2.3 ([Sta01]). ***Honest Verifier Zero Knowledge (HVZK):** We say that Π_Σ is honest-verifier zero-knowledge (HVZK) if there exists a probabilistic polynomial-time (PPT) simulator Sim such that for all $x \in X$ and all w such that $(x, w) \in R$, the distribution of the output of $\text{Sim}(x, e)$ is (computationally or statistically) indistinguishable from a real protocol transcript (com, ch, rsp) produced in an honest interaction between P and V on input (x, w) , where ch is chosen uniformly at random.*

Definition 2.4 ([BDK⁺23]). ***High-min Entropy:** Π_Σ has the high min-entropy property if, for every statement $x \in X$ and witness w such that $(x, w) \in R$, the distribution of the prover's*

first message *com* (i.e., the commitment), generated by the honest prover on input (x, w) , has high min-entropy. The min-entropy of a random variable A is defined as:

$$H_\infty(A) = -\log \max_{com} \Pr[A = com]$$

We say that the protocol has high min-entropy if for all $(x, w) \in R$, we have:

$$H_\infty(A_{x,w}) \geq \lambda$$

for some security parameter λ . This ensures that the prover's commitment is hard to guess, and no particular value occurs with high probability.

A Sigma protocol satisfying all those properties can be turned into a non-interactive protocol (i.e a signature scheme) by replacing the challenge chosen by the Verifier into the output of a hash function on the message (and usually the commitment). The special soundness of the Sigma protocol represents the unforgeability of the resulting signature scheme. If C represents the challenge space, the soundness error is $\epsilon = 1/|C|$. This can be further increased by executing the protocol t_k times. For $C = \{0, 1\}$, we have $t_k = \lambda$.

2.2.2 Digital Signature

A signature scheme includes the following algorithms with the public parameter *params*:

- $(sk, pk) \leftarrow \text{KeyGen}(params)$. It is to generate a secret key sk and a public key pk .
- $\sigma \leftarrow \text{Sign}(sk, M)$. It is to generate a signature σ on a message M by secret key sk .
- $b \leftarrow \text{Verify}(pk, M, \sigma)$. It is to verify a signature σ on M by pk . If σ is generated from $\text{Sign}(sk, M)$, bit $b = 1$ otherwise $b = 0$.

The traditional security model for signature schemes is the existential unforgeability under chosen message attack (EUF-CMA) which is defined by the following game:

Game: Played by the Oracle (functioning as a black box that can be queried) and the forger $\mathcal{F}$.

- **KeyGen:** The Oracle runs $\text{KeyGen}_{\text{once}}$, generating the target key pair:

$$(sk^*, pk^*) \leftarrow \text{KeyGen}(params)$$

The Oracle then sends the public key pk^* to $\mathcal{F}$.

- **Signing Queries:** The forger $\mathcal{F}$ can query the oracle with any message M . The oracle will emulate the honest signer and return the resulting signature σ .
- **Forgery:** We consider that the forger $\mathcal{F}$ succeeds if it is able to produce a signature σ^* on a message M^* that has not been previously queried on the public key pk^* . Formally, $\mathcal{F}$

succeeds if the following condition holds:

$$\text{Verify}(pk^*, M^*, \sigma^*) = 1$$

Definition 2.5 ([KL20]). **EUFCMA**: An adversary $\mathcal{A}$ is said to be a (t, q_s, q_h, ϵ) -forger if $\mathcal{A}$ has at least advantage ϵ in the above game running in time at most t and makes at most q_s signing queries and q_h hash queries to the challenger. A signature scheme $(\text{KeyGen}, \text{KeyAgg}, \text{Sign}, \text{Verify})$ is said to be (t, q_s, q_h, ϵ) -EUFCMA secure in the Random Oracle Model if there is no $(\epsilon, t_k, q_d, q_s)$ -forger that wins in the above game except with negligible probability.

2.2.3 Multi-Signature

A multi-signature scheme includes the following algorithms with the public parameter $params$ and a qualified set of signers Q :

- $(sk, pk) \leftarrow \text{KeyGen}(params)$. It is to generate a secret key sk and a public key pk , run by each signer.
- $Agg \leftarrow \text{KeyAgg}(L_{pk})$. It is to compute the aggregated public key Agg of all signers in Q using the list of public keys L_{pk} .
- $\Sigma \leftarrow \text{Multi-Sign}(\{sk_i\}_{i \in Q}, L_{pk}, M)$. It is run by all signers together to sign a message M by using each signer's secret key sk_i with $i \in Q$ and the list of public keys L_{pk} .
- $b \leftarrow \text{Multi-Verify}(Agg, M, \Sigma)$. It is to verify a signature by the aggregated public key Agg . If Σ is generated from $\text{Multi-Sign}(sk, L_{pk}, M)$, $b = 1$ otherwise $b = 0$.

For Correctness, the scheme requires that for any $(sk, pk) \leftarrow \text{KeyGen}(params)$, any valid message M , any list of public keys L_{pk} , any $Agg \leftarrow \text{KeyAgg}(L_{pk})$, any $\Sigma \leftarrow \text{Multi-Sign}(sk, L_{pk}, M)$, $\text{Multi-Verify}(Agg, M, \Sigma)$ outputs 1 with overwhelming probability. Multi-signatures need to satisfy several security features as introduced in [MPSW18], such as security in the plain public key model and key aggregation which are defined as follows:

Definition 2.6 ([BN06]). **Security in the plain public key model**: In the plain public key model, each signer is identified simply by their public key, which is published without any additional certification or special setup (like a trusted setup or a proof of possession)

Definition 2.7 ([BN06]). **Key Aggregation**: Key aggregation is the process of combining multiple users' public keys into a single aggregated public key, allowing the verification algorithm to be identical to the base scheme using the aggregated public key instead.

We consider the case of a rogue-key attack where the forger can control $n - 1$ of the n signers, i.e., a single honest signer. Like the EUFCMA Model in the digital signature scheme, we define the security notion for multi-signature schemes as EUFCMA specified by the following game:

Game: Played by the Oracle and the forger $\mathcal{F}$.

- **KeyGen:** The Oracle runs **KeyGen**_{once}, generating the honest signer key pair:

$$(sk^*, pk^*) \leftarrow \text{KeyGen}(params)$$

The Oracle then sends the public key pk^* to $\mathcal{F}$.

- **Signing Queries:** The forger $\mathcal{F}$ can query the oracle with any message M and any collection of signer public keys $L_{pk} = \{pk_1, \dots, pk_n\}$, where $pk_i = pk^*$ for some i . The oracle will emulate the honest signer and return the resulting signature σ_i .
- **Forgery:** We consider that the forger $\mathcal{F}$ succeeds if it is able to produce a multi-signature Σ^* on a message M^* that has not been previously queried, using a set of public keys L_{pk} that contains pk^* . Formally, $\mathcal{F}$ succeeds if the following condition holds:

$$\text{Multi-Verify}(\text{KeyAgg}(L_{pk}), M^*, \Sigma^*) = 1$$

Definition 2.8 ([BN06]). **EUFCMA:** An adversary $\mathcal{A}$ is said to be a $(t, q_s, q_h, n, \epsilon)$ -forger if $\mathcal{A}$ has at least advantage ϵ in the above game with n signers running in time at most t and makes at most q_s signing queries and q_h hash queries to the challenger. A multi-signature scheme $(\text{KeyGen}, \text{KeyAgg}, \text{Multi-Sign}, \text{Multi-Verify})$ is said to be $(t, q_s, q_h, n, \epsilon)$ -EUFCMA secure in the Random Oracle Model if there is no $(\epsilon, t_k, q_d, q_s)$ -forger that wins in the above game except with negligible probability.

We additionally define the concept of the *Generalized Forking Lemma* and present the forking algorithm in Algorithm 1, which is employed to establish the unforgeability of multi-signatures.

Lemma 2.1 ([BN06]). **Generalized Forking Lemma-** Fix integers q and l . Let $\mathcal{A}$ be a randomized algorithm that takes as input some main input inp and l -bit strings $\{h_1, \dots, h_q\}$ and returns either $\perp$ or a tuple (i, out, out') , where $i \in \{1, \dots, q\}$ and out, out' are some side outputs. The accepting probability of $\mathcal{A}$, denoted $acc(\mathcal{A})$, is defined as the probability, over the random draw of inp (according to some well-understood distribution), $\{h_1, \dots, h_q\} \leftarrow \{0, 1\}^l$ and the random coins of $\mathcal{A}$, that $\mathcal{A}$ returns a non- $\perp$ output. Consider Algorithm ForkA, taking as input inp , described in Algorithm 1. Let frk be the probability (over the draw of inp and the random coins of ForkA) that ForkA returns a non- $\perp$ output. Then

$$frk \geq acc(\mathcal{A}) \left(\frac{acc(\mathcal{A})}{q} - \frac{1}{2^l} \right) \quad (2.1)$$

2.2.4 Proxy Signature

A proxy signature scheme includes the following algorithms with the public parameter $params$, an original signer, and a proxy signer:

- $(sk, pk) \leftarrow \text{KeyGen}(params)$. It is to generate a secret key sk and a public key pk .

Algorithm 1 ForkA

Input inp
Output (i, out, out')

```

1: pick random coins  $\rho$  for  $\mathcal{A}$ 
2:  $h_1, \dots, h_q \leftarrow \{0, 1\}^l$ 
3:  $\alpha \leftarrow \mathcal{A}(inp, h_1, \dots, h_q, \rho)$ 
4: if  $\alpha = \perp$  then
5:   return  $\perp$ 
6: else
7:   parse  $\alpha$  as  $(i, out)$ 
8:    $h'_1, \dots, h'_q \leftarrow \{0, 1\}^l$ 
9:    $\alpha' \leftarrow \mathcal{A}(inp, h_1, \dots, h_q, h'_1, \dots, h'_q, \rho)$ 
10:  if  $\alpha' = \perp$  then
11:    return  $\perp$ 
12:  else
13:    parse  $\alpha'$  as  $(i', out')$ 
14:    if  $i = i'$  and  $h_i \neq h'_i$  then
15:      return  $(i, out, out')$ 
16:    else
17:      return  $\perp$ 
18:    end if
19:  end if
20: end if

```

- $\sigma \leftarrow \text{Sign}(sk, M)$. It is to generate a signature σ on a message M by secret key sk .
- $b \leftarrow \text{Verify}(pk, M, \sigma)$. It is to verify a signature σ by public key pk . If σ is generated from $\text{Sign}(M, sk)$, $b = 1$ otherwise $b = 0$.
- $\sigma^o \leftarrow \text{Delegate}(sk^o, W)$. It is to generate a certificate σ^o (usually a signature) on a warrant W by secret key of the original signer sk^o . The warrant may describe the identities of the original signer and the designated proxy signer, as well as a description of the message space and the validity period. (W, σ^o) is sent to proxy signer.

In EUF-CMA-PKE model, it is replaced by $(sk^{o'}, pk^{o'}, \sigma^o) \leftarrow \text{Delegate}(sk^o, W)$ which generates an intermediate key pair $(sk^{o'}, pk^{o'})$ sent to the proxy signer.

- $b \leftarrow \text{D-Verify}(pk^o, W, \sigma^o)$. It is to verify a certificate σ^o by the public key of the original signer pk^o and the warrant W . If σ^o is generated from $\text{Delegate}(W, sk^o)$, $b = 1$ otherwise $b = 0$.

In EUF-CMA-PKE model, it is replaced by $(sk^{p'}, pk^{p'}, \sigma_{prx}) \leftarrow \text{P-KeyGen}(sk^p, pk^o, W, \sigma^o)$ which is to generate a proxy signing key pair $(sk^{p'}, pk^{p'})$ using intermediate keys $(sk^{o'}, pk^{o'})$ and a signature σ_{prx} using the secret of the proxy signer sk^p on proxy signing public key $pk^{p'}$.

- $\sigma^p \leftarrow \text{P-Sign}(sk^p, W, M, \sigma^o)$. It is to generate a proxy signature σ^p on a message M by secret key of the proxy signer sk^p with warrant W and certificate σ^o .

In EUF-CMA-PKE model, it is replaced by $\sigma^p \leftarrow \text{P-Sign}(sk^{p'}, W, M, \sigma^o, \sigma_{prx})$ which generates σ^p by proxy signing secret key $sk^{p'}$ on message M and signature σ_{prx} instead.

- $b \leftarrow \text{P-Verify}(pk^o, pk^p, W, \sigma^o, M, \sigma^p)$. It is to verify a certificate σ^o by the public key of the original signer pk^o and the warrant W , σ_{prx} by and proxy signature σ^p by the public key of the proxy signer pk^p . If σ^o is generated from $\text{Delegate}(W, sk^o)$, and σ_{prx} is generated from $\text{P-Sign}(sk^p, W, M, \sigma^o)$, $b = 1$ otherwise $b = 0$.

In EUF-CMA-PKE model, it is replaced by $\text{P-Verify}(pk^o, pk^p, pk^{p'}, W, \sigma^o, \sigma_{prx}, M, \sigma^p)$. If σ^o is generated from $\text{Delegate}(W, sk^o)$, σ_{prx} is generated from $\text{P-KeyGen}(sk^p, pk^o, W, \sigma^o)$ and σ^p is generated from $\text{P-Sign}(sk^{p'}, W, M, \sigma^o, \sigma_{prx})$, $b = 1$ otherwise $b = 0$.

Proxy signatures need to satisfy several security features as introduced in [MUO96, LKK01], such as verifiability, strong identifiability, strong undeniability, and prevention of misuse, which are as follows:

- **Verifiability:** A verifier can be convinced based on the proxy signature of the original signer's agreement on the signed message.
- **Strong Identifiability:** Anyone can determine the identity of the proxy signer from the proxy signature.
- **Strong Undeniability:** After a proxy signer has generated a proxy signature for the original signer, the proxy signer cannot repudiate his signature creation against anyone.
- **Prevention of misuse:** The proxy signature cannot be used for something other than its original purpose.

We first start by introducing the EUF-CMA-CWA security model as defined in [WBZ⁺22] and originally proposed by Boldyreva et al. [BPW03]. We consider the case where the forger can collude with either the original signer or the proxy but not both of them. We say the former is a type I forger, denoted $\mathcal{F}_I$, while the latter is a type II forger denoted $\mathcal{F}_{II}$.

Game 1: Played by the challenger $\mathcal{C}$ and the forger $\mathcal{F}_I$.

- *Setup:* Challenger $\mathcal{C}$ runs KeyGen twice, generating both the original signer's and the proxy signer's key pair: $(sk^o, pk^o), (sk^p, pk^p) \leftarrow \text{KeyGen}(params)$. $\mathcal{C}$ then sends both public keys (pk^o, pk^p) and the original signer secret key sk^o to $\mathcal{F}_I$.
- *Signing Queries:* $\mathcal{F}_I$ is allowed to ask $\mathcal{C}$ the proxy signer's standard signature on any fresh message M . $\mathcal{C}$ then runs $\sigma^p \leftarrow \text{Sign}(sk^p, M)$ and sends σ^p to the forger $\mathcal{F}_I$.
- *Proxy Signing Queries:* The forger $\mathcal{F}_I$ can ask the proxy signer's proxy signature on behalf of the original signer with any valid warrant and certificate pair (W, σ^o) with any fresh message M conforming to the warrant W . Challenger $\mathcal{C}$ then runs $\sigma^p \leftarrow \text{P-Sign}(sk^p, \sigma^o, W, M)$ and sends σ^p to the forger $\mathcal{F}_I$.

- *Forgery*: We say the forger $\mathcal{F}_I$ wins if it can output a warrant-certificate and message-signature pair (W^*, σ^{*o}) and (M^*, σ^{*p}) with

$$\text{P-Verify}(pk^o, pk^p, W^*, \sigma^{*o}, M^*, \sigma^{*p}) = 1$$

while a query has never been made with the tuple (W^*, σ^{*o}, M^*) (i.e a proxy signing query on M^* with W, σ^o or a signing query directly on $M^{*'} = W || M || \sigma^o$).

Game 2: Played by the challenger $\mathcal{C}$ and the forger $\mathcal{F}_{II}$.

- *Setup*: Challenger $\mathcal{C}$ runs **KeyGen** twice, generating both the original signer's and the proxy signer's key pair: $(sk^o, pk^o), (sk^p, pk^p) \leftarrow \text{KeyGen}(params)$. $\mathcal{C}$ then sends both public keys (pk^o, pk^p) and the proxy signer's secret key sk^p to $\mathcal{F}_{II}$.
- *Delegation Generating Queries*: $\mathcal{F}_{II}$ is allowed to ask $\mathcal{C}$ the original signer's delegation on any warrant W . $\mathcal{C}$ then runs $\sigma^o \leftarrow \text{Delegate}(sk^o, W)$ and sends σ^o to the forger $\mathcal{F}_{II}$.
- *Forgery*: We say the forger $\mathcal{F}_{II}$ wins if it can output a warrant-certificate and message-signature pair (W^*, σ^{*o}) and (M^*, σ^{*p}) with

$$\text{P-Verify}(pk^o, pk^p, W^*, \sigma^{*o}, M^*, \sigma^{*p}) = 1$$

while the forger has never made the delegation generating query on the warrant W^* . Let $\mathcal{A}$ be an adversary controlling the forgers in the above two games, and let $\text{Adv}_{\mathcal{A}, \text{CWA}} = \max_{i \in \{1, 2\}} (\Pr[\text{Game}_i])$ be the advantage of $\mathcal{A}$ in the above security games.

Definition 2.9 ([BPW03]). **EUFCMA-CWA**: An adversary $\mathcal{A}$ is said to be a $(\epsilon, t_k, q_d, q_s)$ -forger if $\mathcal{A}$ has at least advantage ϵ in the above games running in time at most t_k and makes at most q_d, q_s delegation and signing queries to the challenger. A proxy signature scheme (**KeyGen**, **Sign**, **Verify**, **Delegate**, **D-Verify**, **P-Sign**, **P-Verify**) is said to be $(\epsilon, t_k, q_d, q_s)$ -EUFCMA-CWA-secure in the Quantum Random Oracle Model if there is no $(\epsilon, t_k, q_d, q_s)$ -forger.

The EUFCMA-PKE model for single-level proxy signature by Tang and Xu [TX14] is considered as a generalization of the EUFCMA-CWA Model. It is a stronger model than EUFCMA-CWA in that it also considers the risk of proxy key exposure. Once again, the game is played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, who first generates (pk^*, sk^*) for an honest signer u^* . $\mathcal{A}$ will control all other key pairs, and it will not be required for $\mathcal{A}$ to prove knowledge of the secret keys corresponding to public keys used in the game. $\mathcal{C}$ will maintain a list of proxy sign keys generated $pskList(w)$. $\mathcal{A}$ has access to the following queries:

- *Ordinary Signing Query*: On input from $\mathcal{A}$ with M , $\mathcal{C}$ returns a signature on M .
- *Delegation to $\mathcal{C}$ Query*: $\mathcal{A}$ can send to $\mathcal{C}$ an intermediate key pair $(sk^{o'}, pk^{o'})$ with a certificate σ^o . $\mathcal{C}$ will then interact with $\mathcal{A}$, run **P-KeyGen** to compute the proxy signing key pair $sk^{p'}, pk^{p'}$ by using the intermediate keys provided by $\mathcal{A}$ and adds $sk^{p'}$ to $pskList$, as well as σ_{prx} .

- *Delegation from $\mathcal{C}$ Query.*
 1. *Delegation of sk^* Query:* On input W from $\mathcal{A}$, $\mathcal{C}$ interacts with $\mathcal{A}$ by running **Delegate**, generating a certificate σ^o on the warrant W using the chosen target public key.
 2. *Self-Delegation Query:* $\mathcal{C}$ interacts with itself through the protocol of **Delegate** and **P-KeyGen**, by generating a warrant and certificate W, σ^o , and an intermediate key pair $(sk^{o'}, pk^{o'})$ and proxy signing key pair $sk^{p'}, pk^{p'}$ together with σ_{prox} . He then adds $sk^{p'}$ to $pskList$, as well as σ_{prox} . $\mathcal{C}$ then sends the transcript to $\mathcal{A}$.
- *Proxy Signature Query:* On input W, M , $\mathcal{C}$ looks up the proxy sign key $pk^{p'}$ and generates the proxy signature $\sigma^p \leftarrow \text{P-Sign}(sk^{p'}, W, M)$. If no such key exists, return $\perp$.
- *Proxy Key Exposure Query:* On input w , $\mathcal{C}$ returns the corresponding proxy sign key or $\perp$ if no such key exists.

At the end, the forger is able to attempt a forgery under several forms:

- ordinary signature from $\mathcal{C}$: The adversary outputs (M, σ) . It wins if $\text{Verify}(pk^*, M, \sigma) = 1$ and M has not been submitted in an ordinary signing query.
- proxy signature of $\mathcal{C}$: The adversary output (W, M, σ^p) where the forgery of the delegator is pk^o . It is valid if **P-Verify** succeeds while W, M has not been submitted in a proxy signature query and w has not been submitted in a proxy key exposure query.
- proxy signature on behalf of $\mathcal{C}$: The adversary outputs (W, M, σ^p) where $W = (pk^*, pk^p, \mathcal{M}, \mathcal{V})$. The forgery is valid if **P-Verify** is successful and the challenger did not generate the proxy signature and W has not been submitted in a proxy key exposure query.

Let $\mathcal{A}$ be the adversary in the above game, and $\text{Adv}_{\mathcal{A}, \text{PKE}} = (\Pr[\text{Game}])$ be the advantage of $\mathcal{A}$ in the above security game.

Definition 2.10 ([TX14]). **EUFCMA-PKE:** An adversary $\mathcal{A}$ is said to be a $(\epsilon, t_k, q_d, q_s)$ -forger if $\mathcal{A}$ has at least advantage ϵ in the above game running in time at most t_k and makes at most q_d, q_s delegation and signing queries to the challenger. A proxy signature scheme (**KeyGen**, **Sign**, **Verify**, **Delegate**, **P-KeyGen**, **P-Sign**, **P-Verify**) is said to be $(\epsilon, t_k, q_d, q_s)$ -EUFCMA-PKE-secure in the Quantum Random Oracle Model if there is no $(\epsilon, t_k, q_d, q_s)$ -forger.

2.2.5 Threshold Signature

A threshold signature scheme includes the following algorithms with the public parameter $params$ and a set of members Q :

- $(sk, pk) \leftarrow \text{KeyGen}(params)$. It is to generate a public key pk and a shared secret sk . The shared secret is divided into secret shares $(sk^{(i)})_{i \in Q}$ and distributed among the signers.

- $\sigma \leftarrow \text{Sign}(\{sk_i\}_{i \in Q_t}, M)$. It is run by a qualified set of signers $Q_t \subseteq Q$ (i.e, a greater number of signers than the required threshold) to generate a threshold signature σ on a given message M .
- $b \leftarrow \text{Verify}(pk, M, \sigma)$. It is to verify a signature σ by the public key pk . If σ is generated from $\text{Sign}(\{sk_i\}_{i \in Q_t}, M)$, $b := 1$ otherwise $b := 0$.

Unforgeability requires that for $(\{sk_i\}_{i \in Q}, pk) \leftarrow \text{KeyGen}(params)$, a threshold signature is unforgeable if given the public key pk , an unqualified set of signers cannot forge a valid pair (M^*, σ^*) i.e. $\text{Verify}(pk, M^*, \sigma^*) = 1$. In addition, it is required that a valid output signature is indistinguishable from the signature produced by the signing algorithm of the underlying non-thresholdized scheme with the same public key.

2.2.6 Ring Signature

A ring signature scheme includes the following algorithms with the public parameter $params$, a group (ring) of members Q of size n :

- $(sk, pk) \leftarrow \text{KeyGen}(params)$. It is to generate a secret key sk and a public key pk .
- $\sigma \leftarrow \text{Sign}(sk, L_{pk}, M)$. It is run by a signer to sign a message by using the secret key sk and the list of public keys L_{pk} for all signers in Q .
- $b \leftarrow \text{Verify}(L_{pk}, M, \sigma)$. It is to verify a signature by the public keys L_{pk} . If σ is generated from $\text{Sign}(sk, L_{pk}, M)$, $b := 1$ otherwise $b := 0$.

Unforgeability requires that for $(sk, pk) \leftarrow \text{KeyGen}(params)$, $L_{pk} = \{pk^{(j)} | j \in Q\}$ a ring signature is secure if given the list of public keys L_{pk} , the attacker cannot forge a valid pair (M^*, σ^*) i.e. $\text{Verify}(L_{pk}, M^*, \sigma^*) = 1$.

A ring signature guarantees the anonymity of the signer.

Definition 2.11 ([SW07]). **Anonymity-** *Given a valid ring signature and the public keys of all group members, no probabilistic polynomial-time adversary can determine which member of the group created the signature with probability non-negligibly greater than $1/n$, where n is the number of members in the group.*

An ID-based ring signature additionally requires a trusted authority (GV) and for all signers to be in possession of an identifier ID. Instead of generating a key pair independently, the signer will receive their key pair from GV with the public key extracted from ID, and their secret key computed by GV.

- **Setup** GV generates master key pair (sk_G, pk_G) and publishes $params$ together with pk_G .
- $(sk, pk) \leftarrow \text{KeyGen}(params, sk_G, ID)$. It is to receive a public key pk , extracted from ID, and a secret key sk , generated using sk_G , in collaboration with GV.

- $\sigma \leftarrow \text{Sign}(sk, L_{ID}, M)$. It is run by a signer to sign a message by using the secret key sk and the list of identifiers L_{ID} for all signers in Q .
- $b \leftarrow \text{Verify}(L_{ID}, M, \sigma)$. It is to verify a signature by the public keys L_{ID} . If σ is generated from $\text{Sign}(sk, L_{pk}, M)$, $b := 1$ otherwise $b := 0$.

2.3 Isogeny-based Cryptography

In this section, we briefly introduce isogeny-based cryptography, CSIDH, and signature schemes relevant to our multi-signature and proxy signature schemes.

2.3.1 Isogenies

A detailed overview of isogeny-based cryptography can be found in the lecture notes by De Feo [Feo17].

Let E be an elliptic curve defined over a finite field $\mathbb{F}_p$, where p is a large prime. The curve E is said to be *supersingular* if and only if $\#E(\mathbb{F}_p) = p + 1$; otherwise, it is referred to as *ordinary*.

Consider two elliptic curves E and E' defined over $\mathbb{F}_p$. An *isogeny* $\phi : E \rightarrow E'$ is a morphism between elliptic curves satisfying $\phi(O_E) = O_{E'}$. The *degree* of ϕ , denoted $\deg(\phi)$, corresponds to the degree of the associated rational map. If ϕ is separable, $\deg(\phi)$ also equals the size of its kernel as a group homomorphism. An isogeny is said to be *cyclic* when its kernel forms a cyclic subgroup of E . An *endomorphism* is an isogeny from a curve to itself. For a given curve E , the set of its endomorphisms forms a ring, denoted $\text{End}(E)$ and called the *endomorphism ring*. We denote by $\text{End}(E)_{\mathbb{F}_p}$ the subring of endomorphisms defined over the base field $\mathbb{F}_p$. In the supersingular case, the inclusion $\text{End}(E)_{\mathbb{F}_p} \subsetneq \text{End}(E)$ is strict: $\text{End}(E)$ is an order in a quaternion algebra, while $\text{End}(E)_{\mathbb{F}_p}$ is an order in the imaginary quadratic field $\mathbb{Q}(\sqrt{-p})$.

The *ideal class group* of $\mathcal{O} = \text{End}(E)$ is defined as the quotient of the set of fractional invertible ideals in $\mathcal{O}$ by the set of principal fractional invertible ideals, and is denoted $\text{Cl}(\mathcal{O})$. Given an ideal $\mathfrak{a} \subset \mathcal{O}$, the *$\mathfrak{a}$ -torsion group* of E is defined as $E[\mathfrak{a}] = \{P \in E \mid \alpha(P) = 0 \forall \alpha \in \mathfrak{a}\}$. This torsion subgroup allows the construction of the isogeny $\phi_{\mathfrak{a}} : E \rightarrow E_{\mathfrak{a}}$ with $E_{\mathfrak{a}} = E/E[\mathfrak{a}]$, which we denote by $\mathfrak{a} \star E$. This isogeny is uniquely defined up to $\mathbb{F}_{p^2}$ -isomorphism, and the group $\text{Cl}(\mathcal{O})$ acts on the set $\mathcal{E}$ of $\mathbb{F}_{p^2}$ -isomorphism classes of elliptic curves with endomorphism ring $\mathcal{O}$ through the operation $\star$. Moreover, $\text{Cl}(\mathcal{O})$ acts *freely and transitively* on $\mathcal{E}$, so that $\mathcal{E}$ forms a *principal homogeneous space* for $\text{Cl}(\mathcal{O})$. The correspondence between ideals and isogenies is formalized by the *Deuring Correspondence*, which associates left ideals in a fixed maximal order $\mathcal{O}$ of the quaternion algebra $B_{p,\infty}$ with isogenies between supersingular elliptic curves over $\mathbb{F}_{p^2}$ whose full endomorphism ring is $\mathcal{O}$.

2.3.2 SQISign

SQI-Sign and its variants such as SQISignHD [DLRW24], SQISign2D–West [BDF⁺24] and SQISign2D–East [NOC⁺24] are currently the most compact and fastest signatures in isogeny-based cryptography. They rely on the Deuring Correspondence between isogeny and ideal to

obtain an identification protocol, which is then turned into a signature by using a Fiat-Shamir Transform. The identification protocol is given in Figure 2.1. The core of SQISign and its variants lies in how to compute the isogeny σ corresponding to the response during the identification protocol without revealing information about τ , the secret key, and ψ , the commitment, as the protocol needs to be zero-knowledge. The signature scheme is then given in Algorithms 2 to 4.

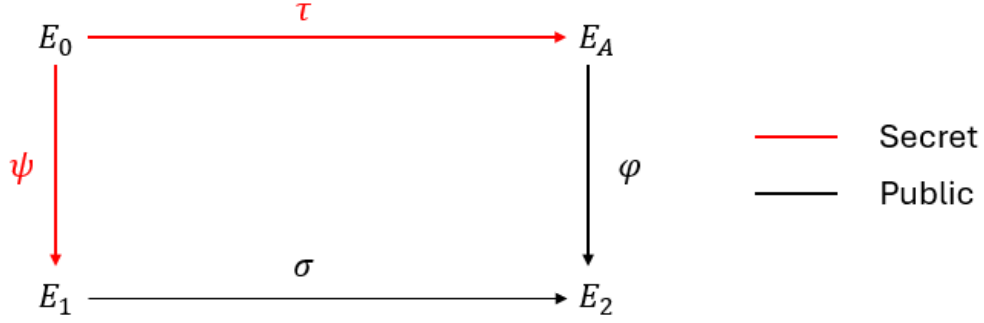


Figure 2.1: SQISign identification protocol

Algorithm 2 SQISign KeyGen

Input $params$

Output sk, pk

- 1: Sample $\mathfrak{a} \in \text{Cl}(\mathcal{O})$
 - 2: Compute $\phi_{\mathfrak{a}} : E_0 \rightarrow E_A := \mathfrak{a} \star E_0$
 - 3: Return $sk = \mathfrak{a}, pk = E_A$
-

Algorithm 3 SQISign Sign

Input sk, M

Output Σ

- 1: Sample $\psi : E_0 \rightarrow E_1$.
 - 2: Compute $s = H(j(E_1), M)$
 - 3: Compute $\Phi_{D_c}(E_1, s) = \varphi : E_1 \rightarrow E_2$
 - 4: From $\mathcal{O}_A$ and $\varphi \circ \psi : E_0 \rightarrow E_2$, construct $\sigma : E_A \rightarrow E_2$ of degree D such that $\hat{\varphi} \circ \sigma$ cyclic.
 - 5: Return $\Sigma = (E_1, \sigma)$
-

Algorithm 4 SQISign Verify**Input** pk, Σ, M **Output** b

- 1: Parse Σ as (E_1, σ)
- 2: From $s = H(j(E_1), M)$, reconstruct $\Phi_{D_c}(E_1, s) = \varphi : E_1 \rightarrow E_2$.
- 3: **if** σ from E_A to E_2 and $\hat{\varphi} \circ \sigma$ cyclic **then**
- 4: Return $b = 1$
- 5: **else**
- 6: Return $b = 0$
- 7: **end if**

Definition 2.12 ([AAA⁺25]). **Hint distribution** Let $R \subset X \times W$ be a relation. A hint distribution H for R is a collection of distributions $H = \{H_x\}_{x \in X}$, where $H_x : \text{HintSet}_x \rightarrow \{0, 1\}$ and the elements (i.e., the hints) of HintSet_x are efficiently representable in $|x|$. The distribution H_x needs not be efficiently sampleable.

Definition 2.13 ([AAA⁺25]). **q -unif-hint-EndRing $_p$** Given a supersingular curve E sampled uniformly at random and q hints $(h_1, \dots, h_q) \leftarrow H_E^{\text{unif}}$ find four endomorphisms in efficient representation that form a basis of $\text{End}(E)$.

SQISign is proven secure in the Random Oracle Model (and conjectured in QROM) under the q -unif-hint-EndRing $_p$ where hints are additional data the simulator has access to. The original SQISign protocol used the KLPT [KLPT14] algorithm to find a random equivalent ideal of smooth norm, in order to prevent any leaks before converting this ideal to the corresponding isogeny, which is costly. Using an ideal of smooth norm was necessary as only such ideals, and correspondingly isogeny of smooth degree could be efficiently represented without revealing the endomorphism ring. After SIDH [FJP14] was broken in 2022, the technique used to break its security which relies on Kani’s Lemma was put to constructive use in order to propose new protocols. The attack proposed in [Rob22] relies on the fact that any isogeny can be efficiently recovered from its evaluation on a sufficiently large torsion subgroup. By using this technique, it is possible to embed information about an isogeny into an isogeny of higher dimension without having to rely on smooth norms. Such a technique led to the proposal of SQISignHD, with a less rigorous but faster variant which uses isogenies of dimension 4 and a more rigorous variant using isogenies of dimension 8. The faster version achieves comparable signature size to SQISign with faster signing but slower verification. More recently, three protocols which uses isogenies of dimension 2 were proposed, SQISign2D-East [NOC⁺24], SQISign2D-West [BDF⁺24] and SQIPrime [DF24]. SQISign2D-West which appears to have the best performance achieves faster signing than SQISign and comparable to the fast version SQISignHD but with faster verification and improved security.

2.3.3 CSIDH

CSIDH serves as the main key exchange protocol in isogeny-based cryptography. Introduced by Castryck et al. [CLM⁺18] in 2018, it leverages an efficient commutative group action $\star$ realized through supersingular elliptic curves that contain many small $\mathbb{F}_p$ -rational subgroups.

In the CSIDH setting, the endomorphism ring is restricted to the field $\mathbb{F}_p$, which is commutative. The group $\text{Cl}(\mathcal{O})$ then acts via $\star$ on the set $\mathcal{E}$ of $\mathbb{F}_p$ -isomorphism classes of elliptic curves with endomorphism ring $\mathcal{O}$. For supersingular curves, we have $\#E(\mathbb{F}_p) = p + 1$, and primes are chosen of the form $p = 4 \cdot l_1 \cdots l_n - 1$ so that $\#E(\mathbb{F}_p) = 4 \cdot l_1 \cdots l_n$, with each l_i a small distinct odd prime. In the field $\mathbb{Q}(\sqrt{-p})$, each l_i splits as $(l_i) = \langle l_i, \pi - 1 \rangle, \langle l_i, \pi + 1 \rangle$, where $\pi = \sqrt{-p}$ is the $\mathbb{F}_p$ -Frobenius endomorphism. We denote the first ideal factor $\langle l_i, \pi - 1 \rangle$ by ℓ_i , and CSIDH uses ideals of the form $\prod \ell_i^{e_i}$, with exponents e_i sampled uniformly from a fixed interval $[-B, B]$.

Let $E_0 : y^2 = x^3 + x$ over $\mathbb{F}_p$. Each isomorphism class of a *supersingular* curve with endomorphism ring $\mathcal{O} = \mathbb{Z}[\pi]$ can be represented by a coefficient A , defining the curve $E_A : y^2 = x^3 + Ax^2 + x$. We denote by $\mathcal{A}$ the set of all such coefficients. The ideal class group $\text{Cl}(\mathcal{O})$ acts freely and transitively on $\mathcal{A}$ through the action $\star : \text{Cl}(\mathcal{O}) \times \mathcal{A} \rightarrow \mathcal{A}$. From now on, we refer to a curve E_A by its isomorphism class via its coefficient A .

