PhD-FSTM-2022-003
The Faculty of Sciences, Technology and Medicine

DISSERTATION

Defence held on 26/01/2022 in Esch-sur-Alzette

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Giuseppe VITTO

SECURITY, SCALABILITY AND PRIVACY IN APPLIED CRYPTOGRAPHY

Dissertation defence committee

Dr Alex Biryukov, dissertation supervisor
Professor, Université du Luxembourg

Dr Jean-Sébastien Coron, Chairman
Professor, Université du Luxembourg

Dr Nadia Heninger,
Associate Professor, University of California, San Diego

Dr Volker Müller, Vice Chairman
Associate Professor, Université du Luxembourg

Dr Frederik Vercauteren,
Associate Professor, KU Leuven

Abstract

In the modern digital world, cryptography finds its place in countless applications. However, as we increasingly use technology to perform potentially sensitive tasks, our actions and private data attract, more than ever, the interest of ill-intentioned actors.

Due to the possible privacy implications of cryptographic flaws, new primitives' designs need to undergo rigorous security analysis and extensive cryptanalysis to foster confidence in their adoption. At the same time, implementations of cryptographic protocols should scale on a global level and be efficiently deployable on users' most common devices to widen the range of their applications.

This dissertation will address the security, scalability and privacy of cryptosystems by presenting new designs and cryptanalytic results regarding blockchain cryptographic primitives and public-key schemes based on elliptic curves. In Part I, I will present the works I have done in regards to accumulator schemes. More precisely, in Chapter 2, I cryptanalyze Au *et al.* [Au+09] Dynamic Universal Accumulator, by showing some attacks which can completely take over the authority who manages the accumulator. In Chapter 3, I propose a design for an efficient and secure accumulator-based authentication mechanism, which is scalable, privacy-friendly, lightweight on the users' side, and suitable to be implemented on the blockchain.

In Part II, I will report some cryptanalytical results on primitives employed or considered for adoption in top blockchain-based cryptocurrencies. In particular, in Chapter 4, I describe how the zero-knowledge proof system and the commitment scheme adopted by the privacy-friendly cryptocurrency Zcash [Zca], contain multiple subliminal channels which can be exploited to embed several bytes of tagging information in users' private transactions. In Chapter 5, instead, I report the cryptanalysis of the Legendre PRF [Dam90], employed in a new consensus mechanism considered for adoption by the blockchain-based platform Ethereum [Woo14], and attacks for further generalizations of this pseudo-random function, such as the Higher-Degree Legendre PRF, the Jacobi Symbol PRF, and the Power-Residue PRF.

Lastly, in Part III, I present my line of research on public-key primitives based on elliptic curves. In Chapter 6, I will describe a backdooring procedure for primes so that whenever they appear as divisors of a large integer, the latter can be efficiently factored. This technique, based on elliptic curves Complex Multiplication theory, enables to eventually generate non-vulnerable certifiable semiprimes with unknown factorization in a multi-party computation setting, with no need to run a statistical semiprimality test common to other protocols. In Chapter 7, instead, I will report some attack optimizations and specific implementation design choices that allow breaking a reduced-parameters instance, proposed by Microsoft, of SIKE [Jao+20], a post-quantum key-encapsulation mechanism based on isogenies between supersingular elliptic curves.

Acknowledgements

I want to express my sincere gratitude to all the people without whom this dissertation would not have been possible.

First of all, I thank Prof. Alex Biryukov for supervising my doctoral studies, supporting my research, and providing me with all the resources I needed to accomplish my work.

I further thank Prof. Jean-Sébastien Coron and Prof. Volker Müller, for the discussions and valuable advice received while following and encouraging my research as thesis supervision committee members.

I am also grateful to Prof. Nadia Heninger and Prof. Frederik Vercauteren for agreeing to serve as jury members during my defense.

I would like to thank the Luxembourg National Research Fund (FNR) for financially supporting the FinCrypt project (C17/IS/11684537), and, through it, all my research, and I thank Anne Ochsenbein for the project coordination.

I am thankful to the University of Luxembourg and the Interdisciplinary Centre for Security, Reliability, and Trust for providing me with an excellent research environment.

I thank all my co-authors for the fruitful exchanges, discussions, and reciprocal support: Aleksei Udovenko, Alex Biryukov, Daniel Feher Luan Cardoso dos Santos, Tim Beyne, Vesselin Velichkov, Ward Beullens.

I am grateful to all my other CryptoLUX colleagues for the enjoyable time spent together and for the countless coffees: Brian Shaft, Christof Beierle, Dag Arne Osvik, Johann Großschädl, Qingju Wang, Sergei Tikhomirov.