In this thesis, unless stated otherwise, $\mathcal{O}$ denotes the restriction of the endomorphism ring to $\mathbb{F}_p$, and $\mathcal{E}$ denotes the $\mathbb{F}_p$ -isomorphism classes of elliptic curves with endomorphism ring $\mathcal{O}$.

2.3.4 Hard-Homogeneous Spaces

Hard Homogeneous Spaces (HHS) were introduced by Couveignes in [Cou06] as a generalization of Diffie-Hellman schemes. A principal homogeneous space is a set $\mathcal{E}$ that is acted upon transitively by a group $\mathbb{G}$. It is defined by the following map

$$\mathbb{G} \times \mathcal{E} \rightarrow \mathcal{E}$$

$$g \star E = E'$$

satisfying the following properties:

- *Compatibility*: $g' \star (g \star E) = (g'g) \star E$ for any $g, g' \in \mathbb{G}$ and $E \in \mathcal{E}$;
- *Identity*: $e \star E = E$ if and only if $e \in \mathbb{G}$ is the identity element;
- *Transitivity*: For any $E, E' \in \mathcal{E}$ there exists a unique $g \in \mathbb{G}$ such that $g \star E = E'$.

Couveignes defines an HHS as a finite principal homogeneous space with additional algorithmic capabilities. In particular, the following tasks must be efficiently computable (i.e., in polynomial time):

- *Group operations*: decide whether a string g represents an element of $\mathbb{G}$, decide whether $g = g'$, compute g^{-1} and gg' ;
- *Sampling*: sample uniformly random elements from $\mathbb{G}$;

- *Membership*: decide whether a string E represents an element of $\mathbf{E}$, decide whether $E = E'$;
- *Action*: Given g and E , compute $g \star E$.

Conversely, the following problems are assumed to be hard (i.e., not efficiently solvable)

- *Vectorization*: Given $E, E' \in \mathcal{E}$, find $g \in \mathbb{G}$ such that $g \star E = E'$;
- *Parallelization*: Given $E, E', F \in \mathcal{E}$, such that $E' = g \star E$, find $F' = g \star F$.

Currently, CSI-FiSh and CSI-SharK are the only known realizations of HHS in isogeny-based cryptography, where $\mathcal{E}$ is the set of supersingular elliptic curves over $\mathbb{F}_p$ and $Cl(\mathcal{O})$ acts on $\mathcal{E}$.

2.3.5 SeaSign

In this section, we describe the signature scheme SeaSign [FG19] by De Feo and Galbraith, which serves as the basis for the proxy signature CSI-PS [DKSD24]. The concept behind SeaSign was originally proposed by Stolbunov in his thesis [Sto10] and Couveignes [Cou06], by using a Fiat-Shamir transform on an identification protocol. As in CSIDH, the public key consists of an elliptic curve $E_A = \mathbf{a} \star E$ with $\mathbf{a}$ the private key represented as $\mathbf{a} = \prod_{i=1}^n \ell_i^{e_i}$ with the exponent vector $\mathbf{e} = (e_1, \dots, e_n) \in [-B, B]^n$. The identification protocol goes as follows:

- The prover will sample t_k random ideals $\mathbf{b}_k = \prod_{i=1}^n \ell_i^{f_{k,i}}$ for $1 \leq k \leq t_k$, compute the curves $\mathcal{E}_k = \mathbf{b}_k \star E$ and send the j -invariants $(j(\mathcal{E}_k) : 1 \leq k \leq t_k)$ to the verifier as commitments.
- The verifier responds with t_k challenge bits $b_1, \dots, b_{t_k}$.
- For $1 \leq k \leq t_k$, if $b_k = 0$, the prover will respond with $\mathbf{f}_k = (f_{k,1}, \dots, f_{k,n})$ and the verifier checks that $j(\mathcal{E}_k) = j(\prod_{i=1}^n \ell_i^{f_{k,i}} \star E)$. If $b_k = 1$ the prover sends a representation of $\mathbf{b}_k \mathbf{a}^{-1}$ to the verifier. The verifier checks that $j(\mathcal{E}_k) = j(\mathbf{b}_k \mathbf{a}^{-1} \star E_A)$.

However, the original scheme assumes that the relation lattice is known, allowing elements of $Cl(\mathcal{O})$ to be sampled uniformly at random. Simply applying their scheme without knowing the relation lattice would lead to secret key leakage. To solve this issue, De Feo and Galbraith use rejection sampling as proposed by Lyubashevski [Lyu09] for lattice signatures. The exponents $(f_{k,1}, \dots, f_{k,n})$ are sampled uniformly in $[-(nt+1)B, (nt+1)B]$. If $b_k = 1$, the prover will instead compute the vector $\mathbf{z}_k = (z_{k,1}, \dots, z_{k,n})$ defined by $z_{k,i} = f_{k,i} - e_i$ and output $\mathbf{z}_k$ only if all its entries satisfy $|z_{k,i}| \leq ntB$ to prevent leakage. The complete protocol is given in Algorithms 5 to 7.

Definition 2.14 ([RS06, Sto10]). **Group action inverse problem (GAIP)**: Given two elliptic curves E and E_0 over the same finite field and with $\text{End}(E) = \text{End}(E_0) = \mathcal{O}$, find an ideal $\mathbf{a} \subset \mathcal{O}$ such that $E_0 = \mathbf{a} \star E$.

SeaSign is proven EUF-CMA in the Quantum Oracle Model under the GAIP Assumption, using the lossy key framework by Kiltz et al. [KLS18], which allows to construct signatures with tight reduction in QROM.

Algorithm 5 SeaSign KeyGen

Input $B, \ell_1, \dots, \ell_n, E$ **Output** sk, pk

- 1: sample $\mathbf{e} \leftarrow [-B, B]^n$
 - 2: Compute $E_A = (\prod_{i=1}^n \ell_i^{\mathbf{e}_i}) \star E$
 - 3: return $sk = e, pk = E_A$
-

Algorithm 6 SeaSign Sign

Input sk, M **Output** σ

- 1: **for** $k = 1, \dots, t_k$ **do**
 - 2: sample $\mathbf{f}_k \leftarrow [-(nt+1)B, (nt+1)B]^n$ and compute $\mathcal{E}_k = (\prod_{i=1}^n \ell_i^{\mathbf{f}_{k,i}}) \star E$
 - 3: **end for**
 - 4: Set $(c_1, \dots, c_t) = H(j(\mathcal{E}_1), \dots, j(\mathcal{E}_{t_k}) || M)$
 - 5: **for** $k = 1, \dots, t_k$ **do**
 - 6: **if** $c_k = 0$ **then**
 - 7: $\mathbf{z}_k = \mathbf{f}_k$
 - 8: **else**
 - 9: $\mathbf{z}_k = \mathbf{f}_k - \mathbf{e}$
 - 10: **end if**
 - 11: **if** $\mathbf{z}_k \notin [-(nt+1)B, (nt+1)B]$ **then**
 - 12: Return $\perp$
 - 13: **end if**
 - 14: **end for**
 - 15: return $\sigma = (\mathbf{z}_k, c_k)_{k=1}^{t_k}$
-

Algorithm 7 SeaSign Verify

Input pk, σ, M
Output b

```

1: Parse  $\sigma$  as  $(\mathbf{z}_k, b_k)_{k=1}^{t_k}$ 
2: for  $k = 1, \dots, t_k$  do
3:   if  $c_k = 0$  then
4:      $(\prod_{i=1}^n \ell_i^{z_{k,i}}) \star E$ 
5:   else
6:      $(\prod_{i=1}^n \ell_i^{z_{k,i}}) \star E_A$ 
7:   end if
8: end for
9: Set  $(c_1^*, \dots, c_t^*) = H(j(\mathcal{E}_1), \dots, j(\mathcal{E}_{t_k}) || M)$ 
10: if  $(c_1^*, \dots, c_{t_k}^*) = (c_1, \dots, c_{t_k})$  then
11:   return  $b = 1$ 
12: end if
13: return  $b = 0$ 
    
```

2.3.6 CSI-SharK

In this section, we briefly describe the isogeny-based signature CSI-SharK [ABCP22], which is the structured counterpart of CSI-FiSh [BKV19].

Definition 2.15 ([ABCP22]). **Exceptional Set:** An exceptional set modulo N is a set $\Xi_{k-1} = \{c_0, \dots, c_{k-1}\} \subseteq \mathbb{Z}_N$ where the pairwise difference $c_i - c_j$ of all elements $c_i \neq c_j$ is invertible modulo N .

Definition 2.16 ([ABCP22]). **Super-Exceptional Set:** A super-exceptional set modulo N is an exceptional set $\Xi_{k-1} = \{c_0, \dots, c_{k-1}\} \subseteq \mathbb{Z}_N$ where the pairwise sum $c_i + c_j$ of all elements $c_i \neq c_j$ is also invertible modulo N .

Instead of sampling $k - 1$ secret keys as in CSI-FiSh, the signer generates a single secret key and computes the corresponding public keys using a public exceptional set $\Xi_{k-1} = c_1 = 1, \dots, c_{k-1}$, as originally introduced in [BCP21]. Both CSI-FiSh and, consequently, CSI-SharK are derived by applying a Fiat-Shamir Transform to a Sigma protocol based on the CSIDH group action. They start from the same curve as CSIDH, $E_0 : y^2 = x^3 + x$ over $\mathbb{F}_p$, with $p = 4 \cdot l_1 \cdots l_n - 1 = 3 \pmod{4}$, where the l_i are small distinct primes; see [CLM⁺18] for details. In CSIDH, only the action of ideals of the form $\prod_{i=1}^n \ell_i^{e_i}$ can be efficiently computed. CSI-FiSh, however, allows for random sampling of class group elements by exploiting the structure of $\text{Cl}(\mathcal{O})$. Assuming $\text{Cl}(\mathcal{O})$ is cyclic of order N with generator $\mathbf{g}$, any $\mathbf{a} \in \text{Cl}(\mathcal{O})$ can be expressed as $\mathbf{a} = \mathbf{g}^a$. This makes it possible to define the group action $[] : \mathbb{Z}_N \times \mathcal{A} \rightarrow \mathcal{A}$, with $\mathcal{A}$ as defined previously. Throughout this thesis, we will write $[a]$ for $\mathbf{g}^a$ and $[a]E$ for $\mathbf{g}^a \star E$, where $E \in \mathcal{E}$ and $a \in \mathbb{Z}_N$. With this notation, we have $[a][b]E = [a + b]E$.

CSI-SharK incorporates several optimizations from [BKV19] to enhance the soundness of the Sigma protocol and, consequently, the security of the scheme, while reducing computational cost:

- During Key Generation, the signer computes the secret key and the corresponding $k - 1$ public keys, which form the challenge space.
- The challenge space is further extended using the quadratic twist. For an elliptic curve $E_A = [a]E_0$, its quadratic twist $E_A^{t_k}$ is isomorphic to $[-a]E_0$. This enlarges the public key set to $E_{-k+1}, \dots, E_{k-1}$, lowering the soundness error to $\frac{1}{2k-1}$.
- Since the challenge space consists of a finite number of curves, the Sigma protocol is repeated t_k times to reach the desired security level.

The parameters k , t_k , and the security parameter λ satisfy $t_k = \lambda \log_2(k)$. The hash function $H : 0, 1^* \rightarrow \mathcal{C}$ is modeled as a random oracle, with $\mathcal{C}$ representing the challenge space; for $i \in [1, t_k]$, $d_i \in [-k + 1, k - 1]$. The complete protocol is given in Algorithms 8, 9, and 10.

Algorithm 8 CSI-SharK KeyGen

Input E_0 , the order N , the exceptional set $\Xi_k = \{c_1 = 1, \dots, c_k\}$

Output sk, pk

- 1: sample $x \leftarrow \mathbb{Z}_N$
 - 2: **for** $i = 1, \dots, k - 1$ **do**
 - 3: Compute $Y_i = [c_i x]E_0$
 - 4: **end for**
 - 5: return $sk = x, pk = (Y_1, \dots, Y_{k-1})$
-

Algorithm 9 CSI-SharK Sign

Input sk, M

Output σ

- 1: **for** $i = 1, \dots, t_k$ **do**
 - 2: sample $b_i \leftarrow \mathbb{Z}_N$ and compute $E_i = [b_i]E_0$
 - 3: **end for**
 - 4: Set $(d_1, \dots, d_t) = H(E_1 \dots E_{t_k} || M)$
 - 5: **for** $i = 1, \dots, t_k$ **do**
 - 6: set $r_i = b_i - \text{sign}(d_i) \cdot c_{|d_i|} x \pmod{N}$
 - 7: **end for**
 - 8: return $\sigma = (r_i, d_i)_{i=1}^{t_k}$
-

Algorithm 10 CSI-SharK Verify**Input** pk, σ, M **Output** b

```

1: for  $i = 1, \dots, t_k$  do
2:   Compute  $E_i = [r_i]Y_{d_i}$ 
3: end for
4: Set  $(d_1^*, \dots, d_t^*) = H(E_1 \cdots E_{t_k} || M)$ 
5: if  $(d_1^*, \dots, d_t^*) = (d_1, \dots, d_t)$  then
6:   return  $b = 1$ 
7: end if
8: return  $b = 0$ 

```

CSI-FiSh can similarly be obtained by generating $k - 1$ secret keys during **KeyGen** and replacing $c_i x$ by x_i , the i -th secret key generated throughout the protocol. CSI-SharK relies on the hardness of inverting a group action, more specifically, random instances of a multi-target inversion problem, as well as a vectorization problem :

Definition 2.17 ([BKV19]). **Multi-Target Group Action Inverse Problem (MT-GAIP):** Given k curves E_i , with $\text{End}(E_1) = \dots = \text{End}(E_k) = \mathcal{O}$, find an ideal $\mathfrak{a} \subset \mathcal{O}$ such that $E_i = \mathfrak{a} \star E_j$ for some $i, j \in \{1, \dots, k\}$ with $i \neq j$.

Definition 2.18 ([ABCP22]). **$\mathcal{C}_{k-1}$ -Vectorization Problem with Auxiliary Inputs ($\mathcal{C}_{k-1}$ -VPwAI):** Given an element $E \in \mathcal{E}$ and the pairs $(c_i, [c_i x]E)_{i=1}^{k-1}$, where $\Xi_{k-1} = \{c_0 = 0, c_1 = 1, \dots, c_{k-1}\}$ is an exceptional set, find $x \in \mathbb{Z}_N$.

CSI-FiSh is sEUF-CMA in the QROM under the MT-GAIP assumption. Similarly, CSI-SharK is sEUF-CMA secure in the QROM under the MT-GAIP assumption and the $\mathcal{C}_{k-1}$ -VPwAI assumption. See [ABCP22, BKV19] for the full explanation.

2.3.7 Signature Comparison

Table 2.2: Comparison between isogeny-based signatures (size in kB)

	CSI-FiSh	CSI-SharK	SeaSign	SQISign
$ sk $	0.016	0.016	16	0.353
$ pk $	256	256	4194	0.065
$ sig $	0.306	0.306	0.978	0.148
Security Assumption	MT-GAIP	MT-GAIP, $\mathcal{C}_{k-1}$ -VPwAI	GAIP	q -unif-hint- EndRing_p
Oracle Model	QROM	QROM	QROM	QROM (conjectured)
Additional Properties	Commutativity	Commutativity, Structured	Commutativity, Reject Sampling	

We compare the isogeny-based signatures in Table 2.2 in terms of keys and signature size (in kB), security assumption and Oracle Model.. MT-GAIP reduces into GAIP. While no strict reduction is known, q -unif-hint- $EndRing_p$ is considered harder than GAIP Problem. All four signatures in Table 2.2 rely on some version of the Fiat-Shamir transform (lossy key setting for SeaSign and hint frameworks for SQISign). The security of Fiat Shamir transform scheme in the QROM is more subtle than in the ROM. CSI-FiSh and CSI-SharK directly deduce security in the QROM by arguing the scheme has unique responses, λ bits of entropy and perfect HVZK, therefore using the setting in [DFMS19]. SeaSign uses the lossy key setting by Kiltz, Lyubashevsky and Schaffner [KLS18]. While SQI-Sign uses the Fiat Shamir with hints framework to deduce the QROM security, there has been no formal proof to our knowledge. While SQI-Sign and its variants currently represent the fastest and most compact isogeny-based signature schemes, they are limited in flexibility due to the lack of commutativity in their underlying structure. This fundamental restriction makes it difficult to extend them to settings involving multiple participants, such as multi-signature schemes (involving a group of signers) or proxy signature schemes (involving original and delegated signers). In contrast, SeaSign offers greater structural flexibility but relies on rejection sampling, which significantly increases both its computational cost and signature size, rendering it less suitable for practical deployment in multi-party scenarios. As a result, we choose to use CSI-SharK as the basis for both our multi-signature and proxy signature schemes for its underlying structure.

2.4 Tracing Components

In this section, we briefly introduce the Elliptic Curve Discrete Logarithm Problem (ECDLP) and Bilinear Pairing, before presenting the two multi-signature schemes and the ring signature scheme that serve as components in our contact tracing protocol in Section 6.5.

2.4.1 ECDLP and Bilinear Pairing

We introduce the ECDLP, which is at the core of elliptic curve-based cryptography and is then extended into bilinear pairing-based cryptography.

Definition 2.19 ([KL20]). **Elliptic Curve Discrete Logarithm Problem (ECDLP)** *Given an elliptic curve E defined over a finite field $(\mathbb{F}_q)$, a base point $P \in E(\mathbb{F}_q)$ of large prime order n , and another point $Q \in E(\mathbb{F}_q)$, the problem is to find an integer $x \in [0, n - 1]$ such that $Q = xP$.*

Bilinear pairings are special maps defined over certain elliptic curve groups that are widely used in cryptography, particularly in multi-signature and identity-based schemes. Let $(\mathbb{G}_1, +)$ and $(\mathbb{G}_2, \cdot)$ be two cyclic groups of prime order p , and let G_T be another cyclic group of the same order. A bilinear pairing is a map

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$$

that satisfies the following properties:

Definition 2.20 ([DGNW20]). **Bilinearity:** For all $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$, we have

$$e(aP, bQ) = e(P, Q)^{ab}.$$

Definition 2.21 ([DGNW20]). **Non-degeneracy:** There exist $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$ such that $e(P, Q) \neq 1$, i.e., the map does not send all pairs to the identity in G_T .

Definition 2.22. **Efficient computability:** There exists an efficient algorithm to compute $e(P, Q)$ for any $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$.

Bilinear Pairing relies on the following assumptions:

Definition 2.23 ([DGNW20]). **Decisional Diffie-Hellman Problem (DDHP)** Given a generator g of a group $\mathbb{G}$ and a 3-tuple (g^a, g^b, g^c) , the Decisional Diffie-Hellman problem is to decide if $c = ab$.

Definition 2.24 ([DGNW20]). **Computational Diffie-Hellman Problem (CDHP)** Given a generator g of a group $\mathbb{G}$ and a 2-tuple (g^a, g^b) , the Computational Diffie-Hellman problem is to compute g^{ab} .

Definition 2.25 ([DGNW20]). **Gap Diffie-Hellman (GDH)** $\mathbb{G}$ is a Gap Diffie-Hellman group if no algorithm can solve CDHP with non-negligible advantage within polynomial time even when the DDH oracle is accessible by the algorithm.

Definition 2.26 ([DGNW20]). **Weak Bilinear Diffie-Hellman Inversion (wBDHI)** Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ be bilinear groups of prime order p , where $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ is a bilinear pairing. For some integer ℓ and randomly chosen exponents $\alpha, \gamma \in \mathbb{Z}_p^*$, let the adversary be given

$$g_1, g_1^\alpha, g_1^{\alpha^2}, \dots, g_1^{\alpha^\ell}, \quad g_2, g_2^\alpha, g_2^{\alpha^2}, \dots, g_2^{\alpha^\ell}, \quad g_1^\gamma, g_2^\gamma$$

where g_1 is a generator of $\mathbb{G}_1$ and g_2 a generator of $\mathbb{G}_2$. Then it is computationally infeasible for the adversary to compute $e(g_1, g_2)^{1/(\alpha+\gamma)}$.

In the rest of this section, let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be cyclic groups of prime order q , equipped with a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, and assume g_1, g_2 are generators of $\mathbb{G}_1, \mathbb{G}_2$.

2.4.2 DGNW Multi-Signature

We review the DGNW multi-signature [DGNW20]. DGNW is a pairing-based multi-signature that rely on the wBDHI problem. One of its advantages lies in the fact that each signer is able to generate an intermediate signature on a message, which are then aggregated into a multi-signature. We take advantage of this fact in our contact tracing protocol. Note that the forward security described in [DGNW20] is beyond the scope of this thesis. We consider

n signers, $H_G : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ a hash function modeled as a random oracle and let $params = (G_1, G_2, e, q, g_1, g_2, h)$ with $h \in \mathbb{G}_1$. The scheme is described below.

KeyGen: Each signer samples $x_i \leftarrow \mathbb{Z}_q^*$ its secret key and computes $y_i = g_2^{x_i}$ the corresponding public key. Return $(sk_i, pk_i) = (x_i, y_i)$.

KeyAgg: Compute $Agg = y_1 \cdots y_n$ the aggregated public key.

Sign: Sample $r \leftarrow \mathbb{Z}_q$, compute $\sigma_i = (\sigma'_i, \sigma''_i) = (h^{x_i} \cdot H_G(M)^r, g_2^r)$ the intermediate signature on M . Compute $\Sigma = (\Sigma' = (\Sigma'', \Sigma''') = (h^{x_1 + \cdots + x_N} \cdot H_G(M)^{r_1 + \cdots + r_N}, g_2^{r_1 + \cdots + r_N})$ the multi-signature.

Verify: Check $e(\Sigma', g_2) = e(h, Agg) \cdot e(H_G(M), \Sigma'')$.

2.4.3 MM Multi-Signature

We describe the MM multi-signature [MM00]. MM is a Discrete Logarithm Problem (DLP)-based multi-signature scheme that stands out compared to other multi-signature schemes by allowing the signers to progressively construct the message on which the multi-signature is generated, which [MM00] refers to as **message flexibility**. Note that MM does not support key aggregation. Let $M_{1,\dots,i}$ be the message modified by signer i , and M_i the corresponding modification (i.e. message chosen by signer i), i.e., $M_{1,\dots,i} = M_{1,\dots,i-1} || M_i$. We adapt the scheme to the ECDLP setting in this thesis. Let $params = (G_2, q, g_2)$, $H_G : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ a hash function modeled as a random oracle, and assume each signer is in possession of an identifier ID_i . The scheme is described below.

KeyGen: Each signer i samples $x_i \leftarrow \mathbb{Z}_q^*$ its secret key and computes $y_i = g_2^{x_i}$ the corresponding public key. Return $(sk_i, pk_i) = (x_i, y_i)$.

Sign: All signers will collaborate to compute the multi-signature Σ on $M = M_{1,\dots,n}$ as follows:

- Signer 1 samples $k_1 \leftarrow \mathbb{Z}_q^*$ and computes $R_1 = g_2^{k_1}$, $r_1 = R_1 + H_G(M_1 || ID_1)$, and $s_1 = (x_1 r_1 + 1)k_1^{-1}$ and sends the intermediate signature $\sigma_1 = (r_1, s_1)$ with M_1 to the following signer.
- For $i = 2, \dots, n$, Signer i adds M_i to $M_{1,\dots,i-1}$ to obtain $M_{1,\dots,i}$. He then computes $R_i = g_2^{k_i}$, $r_i = R_i + H_G(M_i || ID_i)r_{i-1}$, and $s_i = (x_i r_i + 1)k_i^{-1}$, and sends $\sigma_i = (r_i, s_i)$ with $M_{1,\dots,i}$ to signer $i + 1$.
- At the end, signer n outputs $\Sigma = \sigma_n = (r_n, s_n)$ on $M = M_{1,\dots,n}$.

Verify:

- For $j = n, n-1, \dots, 2$, compute $R'_j = g_2^{s_j^{-1}} y_j^{r_j s_j^{-1}}$, $T_j = r_j - R'_j$, and $r_{j-1} = T_j (H_G(M_j || ID_j))^{-1}$, using signer j 's public key y_j .
- Compute $R'_1 = g_2^{s_1^{-1}} y_1^{r_1 s_1^{-1}}$, $T_1 = r_1 - R'_1$. If $T_1 = H_G(M_1 || ID_1)$, $b = 1$, else $b = 0$.

2.4.4 CYH ID-based Ring Signature

We describe the ring signature scheme [CYH05] which we use in our protocol for its ability to keep the identity of the signer anonymous. We assume that $\mathbb{G}_1, \mathbb{G}_2$ are GDH groups. This

signature scheme is an ID-based signature scheme, meaning it relies on a trusted authority GV but allows the use of identifiers without requiring an exchange of public keys while being efficient and requiring a computation amount independent from the group size.

Setup: The Government GV randomly chooses $sk_G = x_G \in \mathbb{Z}_q^*$ as the master secret key and computes the corresponding public key $y_G = g_1^{x_G}$. Let $params = (\mathbb{G}_1, \mathbb{G}_2, e, q, g_1, y_G)$ be the parameters and let $H_G : \{0, 1\}^* \rightarrow \mathbb{Z}_q$, $H_{exp} : \{0, 1\}^* \rightarrow \mathbb{G}_2$ be hash functions modeled as random oracles.

KeyGen: A signer with identity $ID_i \in \{0, 1\}^*$ submits ID_i to GV . GV sets the signer's public key Q_i to be $H_G(ID_i) \in \mathbb{G}_2$, computes the signer's private signing key S_i by $S_i = Q_i^{x_G}$. Then GV sends the private signing key S_i to the signer via a secure channel. Return $(sk, pk) = (S_i, Q_i)$.

Sign: Let $L_{ID} = \{ID_1, ID_2, \dots, ID_n\}$ be the set of all identities of n users. The actual signer, with index s , will follow the following protocol to generate the ring signature on a message M :

- $\forall i \in \{1, 2, \dots, n\} \setminus s$ Choose $U_i \in \mathbb{G}_2$, compute $h_i = H_{exp}(M || L_{ID} || U_i)$.
- Sample $r'_s \leftarrow \mathbb{Z}_q^*$, compute $U_s = Q_s^{r'_s} / \prod_{i \neq s}^n U_i \cdot Q_i^{h_i}$
- Compute $h_s = H_{exp}(M || L || U_s)$ and $V = S_s^{h_s + r'_s}$.
- Output the signature on M as $\sigma = \{\cup_{i=1}^n U_i, V\}$

Verify: A verifier can check the validity of a signature $\sigma = \{\cup_{i=1}^n U_i, V\}$ for the message M and a set of identities L as follows.

- Compute $h_i = H_{exp}(M || L_{ID} || U_i) \forall i \in \{1, 2, \dots, n\}$.
- Check whether $e(y_G, \prod_{i=1}^n U_i \cdot Q_i^{h_i}) = e(g_1, V)$.
- Accept the signature if it is true, reject otherwise.

Both MM and DGNW multi-signatures are used in the contact-tracing protocol described in Section 6.5, in the **Proof Generation Phase** as a way to generate a proof that all participants in the meeting were present at the time, therefore accurately reflecting the reality. CYH ring signature is used in the **Alerting Phase** to alert the close contacts that they have been in contact with a patient, while preserving Patient-Privacy.

Chapter 3

Related Works

In this chapter, we first introduce the isogeny-based DKG protocol CSI-RASHI [BDPV21] and the threshold signature ThreshER SharK [ABCP23b] in Section 3.1 which we use to construct our multi-signature scheme. We then introduce the lattice-based multi-signatures DualMS [Che23] and MuSig-L [BTT22] in Section 3.2 which we compare with our multi-signature scheme in Section 4.4. We then present the isogeny-based proxy-signature CSI-PS [DKSD24] in Section 3.3 which we compare to our proxy signature scheme in Section 5.4. Finally we introduce the contact tracing protocol we refer to as primitive contact tracing [LAY+20] in Section 3.4, which we compare to our scheme in Section 6.4.

3.1 CSI-RASHI and ThreshER SharK

In this section, we briefly introduce the PVP scheme introduced in CSI-RASHI [BDPV21] before presenting the threshold signature ThreshER SharK [ABCP23b] which uses it to obtain the most efficient isogeny-based threshold signature.

3.1.1 Piecewise Verifiable Proofs

PVPs are ZK proofs over secret shared data [BBC⁺19] that consist of a collection of distributed relations $R_0, \dots, R_n$ with the same witness space, where each statement can be verified independently. The goal of a PVP is to prove the existence of a witness w that satisfies $(x_i, w) \in R_i$ for every $i \in \{0, \dots, n\}$. The PVP outputs a proof of the form $(\pi, \pi_i)_{i \in \{0, \dots, n\}}$ where (π, π_0) correspond to the main relation R_0 while $(\pi, \pi_i)_{i \in \{1, \dots, n\}}$ enables verification of x_i in relation to R_i .

The PVPs used in [ABCP23b] and [BDPV21] are used for the following list of distributed relations $R = (R_0, \dots, R_n)$,

$$\begin{aligned} R_0 &= \{((\Xi_k, F_1, F'_1, \dots, F_k, F'_k), f(x)) \mid (F'_l = [c_l f(0)] F_l)_{l=1}^k\}, \\ \forall i = 1, \dots, n : R_i &= \{(x_i, f(x)) \mid f(i) = x_i\} \end{aligned} \tag{3.1}$$

Prover: Given a witness polynomial $f(X) \in \mathbb{Z}_N[X]_t$ and a statement $x = ((\Xi_k, F_1, F'_1, \dots, F_k, F'_k), x_1, \dots, x_n)$, outputs a non-interactive piecewise proof π of the relations in equation 3.1.

1. Parse $\Xi_k = \{c_1, \dots, c_k\}$
2. For $j = 1, \dots, \lambda$: sample $b_j \leftarrow \mathbb{Z}_N[X]_t$ and compute $\hat{F}_j^1 = [c_1 b_j(0)]F_1, \dots, \hat{F}_j^k = [c_k b_j(0)]F_k$
3. Sample $y_0, y'_0 \leftarrow \{0, 1\}^\lambda$ uniformly at random;
4. Set $\mathbf{C}_0 = C(\hat{F}_1^1, \dots, \hat{F}_1^k || \dots || \hat{F}_\lambda^1, \dots, \hat{F}_\lambda^k, y_0)$,
5. Set $\mathbf{C}'_0 = C(F_1, F'_1 || \dots || F_k, F'_k, y'_0)$.
6. For $i = 1, \dots, n$: sample $y_i, y'_i \leftarrow \{0, 1\}^\lambda$ and set $\mathbf{C}_i = C(b_1(i) || \dots || b_\lambda(i), y_i)$ and $\mathbf{C}'_i = C(x_i, y'_i)$;
7. $d = d_1 \dots d_\lambda = H(\mathbf{C}, \mathbf{C}')$ where $\mathbf{C} = (\mathbf{C}_0, \dots, \mathbf{C}_n)$, $\mathbf{C}' = (\mathbf{C}'_0, \dots, \mathbf{C}'_n)$;
8. For $j = 1, \dots, \lambda$: compute $r_j(x) = b_j(x) - d_j f(x) \bmod N$
9. Return $\tilde{\pi} = (\mathbf{C}, \mathbf{C}', r)$ and $\{\pi_i = (y_i, y'_i)\}_{i=0}^n$, where $\mathbf{r} = (r_1, \dots, r_\lambda)$.

Verifier: Given, an index $i \in \{0, \dots, n\}$, a statement piece x_i of the form $x_0 = (\Xi_k, F_1, F'_1, \dots, F_k, F'_k)$ if $i = 0$, or $x_i \in \mathbb{Z}_N$ if $i \neq 0$, as well as a proof piece $(\tilde{\pi} = ((\mathbf{C}, \mathbf{C}', \mathbf{r}), (y_i, y'_i)))$, outputs true or false.

1. If $\mathbf{C}'_i \neq C(x_i, y'_i)$, then return false
2. $d_1 \dots d_\lambda = H(\mathbf{C}, \mathbf{C}')$;
3. If $i = 0$:
 - For $j = 1, \dots, \lambda$:
 - If $d_j = 0$: $\tilde{F}_j^1 \leftarrow [c_1 r_j(0)]F_1, \dots, \tilde{F}_j^k \leftarrow [c_k r_j(0)]F_k$,
 - else $\tilde{F}_j^1 \leftarrow [c_1 r_j(0)]F'_1, \dots, \tilde{F}_j^k \leftarrow [c_k r_j(0)]F'_k$
 - If $\mathbf{C}_0 \neq C(\tilde{F}_1^1, \dots, \tilde{F}_1^k || \dots || \tilde{F}_\lambda^1, \dots, \tilde{F}_\lambda^k, y_0)$ return false, else return true
4. Else, if $\mathbf{C}_i \neq C(r_1(i) + d_1 x_i || \dots || r_\lambda(i) + d_\lambda x_i, y_i)$ return false, else return true

Figure 3.1: Prover and Verifier of the PVP schemes

where the statement of R_0 consists of a public exceptional set $\Xi_k = \{c_1 = 1, c_2, \dots, c_k\}$, a set of curves $(F_1, F'_1, \dots, F_k, F'_k) \in \mathcal{E}^{2k}$, and the statements for the relations $\{R_i\}_{i=1, \dots, n}$ which are elements of $\mathbb{Z}_N$. The witnesses for both PVP schemes is a secret polynomial $f(X) \in \mathbb{Z}_N[X]_t$, which is a polynomial in the variable X with coefficients defined over $\mathbb{Z}_N$ and have a maximum degree of t .

Fig 3.1 describes the prover and verification algorithms of the PVP schemes where $H : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ is a random oracle and $\mathbf{C} : \{0, 1\}^* \times \{0, 1\}^\lambda \rightarrow \{0, 1\}^{2\lambda}$ is a commitment

Verifiable Secret Sharing (BCW VSS)**Computing the Single/ Structured Public Key:**

1. Let $Q = \{1, \dots, n'\}$. Given a superexceptional set $\Xi_k = \{c_1 = 1, c_2, \dots, c_k\}$, a qualified set of parties engage in a round-robin protocol and party P_j computes:

$F_1^{(j)} \leftarrow [x^{(j)}]F_1^{j-1}$, $F_2^{(j)} \leftarrow [c_2 x^{(j)}]F_2^{(j-1)}, \dots, F_k^{(j)} \leftarrow [c_k x^{(j)}]F_k^{(j-1)}$, where $F_1^{(0)} = F_2^{(0)} = \dots = F_k^{(0)} = E_0$. At each step, Party P_j also uses the Structured PVP Scheme given in Fig 3.1 and creates and publishes a proof,

$\pi^{(j)} = (\tilde{\pi}^{(j)}, \pi_1^{(j)}, \dots, \pi_{n'}^{(j)}) \leftarrow \text{Prover}(\Xi_k, (F_1^{(j-1)}, F_1^{(j)}, Y_1^{(j)}, \dots, F_k^{(j-1)}, F_k^{(j)}, Y_k^{(j)}); x^{(j)})$ which contains a main proof $(\tilde{\pi}^{(j)}, \pi_0^{(j)})$ as well as individual proof pieces $\pi_i^{(j)}$ for each other player P_i .

Note that when computing a single, they run the PVP in Fig 3.1 for $k = 1$.

2. Each other player verifies both $\text{Prover}(j, f_j^{(j)}(0), \tilde{\pi}^{(j)}, \pi_0^{(j)})$ and **Verifier** $((\Xi_k, (F_1^{(j-1)}, F_1^{(j)}, \dots, F_k^{(j-1)}, F_k^{(j)}), \tilde{\pi}^{(j)}, \pi_0^{(j)}))$. Whenever a verification of $\pi^{(i)}$ fails, the verifier P_j broadcasts $f_j^{(i)}(X)$ which all other parties verify. If it is correct, since there are at least t honest players, they will be able to reconstitute x^j , compute F_i , and proceed with the protocol (and potentially disqualify P_i). Otherwise, if the checks of $f_j^{(i)}(0)$ fail, the complaint can be ignored (or P_j disqualified). In the latter case, the shares of P_j can also be reconstructed by at least t honest players.
3. At the end of the round-robin, the parties return the structured public key (single) $F_1^{(n')} = [\sum_{j \in Q} x^{(j)}]E_0$, $F_2^{(n')} = [\sum_{j \in Q} c_2 x^{(j)}]E_0, \dots, F_k^{(n')} = [\sum_{j \in Q} c_k x^{(j)}]E_0$.

Figure 3.2: CSI RASHI structured (single) DKG protocol

scheme that is collapsing and quantum computationally hiding. In [BDPV21, ABCP23a] the authors showed that the PVP schemes are correct, sound and ZK against t malicious verifiers.

CSI-RASHI uses the PVP scheme defined above and the Ben-or, Goldwasser, and Wigderson (BGW) VSS protocol (which we do not describe) to build a Distributed Group Protocol (DKG) as described in Figure 3.2.

3.1.2 Threshold Signature Protocol

Thresher SharK builds on the concept of Non-Interactive Threshold Zero Knowledge (NI-TZK), ensuring that zero knowledge is preserved even if up to t verifiers collude, which is then used to propose a Non-Interactive Verifiable Secret Sharing (NI-VSS) protocol based on Shamir Secret Sharing. The resulting NI-VSS is then integrated into CSI-RASHI's DKG protocol by replacing BCW VSS so that each signer holds a share of a joint secret key $x_0 = \sum_{i \in Q} f^{(i)}(0)$, where Q denotes a qualified set. Concretely, each participant possesses a share $x_j = \sum_{i \in Q} f^{(i)}(j)$ with $f^{(i)}(X) \in \mathbb{Z}_N[X]^t$. The participants then execute the structured public key computation step of the DKG protocol, resulting in the structured public key $Y_i = [c_i x_0]E_0 i = 1^k$. The full threshold signature protocol is presented in Algorithms 11 through 13.

Algorithm 11 Thresher SharK KeyGen**Input** $E_0, N, \Xi_k = \{c_1 = 1, \dots, c_k\}$ **Output** $pk, Q_0, sk^{(j)}$ for $j \in Q_0$

- 1: Engage in the Structured DKG Protocol to compute $pk = (Y_1, \dots, Y_{k-1})$ where $Y_i = [c_i x]E_0$ with x the shared secret, and $sk^{(j)} = s^{(j)}$ the secret key of signer j .
- 2: return $pk = (Y_1, \dots, Y_{k-1}), Q_0, sk^{(j)} = s^{(j)}$

Algorithm 12 Thresher SharK Sign**Input** $sk^{(j)}$ for $j \in Q_0, M$ **Output** σ

- 1: **for** $i = 1, \dots, t_k$ **do**
- 2: All parties in Q_0 sample $b_i^{(j)} \leftarrow \mathbb{Z}_N$ and engage in the DKG Protocol with b_i . At each invocation, malicious parties might be disqualified. We assume every step ends with a set $Q_i \subseteq Q_{i-1}$
- 3: **end for**
- 4: Set $(d_1, \dots, d_t) = H(E_1 \dots E_{t_k} || M)$
- 5: **for** $i = 1, \dots, t_k$ **do**
- 6: Each party j in Q_{t_k} computes $r_i^{(j)}(X) - d_i s^{(j)}(X)$
- 7: Using the secret values shared during the NI-VSS protocol, namely $s^{(j)}$ and $b_i^{(j)}$, each other party l verifies $r_i^{(j)}(l) = b_i^{(j)}(l) - d_i s^{(j)}(l)$
- 8: Whenever one of these checks fails, P_l submits a complaint against P_j . If $t+1$ complaints were submitted or more, they are disqualified. The remaining players construct $r_i^{(j)}(0)$ using the information from the DKG.
- 9: For each party P_j in $Q_i \setminus Q_{t_k}$ reconstruct $r_i^{(j)}(0)$ in the same way
- 10: For each party $P_j \in Q_0 \setminus Q_{t_k}$, set and reconstruct $r_i^{(j)}(0) = s^{(j)}(0)$
- 11: All parties build the responses $r_i = \sum_{i \in Q} r_i^{(j)}(0)$
- 12: **end for**
- 13: return $\sigma = (r_i, d_i)_{i=1}^{t_k}$

Algorithm 13 Thresher SharK Verify**Input** pk, σ, M **Output** bit b

- 1: **for** $i = 1, \dots, t_k$ **do**
- 2: Compute $E_i = [r_i]Y_{d_i}$
- 3: **end for**
- 4: Set $(d_1^*, \dots, d_t^*) = H(E_1 \dots E_{t_k} || M)$
- 5: **if** $(d_1^*, \dots, d_t^*) = (d_1, \dots, d_t)$ **then**
- 6: return $b = 1$
- 7: **end if**
- 8: return $b = 0$

ThreshER SharK is currently the state-of-the-art threshold signature in isogeny-based cryptography and achieves security against colluding corrupted signers, i.e., active security. As a threshold signature, it satisfies the robustness property, allowing the protocol to continue even

if some signers are disqualified. On the other hand, it requires the shared secret to be divided into shares at each signing as shares cannot be reused, which separates it from a multi-signature scheme.

3.2 DualMS and MuSig-L

In this section, we describe two lattice multi-signature scheme MuSig-L [BTT22] and DualMS [Che23], which we compare to our multi-signature protocol in Section 5.4.

3.2.1 MuSig-L

MuSig-L [BTT22] is a multi-signature based on the lattice-based signature Dilithium-G [DLL⁺17].

Definition 3.1. MSIS assumption: *Let $\lambda \in \mathbf{N}$ be a security parameter. For a prime $q(\lambda)$, a bound $\beta = \beta(\lambda) > 0$ and positive integers $k = k(\lambda), l = l(\lambda)$ the MSIS assumption holds if for any probabilistic polynomial-time algorithm $\mathcal{A}$, the following advantage is negligible in λ :*

$$\text{Adv}^{\text{MSIS}}(\mathcal{A}) := \Pr[0 < \|x\| \leq \beta \wedge [A|I_k] \cdot x = 0 \bmod q : A \leftarrow R_q^{k \times l}; x \leftarrow \mathcal{A}(A)]$$

Definition 3.2. MLWE assumption: *Let $\lambda \in \mathbf{N}$ be a security parameter. For a prime $q(\lambda)$ and positive integers $k = k(\lambda), l = l(\lambda), \eta = \eta(\lambda)$ the MLWE assumption holds if for any probabilistic polynomial-time algorithm $\mathcal{D}$, the following advantage is negligible in λ :*

$$\begin{aligned} \text{Adv}^{\text{MLWE}}(\mathcal{A}) := & |\Pr[b = 1 : A \leftarrow R_q^{k \times l}; s \leftarrow S_\eta^{k \times l}; t := [A|I_k] \cdot s \bmod q; b \leftarrow (\mathcal{D}(A, t)) \\ & \Pr[b = 1 : A \leftarrow R_q^{k \times l}; s \leftarrow S_\eta^{k \times l}; t \leftarrow R_q^k; b \leftarrow (\mathcal{D}(A, t))]| \end{aligned}$$

MuSig-L follows the offline-online paradigm, which separates the signing algorithm into an offline and online part, Algorithm 18 and 19 respectively. The aggregation of the signature parts output by each signer into a single multi-signature is also explicitly separated into a different algorithm, described in Algorithm 20. The complete protocol is described in Algorithms 14 to 21.

Algorithm 14 Samp

Input r

Output β

- 1: Sample $\beta \leftarrow D_{\sigma_\beta}$ using randomness r
 - 2: **return** β
-

Algorithm 15 RejSamp

Input $(v, z, (\beta^{(j)})_{j \in [m]})$ **Output** b

- 1: $\Sigma = (\sigma_1^2 + \sigma_y^2 \sum_{j=2}^m (\beta^{(j)})^2) \cdot I_{\ell+k}$
 - 2: $\rho \leftarrow [0, 1]$
 - 3: **if** $\rho \geq \min \left(\frac{D^{\ell+k}_{\sqrt{\Sigma}}(z)}{Rep \cdot D^{\ell+k}_{\sqrt{\Sigma}, v}(z)}, 1 \right)$ **then**
 - 4: **return** $b = 0$
 - 5: **end if**
 - 6: **return** $b = 1$
-

Algorithm 16 MuSig-L KeyGen

Input $\bar{A}$ **Output** (sk, pk)

- 1: $s_1 \leftarrow_R \mathcal{S}_\eta^{\ell+k}$
 - 2: $t_1 = \bar{A}s_1 \bmod q$
 - 3: **return** $(sk, pk) = (t_1, s_1)$
-

Algorithm 17 MuSig-L KeyAgg

Input L_{pk} **Output** Agg

- 1: $\{t_1, \dots, t_n\} = L_{pk}$
 - 2: **for** $i \in [1, n]$ **do**
 - 3: $a_i = H_{Agg}(\langle L_{pk} \rangle, t_i)$
 - 4: **end for**
 - 5: $\tilde{t} = \sum_{i=1}^n a_i t_i \bmod q$
 - 6: **return** $Agg = \tilde{t}$
-

Algorithm 18 MuSig-L Sign_{Off}

Input sk_1 **Output** $(\text{off}_1, \text{st}_1)$

```
1:  $s_1 = sk_1$ 
2:  $y_1^{(1)} \leftarrow D_{\sigma_1}^{\ell+k}$ 
3: for  $j \in [2, m]$  do
4:    $y_1^{(j)} \leftarrow D_{\sigma_y}^{\ell+k}$ 
5: end for
6: for  $j \in [1, m]$  do
7:    $w_1^{(j)} = \bar{A}y_1^{(j)} \bmod q$ 
8: end for
9:  $\text{com}_1 = (w_1^{(1)}, \dots, w_1^{(m)})$ 
10:  $\text{off}_1 = (t_1, \text{com}_1)$ 
11:  $\text{st}_1 = (y_1^{(1)}, \dots, y_1^{(m)}, \text{com}_1)$ 
12: return  $(\text{off}_1, \text{st}_1)$ 
```

Algorithm 19 MuSig-L Sign_{O_n}

Input $\text{st}_1, M, sk_1, \mu, (pk_2, \dots, pk_n)$ **Output** on_1

```
1:  $(t_i, \text{com}_i)_{i \in [2, n]} = M$ 
2: if  $\langle t_i \rangle_{i \in [2, n]} \neq \langle \text{pk}_i \rangle_{i \in [2, n]}$  then
3:   return  $\perp$ 
4: end if
5: if  $\exists i \geq 2 : t_i = t_1$  then
6:   return  $\perp$ 
7: end if
8:  $L = \{t_1, \dots, t_n\}$ 
9:  $a_1 = H_{\text{Agg}}(\langle L_{pk} \rangle, t_1)$ 
10:  $\tilde{t} \leftarrow \text{KeyAgg}(L_{pk})$ 
11:  $W = \{t_i \parallel \text{com}_i\}_{i \in [n]}$ 
12:  $(r^{(j)})_{j \in [2, m]} = H_{\text{non}}(\langle W \rangle, \mu, \tilde{t})$ 
13:  $\beta^{(1)} = 1$ 
14: for  $j \in [2, m]$  do
15:    $\beta^{(j)} = \text{Samp}(r^{(j)})$ 
16: end for
17: for  $j \in [1, m]$  do
18:    $w^{(j)} = \sum_{i=1}^n w_i^{(j)} \bmod q$ 
19: end for
20:  $[w_1^{(m)}, \dots, w_k^{(m)}]^T = w^{(m)}$ 
21: if  $w_1^{(m)} \notin \mathbb{R}_q^\times$  then
22:   return  $\perp$ 
23: end if
24:  $\tilde{w} = \sum_{j=1}^m \beta^{(j)} \cdot w^{(j)} \bmod q$ 
25:  $\tilde{y}_1 = \sum_{j=1}^m \beta^{(j)} \cdot y_1^{(j)}$ 
26:  $c = H_{\text{sig}}(\tilde{w}, \mu, \tilde{t})$ 
27:  $v = c \cdot a_1 \cdot s_1$ 
28:  $z_1 = v + \tilde{y}_1$ 
29: if  $\text{RejSamp}(v, z_1, (\beta^{(j)})_{j \in [m]}) = 0$  then
30:    $z_1 \leftarrow \perp$ 
31: end if
32: return  $\text{on}_1 = (z_1, \tilde{w})$ 
```

Algorithm 20 MuSig-L Sign_{Agg}

Input $(on_1, \dots, on_n)$ **Output** σ

- 1: **if** $\exists i \in [1, n] : z_i = \perp$ **then**
 - 2: **return** $\perp$
 - 3: **end if**
 - 4: $z = \sum_{i=1}^n z_i$
 - 5: **return** $\sigma = (\tilde{w}, z)$
-

Algorithm 21 MuSig-L Multi-Verify

Input Agg, σ, μ **Output** b

- 1: $(\tilde{w}, z) = \sigma$
 - 2: $\tilde{t} = \text{Agg}$
 - 3: $c = H_{\text{Sig}}(\tilde{w}, \mu, \tilde{t})$
 - 4: **if** $\bar{A}z - c\tilde{t} = \tilde{w} \bmod q \wedge \|z\|_2 \leq B_n$ **then**
 - 5: **return** $b = 1$
 - 6: **else**
 - 7: **return** $b = 0$
 - 8: **end if**
-

MuSig-L is proven to be EUF-CMA in the Random Oracle Model under the MSIS and MLWE Assumptions.

3.2.2 DualMS

Similarly to MuSig-L, DualMS [Che23] is based on Dilithium-G. DualMS also separates signing into 3 steps, Sign_1 , Sign_2 and the computation of the multi-signature itself Sign_{Agg} , which are described in Algorithms 24, 25, 26 respectively. However, the scheme does not follow the online/offline paradigm. To compensate for this fact, it reduces the key pair by a factor of around 4, and the signature size by a factor of 5. DualMS is proven to be EUF-CMA in the Random Oracle Model under the MSIS and MLWE Assumptions. The algorithms are described in Algorithms 22 to 27.

Algorithm 22 DualMS KeyGen

Input A **Output** (sk, pk)

- 1: $s \leftarrow \mathcal{S}_\eta^{\ell+k}$
 - 2: $t = A\bar{s}$
 - 3: **return** $(sk, pk) = (s, t)$
-

Algorithm 23 DualMS KeyAgg**Input** L_{pk} **Output** Agg

- 1: $\{t_1, \dots, t_n\} = L_{pk}$
- 2: **for** $i \in [n]$ **do**
- 3: $a_i = H_{\text{Agg}}(L_{pk}, t_i) \in \mathcal{C}$
- 4: **end for**
- 5: $\tilde{t} = \sum_{i=1}^n a_i t_i$
- 6: **return** $Agg = \tilde{t}$

Algorithm 24 DualMS Sign₁**Input** sk_1, L_{pk}, μ **Output** st_1

- 1: $s_1 = sk_1$
- 2: $\{t_1, \dots, t_n\} = L_{pk}$
- 3: $a_1 = H_{\text{Agg}}(L_{pk}, t_1)$
- 4: $\tilde{t} = \text{KeyAgg}(L_{pk})$
- 5: $B = H_{\text{com}}(\tilde{t}, \mu) \in \mathbb{R}_q^{k \times \ell'}$
- 6: $\bar{B} = [B \mid I] \in \mathbb{R}_q^{k \times (\ell' + k)}$
- 7: $y_1 \leftarrow D_{\sqrt{s}}^{\ell+k}$
- 8: $r_1 \leftarrow D_{\sqrt{s'}}^{\ell'+k}$
- 9: $w_1 = Ay_1 + B\bar{r}_1 \in \mathbb{R}_q^k$
- 10: **return** $st_1 = (s_1, \tilde{t}, \mu, a_1, y_1, r_1, w_1)$

Algorithm 25 DualMS Sign₂**Input** $st_1, (M_1, M_2, \dots, M_n)$ **Output** σ

- 1: $(s_1, \tilde{t}, \mu, a_1, y_1, r_1, w_1) = st_1$
- 2: **for** $i = 2$ to n **do**
- 3: $w_i = M_i$
- 4: **end for**
- 5: $\tilde{w} = \sum_{i=1}^n w_i$
- 6: $c = H_{\text{Sig}}(\tilde{t}, \mu, \tilde{w}) \in \mathcal{C}$
- 7: $z_1 = y_1 + a_1 cs_1$
- 8: **if** with probability $\min\left(1, \frac{D_s^{\ell+k}(z_1)}{\text{Rep} \cdot D_{a_1 cs_1, s}^{\ell+k}(z_1)}\right)$ **then**
- 9: **return** $\sigma = (c, z_1, r_1)$
- 10: **else**
- 11: **return** $\perp$
- 12: **end if**

Algorithm 26 DualMS Sign_{Agg}

Input $(\sigma_1, \dots, \sigma_n)$
Output Σ

- 1: **for** $i = 1$ to n **do**
- 2: $(c_i, z_i, r_i) = \sigma_i$
- 3: **if** $\exists i$ such that $c_i \neq c_1$ **then**
- 4: **return** $\perp$
- 5: **end if**
- 6: **end for**
- 7: **for** $i = 1$ to n **do**
- 8: $[z'_i, z''_i]^\top = z_i^\top$
- 9: $[r'_i, r''_i]^\top = r_i^\top$
- 10: **end for**
- 11: $\tilde{z} = \sum_{i=1}^n z'_i \in \mathbb{R}^\ell$
- 12: $\tilde{r} = \sum_{i=1}^n r'_i \in \mathbb{R}^{\ell'}$
- 13: $\tilde{e} = \sum_{i=1}^n (z''_i + r''_i) \in \mathbb{R}^k$
- 14: **return** $\Sigma = (c_1, \tilde{z}, \tilde{r}, \tilde{e})$

Algorithm 27 DualMS Multi-Verify

Input Agg, σ, μ
Output b

- 1: $\tilde{t} = Agg$
- 2: $(c, \tilde{z}, \tilde{r}, \tilde{e}) = \tilde{\sigma}$
- 3: $B = H_{\text{com}}(\tilde{t}, \mu)$
- 4: $\tilde{w} = A\tilde{z} + B\tilde{r} + \tilde{e} - c\tilde{t}$
- 5: **if** $[\|\tilde{z}\|_2 \leq B_n \wedge \|\tilde{r}\|_2 \leq B'_n \wedge \|\tilde{e}\|_2 \leq B''_n \wedge H_{\text{Sig}}(\tilde{t}, \mu, \tilde{w}) = c]$ **then**
- 6: **return** $b = 1$
- 7: **else**
- 8: **return** $b = 0$
- 9: **end if**

Both MuSig-L and Dual-MS are state-of-the-art lattice-based multi-signature schemes. MuSig-L in particular, with its offline (preprocessing) round design feature, allows signers to complete most of the computation before knowing the message to be signed, resulting in a single-round online phase when the actual signing occurs. Such a property is highly desirable in practice, as it minimizes interaction and latency during the signing phase, a crucial advantage for systems requiring fast or asynchronous multi-party signing. On the other hand, both signatures suffer from a rather large signature size, which is dependent from the number of signers.

3.3 CSI-PS and Generic Construction

Here, we describe the generic construction by Boldyreva et al. [BPW03] and the isogeny-based proxy signature scheme CSI-PS by Dey et al. [DKSD24], which were proposed concurrently and independently of our schemes. We compare all schemes in Section 5.4.

3.3.1 Generic Construction

Boldyreva et al. [BPW03] propose a generic construction of a proxy signature based on a signature scheme. The resulting proxy signature scheme is proven unforgeable in the EUF-CMA-CWA model based on the unforgeability of the signature scheme itself. We detail the generic construction of a proxy-signature in Figure 3.3 and the corresponding theorem. Let $DS = (\text{Params}, \text{KeyGen}, \text{Sign}_{DS}, \text{Verify}_{DS})$ be a signature scheme. The corresponding delegation-by-certificate proxy signature scheme $(\text{Params}, \text{KeyGen}, \text{Sign}, \text{Verify}, \text{Delegate}, \text{D-Verify/P-KeyGen}, \text{P-Sign}, \text{P-Verify})$ is defined as follows:

Proxy Signature Generic Construction

- **Sign:** $\text{Sign}(\text{sk}, M) = \text{Sign}_{DS}(\text{sk}, 11 \| M)$.
- **Verify:** $\text{Verify}(\text{pk}, M, \sigma) = \text{Verify}_{DS}(\text{pk}, 11 \| M, \sigma)$.
- **Delegate by original signer to proxy signer:** Send the warrant $W = \{\mathcal{M}\}$ corresponding to the message space, the public key pk^o, pk^p and a certificate

$$\sigma^o = \text{Sign}_{DS}(sk^o, 00 \| pk^p \| W)$$

to the proxy signer.

- **Self-delegation by signer:** Run **KeyGen** to obtain (pk^p, sk^p) and create:

$$\sigma^o = \text{Sign}_{DS}(sk^o, 00 \| pk^p \| W)$$

- **P-Sign:** For message $M \in \mathcal{M}$, σ^o, pk^o, pk^p, W , compute:

$$\sigma^p = \text{Sign}_{DS}(sk^p, 01 \| pk^o \| M)$$

and return $pk^o, pk^p, W, \sigma^o, \sigma^p$ If $M \notin \mathcal{M}$, return $\perp$.

- **P-Verify:** For proxy signer with pk^p and original signer with pk^o :

$$\begin{aligned} & \text{Verify}_{DS}(pk^o, \sigma^o, 00 \| j \| pk^p \| W) \\ & \wedge \text{Verify}_{DS}(pk^p, \sigma^p, 01 \| pk^o \| M) \wedge m \in \mathcal{M} \end{aligned}$$

Figure 3.3: Proxy signature generic construction

In order to differentiate a signature from a delegation and a proxy signature, 00, 01, 11 are included with the message. As opposed to most proxy signatures, the certificate σ^o is not

included together with the message in the proxy signature σ^p , making it vulnerable to a mix-and-match attack, which we explain in Section 5.1.

Theorem 3.1 ([BPW03]). *Let DS be a secure digital signature scheme. Then the scheme PS[DS] defined above is a secure proxy signature scheme. Concretely, let $\mathcal{A}$ be an adversary against PS[DS] that makes at most q_d delegation queries, q_{sd} self-delegation queries, q_s standard signature queries, and at most q_{ps} proxy signature queries. Then there exist adversaries $\mathcal{B}, \mathcal{C}, \mathcal{D}$ against DS such that:*

$$\text{Adv}_{\text{PS[DS]}, \mathcal{A}}^{\text{EUF-CMA-CWA}} \leq \text{Adv}_{\text{DS}, \mathcal{B}}^{\text{euf-cma}} + \text{Adv}_{\text{DS}, \mathcal{C}}^{\text{euf-cma}} + q_{sd} \cdot \text{Adv}_{\text{DS}, \mathcal{D}}^{\text{euf-cma}}$$

3.3.2 CSI-PS

CSI-PS [DKSD24] proposed by Dey et al. is based on the signature protocol SeaSign, and is composed of 5 algorithms: **Setup**, **Delegate**, **DelegateVerify**, **ProxySign**, **ProxyVerify**. CSI-PS uses a similar notation to CSI-SharK, where the elliptic curve resulting from the action of the ideal $\mathbf{x}$ on an elliptic curve E is denoted as $E_A = [\mathbf{x}]E$. They also optimize the signature by using L_0 elliptic curves and isogenies as key pairs.

In the Setup phase, the original signer selects the starting curve E_0 as in CSIDH, the interval $[-I_0, I_0]$ (equivalent to $[-B, B]$ in SeaSign), and parameters α_0 to compute the exponent interval $I_1 = \alpha_0 I_0$, γ_0, γ_1 used to compute $L_0 = 2^{\gamma_0} - 1, L_1 = 2^{\gamma_1} - 1$ and integers M_1, M_2 . He then publishes the parameter $params = (E_0, \alpha_0, \gamma_0, \gamma_1, M_1, M_2, I_0, I_1, L_0, L_1)$.

Both the original signer and the proxy signer generates their key pairs, $(sk^o = (\mathbf{x}_i^o)_{i=0}^{L_0}, pk^o = (Y_i^o)_{i=0}^{L_0})$ and $(sk^p = (\mathbf{x}_i^p)_{i=0}^{L_0}, pk^p = (Y_i^p)_{i=0}^{L_0})$ respectively. CSI-PS is correct and proven EUF-CMA under the GAIP Problem. While CSI-PS is only proven EUF-CMA, i.e., unforgeable against message attacks and therefore against an adversary without access to Delegation Queries, it is currently the only proxy signature in isogeny-based cryptography. The detailed algorithms can be found in algorithms 28 to 31.

Algorithm 28 CSI-PS Delegate

Input $sk^o = (\mathbf{x}_i^o)_{i=0}^{L_0}, W$
Output σ^o

- 1: Computes $h^o = H_1(W)$
- 2: Parse h^o as integers $\{h_i^o\}_{i=1}^{M_1}$
- 3: **for** $i = 1, \dots, M_1, j = 1, \dots, M_2$ **do**
- 4: sample $\mathbf{b}_{i,j}^o \leftarrow [-(I_0 + I_1), ((I_0 + I_1))^n]$ and compute $E_{i,j}^o = [\mathbf{b}_{i,j}^o]Y_{h_i^o}^o$
- 5: **end for**
- 6: Set $C^o = H_2(W, \{E_{i,j}^o\}_{i=1,j=1}^{M_1 M_2})$
- 7: Parse C^o as $\{c_{i,j}^o\}_{i=1,j=1}^{M_1 M_2}$
- 8: **for** $i = 1, \dots, M_1, j = 1, \dots, M_2$ **do**
- 9: **if** $c_{i,j}^o = 0$ **then**
- 10: $\mathbf{z}_{i,j}^o = \mathbf{b}_{i,j}^o$
- 11: **else**
- 12: $\mathbf{z}_{i,j}^o = \mathbf{b}_{i,j}^o + \mathbf{x}_{h_i^o}^o$
- 13: **end if**
- 14: **if** $\mathbf{z}_{i,j}^o \in [-I_1, I_1]$ **then**
- 15: Perform next step.
- 16: **else**
- 17: Start from step 3.
- 18: **end if**
- 19: **end for**
- 20: return $\sigma^o = (C^o, \{\mathbf{z}_{i,j}^o\}_{i=1,j=1}^{M_1 M_2})$

Algorithm 29 CSI-PS D-Verify

Input $pk^o = (Y_i^o)_{i=0}^{L_0}, W, \sigma^o$
Output b

- 1: Evaluate $h^o = H_1(W)$
- 2: Parse h^o as integers $\{h_i^o\}_{i=1}^{M_1}$
- 3: Parse C^o as $\{c_{i,j}^o\}_{i=1,j=1}^{M_1 M_2}$
- 4: **for** $i = 1, \dots, M_1, j = 1, \dots, M_2$ **do**
- 5: **if** $c_{i,j}^o = 0$ **then**
- 6: $E_{i,j}^{o*} = [\mathbf{z}_{i,j}^o]Y_{h_i^o}^o$
- 7: **else**
- 8: $E_{i,j}^{o*} = [\mathbf{z}_{i,j}^o]E_0$
- 9: **end if**
- 10: **end for**
- 11: Set $C^{o*} = H_2(W, \{E_{i,j}^{o*}\}_{i=1,j=1}^{M_1 M_2})$
- 12: **if** $C^o = C^{o*}$ **then**
- 13: return $b = 1$
- 14: **end if**
- 15: return $b = 0$

Algorithm 30 CSI-PS P-Sign

Input $sk^p = (\mathbf{x}_i^p)_{i=0}^{L_0}, W, \sigma^o, M$ **Output** σ^p

- 1: Computes $h^p = H_1(\sigma^o)$
 - 2: Parse h^p as integers $\{h_i^p\}_{i=1}^{M_1}$
 - 3: **for** $i = 1, \dots, M_1, j = 1, \dots, M_2$ **do**
 - 4: sample $\mathbf{b}_{i,j}^p \leftarrow [-(I_0 + I_1), ((I_0 + I_1))^n]$ and compute $E_{i,j}^p = [\mathbf{b}_{i,j}^p]Y_{h_i^p}^p$
 - 5: **end for**
 - 6: Set $C^p = H_2(\sigma^o, M, \{E_{i,j}^p\}_{i=1,j=1}^{M_1 M_2})$
 - 7: Parse C^p as $\{c_{i,j}^p\}_{i=1,j=1}^{M_1 M_2}$
 - 8: **for** $i = 1, \dots, M_1, j = 1, \dots, M_2$ **do**
 - 9: **if** $c_{i,j}^p = 0$ **then**
 - 10: $\mathbf{z}_{i,j}^p = \mathbf{b}_{i,j}^p$
 - 11: **else**
 - 12: $\mathbf{z}_{i,j}^p = \mathbf{b}_{i,j}^p + \mathbf{x}_{h_i^p}^p$
 - 13: **end if**
 - 14: **if** $\mathbf{z}_{i,j}^p \in [-I_1, I_1]$ **then**
 - 15: Perform next step.
 - 16: **else**
 - 17: Start from step 3.
 - 18: **end if**
 - 19: **end for**
 - 20: return $\sigma = (C^p, \{\mathbf{z}_{i,j}^p\}_{i=1,j=1}^{M_1 M_2})$
-

Algorithm 31 CSI-PS P-Verify**Input** $pk^o, pk^p, W, \sigma^o, M, \sigma^p$ **Output** b

```

1: Evaluate  $h^o = H_1(W)$ 
2: Parse  $h^o$  as integers  $\{h_i^o\}_{i=1}^{M_1}$ 
3: Parse  $C^o$  as integers  $\{c_{i,j}^o\}_{i=1,j=1}^{M_1 M_2}$ 
4: for  $i = 1, \dots, M_1, j = 1, \dots, M_2$  do
5:   if  $c_{i,j}^o = 0$  then
6:      $E_{i,j}^{o*} = [\mathbf{z}_{i,j}^o] Y_{h_i^o}^o$ 
7:   else
8:      $E_{i,j}^{o*} = [\mathbf{z}_{i,j}^o] E_0$ 
9:   end if
10: end for
11: Set  $C^{o*} = H_2(W, \{E_{i,j}^{o*}\}_{i=1,j=1}^{M_1 M_2})$ 
12: if  $C^o = C^{o*}$  then
13:   Evaluate  $h^p = H_1(\sigma^o)$ 
14:   Parse  $h^p$  as integers  $\{h_i^p\}_{i=1}^{M_1}$ 
15:   Parse  $C^p$  as integers  $\{c_{i,j}^p\}_{i=1,j=1}^{M_1 M_2}$ 
16:   for  $i = 1, \dots, M_1, j = 1, \dots, M_2$  do
17:     if  $c_{i,j}^p = 0$  then
18:        $E_{i,j}^{p*} = [\mathbf{z}_{i,j}^p] Y_{h_i^p}^p$ 
19:     else
20:        $E_{i,j}^{p*} = [\mathbf{z}_{i,j}^p] E_0$ 
21:     end if
22:   end for
23:   Set  $C^{p*} = H_2(\sigma^p, m, \{E_{i,j}^{p*}\}_{i=1,j=1}^{M_1 M_2})$ 
24:   if  $C^p = C^{p*}$  then
25:     return  $b = 1$ 
26:   end if
27: end if
28: return  $b = 0$ 

```

3.4 Privacy-Preserving Contact Tracing Protocol

In this section, we introduce the privacy-preserving contact tracing protocol proposed by Liu et al. [LAY⁺20]. We then shift perspective and consider it in the multi-party setting (meeting between n users) in Section 6.5, where we will refer to it as *Primitive Multi-Party Tracing*. The protocol is classified as a centralized contact tracing app, as information about the close contacts is stored in a central server. Its main advantage lies in its *close contact privacy*, i.e. the identity of close contacts of patients is kept hidden, achieved by using a zero-knowledge

proof technique. provides to the close contacts by using a zero-knowledge proof technique. The protocol is divided into four different phases, **Registration Phase**, **Meeting Phase**, **Medical Treatment Phase** and **Tracing Phase** which are defined as follows:

Registration Phase: The trusted authority, corresponding to the Government (GV) sets up the generators $u, u_1, u_2 \in \mathbb{G}_1$ and $g, g_1, g_2 \in \mathbb{G}_2$, before generating his master key pair $(sk_G, pk_G) = (x_G, y_G)$ with $x_G \leftarrow \mathbb{Z}_q$ and $y_G = g^{x_G}$. He then publishes the public parameters $\{\lambda, H_G, y_G, u, u_1, u_2, g, g_1, g_2\}$ with $H_G : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ a hash function.

Each day, Alice, a negative user, generates her key pair $(sk_A, pk_A) = (x_A, y_A)$ by sampling $x_A \leftarrow \mathbb{Z}_q$ and computing $y_A = g^{x_A}$ and registers her information with y_A on the government website. GV then generates an identifier ID_A and a signature $\sigma_A = \text{Sign}(x_G, M)$ on the message $M = \{"-VE", y_A, ID_A, DATE\}$ with $"-VE"$ indicating a negative user and $DATE$ the current day. Alice verifies σ_A with y_G and stores it if the verification succeeds. The tuple (y, ID, σ) will be used by users to identify one another.

Meeting Phase: Alice's app is periodically broadcasting the hash $h_A = H(" -VE", ID_A, y_A, \sigma_A)$ to the surroundings by Bluetooth. After exchanging a certain amount of hash within a limited amount of time (15 minutes) with another user, Bob (with $h_B = H(" -VE", ID_B, y_B, \sigma_B)$), both will be considered as close contacts and register the information of the other in their app. To do so, the **Meeting Phase** itself is divided into two subphases: **Verification Phase** and **Mutual Commitment Phase**.

- **Verification Phase:** Both Alice and Bob first check the tuple (ID, y, σ) of the other and verify σ generated by GV during the **Registration Phase**. They will then verify the public key by generating a signature σ' on a challenge number r provided by the other. Alice will generate a Schnorr signature σ'_A using $r_B \in \mathbb{Z}$ provided by Bob, and inversely σ_B using $r_A \in \mathbb{Z}$ for Bob. The exchange goes as follows:

1. Randomly choose $k \in \mathbb{Z}_q$
2. Compute $t = H(g^k, r_B)$
3. Compute $s = k - x_A \cdot t \pmod{q}$
4. Output the signature $\sigma' = (s, t)$ for the challenge number r_B

After receiving σ'_A from Alice, Bob computes $h'_A = H(" -VE", ID_A, y_A, \sigma_A)$ and checks if $t = H(g^s y_A^t, r_B)$. If the verification succeeds, Bob registers Alice as a close contact in the app and stores $\{ID_A, y_A, \sigma_A\}$ together with the date. The symmetric protocol happens simultaneously, with Bob generating σ'_B and Alice storing $\{ID_B, y_B, \sigma_B\}$.

- **Mutual Commitment Phase:** In order to hide the identity of the other if one of them becomes a patient, Alice and Bob will additionally exchange a signature σ'' which will then be used during the zero-knowledge proof in the **Medical Treatment Phase**. Alice will use her secret key x_A and Bob's identifier ID_B to compute $\sigma''_A = u^{1/(H(ID_B) + x_A)}$ and send it to Bob. Bob checks if $e(\sigma''_A, g^{H(ID_B)} y_A) = e(u, g)$ and stores σ''_A if the verification succeeds.

The symmetric protocol occurs simultaneously with Bob generating σ_B'' and Alice storing it.

In order to extend this meeting phase to the multi-party setting with n users, the protocol is repeated $n - 1$ times for each pair of close contacts.