I further thank all the people that, in various ways, provided valuable insights and meaningful suggestion to my research: Alice Silverberg, Cyprien Delpech de Saint Guilhem, Dan Boneh, Nadia Heninger and Younes Talibi Alaoui.

I am finally grateful to the University secretaries and to the Doctoral School for their administrative support: Catherine Violet, Cécilia Messager, Fabienne Schmitz, Ida Ienna, Jessica Giro, Naira Bardasaryan.

Contents

Chapter 1

Introduction

1.1 Introduction

When we make a phone call, receive an e-mail, or surf the web looking for the best restaurant in the neighborhood, we are unconsciously using the result of years of advances in *cryptography*.

Cryptography, whose name originates from the two ancient Greek words $\kappa\rho\upsilon\pi\tau\acute{o}\varsigma$ "hidden" and $\gamma\rho\acute{\alpha}\varphi\varepsilon\iota\nu$ "to write", consists of all those methods that can hide a certain piece of information from all but the intended recipients.

The first known use of (a primitive form of) cryptography dates back to a few millennia when Egyptian scribes started adopting some non-standard hieroglyphs with the probable intent to confuse and intrigue the reader rather than hide any meaningful message.

In the successive centuries, cryptography mainly consisted of increasingly more sophisticated methods to convert written messages into unreadable forms. A classic example is the famous Caesar's cipher, named after the Roman emperor Julius Caesar who used it for his private correspondence: here, a written message is *encrypted* by replacing each letter with the one located at some fixed number of positions down in the alphabet.

However, as we intend it today, cryptography was born between the two world wars. Arguably, the introduction of electrical and mechanical cipher machines, the urgent need to decrypt enemies' secret communications while safely exchanging military information were the elements that accelerated unprecedented progress in this new rising science. With Claude Shannon's foundational works *"A mathematical Theory of Communication"* [Sha48] and the subsequent *"Communication theory of secrecy systems"* [Sha49], cryptography is for the first time formalized from an information-theory point of view, opening to a more rigorous mathematical treatment of *ciphers*.

A cipher consists of two sub-primitives: an encryption and a decryption algorithm. Encryption takes a *plaintext* and a *secret* or *public key*, and returns a *ciphertext*.

Decryption, on the contrary, uses a *secret key* to reveal the original plaintext message from a given ciphertext.

When encryption and decryption keys coincide, the cipher is said to be a *symmetric* cipher. If instead encryption employs a public key, which differs from the decryption key to prevent anyone from decrypting messages, we say that the cipher belongs to the class of *public-key* or *asymmetric* cryptosystems: in fact, this classification applies to many other primitives as well.

Security of modern ciphers and, more in general, cryptosystems, relies on designs motivated by some *security assumptions*, that is, mathematical problems believed to be hard to be algorithmically solved anytime soon. When the security of a cryptosystem is *proven under* a particular security assumption, it means that it is formally proven that in the case there exists an algorithm that can break the cryptosystem, then this algorithm can be used to solve the underlying mathematical problem directly.

The art of disproving security assumptions and finding weaknesses in cryptographic designs is called *cryptanalysis*, which not only works towards lowering the security of existing cryptosystems using the tools provided by mathematics, computer science, and physics but also forces cryptography to progress into more and more secure designs.

It turns out that cryptographic systems which resisted for a long time the evolution of cryptanalysis are the ones on which we can rely to secure *today* digital world. However, what can we say about *tomorrow*?

To give an example, the expected advent of quantum computers might represent a game-changer for cryptanalysts. Due to the existence of quantum algorithms specifically designed for these machines (but which is not possible to implement yet), the majority of current well-established *public-key cryptosystems* might be broken.

On the other hand, cryptographers batten down the hatches by designing new schemes which appear to be *post-quantum resistant*, but only with time (and more cryptanalysis!) we can build confidence in them.

In other words, the future of cryptography, and consequently of our digital world, will be shaped by the interplay between more and more secure designs and the advances of cryptanalysis.

1.2 New Challenges

One of the reasons we need encryption is to address a basic form of *privacy*: just protecting private information from prying eyes is not enough to guarantee, for example, that we are communicating with whom we originally intended, or that the communication itself, by just the fact that it is happening, does not reveal something about the participants involved. While the first of these problems can be easily addressed by adopting specialized cryptographic primitives to *authenticate* exchanged information, the second requires a more sophisticated protocol capable of hiding parties' encrypted messages among random constantly-exchanged ones: a much less trivial task to implement in practice.

In fact, modern cryptography is not only about encryption, as opposed to how it was mainly considered in the past: under its hat, we further find signatures schemes, hash functions, pseudo-random number generators, commitments schemes, accumulators, zero-knowledge proof systems, and many others, each addressing a different problem of modern society's complex needs.