Medical Treatment Phase: Alice is together with the Doctor D after testing positive for Covid-19 and needs to alert her close contacts in the past 14 days (i.e., Bob) of the risk of infection while preserving their privacy (i.e., close contact privacy). To do so, she will generate a pseudo-public key and by using the mutual commitment generated during the **Mutual Commitment Phase**, prove, in a zero-knowledge way, to D that Bob is indeed her close contact. The protocol goes as follows:

- D first authenticates Alice by executing the **Verification Phase** in the **Meeting Phase**, and obtains ID_A .
- Alice retrieves Bob's information stored on her smartphone, $(ID_B, y_B, \sigma_B, \sigma_B'')$.
- She then generates the pseudo public key $(h, \hat{B})$ by sampling $k \leftarrow \mathbb{Z}_q$ and computes: $h = e(u, g)^k, \hat{B} = e(u, y_B)^k$.
- Alice needs to prove to D that $(h, \hat{B})$ is correct and that Bob is her close contact. Conceptually, Alice needs to prove that $h = e(u, g)^k, \hat{B} = e(u, y_B)^k$, corresponding to the pseudo public key and $e(\sigma_B'', g^{H(ID_A)} y_B) = e(u, g)$ corresponding to the mutual commitment. To instantiate this proof, Alice samples $s_1, s_2, t \leftarrow \mathbb{Z}_q$ and computes: $A_1 = g_1^{s_1} g_2^{s_2}$, $A_2 = y_B \cdot g_1^{s_1}$, and $C = \sigma_B'' u_1^t$. She then sends A_1, A_2 and C to D and proves the following system of equations $PK\{(s_1, s_2, t, \alpha_1, \alpha_2, \beta_1, \beta_2, x)\}$:

$$\begin{aligned}
 A_1 &= g_1^{s_1} \cdot g_2^{s_2} \\
 A_1^k &= g_1^{\alpha_1} \cdot g_2^{\alpha_2} \\
 A_1^t &= g_1^{\beta_1} \cdot g_2^{\beta_2} \\
 h &= e(u, g)^k \\
 \hat{B} &= e(u, A_2^k \cdot g_1^{-\alpha_2}) \\
 e(C u_1^{-t}, g^{H(ID_A)} A_2 g_1^{-s_2}) &= e(u, g)
 \end{aligned}$$

- If the proof is correct, D generates a group signature $\sigma_D = \text{GSign}(\text{sk}_D, \text{GPK}, M_D)$ with sk_D his secret key, GPK the group public key on the message $M_D = (h, \hat{B}, DATE)$ and publishes $\{\sigma_D, h, \hat{B}, DATE\}$ into the bulletin board BB. D will also inform GV that Alice has been confirmed as positive and update the entry $\{''+VE'', y_A, ID_A, DATE\}$ and the date every day until Alice recovers.

Tracing Phase: To check if Bob is a close contact of a Covid-19 patient, Bob periodically scans new entries on BB and for each entry retrieves the secret key corresponding to that day

$sk_B = x_B$. He then simply confirms if the pseudo public key $(h, \hat{B})$ corresponds to himself by checking the equality $\hat{B} = e(u, g^{x_B})^k = h^{x_B}$ with his secret key.

While the protocol assumes honest-but-curious users, who may attempt to infer additional information but otherwise follow the protocol, it distinguishes itself from prior contact-tracing approaches through its design and security guarantees. Unlike systems that rely on broadcast tokens or pseudonyms, it achieves privacy for both the infected individual and their close contacts through robust cryptographic mechanisms, namely zero-knowledge proofs and group signatures, supported by formal security proofs. Furthermore, the protocol incorporates mechanisms to prevent false-positive reports, ensuring that only verified infection events can trigger exposure notifications. At the same time, the protocol only considers pairwise interactions between users, which does not reflect the reality in which more than two users might be in the process of becoming close contacts at the same time.

Chapter 4

Actively Secure Isogeny-Based Multi-Signature

The current research on post-quantum multi-signatures has been focused on lattice-based multi-signatures, with MuSig-L and DualMS considered state of the art. Both schemes, however, suffer from rather large key and signature sizes, making the possibility of an isogeny-based multi-signature with its relatively small key and signature sizes an attractive option. In isogeny-based cryptography, while no multi-signatures have been proposed as of yet, several threshold signatures have been proposed, with ThreshER SharK being the most efficient. Threshold and multi-signature schemes remain, however, fundamentally apart in structure and design. In threshold schemes, the secret key is distributed among the signers and each partial signature derives from a shared secret, whereas multi-signatures rely on aggregating independently generated keys and signatures. These contrasting foundations influence setup, key management, and security proofs, making the transformation from one primitive to the other far from straightforward and underscoring the complementarity of the two approaches.

In this chapter, we first show the construction of our new proof system and sub-protocol by modifying the PVP protocol and DKG protocol introduced in Section 3.1, necessary to achieve active security, i.e, in the rogue key attack model used for multi-signatures. We then show the construction of the multi-signature protocol itself, which calls the sub-protocol in both Key Aggregation and Signing to perform the necessary group action computations in a round-robin way. The use of round-robin procedures in isogeny-based cryptography introduces additional challenges when establishing active security. This required collaboration between signers creates an extra layer of interaction that makes the standard Double Forking Lemma insufficient, unlike in conventional multi-signature settings. To address this, we adopt a novel ad hoc strategy: we first show that the behavior of malicious signers can be restricted to that of honest-but-curious participants during the group action computations, and then apply the Double Forking Lemma to complete the proof of active security.

4.1 Group Action Sub-Protocol

In this section, we introduce a proof system derived from the PVP schemes described in Section 3.1, offering two versions: a structured version and an unstructured one. We then integrate it into our Group Action Sub-Protocols ($SP^{Structured}$ and SP^{Single}) based on CSI-RASHI's DKG protocol to ensure the signers make the correct computations.

4.1.1 (Structured) Proof System

Let $\Xi_{k_{ps}} = \{c_1 = 1, \dots, c_{k_{ps}}\}$ be the exceptional set the signers have agreed upon, and let Q of size n be the set of signers. As opposed to ThreshER SharK, in which the shared secret is computed collaboratively through a VSS and then divided into shares and distributed among signers, each signer in a multi-signature executes **KeyGen** independently by sampling $x \leftarrow \mathbb{Z}_N$ and computing the corresponding public keys $Y_i = [c_i x]E_0$ for $k \in [1, k-1]$ (i.e., $sk^{(j)} = x^{(j)}, pk^{(j)} = \{Y_1^{(j)}, \dots, Y_{k-1}^{(j)}\}$ for signer P_j). As a result, the relations R_i for $i \in [1, n]$ tied to shares x_i are irrelevant in our scheme, with only the main relation corresponding to the computation of the elliptic curves remaining, corresponding to Relation R_0 .

$$R_0 = \{((\Xi_{k_{ps}}, F_1, F'_1, \dots, F_k, F'_{k-1}), x) | (F'_l = [c_l x]F_l)_{l=1}^{k_{ps}}\} \quad (4.1)$$

This relation is, however, insufficient for the structured proof system, corresponding to **KeyAgg**. As each signer will execute Key Aggregation after generating their key pair, there needs to be an additional condition that must be satisfied: the secret key used to compute the public keys must be identical to the secret key used when those same public keys are aggregated into the aggregated public key. Let $a^{(j)}$ be signer P_j aggregation coefficient, and let the aggregated public key be of the form $\bar{Y}_l = [\sum_{j \in Q} c_l a^{(j)} x^{(j)}]$. To capture this condition, we define the following relation R'_0 :

$$R'_0 = \{((\Xi_{k_{ps}}, F_1, F'_1, F''_1, \dots, F_{k_{ps}}, F'_{k_{ps}}, F''_{k_{ps}}), x) | (F'_l = [c_l a x]F_l \wedge F''_l = [c_l x]E_0)_{l=1}^{k_{ps}}\} \quad (4.2)$$

where $k_{ps} = k-1$, $(F_1, \dots, F_{k-1})$ correspond to the elliptic curves sent by signer P_{j-1} , $(F'_1, \dots, F'_{k-1})$ correspond to the elliptic curves computed by P_j and $(F''_1, \dots, F''_{k-1})$ correspond to P_j 's public keys. In practice, P_j proves $F'_l = [c_l a^{(j)} x^{(j)}]F_l$, proving the computations were correctly made, and $F''_l = Y_l^{(j)} = [c_l x^{(j)}]E_0$ demonstrating the same secret key $x^{(j)}$ was used to compute the public keys for $l \in [1, k-1]$.

To obtain the (structured) proof system, we remove the components relevant to threshold schemes and add the additional condition in the structured version. As the aggregation coefficients are only involved in the computation of the aggregated public key, they only appear in the proof system in the structured version, corresponding to Key Aggregation to establish R'_0 . The non-structured version is invoked t_k times during **Multi-Sign** with ($k_{ps} = 1$), to prove $E_i^{(j)} = [b_i^{(j)}]E_i^{(j-1)}$ for $i \in [1, t_k]$. The full proof system is given in Fig 4.1.

Prover: Given, a witness $x \in \mathbb{Z}_N$, the aggregated coefficient a and a statement $x_0 = (\Xi_{k_{ps}}, F_1, F'_1, F''_1, \dots, F_{k_{ps}}, F'_{k_{ps}}, F''_{k_{ps}})$, outputs a non-interactive proof π of the relations in Eq 4.1(respectively 4.2).

1. Parse $\Xi_{k_{ps}} = \{c_1, \dots, c_{k_{ps}}\}$
2. For $j = 1, \dots, \lambda$: sample $b_j \leftarrow \mathbb{Z}_N$ and compute:
 - $\hat{F}_j^1 = [c_1 a b_j] F_1, \dots, \hat{F}_j^{k_{ps}} = [c_{k_{ps}} a b_j] F_{k_{ps}}$
 - $\tilde{F}_j^1 = [c_1 b_j] E_0, \dots, \tilde{F}_j^{k_{ps}} = [c_{k_{ps}} b_j] E_0$
3. Sample $y_0, y'_0 \leftarrow \{0, 1\}^\lambda$ uniformly at random;
4. Set $\mathbf{C}_0 = C(\hat{F}_1^1, \tilde{F}_1^1, \dots, \hat{F}_1^{k_{ps}}, \tilde{F}_1^{k_{ps}} \parallel \dots \parallel \hat{F}_\lambda^1, \tilde{F}_\lambda^1, \dots, \hat{F}_\lambda^{k_{ps}}, \tilde{F}_\lambda^{k_{ps}}, y_0)$,
5. Set $\mathbf{C}'_0 = C(F_1, F'_1, F''_1 \parallel \dots \parallel F_{k_{ps}}, F'_{k_{ps}}, F''_{k_{ps}}, y'_0)$.
6. $d = d_1 \dots d_\lambda = H(\mathbf{C}_0, \mathbf{C}'_0)$;
7. For $j = 1, \dots, \lambda$: compute $r_j = b_j - d_j x \bmod N$
8. Return $\tilde{\pi} = (\mathbf{C}_0, \mathbf{C}'_0, \mathbf{r})$ and $\pi_0 = (y_0, y'_0)$, where $\mathbf{r} = (r_1, \dots, r_\lambda)$.

Verifier: Given, a statement of the form $x_0 = (\Xi_{k_{ps}}, F_1, F'_1, F''_1, \dots, F_{k_{ps}}, F'_{k_{ps}}, F''_{k_{ps}})$, the aggregated coefficient a as well as a proof $(\tilde{\pi} = ((\mathbf{C}_0, \mathbf{C}'_0, \mathbf{r}), (y_0, y'_0)))$, outputs true or false.

1. If $\mathbf{C}'_0 \neq C(x_0, y'_0)$, then return false
2. $d_1 \dots d_\lambda = H(\mathbf{C}_0, \mathbf{C}'_0)$;
3. For $j = 1, \dots, \lambda$:
 - If $d_j = 0$:
 - $\hat{F}_j^1 = [c_1 a r_j] F_1, \dots, \hat{F}_j^{k_{ps}} = [c_{k_{ps}} a r_j] F_{k_{ps}}$,
 - $\tilde{F}_j^1 = [c_1 r_j] E_0, \dots, \tilde{F}_j^{k_{ps}} = [c_{k_{ps}} r_j] E_0$,
 - else
 - $\hat{F}_j^1 = [c_1 a r_j] F'_1, \dots, \hat{F}_j^{k_{ps}} = [c_{k_{ps}} a r_j] F'_{k_{ps}}$
 - $\tilde{F}_j^1 = [c_1 r_j] F''_1, \dots, \tilde{F}_j^{k_{ps}} = [c_{k_{ps}} r_j] F''_{k_{ps}}$
4. If $\mathbf{C}_0 \neq C(\hat{F}_1^1, \tilde{F}_1^1, \dots, \hat{F}_1^{k_{ps}}, \tilde{F}_1^{k_{ps}} \parallel \dots \parallel \hat{F}_\lambda^1, \tilde{F}_\lambda^1, \dots, \hat{F}_\lambda^{k_{ps}}, \tilde{F}_\lambda^{k_{ps}}, y_0)$, return false, else return true

Figure 4.1: Prover and Verifier of the structured proof system

Theorem 4.1. *The algorithms in Figure 4.1 constitute a non-interactive zero-knowledge quantum (structured) proof of knowledge in the QROM for the relation 3.1 (respectively relation 4.1)*

provided the commitment scheme $\mathbf{C}$ is collapsing and quantum computationally hiding.

Proof. As the proof system is derived from a Fiat Shamir transform which benefits from special soundness and quantum computationally unique responses, we simply need to prove correctness, special soundness, and zero-knowledge. Special Soundness follows by using the technique employed in CSI-RASHI [BDPV21] as we similarly use *weak* Fiat–Shamir Transform in which challenges are derived solely from the commitment, while we invoke Unruh’s result [Unr17] on the Fiat–Shamir Transform in the quantum setting for zero-knowledge. We sketch correctness for the structured proof system (the unstructured case follows from the correctness of the underlying PVP proofs).

Correctness We prove that if the protocol is followed correctly and if the input was a valid statement witness pair $(x, w) \in R'_0$, then the verifier will accept the proof with probability 1. For $j = 1, \dots, \lambda$, for $l = 1, \dots, k$:

- If $d_j = 0$

$$[c_l a r_j] F'_l = [c_l a b_j] F'_l = \hat{F}_j^l$$

$$[c_l r_j] E_0 = [c_l b_j] E_0 = \tilde{F}_j^l$$

- else

$$\begin{aligned} [c_l a r_j] F'_l &= [c_l a (b_j - x)] F'_l = [c_l a (b_j - x)] [c_l a x] F_l \\ &= [c_l a b_j(0)] = \hat{F}_j^l \end{aligned}$$

$$\begin{aligned} [c_l r_j] F''_l &= [c_l (b_j - x)] F''_l = [c_l (b_j - x)] [c_l x] E_0 \\ &= [c_l b_j] E_0 = \tilde{F}_j^l \end{aligned}$$

which concludes the proof of correctness and the proof of Theorem 1.

Comparison We compare our proof systems (PS) with the PVP scheme [BDPV21] introduced in Section 3.1, as summarized in Table 4.1. The comparison is carried out for both the structured case and the unstructured single-case ($k_{ps} = 1$), considering both the Prover and the Verifier ($i = 0$). In Table 4.1, *Proof* denotes the proof size, *Commits* the number of commitments, and *Isogenies* the number of isogeny computations. All quantities are expressed in terms of λ (the security parameter), n (the number of signers), k (the number of curves in the public key), and l such that N is an l -bit integer. Because the witness must be shown to coincide with the secret key generated during **KeyGen**, our structured proof system requires more isogeny computations, while the amount remains unchanged in the other cases. On the other hand, by removing the parts tied to threshold constructions, both the proof size and the number of commitments are reduced compared to PVP.

Table 4.1: Comparison between PVP [BDPV21] and PS in the structured/single case

	Proof	Commits	Isogenies
Single PVP (Prover)	$(6n + l + 4)\lambda$	$2n + 2$	λ
Single PS (Prover)	$(l + 4)\lambda$	2	λ
Single PVP (Verifier)	$(6n + l + 4)\lambda$	2	λ
Single PS (Verifier)	$(l + 4)\lambda$	2	λ
Structured PVP (Prover)	$(6n + l + 4)\lambda$	$2n + 2$	λk
Structured PS (Prover)	$(l + 4)\lambda$	2	$2\lambda k$
Structured PVP (Verifier)	$(6n + l + 4)\lambda$	2	λk
Structured PS (Verifier)	$(l + 4)\lambda$	2	$2k\lambda$

4.1.2 Round-Robin Group Action Computation

Following the approach of Sashimi [CS20] and ThreshER Shark [ABCP23b], we now use the proof systems introduced previously to design a sub-protocol that will be used to compute the round-robin group action computations. To achieve this, we adapt the DKG protocol originally described in [ABCP23a]. The sub-protocol allows the set of signers Q , each holding a secret isogeny s_{P_j} , to jointly compute the action of the aggregated isogeny $s = \sum_{j \in S} s_{P_j}$ on the starting curve E_0 in a zero-knowledge manner. As opposed to the DKG, our sub-protocol does not use a VSS step to divide the shared secret, but simply performs the required computations in a round-robin manner. We propose two variants of the sub-protocol: the structured sub-protocol $SP^{Structured}$ used for **KeyAgg** and the non-structured sub-protocol SP^{Single} used to compute the single $E_i = [b_i]E_0$ with $b_i = \sum_{j=1}^n b_i^{(j)}$ during **Multi-Sign**.

Computing the **single**/ **structured** aggregated public key

1. Let $Q = \{1, \dots, n'\}$. Given a super-exceptional set $\Xi_k = \{c_1 = 1, c_2, \dots, c_k\}$, E_0 the starting curve, a list of aggregated coefficients $(a^{(j)})_{j \in Q}$, a qualified set of parties engage in a round-robin protocol and party P_j computes:

$F_1^{(j)} \leftarrow [a^{(j)}x^{(j)}]F_1^{(j-1)}$, $F_2^{(j)} \leftarrow [a^{(j)}c_2x^{(j)}]F_2^{(j-1)}$, $\dots$, $F_k^{(j)} \leftarrow [a^{(j)}c_kx^{(j)}]F_k^{(j-1)}$, where $F_1^{(0)} = F_2^{(0)} = \dots = F_k^{(0)} = E_0$. At each step, Party P_j also uses the Structured Proof System given in Fig 4.1 and creates and publishes a proof,

$$\pi^{(j)} = (\tilde{\pi}^{(j)}, \pi_0^{(j)}) \leftarrow \text{Prover}(\Xi_k, (F_1^{(j-1)}, F_1^{(j)}, Y_1^{(j)}, \dots, F_k^{(j-1)}, F_k^{(j)}, Y_k^{(j)}); x^{(j)}).$$

Note that when computing a single, they run the proof system in Fig 4.1 for $k = 1$.

2. Each other player verifies **Verifier** $((\Xi_k, (F_1^{(j-1)}, F_1^{(j)}, Y_1^{(j)}, \dots, F_k^{(j-1)}, F_k^{(j)}, Y_k^{(j)}), \tilde{\pi}^{(j)}, \pi_0^{(j)}))$. As a proof generated by P_j is independent from the identity of the signer it is being sent to, the result from **Verifier** should be identical for all players unless a malicious behavior occurs, in which case the honest players can either disqualify the malicious party or withdraw.
3. At the end of the round-robin, the parties return the structured public key $F_1^{(n')} = [\sum_{j \in Q} a_1^{(j)}x^{(j)}]E_0$, $F_2^{(n')} = [\sum_{j \in Q} a_2^{(j)}c_2x^{(j)}]E_0, \dots, F_k^{(n')} = [\sum_{j \in Q} a_k^{(j)}c_kx^{(j)}]E_0$.

 Figure 4.2: SP^{Single} and $SP^{Structured}$

4.2 Construction

Using the previously introduced sub-protocol and proof system, we define our multi-signature scheme. The scheme comprises four algorithms, as detailed in Section 2.2: **KeyGen**, **KeyAgg**, **Multi-Sign**, and **Multi-Verify**. In **KeyGen**, each signer samples $x \leftarrow \mathbb{Z}_N$ and computes the corresponding public key. During **KeyAgg**, aggregation coefficients are derived from the set of participants' public keys, followed by execution of $SP^{Structured}$ to produce the aggregated public key. For **Multi-Sign**, each signer P_j chooses $b_l^{(j)} \leftarrow \mathbb{Z}_N$ for $l \in [1, t_k]$ and performs SP^{Single} t_k times to compute the elliptic curves $E_l = [\sum_{j \in Q_l} b_l^{(j)}]E_0$, then calculates the challenges $d_1, \dots, d_{t_k}$ and responses $r_1, \dots, r_{t_k}$.

During both **KeyAgg** and **Multi-Sign**, executing the (structured) sub-protocols produces proofs verified by all participants. Because verification does not depend on the verifier (i.e., whether P_{j_1}, P_{j_2} for $j_1, j_2 \in [1, n]$), honest signers will reach identical outcomes. If a proof fails verification, honest participants may disqualify the offending signer(s) or withdraw from the protocol. Similarly, false accusations from malicious signers can trigger the same response. Section 4.3 provides further discussion of these cases. As opposed to ThreshER SharK, our scheme does not satisfy the robustness property, requiring that the protocol continues even in the case of disqualifications. Indeed, as our scheme does not employ secret shares, remaining participants cannot recover the secrets of disqualified signers. The full protocol is presented in Algorithm 32 through Algorithm 35.

Algorithm 32 KeyGen

Input $E_0, N, \Xi_k = \{c_1 = 1, c_2, \dots, c_k\}$
Output pk, sk

- 1: sample $x \leftarrow \mathbb{Z}_N$
- 2: **for** $i = 1, \dots, k$ **do**
- 3: compute $Y_i = [c_i x]E_0$
- 4: **end for**
- 5: return $sk = x, pk = (Y_1, \dots, Y_{k-1})$

Algorithm 33 KeyAgg

Input Set of signers $Q, L_{pk} = \{Y_1^{(j)}, \dots, Y_k^{(j)}\}_{j \in Q}, \Xi_k$
Output Agg

- 1: **for** j in Q **do**
- 2: Compute locally $a^{(j)} = H_{Agg}(Y_1^{(j)} \dots Y_{k-1}^{(j)} || L_{pk})$.
- 3: **end for**
- 4: Execute $(\bar{Y}_1, \dots, \bar{Y}_{k-1}) \leftarrow SP^{Structured}$ (refer to Fig 4.2). At the end of the protocol, we obtain a set of qualified parties which we assume to be $Q_0 = \{P_1, \dots, P_{n'}\}$
- 5: return $Agg = (\bar{Y}_1, \dots, \bar{Y}_{k-1})$

Algorithm 34 Multi-Sign**Input** M , set of signers Q_0, Ξ_k, Agg **Output** The multi-signature Σ

-
- 1: **for** $l = 1, \dots, t_k$ **do**
 - 2: Parties sample $b_l^{(j)} \leftarrow \mathbb{Z}_N$
 - 3: Run $E_l \leftarrow SP^{Single}$. At the end, parties will have computed $E_l = [\sum_{j \in Q_l} b_l^{(j)}]E_0$. Note that parties can get disqualified at every turn. We assume to end every step with a set $Q_l \subseteq Q_{l-1}$. Only the parties in Q_{t_k} continue the protocol.
 - 4: **end for**
 - 5: The parties compute the challenge $d_1, \dots, d_{t_k} \leftarrow H_{Sign}(Agg, E_1 \dots E_{t_k} || M)$
 - 6: **for** $l = 1, \dots, t_k$ **do**
 - 7: Signer P_j computes $r_l^{(j)} = b_l^{(j)} - d_l a^{(j)} x^{(j)} \bmod N$
 - 8: **end for**
 - 9: Broadcast $(r_1^{(j)}, \dots, r_{t_k}^{(j)})$ and compute $r_i = \sum_{j \in Q_{t_k}} r_i^{(j)} \bmod N$ for $i \in [1, t_k]$
 - 10: Output $\Sigma = (r_1, \dots, r_{t_k}, d_1, \dots, d_{t_k})$
-

Algorithm 35 Multi-Verify**Input** $\Sigma, Agg = (\bar{Y}_1, \dots, \bar{Y}_{k-1}), M, \Xi_k, Q_{t_k}$ **Output** bit b

-
- 1: **for** $l = 1, \dots, t_k$ **do**
 - 2: $E_l \leftarrow [r_l] \bar{Y}_{d_l}$
 - 3: **end for**
 - 4: Compute $(d_1^*, \dots, d_{t_k}^*) = H_{Sign}(Agg, E_1 \dots E_{t_k} || m)$
 - 5: **if** $(d_1^*, \dots, d_{t_k}^*) = (d_1, \dots, d_{t_k})$ **then**
 - 6: $b \leftarrow 1$
 - 7: **else**
 - 8: $b \leftarrow 0$
 - 9: **end if**
 - 10: Output b
-

Our protocol uses two hash functions $H_{Agg} : \{0, 1\}^* \rightarrow \mathbb{Z}_N$ which is used to compute the aggregation coefficients in **KeyAgg** and $H_{Sign} : \{0, 1\}^* \rightarrow (\Xi_k)^{t_k}$ which returns t_k elements from $\Xi_k = \{c_0 = 0, c_1 = 1, c_2, \dots, c_{k-1}\}$ which will serve as challenges during **Multi-Sign**. The parameters t_k, k are linked by the same relation as CSI-SharK as introduced in Section 2.3, with $t_k = \lambda \log_2(k)$. For $k = 16$ and $t_k = 23$, we get an aggregated public key of 1 KB, and a signature size of 791B.

4.3 Analysis

In this section, we prove the correctness of our multi-signature scheme, before proving it is actively EUF-CMA in the Random Oracle Model.

4.3.1 Correctness

Let $Q = \{P_1, \dots, P_{n'}\}$ be the set of signers attempting to generate a multi-signature and $\Xi_k = \{c_0 = 0, c_1 = 1, c_2, \dots, c_{k-1}\}$ the exceptional set. Each signer P_j first generates their key pair $(x^{(j)}, \{Y_1^{(j)}, \dots, Y_{k-1}^{(j)}\}) = \text{KeyGen}(params)$. They will then locally compute the aggregation coefficients $a^{(j)} = H_{\text{Agg}}(Y_1^{(j)} \dots Y_{k-1}^{(j)} || L_{pk})$ before executing $SP^{\text{Structured}}(\Xi_k, E_0, Q, (\sum_{j \in Q_0} c_i a^{(j)} x^{(j)})_{i \in [1, k-1]})$ to compute the aggregated key $\text{Agg} = (\bar{Y}_1, \dots, \bar{Y}_{k-1})$ as follows

$$\bar{Y}_i = [c_i a^{(1)} x^{(1)}] \dots [c_i a^{(n')} x^{(n')}] E_0 = \left[c_i \sum_{j \in Q} a^{(j)} x^{(j)} \right] E_0$$

Each signer P_j will then generate the multi-signature Σ on a message M by sampling a secret isogeny $b_i^{(j)} \leftarrow \mathbb{Z}_N$ corresponding to the commitment before executing $SP^{\text{Single}}(\Xi_k, Q_0, \sum_{j=1}^n b_i^{(j)})$ together, for $i \in [1, t_k]$. The sub-protocol will return E_i as follows:

$$E_i = [b_i^{(1)}] \dots [b_i^{(n)}] E_0 = \left[\sum_{j \in Q_{t_k}} b_i^{(j)} \right] E_0$$

Each signer P_j will then make the following computations locally for $i \in [1, t_k]$:

$$\begin{aligned} (d_1, \dots, d_{t_k}) &= H_{\text{Sign}}(E_1 \dots E_{t_k} || M) \\ r_i^{(j)} &= b_i^{(j)} - d_i a^{(j)} x^{(j)} \bmod N \end{aligned} \tag{4.3}$$

and broadcast $(r_1^{(j)}, \dots, r_{t_k}^{(j)})$ to the other signers. The signers will then compute locally $r_i = \sum_{j \in Q_{t_k}} r_i^{(j)} \bmod N$ and obtain the multi-signature $\Sigma = (r_i, d_i)_{i=1}^{t_k}$. If any signer was eliminated during $SP^{\text{Structured}}$ or SP^{Single} , the protocol starts again from Key Aggregation.

Given a multi-signature Σ on a message M correctly generated using **Multi-Sign**, A verifier executing **Multi-Verify** will compute $E_i^* = [r_i] \bar{Y}_{d_i}$ for $i \in [1, t_k]$ as follows:

$$\begin{aligned} E_i^* &= [r_i] \bar{Y}_{d_i} \\ &= \left[\sum_{j \in Q_{t_k}} b_i^{(j)} - c_i a^{(j)} x^{(j)} \right] \bar{Y}_{|d_i|} \\ &= \left[\sum_{j \in Q_{t_k}} b_i^{(j)} - c_i a^{(j)} x^{(j)} \right] \left[\sum_{j \in Q_{t_k}} c_i a^{(j)} x^{(j)} \right] E_0 \\ &= \left[\sum_{j \in Q_{t_k}} b_i^{(j)} \right] E_0 \\ &= E_i \end{aligned}$$

The verifier will then compute $(d_1^*, \dots, d_{t_k}^*) = H_{\text{Sign}}(E_1^* \dots E_{t_k}^* || M)$ and check if $(d_1, \dots, d_{t_k}) = (d_1^*, \dots, d_{t_k}^*)$. As $E_i = E_i^*$ for $i \in [1, t_k]$, $(d_1, \dots, d_{t_k}) = (d_1^*, \dots, d_{t_k}^*)$ holds and **Multi-Verify** outputs 1, which concludes the proof of correctness.

4.3.2 Security

In this Section, we prove the unforgeability of our multi-signature scheme in the actively secure setting, i.e., we consider rogue key attacks with a single honest signer. First, we reiterate the definition of $\mathcal{C}_{k-1}$ -VPwAI for convenience.

Definition 2.18 (Restated, [ABCP22]). $\mathcal{C}_{k-1}$ -**Vectorization Problem with Auxiliary Inputs** ($\mathcal{C}_{k-1}$ -VPwAI): Given an element $E \in \mathcal{E}$ and the pairs $(c_i, [c_i x]E)_{i=1}^{k-1}$, where $\Xi_{k-1} = \{c_0 = 0, c_1 = 1, \dots, c_{k-1}\}$ is an exceptional set, find $x \in \mathbb{Z}_N$.

Theorem 4.2. Assume that there exists a $(t, q_s, q_h, n, \epsilon)$ -forger $\mathcal{F}$ against the multi-signature with parameters (N, k, t_k, Ξ_k) with hash functions $H_{Agg} : \{0, 1\}^* \rightarrow \mathbb{Z}_N$ (where N is a l -bit integer), $H_{Sign} : \{0, 1\}^* \rightarrow (\Xi_k)^{t_k}$ (with $2^{l'} = k^{t_k}$) are modeled as random oracles. Then there exist an algorithm C which (t', ϵ') -solves the $\mathcal{C}_{k-1}$ -VPwAI, letting $n' = n(2n\lambda + 1)\lfloor \frac{k}{n} \rfloor + 2n\lambda + n$, with $t' = 4t + 4n't_{iso} + O(n(q_h + q_s + 1))$ where t_{iso} is the time of an isogeny computation and

$$\epsilon' \geq \frac{\epsilon^4}{q^3} - \frac{4(q^2 - q_s t_k + q_s q) - 1}{2^{l'}} - \frac{1}{2^{l'-1}}.$$

The use of round-robin computations in our multi-signature scheme, where signers will collaborate together to perform computations, makes the use of the Double Forking Lemma insufficient. A malicious user successfully falsifying the result of those computations without being detected while behaving honestly during the rest of the protocol would not be prevented by the Double Forking Lemma. To address this, we adopt a novel, ad-hoc approach and prove our scheme in two steps. First, we show that the commitments and zero-knowledge proofs constrain malicious adversaries to behave like honest-but-curious participants during round-robin computations: they must carry out the correct computations, cannot gain additional information, and cannot cause the protocol to fail without being excluded. Then, we apply the Double Forking technique from [MPSW18], treating the round-robin computations as a black box, to complete the proof of security. Note that as the **Double Forking Lemma** relies on rewinding, which would collapse a quantum state, its use is limited to obtaining security in the Random Oracle Model. Moreover, the reduction is not tight, leading to a weaker bound in the final security theorem (of order ϵ^4).

Notations are adapted as follows. The parameters $params = (\Xi_k, E_0, \mathbb{Z}_N)$ are fixed and the key pair (pk^*, sk^*) are generated for the honest signer with $pk^* = (Y_1^*, \dots, Y_{k-1}^*)$ and $sk^* = x^*$ with $Y_l^* = [c_l x^*]E_0$ for $l \in [1, k-1]$. Without loss of generality, we consider the honest signer as P_1 in the qualified set Q . The target public key is given to the forger $\mathcal{F}$, who may then interact with the honest signer by choosing a message M and a list of public keys L_{pk} such that $pk^* \in L_{pk} = \{pk^*, pk^{(2)}, \dots, pk^{(n)}\}$ and $pk^{(2)}, \dots, pk^{(n)}$ are controlled by the $\mathcal{F}$. At the end, $\mathcal{F}$ outputs a forgery Σ^* on the message M^* with L_{pk}^* the list of public keys and Agg^* the corresponding aggregated public key. We consider $\mathcal{F}$ succeeds if M^* was never queried, $pk^* \in L_{pk}$ and

$$Verif(\Sigma^*, Agg^*, m^*) = 1 \tag{4.4}$$

Round Robin Computations: We now analyze $SP^{Structured}$ and SP^{Single} . These occur $t_k + 1$ times: once during **KeyAgg** to compute the aggregated public key Agg using $SP^{Structured}$, and t_k times for SP^{Single} to compute E_i for $i \in [1, t_k]$. The interaction between $\mathcal{F}$ and the oracle simulating the honest signer proceeds as follows:

1. During **KeyAgg**, the oracle and the forger parse $L_{pk} = pk^*, pk^{(2)}, \dots, pk^{(n)}$ and compute aggregation coefficients $a^{(j)} = H_{Agg}(pk^{(j)} || L_{pk})$. They then execute $SP^{Structured}$, outputting a proof at each computation step. Because the proof targets R'_0 , $\mathcal{F}$ is forced to apply the isogeny to which it committed, but also to reuse the same isogeny as in **KeyGen**. If $\mathcal{F}$ deviates by using a different $x^{(j)}$ or alters the computation, the proof verification fails and the signing oracle outputs $\perp$. As a result, we can assume that the aggregated public key is correctly computed, namely $\bar{Y}_i = \left[\sum_{j \in Q} c_i a^{(j)} x^{(j)} \right] E_0$ for $i \in [1, k-1]$.
2. In the subsequent round-robin phase, the honest signer and the forger run SP^{Single} t_k times to compute elliptic curves $E_i = \left[\sum_{j \in Q} b_i^{(j)} \right] E_0$, where $b_i^{(j)}$ is signer j 's i -th commitment. The oracle simulates b_i^* for the honest signer, while $\mathcal{F}$ selects $(b_i^{(2)}, \dots, b_i^{(n)})$ for each $i \in [1, t_k]$. At each step, $\mathcal{F}$ must prove that $E_i^l = [b_i^l] E_i^{l-1}$ according to relation R_0 . Any deviation leads to proof failure and the oracle outputs $\perp$.
3. In both instances of the round-robin computation, the zero-knowledge property of the proof system guarantees that $\mathcal{F}$ gains no information about the secret key x^* or the commitments b_i^* produced by the oracle.

Double Forking Lemma: In the rest of the proof, we consider the round-robin computations as black-boxes and assume we obtain the expected results. The proof follows the construction in [MPSW18] closely. We begin by building a wrapper algorithm A that runs the forger while simulating the random oracles H_{Agg} and H_{Sign} uniformly; A also emulates the signing oracle by programming H_{Sign} . Algorithm A returns the forged output together with execution traces of the forger unless a designated bad event occurs.

We then use A to define an algorithm B which invokes the forking routine $Fork^A$ (see Section 2.2) targeting the H_{Sign} response associated with the forgery. By doing so B is able to extract a multiset of public keys L_{pk} and recover the corresponding aggregated secret $\bar{x} = \sum_{j \in Q} a^{(j)} x^{(j)} \bmod N$, so that $\bar{Y}_i = [c_i \bar{x}] E_0$. Finally, we convert B into an algorithm C that solves the MT-GAIP and the C_{k-1} -VPwAI problems by running $Fork^B$ on the H_{Agg} response tied to the forgery.

For the remainder of the proof we make the usual simplifying assumptions: $\mathcal{F}$ does not repeat queries and issues only “well-formed” queries, meaning that for any H_{Agg} query the listed L_{pk} contains pk^* and the referenced $pk^{(j)}$ satisfy $j \in Q$. We also assume the adversary makes exactly q_h random-oracle queries and exactly q_s signing queries. To shorten notation, let b_i (respectively r_i) denote the sum of the values $b_i^{(j)}$ (respectively $r_i^{(j)}$) over $j \in Q$. Bold symbols denote full lists, i.e., $\mathbf{a}$ and $\mathbf{E}$ stand for $\mathbf{a} = a^{(1)}, \dots, a^{(n)}$ and $\mathbf{E} = E_1, \dots, E_{t_k}$. The detailed reduction is carried out in the following steps.

Lemma 4.1. *Assume there exists a $(t, q_s, q_h, n, \epsilon)$ -forger $\mathcal{F}$ in the random oracle model against the multi-signature with parameters (N, k, t_k, Ξ_k) and let $q = q_h + q_s + 1$. Then, there exists an algorithm A that takes as input k uniformly random elements $Y_i^{(*)}$ and uniformly random elements $h_{0,1}, \dots, h_{0,q}$ (which are elements from $\mathbb{Z}_N$ used to answer queries to H_{Agg}) and $\mathbf{h}_{1,1}, \dots, \mathbf{h}_{1,q}$ (which are sets of t_k elements from Ξ_k used to answer queries to H_{Sign}) and, with accepting probability at least*

$$\epsilon - \frac{q^2 + q_s t_k + q_s q}{N}$$

outputs $(i_0, i_1, L_{pk}, \mathbf{d}, \mathbf{r}, \mathbf{a})$ where $i_0, i_1 \in \{1, \dots, q\}$, $L_{pk} = \{pk^{(1)}, \dots, pk^{(n)}\}$ is a multiset of public keys such that $pk^ \in L_{pk}$, $\mathbf{a} = (a^{(1)}, \dots, a^{(n)})$ is a tuple of values such that $a^{(j)} = h_{0,i_0}$ for any j such that $pk^{(j)} = pk^*$, and for $i \in [1, t_k]$*

$$r_i = b_i - h_{1,i_1,i} \cdot \bar{x} \bmod N \quad (4.5)$$

Proof. We construct A as follows. It initializes 2 empty tables T_{Agg}, T_{Sign} corresponding to the programmed values of H_{Agg} and H_{Sign} , respectively. It also initializes two counters ctr_0, ctr_1 (initially zero), which will be incremented each time $T(\cdot, pk^*)$ is assigned and each time an assignment is made in T_{Sign} , respectively. Finally, it initializes four flags $Badcom, BadProg, BadOrder$, and $Keycoll$ which will be used to track bad events, picks random coins $\rho_{\mathcal{F}}$ and runs the forger $\mathcal{F}$ on input the public key pk^* , answering its queries as follows.

- Hash Query $H_{Agg}(L_{pk}, pk^*)$: If $T_{Agg}(L_{pk}, pk)$ is undefined, A increments ctr_0 , assigns $T_{Agg}(L_{pk}, pk^*) := h_{0,ctr_0}$, and randomly assigns $T_{Agg}(L_{pk}, pk^{(j)}) \leftarrow \mathbb{Z}_N$ for all $pk^{(j)} \in L_{pk} \setminus \{pk^*\}$. Then it returns $T_{Agg}(L_{pk}, pk)$.
- Hash Query $H_{Sign}(Agg, \mathbf{E}, m)$: If $T_{Sign}(Agg, \mathbf{E}, m)$ is undefined, then A increments ctr_1 and assigns $T_{Sign}(Agg, \mathbf{E}, m) := \mathbf{h}_{1,ctr_1}$. Then it returns $T_{Sign}(Agg, \mathbf{E}, m)$.
- Signature query (L_{pk}, m) : If $pk^* \notin L_{pk}$, A return $\perp$. Otherwise, it parses L_{pk} as $\{pk^*, \dots, pk^{(n)}\}$. If $T_{Agg}(L_{pk}, pk^*)$ is undefined, it makes an internal query to $H_{Agg}(L_{pk}, pk^*)$ which ensures that $T_{Agg}(L_{pk}, pk^{(j)})$ is defined for $j \in \{1, \dots, n\}$, sets $a^{(j)} := T_{Agg}(L_{pk}, pk^{(j)})$ and computes, together with $\mathcal{F}$, $Agg = (\bar{Y}_1, \dots, \bar{Y}_{k-1})$ with $\bar{Y}_i = [c_i \bar{x} \bmod N] E_0$. Then, for $i \in [1, t_k]$, A increments ctr_1 , lets $d_i := h_{1,ctr_1,i}$, draws $r_i^* \leftarrow \mathbb{Z}_N$, and computes $b_i^* = r_i^* + d_i a^* x^* \bmod N$. If for any $i \in [1, t_k]$ b_i^* is already defined, set $Badcom = True$ and halts returning $\perp$. Otherwise, it computes $E_i^* = [b_i^*] E_0$ for $i \in [1, t_k]$
- Upon receiving $(\mathbf{E}^{(2)}, \dots, \mathbf{E}^{(n)})$ from $\mathcal{F}$, it computes, together with $\mathcal{F}$, $\mathbf{E} = (E_1, \dots, E_{t_k})$ with $E_i = [\sum_{j \in Q} b_i^{(j)} \bmod N] E_0$. If $T_{Sign}(Agg, \mathbf{E}, m)$ has already been defined, A sets $BadProg = True$ and halts returning $\perp$. Otherwise, it assigns $\mathbf{d} := T_{Sign}(Agg, \mathbf{E}, m)$, computes $\mathbf{r}^*$ and sends $(\mathbf{r}^*, \mathbf{d})$ to the forger, completing the simulation of the honest signature oracle.

If $\mathcal{F}$ outputs $\perp$, A returns $\perp$ as well. Otherwise, the forger returns a forgery $\Sigma^* = (\mathbf{r}, \mathbf{d})$ on

L_{pk} such that $pk^* \in L_{pk}$ on the message M . Then, A parses $L_{pk} = \{pk^*, pk^{(2)}, \dots, pk^{(n)}\}$ and does the following.

- If $T_{Agg}(L_{pk}, pk^*)$ is undefined, A makes an internal query to $H_{Agg}(L_{pk}, pk)$ ensuring $T_{Agg}(L_{pk}, pk^{(j)})$ is defined for each $j \in Q$ and sets $a^{(j)} := T_{Agg}(L_{pk}, pk^{(j)})$.
- It recomputes $\mathbf{E}$ by computing $E_i = [r_i] \overline{Y_{d_i}}$ for $i \in [1, t_k]$. If T_{Sign} is undefined, A makes an internal query to $H_{Sign}(Agg, \mathbf{E}, M)$ and lets $\mathbf{d}^* := H_{Sign}(Agg, \mathbf{E}, M)$. If $\mathbf{d} \neq \mathbf{d}^*$, the forgery is invalid and A outputs $\perp$. Otherwise, let i_0 be the index such that $T_{Agg}(L_{pk}, pk^*) = h_{0, i_0}$ and i_1 such that $T_{Sign}(Agg, \mathbf{E}, M) := \mathbf{h}_{1, i_1}$. If the assignment $T_{Agg}(L_{pk}, pk^*)$ happened after the assignment $T_{Sign}(Agg, \mathbf{E}, m)$, A sets $BadOrder = True$ and returns $\perp$. If a collusion occurs for the public key list L'_{pk} such that $T_{Agg}(L'_{pk}, pk^{(j)'})$ is defined for each $pk^{(j)' } \in L'_{pk}$ and the aggregated keys corresponding to L_{pk}, L'_{pk} are equal, set $Keycoll = True$ and return $\perp$. Otherwise, A returns $(i_0, i_1, L_{pk}, \mathbf{d}, \mathbf{r}, \mathbf{a})$. By construction, $a^{(j)} = h_{0, i_0}$ for each j such that $pk^{(j)} = pk^*$ and the validity of the signature implies Equation 4.5.

We now bound $acc(A)$. Since $h_{0,1}, \dots, h_{0,q}$ and $\mathbf{h}_{1,1}, \dots, \mathbf{h}_{1,q}$ are sampled uniformly at random, the simulation is indiscernible from $\mathcal{F}$'s perspective unless $BadCom$ or $BadProg$ occurs. Moreover, if $\mathcal{F}$ outputs a forgery, A produces a non- $\perp$ result unless either $BadOrder$ or $KeyColl$ is set. Consequently, $acc(A) \geq \epsilon - \Pr[Bad]$ where Bad denotes the event that any of the four flags is triggered.

We start with $BadCom$. Since b_i is determined only at signing time, there are at most q_s possible assignments of b_i . For each $i \in [1, t_k]$, the value r_i is chosen at random, so $b_i = r_i + d_i ax^* \bmod N$ is uniformly distributed over at least N possibilities. Hence, the probability that $BadCom$ occurs is at most $q_s t_k / N$.

Next, consider $BadProg$. We analyze the probability that it occurs, conditioned on $BadCom$ not being triggered. Let $\tilde{A}$ denote the event $BadCom$ and $\tilde{B}$ the event $BadProg$. If $\tilde{A}$ never occurs, then at the time the forger outputs $\mathbf{E}^{(2)}, \dots, \mathbf{E}^{(n)}$, its view is independent of $\mathbf{E}^*$. Thus, for each i , $E_i = \left[\sum_{j \in Q} b_i^{(j)} \bmod N \right] E_0$ is uniformly distributed over N values. Since there are at most q total assignments to T_{Sign} , the probability that $\mathbf{E}$ coincides with a pre-defined entry is at most q/N . Therefore, $\Pr[\tilde{B} \mid \tilde{A}] \leq q_s q / N$.

We then consider $BadOrder$. Following [MPSW18], we rely on the requirement that the aggregated public key Agg is included in the H_{Sign} call for the subsequent argument to hold. Let $T_{Agg}(L'_{pk} := a' \mid pk' \in L'_{pk})$ denote the set of assignments corresponding to a multiset L'_{pk} . There are at most q such sets of T_{Agg} assignments, and each contains a unique mapping of the form $T_{Agg}(L_{pk}, pk^*) := h_{0, i}$ for some $i \in [1, q]$. For $BadOrder$ to be set to true, one such assignment must produce an aggregated public key that coincides with the first argument of a previously defined entry in T_{Sign} . Considering the i -set of assignment, for each $l \in [1, t_k]$, we have

$$\begin{aligned}\overline{Y}_l' &= \left[d_l \left(n^* h_{0,i} x^* + \sum_{j \in Q, x^{(j)} \neq x^*} a^{(j)} x^{(j)} \right) \mod N \right] \\ &= [n^* d_l h_{0,i} x^*] Z\end{aligned}$$

where n^* is the number of times x^* appears in L'_{pk} and $h_{0,i}$ is uniformly random in $\mathbb{Z}_N$ and independent from Z accounting for secret keys different from x^* . Hence, Agg' is uniformly random (with respect to $h_{0,i}$) in a set of at least N elements. Since there are at most q entries in T_{Sign} and at most q sets of T_{Agg} assignments, $BadOrder$ is set to true with probability at most q^2/N .

We now bound the probability that $KeyColl$ is set to *True*. This corresponds to the event where two distinct sets of T_{Agg} assignments, associated with different multisets L'_{pk} and L''_{pk} (not necessarily those appearing in the forgery), yield identical aggregated public keys Agg' and Agg'' . Since each aggregated public key is uniformly distributed over a set of at least N values (determined by $h_{0,i}$) and independent of the others, the probability of such a collision is at most q^2/N .

Combining all of the above, we obtain

$$\begin{aligned}acc(A) &\leq \epsilon - \frac{q_s t_k}{N} - \frac{q_s q}{N} - \frac{q^2}{N} - \frac{q^2}{N} \\ &\leq \epsilon - \frac{q^2 + q_s t_k + q_s q}{N} \\ &= \epsilon - \frac{q^2 + q_s t_k + q_s q}{2^l}\end{aligned}$$

Using A , we now construct an algorithm B which returns a multiset of public keys L_{pk} together with the aggregated secret key.

Lemma 4.2. *Assume there exists a $(t, q_s, q_h, n, \epsilon)$ -forger $\mathcal{F}$ in the random oracle model against the multi-signature with parameters (N, k, t_k, Ξ_k) and let $q = q_h + q_s + 1$. Then, there exists an algorithm B that takes as input k uniformly random elements $Y_i^{(*)}$ and uniformly random elements $h_{0,1}, \dots, h_{0,q}$ and, with accepting probability at least*

$$\frac{\epsilon^2}{q} - \frac{q^2 - q_s t_k + q_s q}{2^l} - \frac{1}{2^{l'}}$$

outputs a tuple $(i_0, L_{pk}, \mathbf{a}, \overline{x})$ where $i_0 \in \{1, \dots, q\}$, $L_{pk} = \{pk^{(1)}, \dots, pk^{(n)}\}$ is a multiset of public keys such that $pk^ \in L_{pk}$, $\mathbf{a} = (a^{(1)}, \dots, a^{(n)})$ is a tuple of values such that $a^{(j)} = h_{0,i_0}$ for any j such that $pk^{(j)} = pk^*$, and $\overline{x}$ is the aggregated secret key corresponding to the aggregated public key, i.e. $\overline{Y}_i = [c_i \overline{x}] E_0$.*

Proof. Algorithm B runs $Fork^A$ with A as defined in Lemma 4.1 and takes a few additional steps described below.

- pk^* and $h_{0,1}, \dots, h_{0,q}$ play the role of *inp*,

- $\mathbf{h}_{1,1}, \dots, \mathbf{h}_{1,q}$ play the role of $h_1, \dots, h_q$,
- i_1 plays the role of i ,
- $(i_0, L_{pk}, \mathbf{r}, \mathbf{d}, \mathbf{a})$ plays the role of *out*.

B picks random coins ρ_A and runs algorithm A on coins ρ_A, pk^* and strings $h_{0,1}, \dots, h_{0,q}$ and $\mathbf{h}_{1,1}, \dots, \mathbf{h}_{1,q}$ where $\mathbf{h}_{1,1}, \dots, \mathbf{h}_{1,q}$ are drawn uniformly at random by B (whereas $h_{0,1}, \dots, h_{0,q}$ are part of the input and will be the same in both runs of A). If A returns $\perp$, B returns $\perp$ as well. Otherwise, A returns a tuple $(i_0, i_1, L_{pk}, \mathbf{r}, \mathbf{d}, \mathbf{a})$. Then, B runs A again with the same random coins on input $pk^*, h_{0,1}, \dots, h_{0,q}$, and $\mathbf{h}_{1,1}, \dots, \mathbf{h}_{1,i_1-1}, \mathbf{h}'_{1,i_1}, \dots, \mathbf{h}'_{1,q}$ where $(\mathbf{h}'_{1,i_1}, \dots, \mathbf{h}'_{1,q})$ are uniformly random. If A returns $\perp$ in the second run, B returns $\perp$ as well. If A returns another tuple $(i'_0, i'_1, L'_{pk}, \mathbf{r}', \mathbf{d}', \mathbf{a}')$, B proceeds as follow. For $i \in [1, t_k]$, let $r_i = b_i - d_i \bar{x} \bmod N$ and $r'_i = b'_i - d'_i \bar{x}' \bmod N$. If $i_1 \neq i'_1$, or $i_1 = i'_1$ and $\mathbf{h}_{1,i_1} = \mathbf{h}'_{1,i_1}$, then B returns $\perp$. Otherwise, if $i_1 = i'_1$ and $\mathbf{h}_{1,i_1} \neq \mathbf{h}'_{1,i_1}$ then we will prove that necessarily

$$i_0 = i'_0, L_{pk} = L'_{pk}, \mathbf{E} = \mathbf{E}', \mathbf{a} = \mathbf{a}' \quad (4.6)$$

which in particular implies $b_i = b'_i$ for $i \in [1, t_k]$.

By Lemma 4.1, the two outputs returned by A are such that

$$\begin{aligned} r_i &= b_i - d_i \bar{x} \bmod N \\ r'_i &= b_i - d'_i \bar{x}' \bmod N \end{aligned}$$

which allows B to compute the aggregated secret key $\bar{x}$ as

$$\bar{x} = (r_i - r'_i)(h_{1,i_1,i} - h'_{1,i_1,i})^{-1} \bmod N$$

Then B returns $(i_0, L_{pk}, \mathbf{a}, \bar{x})$.

B return a non- $\perp$ output iff $For k^A$ does, so that by Lemma 2.1, Lemma 4.1, and letting $\alpha = \frac{q^2 - q_s t_k + q_s q}{N}$ the accepting probability of B is at least

$$\begin{aligned} acc(A) \left(\frac{acc(A)}{q} - \frac{1}{2^l} \right) &\geq \frac{(\epsilon - \alpha)^2}{q} - \frac{(\epsilon - \alpha)}{2^{l'}} \\ &\geq \frac{\epsilon^2}{q} - 2\alpha - \frac{1}{2^{l'}} \\ &= \frac{\epsilon^2}{q} - \frac{q^2 - q_s t_k + q_s q}{2^l} - \frac{1}{2^{l'}} \end{aligned}$$

We now turn to the proof of Equation 4.6. During A 's first execution, $\mathbf{h}_{1,i_1}$ is assigned to $T_{Sign}(Agg, \mathbf{E}, m)$, while in the second, $\mathbf{h}'_{1,i_1}$ is assigned to $T_{Sign}(Agg', \mathbf{E}', M')$. These assignments may arise either from a direct query of the forger to H_{Sign} , from a signature request, or from

the final verification step validating the forgery. Up to these two assignments, both executions are identical: A runs $\mathcal{F}$ with the same random coins and inputs, employs identical values $h_{0,1}, \dots, h_{0,q}$ for $T_{\text{Agg}}(\cdot, pk^*)$ assignments, the same $\mathbf{h}_{1,1}, \dots, \mathbf{h}_{1,i_1-1}$ for T_{Sign} , and identical random coins for $T_{\text{Agg}}(\cdot, pk^{(j)} \neq pk^*)$ as well as for the random draws of $\mathbf{r}^*$ during signing queries. Since the two executions are indistinguishable up to the assignments $T_{\text{Sign}}(\text{Agg}, \mathbf{E}, M) := \mathbf{h}_{1,i_1}$ and $T_{\text{Sign}}(\text{Agg}', \mathbf{E}', M') := \mathbf{h}'_{1,i_1}$, their arguments must coincide. This implies in particular that $\mathbf{E} = \mathbf{E}'$ and $\text{Agg} = \text{Agg}'$. Because $\text{Agg} = \text{Agg}'$, assuming $L_{pk} \neq L'_{pk}$ would contradict the fact that KeyColl is not set to True . Given $\mathbf{E} = \mathbf{E}'$, it follows that $b_i = b'_i$. Furthermore, since BadOrder is not triggered, the assignments h_{0,i_0} and h_{0,i'_0} must both have occurred before the Fork, yielding $i_0 = i'_0$ and $\mathbf{a} = \mathbf{a}'$. We now proceed to establish Theorem 4.2, restated here for clarity, by constructing from B an algorithm C that solves the $\mathbf{C}_{k-1}$ -VPwAI problem. $\blacksquare$

Theorem 4.2 (Restated). *Assume that there exists a $(t, q_s, q_h, n, \epsilon)$ -forger $\mathcal{F}$ against the multi-signature with parameters (N, k, t_k, Ξ_k) with hash functions $H_{\text{agg}} : \{0, 1\}^* \rightarrow \mathbb{Z}_N$ (where N is a l -bit integer), $H_{\text{sign}} : \{0, 1\}^* \rightarrow (\Xi_k)^{t_k}$ (with $2^{l'} = k^{t_k}$) are modeled as random oracles. Then there exist an algorithm C which (t', ϵ') -solves the $\mathbf{C}_{k-1}$ -VPwAI, letting $n' = n(2n\lambda + 1)\lfloor \frac{k}{n} \rfloor + 2n\lambda + n$, with $t' = 4t + 4n't_{\text{iso}} + O(n(q_h + q_s + 1))$ where t_{iso} is the time of an isogeny computation and*

$$\epsilon' \geq \frac{\epsilon^4}{q^3} - \frac{4(q^2 - q_s t_k + q_s q) - 1}{2^l} - \frac{1}{2^{l'-1}}.$$

Proof. Algorithm C runs Fork^B with B as defined in Lemma 4.2 and takes a few additional steps described below.

- pk^* play the role of inp ,
- $h_{0,1}, \dots, h_{0,q}$ play the role of $h_1, \dots, h_q$,
- i_0 plays the role of i ,
- $(L_{pk}, \mathbf{a}, \bar{x})$ plays the role of out .

C picks random coins ρ_B and runs algorithm B twice, once of random coins $h_{0,1}, \dots, h_{0,q}$ and once on $h_{0,1}, \dots, h_{0,i_0-1}, h'_{0,i_0}, \dots, h'_{0,q}$. If B does not return $\perp$ in the two runs, we obtain two tuples $(i_0, L_{pk}, \mathbf{a}, \bar{x})$ and $(i'_0, L'_{pk}, \mathbf{a}', \bar{x}')$. Let n^* be the number of times pk^* appears in L_{pk} . If $i_0 \neq i'_0$ or $i_0 = i'_0$ and $h_{0,i_0} \neq h'_{0,i_0}$, return $\perp$. Otherwise, we will prove that if $i_0 = i'_0$ and $h_{0,i_0} \neq h'_{0,i_0}$ then necessarily

$$L_{pk} = L'_{pk} \text{ and } a^j = a'^j \text{ for } j \text{ such that } x^{(j)} \neq x^* \quad (4.7)$$

We have

$$\begin{aligned}\bar{x} &= \sum_{j \in Q} a^{(j)} x^{(j)} \bmod N = h_{0,i_0} x^* + \sum_{j \in Q \setminus \{1\}} a^{(j)} x^{(j)} \bmod N \\ \bar{x}' &= \sum_{j \in Q} a^{(j)'} x^{(j)} \bmod N = h'_{0,i_0} x^* + \sum_{j \in Q \setminus \{1\}} a^{(j)} x^{(j)} \bmod N\end{aligned}$$

Thus, C can compute x which is a solution to the C_{k-1} -VPwAI problem as

$$x^* = (\bar{x} - \bar{x}')(n^*)^{-1}(h_{0,i_0} - h'_{0,i_0})^{-1} \bmod N$$

Neglecting the time to compute $\bar{x}$ and x^* , C takes twice the running time of B , which itself is twice the running time of A . The running time of A is the running time of $\mathcal{F}$ plus the time needed to maintain tables T_{Agg}, T_{Sign} , which are of sizes at most qN and q . Answering signatures queries is dominated by the time needed to compute Agg and $\mathbf{E}$ which is at most $n' t_{iso}$ with $n' = n(2n\lambda + 1)\lfloor \frac{k}{n} \rfloor + 2n\lambda + n$. Therefore, $t' = 4t + 4n' t_{iso} + O(n(q_h + q_s + 1))$.

C is successful iff $ForK^B$ returns a non- $\perp$ answer. By Lemma 2.1, Lemma 4.2 and letting $\beta = \frac{q^2 - q_s t_k + q_s q}{2^l} - \frac{1}{2^{l'}}$, the success probability ϵ' of C is at least

$$\begin{aligned}\epsilon' &\geq acc(B) \left(\frac{acc(B)}{q} - \frac{1}{2^l} \right) \\ &\geq \frac{\epsilon^4}{q^3} - 2\beta - \frac{1}{2^l} \\ &= \frac{\epsilon^4}{q^3} - \frac{4(q^2 - q_s t_k + q_s q) - 1}{2^l} - \frac{1}{2^{l'-1}}\end{aligned}$$

We next justify the equalities given in Equation 4.7. In the two runs of A within the first execution of B , h_{0,i_0} is assigned to $T_{Agg}(L_{pk}, pk^*)$, whereas in the two runs of A during the second execution of B , h'_{0,i_0} is assigned to $T_{Agg}(L'_{pk}, pk^*)$. These assignments may occur either via a forger query to H_{Agg} , during a signing query, or at the final verification stage. Up to these assignments, the four runs of $\mathcal{F}$ are perfectly synchronized: A executes $\mathcal{F}$ using identical random coins and inputs, reuses the same values $h_{0,1}, \dots, h_{0,i_0-1}$ for $T_{Agg}(\cdot, pk^*)$, the same $\mathbf{h}_{1,1}, \dots, \mathbf{h}_{1,q}$ for T_{Sign} , and identical assignments for $T_{Agg}(\cdot, pk \neq pk^*)$ and $\mathbf{r}^*$ in the signing phase. As the executions differ only in the $T_{Agg}(\cdot, pk^*)$ assignments, the arguments must coincide, implying $L_{pk} = L'_{pk}$. All remaining values are sampled uniformly by A using the same random coins in all four runs, ensuring $a^j = a^{j'}$ for every j such that $pk^{(j)} \neq pk^*$, which completes the proof. $\blacksquare$

4.4 Comparison

In this section, we compare our proposed multi-signature scheme to two lattice-based multi-signature protocols, MuSig-L [BTT22] and DualMS [Che23], as well as the isogeny-based threshold signature ThreshER SharK, which is considered the most efficient isogeny-based threshold signature.

4.4.1 Multi-Signature Comparison

MuSig-L [BTT22] and DualMS [Che23] are two-round multi-signatures based on Dilithium-G and proven secure in the Random Oracle Model. Table 4.2 presents the comparison of public key and signature sizes for the two lattice-based schemes and our protocol (Multi-I), using two parameter sets ($k = 2^4, 2^{12}$). Here, $|sk|$, $|pk|$, and $|sig|$ refer to the estimated secret key size, public key size, and signature size, all measured in kB. Parameters are taken from [Che23] (Table 2). The parameter k represents the number of elliptic curves in the public key, as defined in Section 4.2, and we consider a 128-bit security level with $n = 32$ and $n = 1024$ signers.

Table 4.2: Comparison between Multi-I ($k = 2^4, 2^{12}$), MuSig-L [BTT22] and DualMS [Che23] (size in kB)

Params	MuSig-L	DualMS	Multi-I ($k = 2^4$)	Multi-I ($k = 2^{12}$)
$ sk $ ($n = 32$)	104	36.9	0.016	0.016
$ pk $	26	6.94	1	256
$ sig $	98.47	19.72	0.791	0.306
$ sk $ ($n = 1048$)	3,325	1536	0.016	0.016
$ pk $	32.5	10.5	1	256
$ sig $	106.53	30.91	0.791	0.306
Security Assumption	MSIS, MLWE	MSIS, MLWE	MT-GAIP, C_{k-1} -VPwAI	MT-GAIP, C_{k-1} -VPwAI

More specifically, in our protocol, both the public key and signature sizes depend on the parameters k and t_k , which are related by $t_k = \lambda \log_2(k)$. Here, k denotes the number of curves included in the public key, while t_k is used during signing. Selecting $k = 2^4$ (corresponding to $t_k = 23$) provides a balanced trade-off between public key and signature sizes, whereas $k = 2^{12}$ ($t_k = 9$) minimizes the signature size at the cost of a larger public key. Since the public key grows faster than the signature shrinks, setting $k = 2^{12}$ yields a signature approximately $64\times$ and $321\times$ smaller than those of DualMS and MuSig-L, respectively, but increases the public key size by roughly $36\times$ and $9\times$ when $n = 32$. In contrast, choosing $k = 2^4$ offers a more balanced compromise, reducing the public key size by $7\times$ and $26\times$ and the signature by $25\times$ and $125\times$, relative to DualMS and MuSig-L, respectively. Another advantage of our multi-signature scheme is that the sizes of both the public key and signature remain independent of the number of signers. However, we note that both DualMS and MuSig-L are two-round protocols that do not require a round-robin procedure during Signing or Key Aggregation, with

MuSig-L even allowing the first round to be executed offline. In contrast, our multi-signature relies on a round-robin process, making it effectively a $2n + 1$ -round protocol.

4.4.2 Our Protocol and ThreshER SharK

We also compare our multi-signature to the state-of-the-art isogeny-based threshold signature protocol ThreshER SharK. In a threshold signature, the secret shares are revealed during signing, preventing their reuse. Consequently, the signers must execute the full protocol each time they wish to generate a signature. In contrast, in a multi-signature, participants' secret keys are never exposed, allowing them to be reused. Since the aggregated public key depends on all signers' key pairs, reusing the same key pairs ensures that the aggregated public key remains unchanged. This enables a fixed group of signers to generate multiple signatures without rerunning Key Aggregation, a significant advantage for applications such as multi-signature wallets or federated signing systems.

We compare our protocol (Multi-I) to the full threshold ThreshER SharK protocol, both with and without **KeyAgg**, in Table 4.3, considering the number of isogeny computations for the same parameters. Here, n denotes the group size, k the number of curves in the public key, and λ the security parameter. Since the computation cost is dominated by isogeny evaluations, we only compare the number of isogenies.

Table 4.3: Comparison between Multi-I and ThreshER SharK [ABCP23b]
(in terms of isogeny computations)

	ThreshER SharK	Multi-I
inc. KeyAgg	$n[t_k(2\lambda + 1) + (n\lambda + 1)\lfloor \frac{k}{n} \rfloor]$	$n[t_k(2\lambda + 1) + (2n\lambda + 1)\lfloor \frac{k}{n} \rfloor]$
exc. KeyAgg	$n[t_k(2\lambda + 1) + (n\lambda + 1)\lfloor \frac{k}{n} \rfloor]$	$nt_k(2\lambda + 1)$
Properties	Robustness	

When including **KeyAgg**, our protocol requires more isogeny computations overall. This overhead arises because each signer must prove that the secret key used during **KeyAgg** matches the key used in **KeyGen**, adding approximately $(n\lambda)\lfloor \frac{k}{n} \rfloor$ isogenies. Given that isogeny computations dominate runtime, this means that our protocol is slower than ThreshER SharK when both protocols are executed fully. Moreover, the lack of secret shares removes the robustness feature of ThreshER SharK, as honest signers are unable to continue the protocol if any malicious signer is eliminated.

However, our protocol outperforms ThreshER SharK when the same group of signers needs to produce multiple signatures. By reusing the aggregated public key, the costly computations associated with the Structured DKG in ThreshER SharK and $PS^{Structured}$ in our protocol are avoided. Skipping **KeyAgg** significantly reduces computational overhead, making our multi-signature protocol substantially faster in these scenarios.

4.5 Implementation

In this Section, we share the results of our implementation. The scheme was developed in C, extending the CSI-FiSh repository available on GitHub [Beu19]. We first integrated CSI-SharK into this framework and then incorporated our own protocol. Experiments were performed on a Kali virtual machine configured with 10 GB of RAM and 8 cores, hosted on a laptop powered by an AMD Ryzen 7 5800H with Radeon Graphics at 3.20 GHz. The computation times (in seconds) summarized in Table 4.4 should be interpreted mainly as indicative, since the implementation is not optimized. The boolean flag ZK indicates whether the proof systems are enabled ($ZK = 1$) or disabled ($ZK = 0$), while n denotes the number of signers. As in CSI-SharK, additional optimization, such as multi-thread parallelization and minimizing idle phases, should substantially improve efficiency. All experimental data correspond to parameters $k = 2^4$, $t_k = 23$, and $h = 13$, where k and t_k are defined in Section 4.2, and h represents the slow factor of the slow hash function introduced in CSI-FiSh. This configuration, consistent with the balanced setting discussed in Section 4.4, yields a 1 KB public key and a 0.791 KB signature.

Table 4.4: Multi-I benchmark results for $n = 2, 3, 4$ with ($ZK = 0$) and without ($ZK = 1$) proof systems (in seconds)

n	KeyAgg ($ZK = 0$)	KeyAgg ($ZK = 1$)	Multi-Sign ($ZK = 0$)	Multi-Sign ($ZK = 1$)	Multi-Verify
2	0.650	686.1	1.902	532.1	0.946
3	1.988	1657	2.858	1156	0.953
4	2.608	3371	3.817	2257	0.926

Both **KeyAgg** and **Multi-Sign** exhibit roughly linear growth in computation time as the number of signers increases when no proof systems are used. However, incorporating proof systems causes a significant rise in cost. For each execution of the Prover side in the structured proof system, performed n times in total, the Verifier side runs $n - 1$ times. Likewise, for the single proof system, the Prover executes $n * t_k$ times and the Verifier $n - 1$ times. In contrast, **Multi-Verify** requires the same computation time in both settings and remains unaffected by the number of participants.

4.6 Conclusion

We propose a multi-signature scheme by extending the isogeny-based signature CSI-SharK. Our scheme incorporates Key Aggregation, enabling the multi-signature to be verified against the aggregated public key using the same verification algorithm as in the base signature scheme. Both the aggregated public key and the multi-signature are computed via a sub-protocol, which allows the signers to perform the necessary computations in a round-robin manner. At each step, the signers execute a non-interactive zero-knowledge proof system to ensure correct computations and guarantee active security. We prove that our scheme is correct and actively EUF-CMA using the Double Forking Technique. We compare its memory requirements to two state-of-

the-art lattice-based multi-signatures, showing that, with appropriate parameter choices, our multi-signature is significantly more compact. We further compare the computational cost to the most efficient isogeny-based threshold signature, ThreshER SharK. While our protocol lacks the robustness of ThreshER SharK and is slower when the full protocol is executed, it becomes advantageous in scenarios where the same group of signers repeatedly participates in a multi-party signing process, as the key aggregation step can be skipped. Finally, we implement our multi-signature scheme in C, using the CSI-FiSh repository as a foundation.

Chapter 5

Isogeny-based Proxy Signature

In isogeny-based cryptography, only a single scheme, CSI-PS, has been proposed based on the signature SeaSign. Due to the limitations of its base signature which uses rejection sampling, it suffers from relatively slow signing and key generation, as well as large key and signature size. Moreover, CSI-PS has only been proven secure in the EUF-CMA model which is not adapted to proxy signatures. A generic construction has also been proposed which can be used to obtain a proxy signature scheme based on any digital signatures. Unlike most proxy signatures, however, the resulting scheme does not bind the certificate with the proxy signature, which leads to a potential mix-and-match issue. Additionally, while both schemes mention the usual security properties of proxy signatures, such as prevention of misuse and verifiability, it does not provide a formal definition.

In this chapter, we propose the first formal definitions of security properties of proxy signatures. We identify a new attack on schemes that do not bind certificates and proxy signatures, showing that it weakens the notions of verifiability and undeniability, before proposing a new security property to prevent this issue. We present our proxy signature scheme based on the CSI-SharK signature scheme, which comes in two versions: a modified instantiation of the generic construction, which binds the certificate and proxy signature, in the EUF-CMA-CWA model and the first isogeny-based proxy signature in the EUF-CMA-PKE model. We show that both versions are unforgeable in the Quantum Random Oracle Model by arguing the unforgeability of CSI-SharK lifts to the proxy signature.

5.1 Properties of Proxy Signature

5.1.1 Formal Definition of Security Properties

In this section, we propose formal definitions for the properties of proxy signatures: Identifiability, Undeniability, Verifiability and Prevention of misuse. We first start by reiterating the notions as defined in Section 2.2 before proposing a security game and formal definition.

Verifiability: A verifier can be convinced based on the proxy signature of the original's signer agreement on the signed message.

The Verifiability notion defined above captures two components, verification succeeds if the certificate and the warrant (representing the original signer's agreement) are emitted by an honest original signer, but also that a certificate not emitted by the original signer will have its verification fail. As the first notion is part of the correctness itself, we focus on the second component and define the Verifiability Game as follows:

$Game_{\Sigma}^{Verif}$: Played by the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$. $\mathcal{C}$ will maintain a list of certificates emitted $CertList$.

- *Setup*: Challenger $\mathcal{C}$ runs **KeyGen** twice, generating both the original signer's and the proxy signer's key pair: $(pk^o, sk^o, pk^p, sk^p) \leftarrow \text{KeyGen}()$. $\mathcal{C}$ then sends both public keys (pk^o, pk^p) to $\mathcal{A}$.
- *Delegate Query*: On input $(pk^{p'}, W)$, $\mathcal{C}$ outputs a certificate $\sigma^o \leftarrow \text{Delegate}(sk^o, W, pk^{p'})$, sends σ^o to $\mathcal{A}$ and adds $(W, \sigma^o, pk^{p'})$ it to $CertList$.
- *Proxy Signing Queries*: On input (m, w, pk^p) , $\mathcal{C}$ checks $(W, \sigma^o) \in CertList$. If it is the case it runs $\sigma^p \leftarrow \text{P-Sign}(sk^p, \sigma^o, W, M)$ and sends it to $\mathcal{A}$.
- *Win Condition*: $\mathcal{A}$ outputs $(\sigma^{o*}, W, \sigma^{p*}, M)$. $\mathcal{A}$ wins if

$$\text{P-Verify}(pk^o, pk^{p'}, W^*, \sigma^{o*}, M^*, \sigma^{p*}) = 1$$

such that $(W, \sigma^o, pk^{p'}) \notin CertList$. Let $Adv_{\Sigma}^{verif}(\mathcal{A}) = Pr[\mathcal{A} \text{ wins}]$ be the advantage of $\mathcal{A}$.

Definition 5.1. Verifiability A proxy-signature is verifiable if the advantage $Adv_{\Sigma}^{verif}(\mathcal{A}) \leq \text{negl}(\lambda)$ for all PPT $\mathcal{A}$.

Strong Identifiability: Anyone can determine the identity of the proxy signer from the proxy signature.

Similarly, Strong Identifiability captures the notion of honest identification succeeding, which is encompassed in the correctness, but also the notion of no hiding or misbinding, meaning a proxy signature with no identities or wrong identities can be verified successfully. To do so, we assume the existence of a deterministic extractor algorithm ID that outputs the identity of both signers (pk^o, pk^p) . In practice, the identity of both signers can be recovered from the warrant. We define the Identifiability game as follows:

$Game_{\Sigma}^{ID}$: Played by the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$.

- The Setup, Delegate Queries and Proxy Signing queries are identical to $Game_{\Sigma}^{Verif}$.
- *Win Condition*: $\mathcal{A}$ outputs $(\sigma^{o*}, W, \sigma^{p*}, M)$. $\mathcal{A}$ wins if

$$\text{P-Verify}(pk^o, pk^p, W^*, \sigma^{o*}, M^*, \sigma^{p*}) = 1$$

such that $ID(W, \sigma^o, M, \sigma^p) = \{\perp\}$ or $ID(W, \sigma^o, M, \sigma^p) = (pk^{o'}, pk^{p'})$ such that $pk^{o'} \neq pk^o$ or $(W, \sigma^o, pk^{p'}) \notin CertList$.

Let $Adv_{\Sigma}^{ID}(\mathcal{A}) = Pr[\mathcal{A} \text{ wins}]$ be the advantage of $\mathcal{A}$.

Definition 5.2. Strong Identifiability *A proxy-signature satisfies strong identifiability if the advantage $Adv_{\Sigma}^{ID}(\mathcal{A}) \leq \text{negl}(\lambda)$ for all PPT $\mathcal{A}$.*

Prevention of Misuse: The proxy signature cannot be used for something other than its original purpose.

As stated previously, we assume the existence of a deterministic extractor algorithm $Cond$ that outputs the condition of use of the proxy signature. In practice, $Cond$ takes as input the warrant W and outputs the message space $\mathcal{M}$ and the validity period $\mathcal{V}$.

$Game_{\Sigma}^{Misuse}$: Played by the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$.

- The Setup, Delegate Queries and Proxy Signing queries are identical to $Game_{\Sigma}^{Verif}$.
- *Win Condition:* $\mathcal{A}$ outputs $(\sigma^{o*}, W, \sigma^{p*}, M)$. $\mathcal{A}$ wins if

$$\text{P-Verify}(pk^o, pk^p, W^*, \sigma^{*o}, M^*, \sigma^{*p}) = 1$$

such that $Cond(W, \sigma^o, M, \sigma^p) = \{\perp\}$ or $Cond(W, \sigma^o, M, \sigma^p) = (\mathcal{M}, \mathcal{V})$ such that $M \notin \mathcal{M}$ or $date \notin \mathcal{V}$.

Let $Adv_{\Sigma}^{Misuse}(\mathcal{A}) = Pr[\mathcal{A} \text{ wins}]$ be the advantage of $\mathcal{A}$.

Definition 5.3. Prevention of Misuse *A proxy-signature satisfies prevention of misuse if the advantage $Adv_{\Sigma}^{Misuse}(\mathcal{A}) \leq \text{negl}(\lambda)$ for all PPT $\mathcal{A}$.*

Strong Undeniability: After a proxy signer has generated a proxy signature for the original signer, the proxy signer cannot repudiate their signature creation against anyone.

Strong undeniability is the most complex to define. We need to assume the existence of two deterministic algorithms, $VerifyDenyProxy$ and $VerifyDenyDelegate$, which would take as input W, pk^o, pk^p , some proof statement π and the respective signature σ^o in the case of $VerifyDenyDelegate$, σ^p in the case of $VerifyDenyProxy$, and output 1 if the proof succeeds, 0 if it does not. Those two functions then output accept if they accept the denial, reject otherwise. The game is then as follows:

$Game_{\Sigma}^{Und}$: Played by the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$.

- *Setup:* Challenger $\mathcal{C}$ runs **KeyGen** twice, generating both the original signer's and the proxy signer's key pair: $(pk^o, sk^o, pk^p, sk^p) \leftarrow \text{KeyGen}()$. The adversary either plays the role of the original signer or the proxy signer, and receives one of the secret key sk^p or sk^o , as well as both public keys (pk^o, sk^o, pk^p) .
- *Queries:* If $\mathcal{A}$ is the original signer, he has access to Proxy Signing Queries as detailed previously. If he plays the role of the proxy signer, he has access to Delegate queries.
- *Win Condition:* $\mathcal{A}$ outputs $(\sigma^{o*}, W, \sigma^{p*}, M)$, as well as a proof of denial π . $\mathcal{A}$ wins if $\text{P-Verify}(pk^o, pk^p, W^*, \sigma^{*o}, M^*, \sigma^{*p}) = 1$ with $VerifyDenyProxy(\pi) = \text{accept}$ if he plays the role of the proxy signer or $VerifyDenyDelegate(\pi)$ if he plays the role of the original signer.

Let $\text{Adv}_{\Sigma}^{\text{Und}}(\mathcal{A}) = \Pr[\mathcal{A} \text{ wins}]$ be the advantage of $\mathcal{A}$.

Definition 5.4. Strong Undeniability *A proxy-signature satisfies Strong Undeniability if the advantage $\text{Adv}_{\Sigma}^{\text{Und}}(\mathcal{A}) \leq \text{negl}(\lambda)$ for all PPT $\mathcal{A}$.*

The issue is in how to modelize the algorithms *VerifyDenyProxy* and *VerifyDenyDelegate* and the denial proof itself. For simplicity, we will consider them as checking the public key of the original signer or proxy signer and rejecting if it corresponds, accepting otherwise. As a verifier can be convinced of the identity of both signers, neither the original signer nor the proxy signer can deny having output the signature, making strong undeniability similar to identifiability.

An alternative is to use the concept of undeniable signatures as defined in [CA89] and define explicitly a confirmation and denial protocol in an interactive zero-knowledge way. However, we choose to take the first approach and do not develop it in this thesis.

5.1.2 Certificate and Signature Binding

An interesting point to note is that none of the properties strictly requires the certificate and the proxy signature to be bound together, which is the approach taken in the generic construction [BPW03]. While it is possible for a scheme to satisfy those properties without doing so, the notions of verifiability and undeniability become weakened. To illustrate this point, we consider the following example. Let there be an original signer and a proxy signer, collaborating to generate a proxy signature. However, instead of generating a single certificate, the original signer emits two certificates and warrants pairs (W, σ^o) and $(W', \sigma^{o'})$ with overlapping validity period and message space, and only sends (W, σ^o) to the proxy signer. After the proxy signer has produced the proxy signature, the original signer could then mismatch the certificates and argue that the proxy signature was emitted by using $(W', \sigma^{o'})$ instead. As the proxy signature was generated by the proxy signer, on a valid certificate and warrant, the verification will succeed, and the proxy signer would be unable to deny it. Additionally, verifiability is weakened, as while the verifier can be convinced of the original signer and proxy signer agreement on the signed message, the verifier cannot identify under which condition the certificate was exercised. In order to prevent this attack, we propose a new property, Certificate-Signature Binding which we define as follows:

Definition 5.5. *A proxy signature satisfies Certificate-Signature Binding only if it is cryptographically bound to a unique warrant (and corresponding certificate). That is, for every valid proxy signature, there exists exactly one warrant–certificate pair such that the verifier can be convinced the signature was generated under that warrant.*

We define the Binding Game as follows:

$\text{Game}_{\Sigma}^{\text{Binding}}$: Played by the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$.

- The Setup, Delegate Queries and Proxy Signing queries are identical to $\text{Game}_{\Sigma}^{\text{Verif}}$.

- *Win Condition:* $\mathcal{A}$ outputs $(\sigma^{o*}, W^*, \sigma^{p*}, M)$. $\mathcal{A}$ wins if

$$\text{P-Verify}(pk^o, pk^p, W^*, \sigma^{o*}, M^*, \sigma^{p*}) = 1$$

with (W^*, σ^{o*}) and $\sigma^{p*} \leftarrow \text{ProxySign}(sk^p, W', \sigma^{o'}, M)$ such that $(W^*, \sigma^{o*}) \neq (W', \sigma^{o'})$
 Let $\text{Adv}_{\Sigma}^{\text{Binding}}(\mathcal{A}) = \Pr[\mathcal{A} \text{ wins}]$ be the advantage of $\mathcal{A}$.

Definition 5.6. Certificate-Signature Binding *A proxy-signature satisfies Certificate-Signature Binding if the advantage $\text{Adv}_{\Sigma}^{\text{Binding}}(\mathcal{A}) \leq \text{negl}(\lambda)$ for all PPT $\mathcal{A}$.*

5.2 Protocol

In this section, we first explain our choice to use CSI-SharK as the base of our proxy signature scheme before we introduce our proxy signature protocols, CWA and PKE.

5.2.1 Choice of Base Signature

In this section, we explain why we chose to use CSI-SharK as the base signature for our proxy signature scheme. The isogeny-based signature schemes can be mainly divided into two different kinds: the ones based on CSIDH, such as SeaSign, CSI-FiSH and CSI-SharK, and SQISign and its variations, such as SQISignHD and AprèsSQL. While the latter is the fastest and most compact of the isogeny signatures, the signatures lose an important property compared to the CSIDH-based signatures, commutativity, with isogenies having a determined starting and ending curve. In the context of proxy signatures, this becomes an issue because the proxy signer does not know a path which links the starting curve to the certificate and therefore cannot use the certificate to compute a proxy signature.

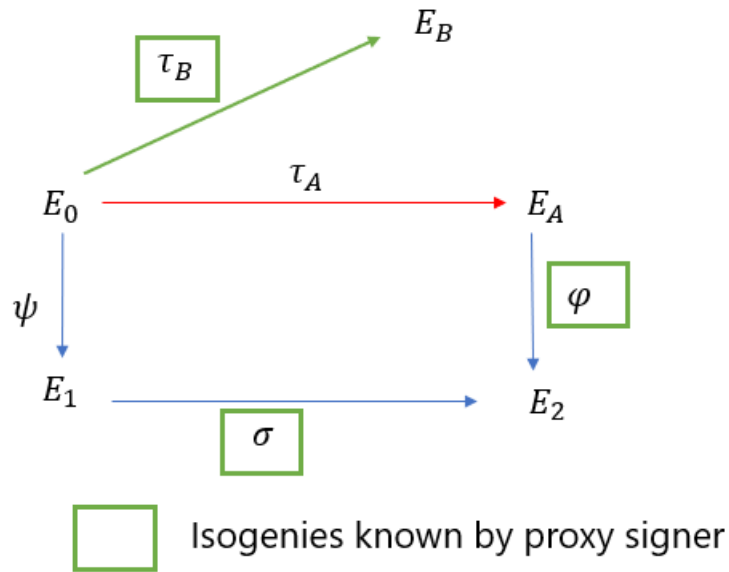


Figure 5.1: Proxy signature setting based on SQI-Sign

All SQISign-based signatures are obtained using a Fiat-Shamir transform on a Sigma protocol, which gives a 3-step signature, as represented in Figure 5.1. The isogeny τ_A represents the original signer's secret key, while τ_B represents a proxy signer's secret key. In the context of a proxy signature, the proxy signer would have access to both the signature σ and the challenge φ while the original signer keeps both τ_A and ψ secret, as it would compromise security. The issue is that a proxy signer would not know a path from E_0 to E_A and E_1 , which would not cause an issue if using the signature as is in the hash (as in CWA), but would cause one when generating a proxy signing key pair based on the original signer (as in PKE). As a result, any resulting scheme would not satisfy the necessary properties of a proxy signature with proxy key leakage, making SQISign-based signature schemes unusable to our knowledge.

As a result, we only consider CSIDH-based signatures for our proxy-signature scheme. Although SeaSign does not require precomputations to compute the structure of the ideal class group, it suffers from larger signature size and key sizes, making it in our opinion less preferable than CSI-FiSh or CSI-SharK. To facilitate future extensions of the scheme, such as a threshold proxy signature scheme, we choose CSI-SharK which thanks to its structured public key allows for faster implementation as proposed in the fastest threshold isogeny-based signature scheme, Thresher-SharK [ABCP23b].

5.2.2 CWA Proxy Signature

As introduced in Section 2.2, our proxy signature scheme is composed of 7 algorithms with KeyGen, Sign, Verify being the same as the original signature scheme CSI-SharK, see Algorithms 8, 9 and 10. We briefly detail the remaining algorithms. The full algorithms can be found in Algorithms 36 to 39. In the remainder of the section, we assume that the proxy and the original signer have agreed on an exceptional set $\Xi_k = \{c_1 = 1, c_2, \dots, c_k\}$.

Delegate: Delegate uses warrants $W = (pk^o, pk^p, \mathcal{M}, \mathcal{V})$ with $pk^o = (Y_1^o, \dots, Y_{k-1}^o)$ the public key of the original signer, $pk^p = (Y_1^p, \dots, Y_{k-1}^p)$ the proxy signer's public key, $\mathcal{M}$ the message space, and $\mathcal{V}$ the validity period. The algorithm itself is identical to the signing algorithm of CSI-SharK, taking the warrant as message, and outputs $\sigma^o = (r_i^o, d_i^o)_{i=1}^{t_k} = \text{Sign}(sk^o, W)$

D-Verify: Similarly to Delegate, the algorithm uses the verification algorithm of CSI-SharK and outputs $b \leftarrow \text{Verify}(pk^o, \sigma^o, W)$.

P-Sign: P-Sign takes as input the warrant and certificate pair (W, σ^o) , the proxy signer's secret key sk^p and the message M . The value t_k is taken similarly to the original scheme. P-Sign differs from Sign in that, when computing the challenges, the proxy signer will use both the warrant, the message to be sent, and the certificate σ^o instead of just the message.

P-Verify: P-Verify is a three-step verification algorithm as follows:

- The verifier will first verify the proxy signature respects the restrictions described in the warrant, i.e, if the message $M \in \mathcal{M}$, if the current date is within the validity period $v \in \mathcal{V}$, and both public keys correspond to the one in the warrant.
- They will then verify the certificate's validity by running $\text{Verify}(\sigma^o, W, pk^o) = 1$.

- Finally, they will check the proxy signature itself.

Algorithm 36 Delegate

Input $\Xi_k = \{c_1 = 1, \dots, c_k\}, sk^o = x^o, W$

Output σ^o

```

1: for  $i = 1, \dots, t_k$  do
2:   sample  $b_i^o \leftarrow \mathbb{Z}_N$  and compute  $E_i^o = [b_i^o]E_0$ 
3: end for
4: Set  $(d_1^o, \dots, d_{t_k}^o) = H(E_1^o \dots E_{t_k}^o || W)$ 
5: for  $i = 1, \dots, t_k$  do
6:   set  $r_i^o = b_i^o - \text{sign}(d_i^o) \cdot x^o c_{|d_i^o|} \pmod{N}$ 
7: end for
8: return  $\sigma^o = (r_i^o, d_i^o)_{i=1}^{t_k}$ 
    
```

Algorithm 37 D-Verify

Input $\Xi_k, pk^o = (Y_1, \dots, Y_{k-1}), \sigma^o, W$

Output b

```

1: for  $i = 1, \dots, t_k$  do
2:   Compute  $E_i^* = [r_i^o]Y_{d_i^o}^o$ 
3: end for
4: Set  $(d_1^*, \dots, d_{t_k}^*) = H(E_1^* \dots E_{t_k}^* || W)$ 
5: if  $(d_1^*, \dots, d_{t_k}^*) = (d_1^o, \dots, d_{t_k}^o)$  then
6:   return  $b = 1$ 
7: end if
8: return  $b = 0$ 
    
```

Algorithm 38 P-Sign

Input $\Xi_k, \sigma^o, W, sk^p, M$

Output σ^p

```

1: for  $i = 1, \dots, t_k$  do
2:   sample  $b_i^p \leftarrow \mathbb{Z}_N$  and compute  $E_i^p = [b_i^p]E_0$ 
3: end for
4: Set  $(d_1^p, \dots, d_{t_k}^p) = H(E_1^p \dots E_{t_k}^p || W || M || \sigma^o)$ 
5: for  $i = 1, \dots, t_k$  do
6:   set  $r_i^p = b_i^p - \text{sign}(d_i^p) \cdot c_{|d_i^p|} x^p \pmod{N}$ 
7: end for
8: return  $\sigma^p = (r_i^p, d_i^p)_{i=1}^{t_k}$ 
    
```

Algorithm 39 P-Verify

Input $\Xi_k, \sigma^o, W, M, \sigma^p, pk^o, pk^p$
Output bit b

- 1: **if** $M \notin \mathcal{M} \vee (pk^o, pk^p) \notin W \vee v \notin \mathcal{V}$ **then**
- 2: return 0
- 3: **end if**
- 4: **for** $i = 1, \dots, t_k$ **do**
- 5: Compute $E_i^{o*} = [r_i^o]Y_{d_i^o}^o$
- 6: **end for**
- 7: Set $(d_1^{o*}, \dots, d_{t_k}^{o*}) = H(E_1^{o*} \dots E_{t_k}^{o*} || W)$
- 8: **if** $(d_1^{o*}, \dots, d_{t_k}^{o*}) \neq (d_1^o, \dots, d_{t_k}^o)$ **then**
- 9: return 0
- 10: **end if**
- 11: **for** $i = 1, \dots, t_k$ **do**
- 12: Compute $E_i^{p*} = [r_i^p]Y_{d_i^p}^p$
- 13: **end for**
- 14: Set $(d_1^{p*}, \dots, d_{t_k}^{p*}) = H(E_1^{p*} \dots E_{t_k}^{p*} || W || M || \sigma^o)$
- 15: **if** $(d_1^{p*}, \dots, d_{t_k}^{p*}) = (d_1^p, \dots, d_{t_k}^p)$ **then**
- 16: return 1
- 17: **end if**
- 18: return 0

5.2.3 PKE Proxy Signature

Our PKE proxy signature scheme is also composed of 7 algorithms with **KeyGen**, **Sign**, **Verify** being identical to CSI-SharK. We briefly detail the differences with the previous protocol, the full algorithms can be found in Algorithms 40 to 43.

Delegate: The scheme is run by the original signer and the inputs are identical to the previous scheme. The original signer samples $x^{o'} \leftarrow \mathbb{Z}_N$ and computes $X^o = x^o + x^{o'} \pmod{N}$ and the curves $Y_i^{o'} = [c_i X^o]E_0$ for $i \in [1, k-1]$ which will serve as the intermediate key pair $(sk^{o'}, pk^{o'})$. Finally, the original signer generates the certificate, which is a signature on the warrant using his secret key and the intermediate public key $pk^{o'} = \{Y_i\}_{i=1}^{k-1}$. He then sends $sk^{o'} = X^o$, $pk^{o'}$ and the certificate σ^o to the proxy signer.

P-KeyGen: The algorithm is similar to **D-Verify** in that it is run by the proxy signer and starts by verifying the validity of the certificate. The proxy signer then samples $x^{p'} \leftarrow \mathbb{Z}_p$, and computes $X^p = x^p + x^{p'}$. Finally, the proxy signer computes $X = X^o + X^p \pmod{N}$ and $Y_i^{p'} = [c_i X]E_0$, which will serve as the proxy signing key pair $(pk^{p'}, sk^{p'})$, as well as the signature $\sigma_{prx} \leftarrow \text{Sign}(sk^p, W || \sigma^o || Y_1^{p'} \dots Y_{k-1}^{p'})$ which he generates with his secret key.

P-Sign: The algorithm is identical to the previous version, except that the proxy signature is generated using the proxy signing secret key $sk^{p'}$: $\sigma^p \leftarrow \text{Sign}(sk^{p'}, M || W || \sigma_{prx})$.

P-Verify: The algorithm is similar to the previous one, by first checking if the proxy signature conforms with the information in the warrant (i.e. correct public keys, message space, and validity period), before checking the signatures.

Algorithm 40 Delegate(PKE)

Input Ξ_k, sk^o, W

Output $\sigma^o, sk^{o'}, pk^{o'}$

- 1: sample $x^{o'}$ and compute $X^o = x^o + x^{o'} \pmod{N}$
 - 2: **for** $l = 1, \dots, k - 1$ **do**
 - 3: compute $Y_l^{o'} = [c_l X^o]E_0$
 - 4: **end for**
 - 5: **for** $i = 1, \dots, t_k$ **do**
 - 6: sample $b_i^o \leftarrow \mathbb{Z}_N$ and compute $E_i^o = [b_i^o]E_0$
 - 7: **end for**
 - 8: Set $(d_1^o, \dots, d_t^o) = H(E_1^o \dots E_{t_k}^o || W || Y_1^{o'} \dots Y_{k-1}^{o'})$
 - 9: **for** $i = 1, \dots, t_k$ **do**
 - 10: set $r_i^o = b_i^o - \text{sign}(d_i^o) \cdot x^o c_{|d_i^o|} \pmod{N}$
 - 11: **end for**
 - 12: return $sk^{o'} = X^o, pk^{o'} = (Y_l^{o'})_{l=1}^{k-1}, \sigma^o = (r_i^o, d_i^o)_{i=1}^{t_k}$
-

Algorithm 41 P-KeyGen

Input $\Xi_k, sk^{o'}, pk^{o'}, x^p, \sigma^o, W$
Output $\sigma_{prx}, sk^{p'}, pk^{p'}$

- 1: **for** $i = 1, \dots, t_k$ **do**
- 2: Compute $E_i^* = [r_i^o]Y_{d_i^o}^o$
- 3: **end for**
- 4: Set $(d_1^*, \dots, d_t^*) = H(E_1^* \dots E_{t_k}^* || W || Y_1^{o'} \dots Y_{k-1}^{o'})$
- 5: **if** $(d_1^*, \dots, d_t^*) \neq (d_1^o, \dots, d_t^o)$ **then**
- 6: return $\perp$
- 7: **end if**
- 8: sample $x^{p'} \leftarrow \mathbb{Z}_N$ and compute $X^p = x^p + x^{p'} \pmod{N}$, $X = X^o + X^p \pmod{N}$
- 9: **for** $l = 1, \dots, k - 1$ **do**
- 10: compute $Y_l^{p'} = [c_l X]E_0$
- 11: **end for**
- 12: **for** $i = 1, \dots, t_k$ **do**
- 13: sample $b_i^p \leftarrow \mathbb{Z}_N$ and compute $E_i^p = [b_i^p]E_0$
- 14: **end for**
- 15: Set $(d_1^p, \dots, d_t^p) = H(E_1^p \dots E_{t_k}^p || W || \sigma^o || Y_1^{p'} \dots Y_{k-1}^{p'})$
- 16: **for** $i = 1, \dots, t_k$ **do**
- 17: set $r_i^p = b_i^p - \text{sign}(d_i^p) \cdot x^p c_{|d_i^p|} \pmod{N}$
- 18: **end for**
- 19: return $sk^{p'} = X, pk^{p'} = (Y_l^{p'})_{l=1}^{k-1}, \sigma_{prx} = (r_i^p, d_i^p)_{i=1}^{t_k}$

Algorithm 42 P-Sign(PKE)

Input $\Xi_k, \sigma_{prx}, W, sk^{p'}, M$
Output σ^p

- 1: **for** $i = 1, \dots, t_k$ **do**
- 2: sample $b_i \leftarrow \mathbb{Z}_N$ and compute $E_i = [b_i]E_0$
- 3: **end for**
- 4: Set $(d_1, \dots, d_t) = H(E_1 \dots E_{t_k} || W || M || \sigma_{prx})$
- 5: **for** $i = 1, \dots, t_k$ **do**
- 6: set $r_i = b_i - \text{sign}(d_i) \cdot c_{|d_i|} X \pmod{N}$
- 7: **end for**
- 8: return $\sigma = (r_i, d_i)_{i=1}^{t_k}$

Algorithm 43 P-Verify(PKE)

Input $\Xi_k, pk^o, W, \sigma^o, pk^{o'}, pk^p, M, \sigma_{prx}, pk^{p'}, \sigma^p$
Output bit b

- 1: **if** $M \notin \mathcal{M} \vee (pk^o, pk^p) \notin W \vee v \notin \mathcal{V}$ **then**
- 2: return $b = 0$
- 3: **end if**
- 4: **for** $i = 1, \dots, t_k$ **do**
- 5: Compute $E_i^{o*} = [r_i^o]Y_{d_i^o}^o$
- 6: **end for**
- 7: Set $(d_1^{o*}, \dots, d_t^{o*}) = H(E_1^{o*} \dots E_{t_k}^{o*} || W || Y_1^{o'} \dots Y_{k-1}^{o'})$
- 8: **if** $(d_1^{o*}, \dots, d_t^{o*}) \neq (d_1^o, \dots, d_t^o)$ **then**
- 9: return $b = 0$
- 10: **end if**
- 11: **for** $i = 1, \dots, t_k$ **do**
- 12: Compute $E_i^{p*} = [r_i^p]Y_{d_i^p}^p$
- 13: **end for**
- 14: Set $(d_1^{p*}, \dots, d_t^{p*}) = H(E_1^{p*} \dots E_{t_k}^{p*} || W || \sigma^o || Y_1^{p'} \dots Y_{k-1}^{p'})$
- 15: **if** $(d_1^{p*}, \dots, d_t^{p*}) \neq (d_1^p, \dots, d_t^p)$ **then**
- 16: return $b = 0$
- 17: **end if**
- 18: **for** $i = 1, \dots, t_k$ **do**
- 19: Compute $E_i^* = [r_i]Y_{d_i}^{p'}$
- 20: **end for**
- 21: Set $(d_1, \dots, d_t) = H(E_1^* \dots E_{t_k}^* || W || M || \sigma_{prx})$
- 22: **if** $(d_1^*, \dots, d_t^*) \neq (d_1, \dots, d_t)$ **then**
- 23: return $b = 0$
- 24: **end if**
- 25: return $b = 1$

5.3 Security

In this section, we first prove the correctness of the schemes before analyzing the security.

5.3.1 Correctness

For conciseness, we only detail the correctness of the CWA protocol. As described in Section 5.2, we consider an original signer and proxy signer pair, each in possession of their respective key pair (sk^o, pk^o) and (sk^p, pk^p) as well as the agreed upon exceptional set $\Xi_k = \{c_1 = 1, c_2, \dots, c_k\}$.

The original signer will first emit a warrant describing the two public keys and the allowed validity period and message space, before outputting a signature using the warrant as the

message. The proxy signer will verify this signature and, if accepted, use it as the certificate. As the correctness of **Delegate** and **D-Verify** directly depends on CSI-SharK's correctness, we do not detail it here.

The proxy signer will then output the proxy signature $\sigma^p = (r_i^p, d_i^p)_{i=1}^{t_k}$. We now consider a verifier checking the validity of a proxy signature $\sigma^p \leftarrow \text{P-Sign}(sk^p, \sigma^o, W, M)$ by running $\text{P-Verify}(pk^o, pk^p, W, \sigma^o, M, \sigma^p)$. As the proxy signature was emitted using a valid certificate and warrant, the first two steps of the verification will succeed. We now detail the verification of the proxy signature itself. The verifier will first compute the curves E_i^p for $i \in [1, t_k]$.

$$\begin{aligned} E_i^{p*} &= [r_i^p]Y_{d_i^p}^p \\ &= [b_i^p - \text{sign}(d_i^p)c_{|d_i^p|x^p}]Y_{d_i^p}^p \\ &= [b_i^p - \text{sign}(d_i^p)c_{|d_i^p|x^p}] [\text{sign}(d_i^p)c_{|d_i^p|x^p}] E_0 \\ &= [b_i^p]E_0 \\ &= E_i^p \end{aligned}$$

Since we have $E_i^{p*} = E_i^p$, the values of the hash will be identical, and the verification will succeed. We have both $\text{D-Verify}(\sigma^o, pk^o, W) = 1$, and $\text{P-Verify}(pk^o, pk^p, W, \sigma^o, M, \sigma^p) = 1$, so the scheme is correct.

5.3.2 Security

We prove that the first proxy signature scheme achieves EUF-CMA-CWA in the Quantum Random Oracle Model under the MT-GAIP assumption and the $\mathbf{C}_{k-1}$ -VPwAI assumption.

Theorem 5.1. *If CSI-SharK is (ϵ, t, q_s) -EUF-CMA-secure in the Quantum Random Oracle Model under the MT-GAIP assumption and the $\mathbf{C}_{k-1}$ -VPwAI assumption, then CWA is $(\epsilon', t', q'_d, q'_s)$ -EUF-CMA-CWA-secure under the same model and assumptions.*

Proof. As described in Section 2.2, the proof is in two parts, where the adversary $\mathcal{A}$ will first play the role of a type I forger, then a type II. In both cases, the idea is similar in that we consider a wrapping algorithm $\mathcal{B}$ playing the role of the challenger in the EUF-CMA-CWA Game. $\mathcal{B}$ itself has access to his own challenger $\mathbf{C}$ who controls a CSI-SharK Signing Oracle, and with whom $\mathcal{B}$ is playing the EUF-CMA Game of CSI-SharK. By using $\mathcal{C}$, $\mathcal{B}$ answers all of $\mathcal{A}$'s queries such that the experience is indistinguishable from the usual EUF-CMA-CWA game from $\mathcal{A}$'s point of view. At the end, $\mathcal{A}$ will in both cases win the EUF-CMA-CWA Game and output his own proxy signature forgery, which $\mathcal{B}$ will use to win his own game with $\mathbf{C}$.

We first start with type I. As the forger will choose a valid warrant and certificate as input, the difficulty of forging a proxy signature lies in the third part of **P-Verify**, the verification of the proxy signature itself. The forger $\mathcal{A}$ can make both proxy signing queries and signing queries to $\mathcal{B}$. In the context of our proxy signature scheme, this means that the forger can ask the challenger to output $\text{Sign}(sk^p, M)$ using the CSI-SharK signing algorithm for any fresh

message M , as well as a P-Sign query in which case $\mathcal{A}$ will send a valid pair (W, σ^o) to the challenger, after which the challenger outputs $\sigma^p \leftarrow \text{P-Sign}(sk^p, \sigma^o, W, M)$ and sends σ^p to the forger. The process goes as follows:

- $\mathcal{C}$ will first generate a target (sk^*, pk^*) which will serve as sk^p the proxy signer's secret key and send pk^* to $\mathcal{B}$.
- $\mathcal{B}$ will generate a key pair $(sk^o, pk^o) \leftarrow \text{KeyGen}(params)$ and send (pk^*, pk^o, sk^o) to $\mathcal{A}$.
- Upon receiving a signing query from $\mathcal{A}$ on a message M , $\mathcal{B}$ will query his own challenger $\mathcal{C}$ who will run $\sigma = \text{Sign}(sk^*, M)$ and give the result to $\mathcal{B}$. $\mathcal{B}$ will then transmit the result to $\mathcal{A}$ as is.
- Upon receiving a Proxy Signing query with the pair (W, σ^o) on the message M , $\mathcal{B}$ will first query his own challenger on the message $M' = M || W || \sigma^o$. $\mathcal{C}$ will run $\sigma = \text{Sign}(sk^*, M') = (d_i, r_i)_{i=1}^{t_k}$ and send $\sigma = \sigma^p$ to $\mathcal{B}$, which $\mathcal{B}$ will then send to $\mathcal{A}$.
- $\mathcal{A}$ will output a valid proxy signature forgery $\sigma^* = (d_i^*, r_i^*)$ on a message M^* with warrant and certificate pair (W, σ^o) .
- $\mathcal{B}$ will output $\sigma^{*'} = (d_i^*, r_i^*)_{i=1}^{t_k}$ on the message $M^{*'} = M^* || W || \sigma^o$.

Clearly, the environment for $\mathcal{A}$ is indistinguishable from the EUF-CMA-CWA setting; all of his queries are answered by using the actual signing oracle run by $\mathcal{B}$'s challenger $\mathcal{C}$. If $\mathcal{A}$'s forgery is valid, $\mathcal{B}$ is able to output a valid forgery unless the message M^{*} has already been queried. This can only happen if $\mathcal{A}$ has already sent a proxy signing query on $(W, \sigma^{o'}, M)$ or a signing query on the message M^{*} , which would both contradict $\mathcal{A}$ winning the game.

The second part of the proof is similar to the first, albeit more straightforward. $\mathcal{A}$ can ask Delegation Generating Queries on a chosen warrant to the challenger, where it will output $\sigma^o \leftarrow \text{Delegate}(sk^o, W)$ and send back σ^o to the forger. As the second part of P-Verify is the verification of the certificate, $\mathcal{A}$ must forge a certificate on a fresh warrant for the forgery to succeed. As Delegation Generating Queries are essentially Signing Queries, $\mathcal{B}$ can simply transmit any query he receives from $\mathcal{A}$ to his own challenger $\mathcal{C}$ and answer with $\mathcal{C}$'s answer as is. Once $\mathcal{A}$ outputs his forgery, which will include a certificate, which is a valid CSI-SharK signature on a fresh warrant, $\mathcal{B}$ can directly use it and win the EUF-CMA Game. In both cases, we were able to construct an algorithm $\mathcal{B}$ winning the EUF-CMA Game in approximately the same amount of time and the same advantage ϵ , which concludes the proof of the theorem.

Because we consider a quantum adversary in both cases, $\mathcal{A}$ is able to ask Quantum Random Oracle (QRO) Queries to $\mathcal{B}$, in which case $\mathcal{B}$ simply applies the QRO unitary provided by its base challenger $\mathcal{C}$. That is, $\mathcal{B}$ does not reprogram or modify the QRO; it simply forwards all quantum queries from $\mathcal{A}$ to the challenger's QRO, and returns the resulting states to $\mathcal{A}$. Because the QRO itself is never programmed or modified by the reduction, and any forwarding decisions depend only on the classical input (and not on the history of prior queries), the reduction is

history-free in the sense of [BDF⁺10]. Therefore, the proxy signature reduction lifts to the QROM, assuming CSI-SharK is QROM-secure. ■

Theorem 5.2. *If CSI-SharK is (ϵ, t, q_s) -EUF-CMA-secure in the Quantum Random Oracle Model under the MT-GAIP assumption and the C_{k-1} -VPwAI assumption, then PKE is $(\epsilon', t', q'_d, q'_s)$ -EUF-CMA-PKE-secure under the same model and assumptions.*

Proof. The idea is similar to the EUF-CMA-CWA proof, let $\mathcal{A}$ be the adversary attempting to output a forgery in the EUF-CMA-PKE Game, $\mathcal{B}$ playing the role of the challenger and playing the EUF-CMA of CSI-SharK with $\mathcal{C}$. $\mathcal{B}$ will maintain $OSList, delList(w), pskList(w)$ for the ordinary signature query, the delegation query and the proxy signing secret key, respectively. $\mathcal{B}$ first receives the target public key pk^* from $\mathcal{C}$, and flips a coin c . If $c = 0$, $\mathcal{B}$ sets $pk^* = pk^t$, i.e without knowing the corresponding secret key sk^t . If $c = 1$, $\mathcal{B}$ generates a key pair pk^*, sk^* instead and chooses $i^* \in \{1, \dots, q'_d\}$ (corresponding to the number of delegation queries). The process goes as follows:

- *Ordinary Signing Query:* On input from $\mathcal{A}$, $\mathcal{B}$ either queries $\mathcal{C}$ if $c = 0$, or generates a signature itself if $c = 1$. $\mathcal{B}$ adds the resulting signature σ to $OSList$.
- *Delegation to $\mathcal{C}$ Query:* $\mathcal{B}$ receives from $\mathcal{A}$ an intermediate key pair $pk^{o'} = (Y_i^{o'})_{i=1}^{k-1}, sk^{o'} = X^o$ with and a certificate σ^o . $\mathcal{B}$ first verifies $Y_i = [c_i X^o]E_0$ for $i \in [1, k-1]$ and $\text{Verify}(pk^{o'}, W || pk^{o'}, \sigma^o) = 1$ and abort otherwise. $\mathcal{B}$ then samples a random isogeny $X^p \leftarrow \mathbb{Z}_N$ and computes $Y_i^{p'} = [c_i X^p]E_0$ which will serve as his proxy signing key pair. If $c = 0$, he then queries $\mathcal{C}$ on $W || \sigma^o || Y_1^{p'} \dots Y_{k-1}^{p'}$ and obtains σ_{prx} . If $c = 1$ and this is not the i^* -th query, he generates σ_{prx} on $W || \sigma^o || pk^{p'}$, with sk^* instead. If it is the i^* -th query, $\mathcal{B}$ lets $pk^{p'} = pk^t$, leaves $sk^{p'}$ empty and generates $\sigma_{prx} \leftarrow \text{Sign}(sk^*, W || \sigma^o || pk^t)$. $\mathcal{B}$ then adds $sk^{p'}$ to $pskList(w)$ and $W, \sigma^o, pk^{o'}$ to $delList(w)$.
- *Delegation from $\mathcal{C}$ Query:*
 1. *Delegation of sk^* Query:* On input W from $\mathcal{A}$, $\mathcal{B}$ will sample $X^o \leftarrow \mathbb{Z}_N$, compute $Y_i^{o'} = [c_i X^o]E_0$ and obtain the certificate σ^o by either asking $\mathcal{C}$ if $c = 0$ or generating it itself if $c = 1$. He then sends $(pk^{o'}, sk^{o'}, \sigma^o)$ to $\mathcal{A}$ and adds $w, \sigma^o, pk^{o'}$ to $delList(w)$.
 2. *Self-Delegation Query:* On input W from $\mathcal{A}$, if $c = 0$ or $c = 1$ and this is not the i^* -th query, $\mathcal{B}$ samples $X^o, X^p \leftarrow \mathbb{Z}_N$ who will serve as $sk^{o'}, sk^{p'}$, computes $pk^{o'}, pk^{p'}$ and obtains σ^o, σ_{prx} either from $\mathcal{C}$ if $c = 0$ or by generating them itself if $c = 1$. If it is the i^* -query, $\mathcal{B}$ only generates $X^o \leftarrow \mathbb{Z}_N$, computes $pk^{o'}, \sigma^o$ and lets $pk^{p'} = pk^t$ (leaving $sk^{p'}$ empty). It then generates $\sigma_{prx} = \text{Sign}(sk^*, W || \sigma^o || pk^t)$. $\mathcal{B}$ then adds $W, \sigma^o, pk^{o'}$ and $W, sk^{p'}$ to $delList(w), pskList(w)$ respectively. If σ^o was obtained from a query to $\mathcal{C}$, $\mathcal{B}$ adds it to $OSList$.
- *Proxy Signature Query:* After receiving $M, (W, \sigma^o)$ from $\mathcal{A}$, $\mathcal{B}$ checks the information relevant to w from $delList(w), pskList(w)$ and aborts if there are not any. Otherwise, it

generates $\sigma^p \leftarrow \text{Sign}(sk^{p'}, W || M || \sigma_{prx})$ (or queries the oracle if $sk^{p'}$ was left empty) and returns σ^p to $\mathcal{A}$.

- *Proxy Key Exposure Query*: On input w from $\mathcal{A}$, $\mathcal{B}$ finds the relevant information from $\text{delList}(w), \text{pskList}(w)$ and aborts if it does not exist. If the corresponding $sk^{p'}$ was left empty, it also aborts. Otherwise, $\mathcal{B}$ returns $(sk^{p'}, W, \sigma^o, pk^{p'}, \sigma_{prx})$ to $\mathcal{A}$.

The experience from $\mathcal{A}$ is indistinguishable from the usual EUF-CMA-PKE Game, $\mathcal{B}$ answers $\mathcal{A}$'s queries as a challenger would. The only exception are during **Delegate** and **P-KeyGen** where $\mathcal{B}$ directly samples $sk^{o'}$ and $sk^{p'}$ instead of using his secret key, uses pk^t as $pk^{p'}$, all of which $\mathcal{A}$ is unable to discern, and if $\mathcal{A}$ makes a proxy key exposure when $sk^{p'}$ is left empty in which case $\mathcal{B}$ aborts. If the above game is not forced to abort by $\mathcal{B}$, $\mathcal{A}$ eventually outputs a forgery, which are classified in 2 cases.

- Case 1: $\mathcal{A}$ forges (1) a valid CSI-SharK signature (m, σ) , (2) a valid proxy signature $(M, (W, \sigma^o, pk^{p'}, \sigma_{prx}), \sigma^p)$ where $pk^{p'}$ was not generated by $\mathcal{B}$, (3) a valid proxy signature where W was not submitted to the ordinary signature query.
- Case 2: All valid forgeries not in Case 1, i.e. type (ii) (proxy signature of u^*) or type (iii) (proxy signature on behalf of u^*) where $\mathcal{B}$ has generated the proxy signing public key $pk^{p'}$.

We first consider the case $c = 0$, where pk^* was set to pk^t . If $\mathcal{A}$ constructs a forgery Case 2, $\mathcal{B}$ aborts. Otherwise, if $\mathcal{A}$ constructs a valid Case 1 forgery:

- If the forgery is of type (i) (ordinary signature forgery), i.e. (m, σ) , $\mathcal{A}$ will not have queried M to $\mathcal{B}$, who will not have queried $\mathcal{C}$. Hence, $\mathcal{B}$ can output the signature as a valid forgery in the EUF-CMA Game.
- If the forgery is of type (ii) (proxy signature of u^*), i.e. $(M, (W, \sigma^o, pk^{p'}, \sigma_{prx}), \sigma^p)$, then σ_{prx} is a valid signature under the public key pk^t , and $\mathcal{C}$ will not have submitted $(W || \sigma^o || pk^{p'})$ to $\mathcal{C}$, so $\mathcal{B}$ can use it as a forgery.
- If the forgery is of type (iii) (proxy signature on behalf of u^*) $(M, (W, \sigma^o, pk^{p'}, \sigma_{prx}), \sigma^p)$, σ^o is a valid signature on pk^t on the warrant W which $\mathcal{B}$ will not have submitted to his own oracle, so he can use it as forgery.

We now consider the case $c = 1$ where $\mathcal{B}$ inserts pk^t as the proxy signing public key $pk^{p'}$. If the forgery is Case 1, $\mathcal{B}$ aborts the game. If the forgery is Case 2 and $(M, (W, \sigma^o, pk^{p'}, \sigma_{prx}), \sigma^p)$ with $pk^{p'} = pk^t$, then σ^p is a valid signature on the message m and $\mathcal{B}$ will not have queried the oracle, so $\mathcal{B}$ can output (m, σ^p) as a valid forgery. Otherwise, $\mathcal{B}$ aborts the game.

We define the following events associated with the above game. Let E_1 be the event that $\mathcal{A}$ constructs a forgery in Case 1, E_2 that $\mathcal{A}$ constructs a forgery in Case 2, and E_3 that $\mathcal{A}$ uses the proxy signing key provided on the i^* -th query in a forgery for Case 2. Clearly, the probability

of $\mathcal{A}$ succeeding is $Pr[E_1] + Pr[E_2]$. Then the probability of success of $\mathcal{B}$ in the EUF-CMA CSI-SharK is $Pr[c = 0 \wedge E_1] + Pr[c = 1 \wedge E_2 \wedge E_3]$. We obtain

$$\begin{aligned}
 \epsilon &= Pr[c = 0 \wedge E_1] + Pr[c = 1 \wedge E_2 \wedge E_3] \\
 &= 1/2 \cdot Pr[E_1] + Pr[E_3|c = 1 \wedge E_2] \cdot Pr[c = 1|E_2] \cdot Pr[E_2] \\
 &= 1/2 \cdot Pr[E_1] + 1/q'_d \cdot 1/2 \cdot Pr[E_2] \\
 &\geq \frac{\epsilon'}{2q'_d}
 \end{aligned} \tag{5.1}$$

Finally, we use the same reasoning as CWA for QROM security, which concludes the security proof. ■

Theorem 5.3. *The proposed schemes satisfy verifiability, strong identifiability, and strong undeniability, certificate signature binding as well as the prevention of misuse.*

Proof. We show that those properties are satisfied for both of our schemes by construction. In both our propositions, the ProxyVerify algorithm is constructed in such a way that it encompasses the verification of the certificate, the verification of the proxy signature, and the verification of the use conditions (as in message space and validity period). As such, the prevention of misuse is directly satisfied by construction. Similarly, as the proxy signature is signed in the certificate and the warrant in both our schemes, the certificate-signature binding is satisfied by construction.

We now look at the remaining properties. As the queries available to $\mathcal{A}$ in the various games are also available to the forger in both EUF-CMA-CWA and EUF-CMA-PKE models, all 3 remaining properties are directly implied from the unforgeability of both the proxy signature and the unforgeability of the base signature. To do so, let $\mathcal{A}$ be a forger winning the EUF-CMA-PKE Game. We show that such a Forger is able to win all 3 games as defined in 5.1.

- **Verifiability:** We consider that $\mathcal{A}$ succeeds in the EUF-CMA-PKE by either using a forgery under the form of a proxy-signature on behalf of the honest signer. Let $(\sigma^{o*}, \sigma^{p*}, W, M)$ be such a forgery. $\mathcal{A}$ can directly use such a forgery to win $Game_{\Sigma}^{Verif}$.
- **Strong Undeniability/ Identifiability:** Similarly, let $(\sigma^{o*}, \sigma^{p*}, W, M)$ be the forgery by $\mathcal{A}$. As the certificate has not been emitted by the honest signer, $\mathcal{A}$ can use the forgery to win $Game_{\Sigma}^{ID}$. ■

5.4 Comparison

In this section, we compare our proxy signature schemes CWA, PKE to CSI-PS [DKSD24], and the generic construction from [BPW03], in terms of both key sizes and computation time in

Table 5.1. CSI-PS is based on SeaSign and as such, relies on the GAIP Problem, on the contrary to our protocol which relies on MT-GAIP and C_{k-1} -VPwAI assumptions. Both schemes are proven unforgeable under the Random Oracle Model and satisfies the properties highlighted in Section 2.2, although Dey et al. seem to use an adaptation of the traditional EUF-CMA to the proxy signature setting, without taking into account warrant attacks (i.e Delegation queries) or key exposure.

For an appropriate comparison, we make a similar choice to CSI-PS and choose a set of parameters $(k, t_k) = (2^{15}, 7)$ corresponding to the smallest signature size, at the cost of a larger key pair, indicated as $|pk|$ for the public key and $|sk|$ for the secret key respectively. While all key sizes and signature sizes are identical in both PKE and CWA schemes, the former requires an additional key pair for both the proxy and the original signers, as well as an additional signature in σ_{prx} .

The computation time is estimated based on the computation time of a single group action measured in [DKSD24], 0.021624 seconds. Given the computation time is approximately identical during Delegate and P-KeyGen/D-Verify for all 3 schemes, we only compare KeyGen, Delegate, P-Sign and P-Verify.

While both the generic construction and CWA have identical computation time (same number of isogeny computations) and identical size (same as a regular signature), the generic construction suffers from several issues. The first is that it does not take into account the validity period, which can be easily fixed by adding it to the warrant. The second is the fact that the proxy signature and the certificate/warrant are not cryptographically linked. Because the proxy signature is signed on $pk^o||M$, while the certificate is signed on $pk^p||W$, the certificate and the warrant are not included in the proxy signature itself. As a result, the proxy signature is not associated with the certificate, but simply with the original signer's public key. As we argue in Section 5.1, such an issue weakens the undeniability and verifiability of the scheme, which we incorporate into the notion of Certificate-Signature Binding. To fix those issues, we modify the scheme to satisfy this property.

We obtain a proxy signature that satisfies the necessary properties of an EUF-CMA-CWA proxy signature without increasing memory or computation cost. CWA is faster and significantly less memory costly than CSI-PS while being proven secure in a more rigorous model. Our second protocol, PKE obtains security in a stronger security model at the cost of an increased computation time and an additional key pair per signer.

5.5 Conclusion

In this chapter, we propose a formal definition of the properties of proxy signatures, verifiability, strong undeniability, strong identifiability and prevention of misuse. Additionally, we propose a new security property, which we name certificate-signature binding. We then propose proxy signature schemes based on the isogeny-based signature scheme CSI-SharK. We prove our schemes are EUF-CMA-CWA, respectively EUF-CMA-PKE in the Quantum Random Oracle Model by reducing the unforgeability of our scheme to the unforgeability of CSI-SharK. We also prove

Table 5.1: Comparison between CSI-PS, PKE, CWA (in kB, seconds)

	CSI-PS	CWA	PKE	Generic
$ sk $	2097	0.016	0.016	0.016
$ pk $	4178	2097	2097	2097
$ sig $	3	0.263	0.263	0.263
KeyGen	1422.12	708.58	708.58	708.58
Delegate	0.3472	0.1514	708.73	0.1514
P-Sign	0.3472	0.1514	0.1537	0.1514
P-Verify	0.6944	0.3027	0.4541	0.3027
Security	EUFCMA	EUFCMA-CWA	EUFCMA-PKE	EUFCMA-CWA
Properties	✓	✓	✓	Binding

that our scheme is correct and satisfies verifiability, strong identifiability, strong undeniability, prevention of misuse and certificate-signature binding.

Chapter 6

Efficient Multi-Party Contact Tracing Protocol

During the Covid-19 pandemic, many contact tracing applications were proposed as a way to slow down the spread of the virus. While the proposal described in Section 3.4 stands out by its use of cryptographic techniques and formal security proofs, it, just like other contact tracing protocols, only considers interaction between users in pairs. This leads to inaccurate information being stored in the app, as more than two users can interact together at the same time. Moreover, when considering the interactions in their entirety, only considering pairs leads to a computation and communication overhead.

In this chapter, we take a new approach and shift perspective by considering the interactions in their entirety, i.e. all users interacting in a way that constitutes a risk of infection. We then propose a new contact tracing application adapted to this multi-party perspective by leveraging the compactness of multi-signatures compared to repeated use of signatures. To guarantee that only patients who test positive can send alerts to their close contacts, we take advantage of the underlying authority-based control of ID-based ring signatures by setting the Government as the trusted authority. The anonymity property of ring signatures additionally ensures that patient privacy is satisfied. We propose two instantiations of the meeting phase, i.e. the phase during which the users interact directly, by using a pairing-based multi-signature and an ECDH-based multi-signature. While the former requires fewer computations and communications, as a single user can gather the intermediate computations to generate the multi-signature, the latter adds flexibility by allowing users to join the protocol midway. Both versions satisfy patient privacy and prevent false positives by relying on the unforgeability of the respective multi-signature and ID-based ring Signature.

6.1 Security Model

In this section, we define the assumptions we use throughout the chapter before defining our Threat Model.

6.1.1 System Assumptions

Our protocol uses the following entities:

- User: A user of the contact tracing app on his smartphone. Will be further classified into close contacts and patients.
- Government (GV): Trusted entity responsible for the users' registrations with their app, and will act as the trusted authority for the ID-based ring signature.
- Bulletin Board (BB): Used to publish public information (i.e., alerts for close contacts). Can be accessed by anyone.

We make the following assumptions concerning the user:

- Every user has Bluetooth and Internet connectivity on their smartphone, accessible by the contact tracing app.
- After first downloading the app, the user accepts that the app will upload their information to BB if they are tested as Covid-19 positive.
- The user is unable to add incomplete information to the app (namely, a list of identifiers without the corresponding multi-signature), and cannot modify it afterwards.
- A user who tested positive for Covid-19, referred to as “patient”, will use all information on the app while alerting their close contacts with only the information on the app (i.e., all lists of identifiers will be used to compute ring signatures, and only those lists).

Apart from those assumptions, we do not limit the users' behaviors during the contact tracing protocol.

6.1.2 Threat Model

We only consider cryptographic attacks, i.e, other attacks, such as networking attacks are out of scope in this thesis. We define the following threat model, which uses concepts defined in [LAY⁺20], but we redefine these concepts. The difference between the two threat models is explained in Section 6.3.1. We then provide a security proof based on that threat model in Section 6.3.2.

Definition 6.1. Traceability Completeness: *Let there be a Covid-19 patient. The protocol satisfies traceability completeness if, after correct execution, all users identified as close contacts by the patient during the Meeting Phase are able to receive an alert in the Tracing Phase.*

Definition 6.2. False Positive Case 1: *Let there be a Covid-19 patient. A False Positive Case 1 occurs if a malicious user can falsely claim to be a close contact of that patient, thereby receiving the alert intended for actual close contacts without having been one.*

Definition 6.3. False Positive Case 2: *Let there be a negative malicious user. A False Positive Case 2 occurs if that malicious user can falsely claim to be a Covid-19 patient, thereby triggering false alerts to all of their supposed close contacts.*

Definition 6.4. Patient Privacy: *Let there be a Covid-19 patient. In a decentralized app, the Contact Tracing system uploads information related to the patient; in a centralized app, it uploads information related to their close contacts. Patient privacy is preserved if no entity (except the government, if applicable) can determine the patient's identity from this information with probability better than random guessing.*

6.2 Proposed Contact Tracing Protocols

This section describes the two variants of our protocol based on DGNW-multi-signature [DGNW20], which we refer to as OF-MP Tracing and MM-multi-signature [MM00], which we refer to as PF-MP Tracing.

6.2.1 Design of our Multi-Party Tracing Protocol

We design our protocol in four phases as follows:

- **Registration Phase:** Users receive the public parameters and their identifiers from the government website. Each user then generates a fresh public/secret key pair at the start of each day.
- **Meeting Phase:** Consider a meeting among n users staying close enough to be considered close contacts, with $n > 1$. The **Meeting Phase** is divided into two subphases: a **Verification Phase** and a **Proof Generation Phase**.
- **Alerting Phase:** Assume that one of the n meeting participants tests positive for Covid-19. Information about the patient is then uploaded to the database.
- **Tracing Phase:** The close contacts of the patient receive alerts notifying them that they have been exposed to a Covid-19 patient.

Additionally, we define the following properties for contact tracing protocols:

Definition 6.5 (Order Flexibility (OF)). *A contact tracing protocol satisfies Order Flexibility if the order in which users complete the **Meeting Phase** does not affect the protocol's correctness. In other words, the protocol functions correctly regardless of the sequence of participants.*

Definition 6.6 (User Flexibility (UF)). *A contact tracing protocol satisfies User Flexibility if the set of close contacts during the **Meeting Phase** does not need to be predetermined for the protocol to function correctly.*

Definition 6.7 (Multi-party Proof (MP)). *A contact tracing protocol satisfies Multi-party Proof if, at the end of the **Meeting Phase**, all participants store an identical proof on their devices confirming that they were all present.*

As the multi-signature in the **Meeting Phase** will be generated on the list of identifiers of the users, User Flexibility (UF) corresponds to message flexibility of the underlying multi-signature, which is achieved by MM. OF-MP Tracing and PF-MP Tracing use the same Registration, Alerting, and Tracing Phases, but some differences exist in the **Meeting Phase** during the **Proof Generation Phase**. We compare the two of them in Section 6.4.

6.2.2 Registration Phase

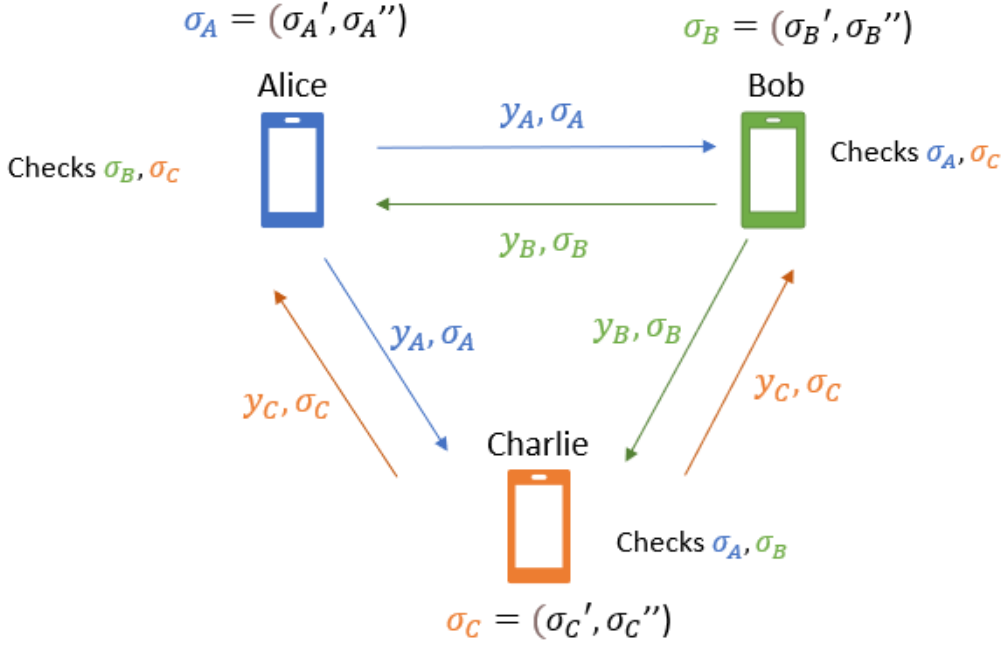
The **Registration Phase** corresponds to the initial phase that the users perform when downloading the app and updating the keys. Each user receives an identifier ID generated by GV, together with the public parameters $params = (\mathbb{G}_1, \mathbb{G}_2, e, q, g_1, g_2, h, y_G, H_{exp}, H_G)$ as defined in Section 2.4. We assume $\mathbb{G}_1, \mathbb{G}_2$ are GDH group as defined in Section 2.4, and that the wBDHI problem is hard, to ensure DGNW, MM and CYH are unforgeable. Each day, the users generate their key pair $pk = y$ and $sk = x$, with $y = g_2^x$ and publish pk on GV's website. The pair (sk, pk) is stored in the smartphone for 14 days, corresponding to the period with a risk of transmission.

6.2.3 Meeting Phase

The **Meeting Phase** corresponds to the phase when users directly interact with one another and store information about close contacts in their smartphones. We consider that every app broadcasts a package to nearby devices containing the user's identifier ID and public key y , in the form $H(y, ID)$. A package is accepted only if the receiving smartphone is within 2 meters of the sender. The app estimates the distance between two smartphones using the RSSI (Received Signal Strength Indicator). Once a user has received packages continuously for at least 15 minutes from another user, that sender is recognized as a close contact.

In [LAY⁺20], such an interaction between users is only considered in pairs by the app. However, because more than two users can interact in such a way that they are recognized at the same time, we shift perspective and consider the users globally to more accurately reflect the situation. Let n denote the number of users who are within sufficient proximity for a long enough period to recognize each other as close contacts. In the rest of this chapter, we label the users with indices i, j such that $i, j \leq n$. Here, y_i and x_i are user i 's public and secret keys, respectively, with $y_i = g_2^{x_i}$.

The **Meeting Phase** consists of two subphases: a **Verification Phase**, illustrated in Figure 6.1, in which each user verifies the identity of all other participants, and a **Proof Generation Phase**, in which participants generate a multi-signature as a multi-party proof of being close contacts, illustrated in Figure 6.2.


 Figure 6.1: **Verification Phase** with $n = 3$

Verification Phase: We describe the protocol from the viewpoint of user i ; all other users execute the same steps simultaneously. Let the message be the list of all users' identifiers, $M = \{\text{ID}_1, \dots, \text{ID}_n\}$. To reduce computation, all users also precompute and store $H(M)$. The protocol goes as follows:

1. Sample $r_i \leftarrow \mathbb{Z}$, and compute the signature σ_i of the following form: $\sigma_i = (\sigma'_i, \sigma''_i) = (h^{x_i} \cdot H(M)^{r_i}, g_2^{r_i})$.
2. Broadcast (y_i, σ_i) to all users $j \in [1, n] \setminus i$.
3. User j checks $e(\sigma'_i, g_2) = e(h, y_i) \cdot e(H(M), \sigma''_i)$.

Correctness:

$$\begin{aligned}
 e(\sigma'_i, g_2) &= e(h^{x_i} \cdot H(M)^{r_i}, g_2) \\
 &= e(h^{x_i}, g_2) \cdot e(H(M)^{r_i}, g_2) \\
 &= e(h, g_2^{x_i}) \cdot e(H(M), g_2^{r_i}) \\
 &= e(h, y_i) \cdot e(H(M), \sigma''_i)
 \end{aligned}$$

If the verification holds, the other users accept user i as a close contact. In order to speed up the process, we use a batch verification scheme [FGHP08]: Let $(\delta_1, \dots, \delta_{n-1})$ be a random vector of l_b bit elements from $\mathbb{Z}_q$ representing all of the other participants of the meeting. Accept if $e(\prod_{j=1}^{n-1} \sigma_j^{\delta_j}, g_2) = \prod_{j=1}^{n-1} e(h, y_j)^{\delta_j} \cdot e(H(M), \sigma_j^{\delta_j})$. User i will then store the signatures $\{\sigma_1, \dots, \sigma_{n-1}\}$ into the smartphone.

Proof Generation Phase: The users will generate a multi-party proof to reflect the reality of the meeting accurately. To do so, we propose two alternatives with DGNW-multi-signature [DGNW20] and MM-multi-signature [MM00] as described in Section 2.4. DGNW is a pairing-based signature that can be generated by having each signer output an intermediate signature, which can then be aggregated into a multi-signature. This makes it especially suitable for our protocol: as each user already generated a pairing-based signature on the list of all signers during the **Verification Phase**, those signatures can be used as the intermediate signatures that can then be aggregated into the multi-signature(i.e., the multi-party proof) as is.

While MM does not benefit from the same advantage, its message flexibility property allows for greater flexibility in the contact tracing protocol itself. In a situation where $n - 1$ users would register each other as close contacts and start the **Meeting Phase**, a n -th signer being recognized as a close contact midway could join the **Proof Generation Phase** seamlessly by simply adding his identifier to the message, and update the multi-signature by making the necessary computations. We will compare both in Section 6.4.

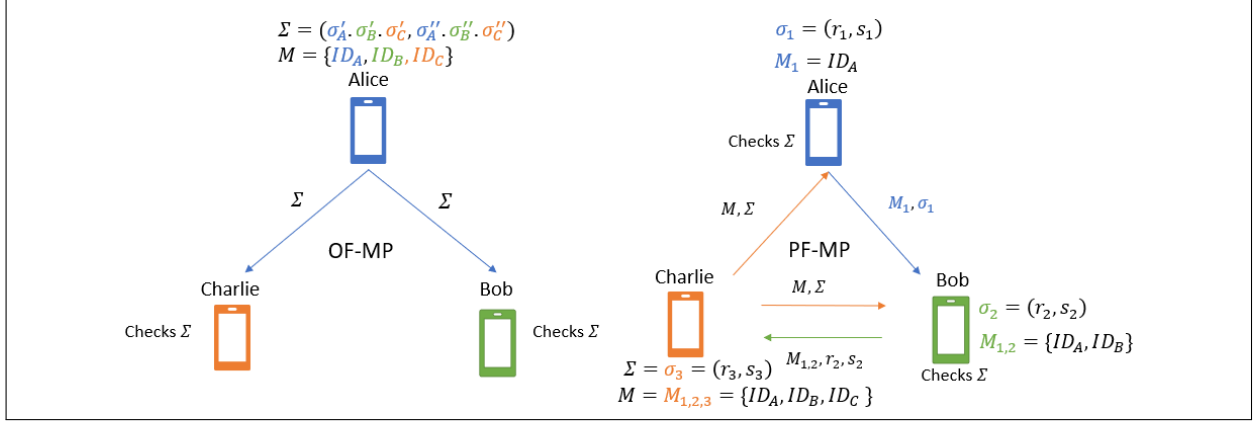
OF-MP Tracing - To ensure that OF-MP Tracing functions correctly, the message $M = \{\text{ID}_1, \dots, \text{ID}_n\}$ is fixed at the beginning of the **Meeting Phase**. User i computes the multi-signature Σ as the product of the signatures $\{\sigma_j\}_{j \in [1, n] \setminus i}$ generated in the **Verification Phase**, $\Sigma = (\sigma'_1 \dots \sigma'_n, \sigma''_1 \dots \sigma''_n) = (h^{x_1 + \dots + x_n} \cdot H(M)^{r_1 + \dots + r_n}, g_2^{r_1 + \dots + r_n})$. They then send Σ to the other $(n - 1)$ users. To verify the multi-signature, user $j \in [1, n] \setminus i$ computes the aggregated public key $\text{Agg} = y_1 \dots y_n$. We then have $e(\Sigma', g_2) = e(h, \text{Agg}) \cdot e(H(M), \Sigma'')$.

Correctness:

$$\begin{aligned}
 e(\Sigma', g_2) &= e(h^{x_1 + \dots + x_n} \cdot H(M)^{r_1 + \dots + r_n}, g_2) \\
 &= e(h^{x_1 + \dots + x_n}, g_2) \cdot e(H(M)^{r_1 + \dots + r_n}, g_2) \\
 &= e(h, g_2^{x_1 + \dots + x_n}) \cdot e(H(M), g_2^{r_1 + \dots + r_n}) \\
 &= e(h, y_1 \dots y_n) \cdot e(H(M), \Sigma'') \\
 &= e(h, \text{Agg}) \cdot e(H(M), \Sigma'')
 \end{aligned}$$

PF-MP Tracing - PF-MP Tracing satisfies *user flexibility* as close contacts can be added during the **Proof Generation Phase**, even after the generation of the multi-party proof has started. At each stage, user $i \in [1, n]$ adds their identifier ID_i to $M_{1, \dots, i-1}$ such that $M = M_{1, \dots, n} = L_{ID} = \{\text{ID}_1, \text{ID}_2, \dots, \text{ID}_n\}$. The protocol is as follows:

- User 1 samples $k_1 \leftarrow \mathbb{Z}_q^*$ and computes $R_1 = g_2^{k_1}$, $r_1 = R_1 + H_G(\text{ID}_1)$, and $s_1 = (x_1 r_1 + 1)k_1^{-1}$ and sends the intermediate signature $\sigma_1 = (r_1, s_1)$ with M_1 to user 2.
- For $i = 2, \dots, n$, user i adds $M_i = \text{ID}_i$ to $M_{1, \dots, i-1}$ to obtain $M_{1, \dots, i}$. They then compute $R_i = g_2^{k_i}$, $r_i = R_i + H_G(\text{ID}_i)r_{i-1}$, and $s_i = (x_i r_i + 1)k_i^{-1}$, and send $\sigma_i = (r_i, s_i)$ with $M_{1, \dots, i}$ to signer $i + 1$.
- At the end, user n outputs $\Sigma = \sigma_n = (r_n, s_n)$ on $M = M_{1, \dots, n} = \{\text{ID}_1, \text{ID}_2, \dots, \text{ID}_n\}$ and sends Σ to the other users.


 Figure 6.2: **Proof Generation Phase: OF-MP Tracing and PF-MP Tracing**

- All other users compute for $j = n, n-1, \dots, 2$, $R'_j = g_2^{s_j^{-1}} y_j^{r_j s_j^{-1}}$, $T_j = r_j - R'_j$, and $r_{j-1} = T_j (H_G(M_j || ID_j))^{-1}$, using user j 's public key y_j .
- Compute $R'_1 = g_2^{s_1^{-1}} y_1^{r_1 s_1^{-1}}$, $T_1 = r_1 - R'_1$. If $T_1 = H_G(M_1 || ID_1)$, $b = 1$, else $b = 0$.

6.2.4 Alerting Phase

In the **Alerting Phase**, we assume that user i is a Covid-19 patient. In order to alert his close contacts, user i retrieves each list of identifiers $L_{ID} = \{ID_1, ID_2, \dots, ID_n\}$ stored in his smartphone in the last 14 days and computes an ID-based ring signature, as introduced in Section 2.4, on the message $M = L_{ID}$, corresponding to the list of identifiers as follows:

- Compute $Q_i = H_G(ID_i)$ his ring public key and receive his private ring key $S_s = Q_s^{x_G}$ via a secure channel from GV.
- $\forall j \in \{1, 2, \dots, n\} \setminus i$, choose $U_j \in \mathbb{G}_1$, compute $h_j = H_{exp}(L_{ID} || U_j)$.
- Choose $r'_i \in \mathbb{Z}_q^*$, compute $U_i = Q_i^{r'_i} / \prod_{j \neq i} U_j \cdot Q_j^{h_j}$.
- Compute $h_i = H_{exp}(L_{ID} || U_s)$ and $V = S_s^{(h_s + r'_s)}$.
- Output the signature on $M = L_{ID}$ as $\sigma_r = \{\cup_{j=1}^n U_j, V\}$.
- Upload σ_r on the bulletin board BB.

As the secret key necessary to output a valid ID-based ring signature can only be received from GV, only users who received the authorization of GV (i.e., patients) are able to send an alert.

6.2.5 Tracing Phase

In the **Tracing Phase**, the users access BB and checks every new entry $\sigma_r = \{\cup_{i=1}^n U_i, V\}$. They will then retrieve

- Compute $h_i = H_{exp}(L_{ID} || U_i) \forall i \in \{1, 2, \dots, n\}$.
- Check whether $e(y_G, \prod_{i=1}^n U_i \cdot Q_i^{h_i}) = e(g_1, V)$.
- Accept the signature if it is true, reject otherwise.

If any ring signature is valid, it means that the user has been in contact with a positive patient. The user will be able to know in which meeting the user was in contact with the patient, but not the identity of the patient among the participants.

Note that while we propose two alternatives for the **Meeting Phase** with DGNW [DGNW20] and MM [MM00], any multi-signature, including the multi-signature scheme proposed in Section 4.6, can be integrated into the protocol by adjusting the **Verification Phase**. Similarly, any ID-based ring signature could be used to replace CYH [CYH05]. However, as there are currently no ID-based ring signatures in isogeny-based cryptography, replacing CYH by an isogeny-based ring signature would require significantly changing the **Alerting Phase** and **Tracing Phase** to account for **False Positive Case 2**, which is currently guaranteed by the ID-based property, see Section 6.3 for the explanation.

6.3 Security

In this section, we first justify the threat model defined in Section 6.1.2. Then, we present the security analysis of OF-MP Tracing and PF-MP Tracing..

6.3.1 Threat Model Justification

As stated in Section 6.1, we assume that users cannot insert or modify incomplete data in the app, for instance, by storing a list of identifiers without the corresponding multi-party proof, or by altering stored identifiers afterward. We also assume that all identifier lists are used during the **Alerting Phase**. These assumptions are reasonable, as they can be enforced at the software level, and they prevent users from maliciously disrupting alerts to close contacts. Our Threat Model builds upon concepts from [LAY⁺20], but we adapt and simplify them. In particular, our definition of **Patient Privacy** is weaker: while participants in a meeting with a patient may be identifiable, the identity of the patient cannot be singled out. We do not consider **Contact Privacy**. Both notions could be strengthened by introducing a trusted entity (e.g., a doctor) to supervise the **Tracing Phase** and by employing zero-knowledge proofs to further protect contact privacy. However, this extension lies beyond the scope of this thesis.

We now justify our Threat Model. Its purpose is to identify possible threats and define security requirements for a contact tracing protocol. The first requirement is user privacy, expressed through **Patient Privacy**. The second is that all users registered as close contacts of a patient during the **Meeting Phase** receive an alert in the **Tracing Phase**, a property referred to as **Traceability Completeness**. Because alerts prompt self-isolation and testing, it is equally important to prevent false alerts. False alerts arise if a user receives an alert from (1)

a patient who was not their close contact, or (2) a close contact who is not actually a patient. These cases are defined as **False Positive Case 1** and **False Positive Case 2**, respectively.

6.3.2 Security Analysis

We first analyze **False Positive Case 1** and **False Positive Case 2**. **False Positive Case 1** occurs when a user receives an alert from a patient who was not a close contact. Since each user checks entries in the bulletin board (BB) using their locally stored list of identifiers L_{ID} , this would require the patient's data to appear in an honest user's list without any real meeting. Such a case can only occur if a malicious user successfully impersonates the patient, i.e., forges a multi-party proof during the **Meeting Phase**.

False Positive Case 2 occurs when a malicious user who tested negative manages to send an alert (i.e., an ID-based ring signature) in the **Alerting Phase** and convinces their close contact of its validity in the **Tracing Phase**. As the alert is only valid if the ID-based ring signature is correct, and since the corresponding secret key can only be issued by the trusted authority, this attack would require forging a valid ID-based ring signature without authorization. Therefore, the security against **False Positive Case 1** and **False Positive Case 2** respectively reduces to the unforgeability of the underlying multi-signature and ring signature schemes. Similarly, **Patient Privacy** reduces to the anonymity of the ring signature scheme.

Theorem 6.1. *False Positive Case 1 and False Positive Case 2 are negligible if the underlying multi-signature and ring signature schemes are unforgeable.*

Proof. The security of **False Positive Case 1** is defined by a sequence of games between a challenger $\mathcal{C}$ and a polynomial-time adversary $\mathcal{A}$. In both games, $\mathcal{C}$ generates a target key pair (sk^*, pk^*) representing the honest user and publishes pk^* .

- Game 0 is the security game of the corresponding multi-signature scheme, where a challenger $\mathcal{C}$ simulates all system users' behaviors for an adversary $\mathcal{A}$.
- Game 1 is the security game in the setting of the **Meeting Phase**, where the adversary will attempt to forge a multi-party proof (i.e, a multi-signature) on a list of identifiers that includes the honest signer.

We show that if $\mathcal{A}$ is able to win Game 1 by forging a multi-party proof which includes the target user represented by the honest signer, $\mathcal{C}$ can use the forgery to win Game 0 and break the EUF-CMA of the underlying multi-signature, meaning if the multi-signature is unforgeable, a multi-party proof cannot be generated. Note that the generated multi-signature was not previously simulated by $\mathcal{C}$.

Similarly, we define Game 2, Game 3 corresponding to **False Positive Case 2** between a challenger $\mathcal{C}$ simulating the trusted authority (i.e, the Government) and a polynomial-time adversary $\mathcal{A}$.

- Game 2 is the security game of the corresponding ID-based ring signature scheme, where a challenger $\mathcal{C}$ simulates all system users' behaviors for an adversary $\mathcal{A}$.

- Game 3 is the security game in the setting of the **Alerting Phase**, where the adversary will attempt to forge a valid ring signature on a list of identifiers, corresponding to a valid alert on the app.

We show that if $\mathcal{A}$ is able to win Game 3 by forging a valid alert for the contact tracing app (i.e., a ring signature on a list of identifiers), $\mathcal{C}$ can use the forgery to win Game 2 and break the EUF-CMA of the underlying ring signature, meaning if the ring signature is unforgeable, a valid alert for the **Alerting Phase** cannot be generated. Note that the generated ring signature was not previously simulated by $\mathcal{C}$.

We will first prove the security of False Positive Case 1. In order to pretend to be a close contact, a user needs to generate a multi-party proof in the Meeting Phase. We prove the security of OF-MP Tracing. For simplicity, we consider the honest user as user 1, and consider $\mathcal{A}$ controls the other users. We define Game 0 as follows:

1. $\mathcal{C}$ generates a key pair for the honest signer $(x^*, y^*) \leftarrow \text{KeyGen}(params)$ with $(sk^*, pk^*) = (x^*, y^*)$ and sends y^* to $\mathcal{A}$.
2. For $2 \leq i \leq n$, $\mathcal{A}$ generates $(x_i, y_i) \leftarrow \text{KeyGen}(params)$ corresponding to the $n - 1$ other users.
3. Upon receiving signing query on M , $\mathcal{C}$ generates $\sigma^* \leftarrow \text{Sign}(x^*, M)$ and send σ_j^* to $\mathcal{A}$.
4. At the end $\mathcal{A}$ outputs a forgery $\Sigma^* = \text{Multi-Sign}(\sigma_1^*, \dots, \sigma_n^*)$ on the message M^* .
5. If $\text{Multi-Verify}(Agg, \Sigma^*, M^*) = 1$ for Agg aggregate public key of $(y^*, y_2, \dots, y_n)$, and $\mathcal{A}$ never made a signing query on M^* , output 1.
6. else, output 0

We define Game 1 analogously to Game 0, but we introduce an additional winning condition corresponding to the ability to compute a valid multi-party proof during the **Meeting Phase**. Specifically, we modify Step 5 as follows: If $\text{Multi-Verify}(Agg, \Sigma^*, M^*) = 1$ for Agg aggregate public key of $(y_1^*, \dots, y_n^*)$, $y_1 \in \{y_1^*, \dots, y_n^*\}$, $\mathcal{A}$ never made a signing query on M^* and M^* is of the form $\{ID_1, \dots, ID_n\}$, output 1. Let W_0 and W_1 denote the events that $\mathcal{A}$ outputs 1 in Game 0 and Game 1, respectively, with probabilities $\Pr[W_0]$ and $\Pr[W_1]$. If W_1 occurs, $\mathcal{A}$ can output the generated multi-signature on the same message to win Game 0, and thus W_0 also occurs. Therefore, $\Pr[W_1] \leq \Pr[W_0]$. Game 0 corresponds to the security game defined in DGNW, so if W_1 occurs, W_0 occurs, and $\mathcal{C}$ can use the generated multi-signature to break the EUF-CMA security of the DGNW multi-signature scheme. Hence, **False Positive Case 1** is negligible if the underlying multi-signature scheme is unforgeable.

We now show that **False Positive Case 2** is also negligible. To impersonate a positive patient, a malicious user must generate a valid ring signature on a list of identifiers previously stored in the app, which is then uploaded to the server during the **Alerting Phase**. The adversary $\mathcal{A}$ is allowed to make the following queries:

- H_G, H_{exp} , which takes an input as parameter and returns the value of the corresponding hash (simulated as random oracles)
- $KeyGen$ which takes as input an ID and returns the corresponding secret key $KeyGen(ID) = S_{ID}$.
- $Sign$ which takes as input a message M , a list of identifiers $\{ID_1, \dots, ID_n\}$ and returns a ring signature.

We define Game 2 in the following way:

1. $\mathcal{C}$ sets up the identifiers, for $1 \leq i \leq n$, $ID_i \leftarrow \{0, 1\}^*$ and the master secret key (corresponding to the Government's secret key) x_G
2. Upon receiving a **KeyGen** query on ID , $\mathcal{C}$ generates $S = KeyGen(ID)$ and sends S to $\mathcal{A}$.
3. Upon receiving a signing query on $\{ID_1, \dots, ID_n\}$ and the message M , $\mathcal{C}$ generates $\sigma_r = Sign(\{ID_1, \dots, ID_n\}, M)$ and sends σ_r to $\mathcal{A}$.
4. At the end $\mathcal{A}$ outputs σ_r on the message M^* .
5. If $Verify(L_{ID}^*, M^*, \sigma_r) = 1$, and $\mathcal{A}$ never made a **KeyGen** query on an identifier in $L_{ID}^* = \{ID_1, \dots, ID_n\}$, output 1.
6. else, output 0

We define Game 3 similarly to the previous games, but with a winning condition corresponding to the generation of a valid alert in the **Alerting Phase**. Specifically, we modify Step 5.: If $Verify(\sigma_r, M^*) = 1$, $\mathcal{A}$ never made a **KeyGen** query on any identifier in $\{ID_1, \dots, ID_n\}$, and M^* is equal to the list of identifiers $\{ID_1, \dots, ID_n\}$. Let W_2 and W_3 denote the events that $\mathcal{A}$ outputs 1 in Game 2 and Game 3, respectively, with probabilities $\Pr[W_2]$ and $\Pr[W_3]$. If W_3 occurs, $\mathcal{A}$ can use the generated ring signature on the same message to win Game 2, implying W_2 also occurs. Therefore, $\Pr[W_3] \leq \Pr[W_2]$. Game 2 corresponds to the security game of the ID-based ring signature scheme. Thus, if W_3 occurs, W_2 occurs, and $\mathcal{C}$ can use the generated alert to break the EUF-CMA security of the underlying ring signature scheme. Consequently, **False Positive Case 2** is negligible if the ring signature scheme is unforgeable, completing the proof of Theorem 1.

An alternative to ensure no **False Positive Case 2** occurs while replacing the ID-based ring signature with a ring signature would be to introduce another trusted entity (i.e., the Doctor D), as was done in Primitive multi-party tracing in Section 3.4, to prevent users from sending alerts without explicit authorization from the Government (GV). Additionally, some information about the multi-signature would need to be linked to the ring signature (i.e., included in the message) together with pseudo-public keys by using zero-knowledge proofs, but we do not detail it in this thesis. ■

Theorem 6.2. Patient Privacy *is maintained if the underlying ID-based ring scheme has the anonymity property.*

Proof. As anyone can access the bulletin board to check the new entries, the adversary can either be a close contact of the patient or an outsider. As the close contact will strictly be in possession of more information than the outsider, if a close contact is unable to deduce the identity of the patient, an outsider would similarly be unable to. Let $\mathcal{A}$ be a polynomial-time adversary, let σ_r be the ID-based ring signature uploaded on BB, and let $Adv_{\mathcal{A}}$ as the probability of finding the identity of the patient. If $\mathcal{A}$ can break the patient privacy, $Adv_{\mathcal{A}} > 1/n$, corresponding to a wild guess, and therefore would break the anonymity property of the ID-based ring signature. We deduce that **Patient Privacy** is maintained if the underlying ID-ring signature scheme's anonymity holds. Note that while an ID-based ring signature is necessary to ensure no **False Positive Case 2** occurs, the anonymity of a ring signature is sufficient to ensure **Patient Privacy**. ■

Theorem 6.3. *The protocol has **Traceability Completeness** if it is correctly executed.*

Proof. As stated in Section 6.1, we consider that no information can be modified after being stored on the app, and must be entirely used in the **Alerting Phase**. Let $L_{ID} = \{ID_1, \dots, ID_n\}$ be the list of identifiers stored on the smartphone of the patient, and i be its index in L_{ID} (i.e., user i). If the protocol is correctly executed, user i generates an ID-based ring signature on the message $M = L_{ID}$ and uploads it on BB. During the **Tracing Phase**, a close contact of the patient, corresponding to user $j \in [1, n] \setminus i$ will retrieve σ_r and check its validity. As both the patient and the user are close contacts and the protocol was correctly executed, σ_r is successfully verified and the close contact is informed they have been in contact with a patient, which concludes the proof of **Traceability Completeness**.

6.4 Comparison

In this section, we compare the characteristics, computations, and communications of PF-MP Tracing, OF-MP Tracing and Primitive Multi-Party Tracing Protocol (Primitive) introduced in Section 3.4, in Table 6.1 from the point of view of **Meeting Phase**. As in the rest of the protocol, we consider a meeting between n participants, $n \geq 2$. We have shown in Section 6.3 that all three protocols achieve Patient Privacy and False Positive Case 1 and 2. Moreover, Primitive Multi-Party protocol achieves Contact Privacy by using a zero-knowledge proof.

We first compare the characteristics of the protocols. In both OF-MP Tracing and PF-MP Tracing, a multi-party proof is generated, and the order of the users has no impact on the protocol, so both OF and MP are satisfied. As PF-MP Tracing uses MM-multi-signature scheme, which has the message flexibility property, it satisfies UF as well which is not the case for OF-MP Tracing. As Primitive Multi-Party Tracing will consider the meeting between the n users in pair, all of them independent from one another, both UF and OF are satisfied.

We then compare the protocols in terms of communications and computations. In both PF-MP Tracing and OF-MP Tracing, the communication amount is fixed and does not depend

Table 6.1: Comparison between OF-MP Tracing, PF-MP Tracing and Primitive [LAY⁺20]

	Communications	Computations	Characteristics
OF-MP Tracing	2	$2(n-1)E + (2n+1)P$	OF, MP
PF-MP Tracing	4	$(4n-1)E + 2(n-1)P$	UF, OF, MP
Primitive	$4(n-1)$	$5(n-1)E + 2(n-1)P$	OF, UF

E : Exponentiations P : Pairings

n : number of Participants OF : Order Flexibility

MP : Multi-party Proof UF : User Flexibility

on the number of users, as we use broadcasts, while the number of communications in Primitive Multi-Party Tracing is linear. The number of computations required in PF-MP Tracing and OF-MP Tracing is also less than Primitive Multi-Party Tracing. PF-MP Tracing requires slightly more computations, which is compensated for by its additional user flexibility property.

6.5 Conclusion

In this chapter, we propose a contact tracing protocol in a multi-party setting with two variants, OF-MP Tracing and PF-MP Tracing, depending on the multi-signature used in the **Meeting Phase**. The protocol is divided into four phases and guarantees that any close contact of a Covid-19 patient can be alerted (Traceability Completeness) while ensuring no false positives occur. In order to guarantee the information stored in the app reflects the actual situation, a multi-party proof is generated during the **Meeting Phase** and stored in the app together with information of the close contacts, ensuring a differentiation between a meeting between several participants and a repetition of smaller meetings. Moreover, our protocol satisfies the privacy of the patient by taking advantage of the anonymity property of ring signatures during the **Tracing Phase**.

Chapter 7

Conclusion and Future Works

7.1 Summary of our Results

Digital signatures are a crucial component of modern cryptographic protocols, ensuring the authenticity, integrity, and non-repudiation of messages. Over the years, numerous specialized signature schemes have been proposed to extend the functionality of digital signatures, addressing specific needs. Among them, multi-signatures are used when a group of signers wishes to output a shared signature on a common message, proving that all members of the group participated in the generation of the multi-signature. They are mostly used in contexts such as federated learning and blockchains, and are a part of the cryptocurrency Bitcoin following its Taproot upgrade. Proxy signatures are used when an (original) signer wishes to delegate its signing right to a proxy signer, ensuring that operations can continue even in the absence of the designated signer. They have their use in e-commerce or distributed systems, or in company settings. In this dissertation, we focus on proposing new variants of those two signature schemes using isogeny-based cryptography, taking advantage of its post-quantum security and its relatively low key and signature sizes. Our proposal of an isogeny-based multi-signature is given in chapter 4, while our proposal of an isogeny-based proxy signature is given in chapter 5. Additionally, we take advantage of the compactness of multi-signatures compared to digital signatures, and of the anonymity and underlying authority-based control of ID-based ring signatures to propose a new efficient multi-party contact tracing protocol in chapter 6.

Both our multi-signature and proxy signature are based on the signature CSI-SharK for its underlying structure and flexibility. While our multi-signature is proven EUF-CMA in the Random Oracle model, our proxy signature comes in two variants, respectively EUF-CMA-CWA and EUF-CMA-PKE, which are both proven in the Quantum Random Oracle Model. Both our proposals satisfy the properties of the corresponding scheme, such as security in the rogue key attack model and plain public key model for multi-signatures, and prevention of misuse and strong identifiability for proxy signatures. Similarly, our contact tracing protocol satisfies properties such as traceability completeness, no false positives, and patient privacy. Additionally, we introduce new security properties throughout this dissertation: certificate-signature binding for proxy signatures, making sure that the certificate and the proxy signature

are bound together to prevent any mix-and-match issue, and multi-party proof, making sure that the information stored in the contact tracing app accurately corresponds to reality, which is not limited to pairwise interactions.

To achieve the multi-party proof property, we use a multi-signature scheme, making sure information about all participating users is stored while solving the overhead issue at the same time. While we propose two variants using a pairing-based multi-signature and an ECDH-based multi-signature, the protocol could be easily adapted to Post-Quantum Cryptography by replacing the multi-signature and the ID-based ring signature with suitable post-quantum schemes. Note that while this includes our isogeny-based multi-signature protocol, the contact tracing protocol would need to be modified to account for the lack of ID-based ring signatures in isogeny-based cryptography.

7.2 Future Works

In this dissertation, we propose an isogeny-based multi-signature, an isogeny-based proxy signature, and a contact tracing protocol. We believe that both our signature schemes are meaningful contributions to the development of isogeny-based cryptography as an alternative to other post-quantum schemes, by either proposing the first of its kind using isogenies in the case of multi-signatures or enhancing the security of existing ones in the case of proxy signatures. Similarly, our contact tracing protocol proposes a novel use of specialized signatures such as multi-signatures and ID-based ring signatures to achieve the desired properties.

However, significant open problems remain that are directly relevant to this dissertation. While the signature CSI-SharK on which our multi-signature protocol is based is unforgeable in the Quantum Random Oracle Model, the multi-signature protocol in its current version has only been proven unforgeable in the Random Oracle Model. Additionally, while the necessity of multi-party group action computations in isogeny-based multi-signatures makes the use of less flexible signatures, such as SQISign or PRISM, impossible to our knowledge, proposing a proxy signature using PRISM as a basis would significantly lower the computation time and memory size of the scheme. Finally, significant room for improvement remains for our contact tracing protocol, such as satisfying close-contact privacy by employing zero-knowledge techniques and proposing an ID-based ring signature using isogenies, which could then be incorporated in the protocol.

List of Publications

Journals

1. Mathieu de Goyon and Atsuko Miyaji. Efficient Multi-Party Contact Tracing. *Journal of Information Processing*, pages 878-887, 2022. (Ref. [dGMT22])
2. Mathieu de Goyon and Atsuko Miyaji. Actively Secure Isogeny-based Multi-Signature Scheme. *Journal of Information Processing*, pages 594-607, 2025. (Ref. [dGM25]),

International Conferences (Refereed Papers)

1. Mathieu de Goyon, Atsuko Miyaji and Tian Yangguang. Efficient Multi-Party Contact Tracing. *Ninth International Symposium on Computing and Networking (CANDAR2021)*, pages 10-18, 2021 (Ref. [dGMT21a]).
2. Mathieu de Goyon and Atsuko Miyaji. Isogeny-based Multi-Signature Scheme. *Information Security Practice and Experience: 18th International Conference, (ISPEC 2023)*, pages 477-491, 2023 (Ref. [dGM23]).
3. Mathieu de Goyon and Atsuko Miyaji. Isogeny-based Proxy Signature. *Twelfth International Symposium on Computing and Networking Workshops (CANDARW2024)*, pages 252-258, 2024 (Ref. [dGM24a]).

Domestic Conferences (No Review)

1. Mathieu de Goyon, Atsuko Miyaji and Tian Yangguang. Generalization of Contact Tracing. *186th Multimedia Communication and Distributed Processing/92nd Computer Security Joint Research Symposium (186DPS, 92CSEC)*, 2021 (Ref. [dGMT21b]).
2. Mathieu de Goyon and Atsuko Miyaji. Proposed Isogeny Multi-signature. *The 41st Symposium on Cryptography and Information Security (SCIS2024)*, 2024 (Ref. [dGM24b]).

References

- [AAA⁺25] Marius A. Aardal, Gora Adj, Diego F. Aranha, Andrea Basso, Isaac Andrés Canales Martínez, Jorge Chávez-Saab, Maria Corte-Real Santos, Pierrick Dartois, Luca De Feo, Max Duparc, Jonathan Komada Eriksen, Tako Boris Fouotsa, Décio Luiz Gazzoni Filho, Basil Hess, David Kohel, Antonin Leroux, Patrick Longa, Luciano Maino, Michael Meyer, Kohei Nakagawa, Hiroshi Onuki, Lorenz Panny, Sikhar Patranabis, Christophe Petit, Giacomo Pope, Krijn Reijnders, Damien Robert, Francisco Rodríguez-Henríquez, Sina Schaeffler, and Benjamin Wesolowski. SQIsign. Technical report, National Institute of Standards and Technology, 2025.
- [ABCP22] Shahla Atapoor, Karim Baghery, Daniele Cozzo, and Robi Pedersen. Csi-shark: Csi-fish with sharing-friendly keys. *IACR Cryptol. ePrint Arch.*, page 1189, 2022.
- [ABCP23a] Shahla Atapoor, Karim Baghery, Daniele Cozzo, and Robi Pedersen. Practical robust DKG protocols for CSIDH. In *Applied Cryptography and Network Security - 21st International Conference, ACNS, 2023*, volume 13906 of *Lecture Notes in Computer Science*, pages 219–247. Springer, 2023.
- [ABCP23b] Shahla Atapoor, Karim Baghery, Daniele Cozzo, and Robi Pedersen. VSS from distributed ZK proofs and applications. In *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, 2023*, volume 14438 of *Lecture Notes in Computer Science*, pages 405–440. Springer, 2023.
- [BBC⁺19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear pcps. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, 2019*, volume 11694 of *Lecture Notes in Computer Science*, pages 67–97. Springer, 2019.
- [BBC⁺25] Andrea Basso, Giacomo Borin, Wouter Castryck, Maria Corte-Real Santos, Riccardo Invernizzi, Antonin Leroux, Luciano Maino, Frederik Vercauteren, and Benjamin Wesolowski. PRISM: simple and compact identification and signatures from large prime degree isogenies. In *Public-Key Cryptography - PKC 2025 - 28th IACR International Conference on Practice and Theory of Public-Key Cryptography*, volume 15676 of *Lecture Notes in Computer Science*, pages 300–332. Springer, 2025.

-
- [BCP21] Karim Bagheri, Daniele Cozzo, and Robi Pedersen. An isogeny-based ID protocol using structured public keys. In *Cryptography and Coding - 18th IMA International Conference, IMACC, 2021*, volume 13129 of *Lecture Notes in Computer Science*, pages 179–197. Springer, 2021.
- [BDF⁺10] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. *IACR Cryptol. ePrint Arch.*, page 428, 2010.
- [BDF⁺24] Andrea Basso, Pierrick Dartois, Luca De Feo, Antonin Leroux, Luciano Maino, Giacomo Pope, Damien Robert, and Benjamin Wesolowski. Ssign2d-west - the fast, the small, and the safer. In *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security*, volume 15486 of *Lecture Notes in Computer Science*, pages 339–370. Springer, 2024.
- [BDK⁺23] Ward Beullens, Samuel Dobson, Shuichi Katsumata, Yi-Fu Lai, and Federico Pintore. Group signatures and more from isogenies and lattices: generic, simple, and efficient. *Des. Codes Cryptogr.*, 91(6):2141–2200, 2023.
- [BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, 2018*, volume 11273 of *Lecture Notes in Computer Science*, pages 435–464. Springer, 2018.
- [BDPV21] Ward Beullens, Lucas Disson, Robi Pedersen, and Frederik Vercauteren. Csi-rashi: Distributed key generation for CSIDH. In *Post-Quantum Cryptography - 12th International Workshop, PQCrypto, 2021*, volume 12841 of *Lecture Notes in Computer Science*, pages 257–276. Springer, 2021.
- [Beu19] Ward Beullens. Csi-fish github repository. <https://github.com/KULeuven-COSIC/CSI-FiSh>, 2019.
- [BKV19] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. Csi-fish: Efficient isogeny based signatures through class group computations. *IACR Cryptol. ePrint Arch.*, page 498, 2019.
- [BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS, 2006*, pages 390–399. ACM, 2006.
- [BPW03] Alexandra Boldyreva, Adriana Palacio, and Bogdan Warinschi. Secure proxy signature schemes for delegation of signing rights. *IACR Cryptol. ePrint Arch.*, page 96, 2003.

-
- [BTT22] Cecilia Boschini, Akira Takahashi, and Mehdi Tibouchi. Musig-l: Lattice-based multi-signature with single-round online phase. In *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, 2022*, volume 13508 of *Lecture Notes in Computer Science*, pages 276–305. Springer, 2022.
- [CA89] David Chaum and Hans Van Antwerpen. Undeniable signatures. In *Advances in Cryptology - CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 212–216. Springer, 1989.
- [CD22] Wouter Castryck and Thomas Decru. An efficient key recovery attack on SIDH (preliminary version). *IACR Cryptol. ePrint Arch.*, page 975, 2022.
- [Che23] Yanbo Chen. Dualms: Efficient lattice-based two-round multi-signature with trapdoor-free simulation. In *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, 2023*, volume 14085 of *Lecture Notes in Computer Science*, pages 716–747. Springer, 2023.
- [CLM⁺18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. *IACR Cryptol. ePrint Arch.*, page 383, 2018.
- [CLP24] Mingjie Chen, Antonin Leroux, and Lorenz Panny. SCALLOP-HD: group action from 2-dimensional isogenies. In *Public-Key Cryptography - PKC 2024 - 27th IACR International Conference on Practice and Theory of Public-Key Cryptography*, volume 14603 of *Lecture Notes in Computer Science*, pages 190–216. Springer, 2024.
- [Cou06] Jean Marc Couveignes. Hard homogeneous spaces. *IACR Cryptol. ePrint Arch.*, page 291, 2006.
- [CS20] Daniele Cozzo and Nigel P. Smart. Sashimi: Cutting up csi-fish secret keys to produce an actively secure distributed signing protocol. In *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020, Paris, France, April 15-17, 2020, Proceedings*, volume 12100 of *Lecture Notes in Computer Science*, pages 169–186. Springer, 2020.
- [CYH05] Sherman S. M. Chow, Siu-Ming Yiu, and Lucas Chi Kwong Hui. Efficient identity based ring signature. In *Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings*, volume 3531 of *Lecture Notes in Computer Science*, pages 499–512, 2005.
- [DEF⁺25] Pierrick Dartois, Jonathan Komada Eriksen, Tako Boris Fouotsa, Arthur Herlédan Le Merdy, Riccardo Invernizzi, Damien Robert, Ryan Rueger, Frederik Vercauteren, and Benjamin Wesolowski. PEGASIS: practical effective class group action using 4-dimensional isogenies. *IACR Cryptol. ePrint Arch.*, page 401, 2025.

-
- [DF24] Max Duparc and Tako Boris Fouotsa. Sqiprime: A dimension 2 variant of sqisignhd with non-smooth challenge isogenies. In *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security*, volume 15486 of *Lecture Notes in Computer Science*, pages 396–429. Springer, 2024.
- [DFMS19] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the fiat-shamir transformation in the quantum random-oracle model. *IACR Cryptol. ePrint Arch.*, page 190, 2019.
- [dGM23] Mathieu de Goyon and Atsuko Miyaji. Isogeny-based multi-signature scheme. In *Information Security Practice and Experience - 18th International Conference, ISPEC, 2023*, volume 14341 of *Lecture Notes in Computer Science*, pages 477–491. Springer, 2023.
- [dGM24a] Mathieu de Goyon and Atsuko Miyaji. Isogeny-based proxy signature. In *Twelfth International Symposium on Computing and Networking, CANDAR, 2024*, pages 252–258. IEEE, 2024.
- [dGM24b] Mathieu de Goyon and Atsuko Miyaji. Proposed isogeny multi-signature. In *Symposium on Cryptography and Information Security 2024, SCIS, 2024*, 2024.
- [dGM25] Mathieu de Goyon and Atsuko Miyaji. Actively secure isogeny-based multi-signature. *J. Inf. Process.*, X:594–607, 2025.
- [dGMT21a] Mathieu de Goyon, Atsuko Miyaji, and Yangguang Tian. Efficient multi-party contact tracing. In *Ninth International Symposium on Computing and Networking, CANDAR2021*, pages 10–18. IEEE, 2021.
- [dGMT21b] Mathieu de Goyon, Atsuko Miyaji, and Yangguang Tian. Generalization of contact tracing. In *Computer Security Research Group 2021, CSEC, 2021*, 2021.
- [dGMT22] Mathieu de Goyon, Atsuko Miyaji, and Yangguang Tian. Efficient multi-party contact tracing. *J. Inf. Process.*, 30:878–887, 2022.
- [DGNW20] Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. Pixel: Multi-signatures for consensus. In *USENIX*, pages 2093–2110, 2020.
- [DKSD24] Kunal Dey, Somnath Kumar, Vikas Srivastava, and Sumit Kumar Debnath. A new approach to delegate signing rights to proxy signers using isogeny-based cryptography. *CoRR*, abs/2407.13318, 2024.
- [DLL⁺17] Léo Ducas, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS - dilithium: Digital signatures from module lattices. *IACR Cryptol. ePrint Arch.*, page 633, 2017.

-
- [DLRW24] Pierrick Dartois, Antonin Leroux, Damien Robert, and Benjamin Wesolowski. Squisignhd: New dimensions in cryptography. In *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 14651 of *Lecture Notes in Computer Science*, pages 3–32. Springer, 2024.
- [Feo17] Luca De Feo. Mathematics of isogeny based cryptography. *CoRR*, abs/1711.04062, 2017.
- [FFK⁺23] Luca De Feo, Tako Boris Fouotsa, Péter Kutas, Antonin Leroux, Simon-Philipp Merz, Lorenz Panny, and Benjamin Wesolowski. SCALLOP: scaling the csi-fish. *IACR Cryptol. ePrint Arch.*, page 58, 2023.
- [FG19] Luca De Feo and Steven D. Galbraith. Seasign: Compact isogeny signatures from class group actions. In *EUROCRYPT (3)*, volume 11478 of *Lecture Notes in Computer Science*, pages 759–789. Springer, 2019.
- [FGHP08] Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Ostergaard Pedersen. Practical short signature batch verification. *IACR Cryptol. ePrint Arch.*, 2008:15, 2008.
- [FJP14] Luca De Feo, David Jao, and Jerome Plut. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *J. Math. Cryptol.*, 8(3):209–247, 2014.
- [FKL⁺20] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. Squisign: Compact post-quantum signatures from quaternions and isogenies. In *ASIACRYPT (1)*, volume 12491 of *Lecture Notes in Computer Science*, pages 64–93. Springer, 2020.
- [FM19] Luca De Feo and Michael Meyer. Threshold schemes from isogeny assumptions. *IACR Cryptol. ePrint Arch.*, page 1288, 2019.
- [Gvi20] Yaron Gvili. Security analysis of the COVID-19 contact tracing specifications by apple inc. and google inc. *IACR Cryptol. ePrint Arch.*, 2020:428, 2020.
- [Has20] Farkhondeh Hassandoust. Not just complacency: why people are reluctant to use covid-19 contact-tracing apps, 2020.
- [HS04] Javier Herranz and Germán Sáez. Revisiting fully distributed proxy signature schemes. In *Progress in Cryptology - INDOCRYPT 2004*, volume 3348 of *Lecture Notes in Computer Science*, pages 356–370. Springer, 2004.
- [IN83] K. Itakura and K. Nakamura. A public key cryptosystem suitable for digital multisignatures. In *NEC J. Res. Dev.*, pages 1–8. NEC Research and Development, 1983.

-
- [JF11] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *PQCrypto*, volume 7071 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2011.
- [KL20] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Third Edition*. CRC Press, 2020.
- [KLPT14] David Kohel, Kristin E. Lauter, Christophe Petit, and Jean-Pierre Tignol. On the quaternion l-isogeny path problem. *IACR Cryptol. ePrint Arch.*, page 505, 2014.
- [KLS18] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of fiat-shamir signatures in the quantum random-oracle model. In *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 10822 of *Lecture Notes in Computer Science*, pages 552–586. Springer, 2018.
- [KPW97] Seungjoo Kim, Sangjoon Park, and Dongho Won. Proxy signatures, revisited. In *Information and Communication Security, First International Conference, ICICS’97*, volume 1334 of *Lecture Notes in Computer Science*, pages 223–232. Springer, 1997.
- [LAY⁺20] Joseph K. Liu, Man Ho Au, Tsz Hon Yuen, Cong Zuo, Jiawei Wang, Amin Sakzad, Xiapu Luo, and Li Li. Privacy-preserving COVID-19 contact tracing app: A zero-knowledge proof approach. *IACR Cryptol. ePrint Arch.*, 2020:528, 2020.
- [LKK01] Byoungcheon Lee, Heesun Kim, and Kwangjo Kim. Strong proxy signature and its applications. In *Symposium on Cryptography and Information Security 2002, SCIS 2001, Oiso, Japan, January 23-26, 2001*, 2001.
- [Lyu09] Vadim Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*, pages 598–616. Springer, 2009.
- [MM00] Shirow Mitomi and Atsuko Miyaji. A multisignature scheme with message flexibility, order flexibility and order verifiability. In *Information Security and Privacy, 5th Australasian Conference, ACISP 2000, Brisbane, Australia, July 10-12, 2000, Proceedings*, volume 1841 of *Lecture Notes in Computer Science*, pages 298–312. Springer, 2000.
- [MOY04] Tal Malkin, Satoshi Obana, and Moti Yung. The hierarchy of key evolving signatures and a characterization of proxy signatures. In *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 306–322. Springer, 2004.

-
- [MPSW18] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. *IACR Cryptol. ePrint Arch.*, page 68, 2018.
- [MUO96] Masahiro Mambo, Keisuke Usuda, and Eiji Okamoto. Proxy signatures for delegating signing operation. In Li Gong and Jacques Stearn, editors, *CCS '96, Proceedings of the 3rd ACM Conference on Computer and Communications Security, 1996*, pages 48–57. ACM, 1996.
- [NOC⁺24] Kohei Nakagawa, Hiroshi Onuki, Wouter Castryck, Mingjie Chen, Riccardo Invernizzi, Gioella Lorenzon, and Frederik Vercauteren. Squisign2d-east: A new signature scheme using 2-dimensional isogenies. In *Advances in Cryptology - ASIACRYPT 2024 - 30th International Conference on the Theory and Application of Cryptology and Information Security*, volume 15486 of *Lecture Notes in Computer Science*, pages 272–303. Springer, 2024.
- [Rob22] Damien Robert. Breaking SIDH in polynomial time. *IACR Cryptol. ePrint Arch.*, page 1038, 2022.
- [RS06] Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies. *IACR Cryptol. ePrint Arch.*, page 145, 2006.
- [SHY⁺24] Jakub Sowa, Bach Hoang, Advait Yeluru, Steven Qie, Anita Nikolich, Ravishankar K. Iyer, and Phuong Cao. Post-quantum cryptography (PQC) network instrument: Measuring PQC adoption rates and identifying migration pathways. In *IEEE International Conference on Quantum Computing and Engineering, QCE, 2024*, pages 1835–1846. IEEE, 2024.
- [SMP08] Jacob C. N. Schuldt, Kanta Matsuura, and Kenneth G. Paterson. Proxy signatures secure against proxy key exposure. In *Public Key Cryptography - PKC 2008*, volume 4939 of *Lecture Notes in Computer Science*, pages 141–161. Springer, 2008.
- [Sta01] Markus Stadler. Proof systems for general statements about discrete logarithms. pages 1–13, 12 2001.
- [Sto10] Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Adv. Math. Commun.*, 4(2):215–235, 2010.
- [SW07] Hovav Shacham and Brent Waters. Efficient ring signatures without random oracles. In *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 166–180. Springer, 2007.
- [Tan20] Qiang Tang. Privacy-preserving contact tracing: current solutions and open questions. *IACR Cryptol. ePrint Arch.*, 2020:426, 2020.

-
- [TPH⁺20] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salathé, James R. Larus, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, Ludovic Barman, Sylvain Chatel, Kenneth G. Paterson, Srdjan Capkun, David A. Basin, Jan Beutel, Dennis Jackson, Marc Roeschlin, Patrick Leu, Bart Preneel, Nigel P. Smart, Aysajan Abidin, Seda Gurses, Michael Veale, Cas Cremers, Michael Backes, Nils Ole Tippenhauer, Reuben Binns, Ciro Cattuto, Alain Barrat, Dario Fiore, Manuel Barbosa, Rui Oliveira, and José Pereira. Decentralized privacy-preserving proximity tracing. *IEEE Data Eng. Bull.*, 43(2):36–66, 2020.
- [TX14] Shaohua Tang and Lingling Xu. Towards provably secure proxy signature scheme based on isomorphisms of polynomials. *Future Gener. Comput. Syst.*, 30:91–97, 2014.
- [Unr17] Dominique Unruh. Post-quantum security of fiat-shamir. In *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, 2017*, volume 10624 of *Lecture Notes in Computer Science*, pages 65–95. Springer, 2017.
- [Vau20a] Serge Vaudenay. Analysis of DP3T. *IACR Cryptol. ePrint Arch.*, 2020:399, 2020.
- [Vau20b] Serge Vaudenay. Centralized or decentralized? the contact tracing dilemma. *IACR Cryptol. ePrint Arch.*, 2020:531, 2020.
- [WBZ⁺22] Zecheng Wang, Huaizhong Bao, Tao Zhao, Faze Du, and Xuemin Chen. New lattice-based short proxy signature scheme with fewer system parameters in the standard model. *IEEE Access*, 10:57180–57191, 2022.
- [XZL21] Yue Xiao, Peng Zhang, and Yuhong Liu. Secure and efficient multi-signature schemes for fabric: An enterprise blockchain platform. *IEEE Trans. Inf. Forensics Secur.*, 16:1782–1794, 2021.
- [ZSL03] Fangguo Zhang, Reihaneh Safavi-Naini, and Chih-Yin Lin. New proxy signature, proxy blind signature and proxy ring signature schemes from bilinear pairing. *IACR Cryptol. ePrint Arch.*, page 104, 2003.