With the advent of technology, increasing computers capabilities, and the giant steps made by research, it is now possible to pay for groceries using our watches, video-call a friend thousands of kilometers away from us, or even watch live what is happening on Mars. All these would not be possible without the development of many different new cryptographic primitives, which not only prevent our bank information or family pictures in the cloud to be easily stolen by some ill-intentioned individual, but prominently contribute to ensure the daily functioning of our entire digital world.

However, the urge for leading-edge cryptographic primitives and protocols becomes more and more pressing as we increasingly use technology to perform potentially sensitive tasks: these do not include only financial transactions, private chats, or video calls, but also schooling, working, access to medical records, voting, and so on.

The threat represented by the mass adoption of technology is that our actions and sensitive information may be more easily targeted by attackers who wish to collect, analyze, and ultimately use this data, mainly for profit but often for reasons we cannot immediately predict.

It is not a coincidence that we are observing a rising trend of cyber-attacks aimed at exfiltrating or sabotaging data of companies and individuals, such as financial transactions, business documents, or even software running on industrial machines. To make matters worse, such attackers often take advantage of the features and guarantees provided by modern cryptography to ensure their goals, possibly undetected.

In this context, wide and cautious use of cryptography is paramount, but it is not the only mitigation possible. For example, if a certain service provider collects all its core-business information safely encrypted in a central database, the latter would represent the single point of failure of the whole organization. If instead multiple copies of such database are *distributed* among different servers, the probability that corruptions will have catastrophic consequences for both the business and its customers is drastically reduced.

However, in this example, we still have one more subtlety: what if an attacker completely takes over the company and starts acting maliciously against its customers? In this situation, the fact that there exist multiple copies of our database becomes irrelevant, as well as the fact that the information stored is encrypted: the attacker will have access to the decryption keys, will read all the data in clear, and eventually will modify it.

How can we possibly address this? A solution is offered by *decentralization*: to apply this concept to our example, not only we should distribute the core information as above, but company operations as well. Moreover, we should do so by ensuring that even when multiple end-points –or *nodes*– go malicious, they can agree *all together* only on honest activities. This is indeed a fundamental concept of decentralized networks: each node's action, to be considered valid, needs to be approved through a *consensus mechanism*, which possibly involves all nodes. Thus, as long as most nodes remain honest, the whole network will continue behaving honestly.

In decentralized networks, another relevant aspect is *how* nodes can join: if they need to be *authorized*, there would be more control on nodes and their actions, but the authorization process itself may be attacked to allow a large number of malicious nodes joining. If instead access is *permissionless*, virtually everyone can join the network and contribute to its functionalities –thus making it *fully* decentralized– but consensus becomes harder to reach due to a more complicated attack model.

At this point, our requirements for a decentralized service provider start getting so hard to implement in practice that we may wonder if there exists any *secure* construction which satisfies all of them. Remarkably, the answer is affirmative: Bitcoin

[Nak08] was the first implementation of a global-scale payment system, which adopts a permissionless decentralized peer-to-peer network based on the the *blockchain* technology and a *proof of work* consensus mechanism.

As its name suggests, the blockchain consists of one or multiple chains of blocks, where each block contains some piece of information (in Bitcoin, *signed* financial transactions). To provide blocks a chronological order, each new one includes the hash of the last block it refers to, thus creating multiple linked chains, but where only the longest is considered as the *main* one. In the economic context, such a chain is also called *distributed ledger*, and each network node keeps a complete and updated copy of it.

The great innovation introduced by Bitcoin is its consensus mechanism: a node, to add a new block to the blockchain, needs to include a random value so that the resulting hash of the block has a certain number of leading zeroes. Since such nonce can only be found by brute-force, which requires a considerable commitment of nodes' resources, this process is also called proof of work.

When updating its local copy of the blockchain, an honest node verifies the validity of new blocks and rejects those that do not have a valid proof of work or contain incorrect (or contradicting) information for the chains they are linked to (regardless of the validity of the proof of work). It follows that an attacker who wishes to change the contents of a block in the past needs to re-compute all subsequent blocks' proof of works in order to convince other nodes to accept as valid a different, longer chain, a goal reachable only if he controls the majority of network resources.

In other words, a proof of work, when implemented and properly incentivized (cryptocurrencies usually reward coins to nodes which *mine* valid blocks to the blockchain), can be effective to ensure, globally, honest network operations.

The increasing adoption of the technology behind Bitcoin, however, revealed some shortcomings in its original design: first of all, since all on-chain operations need to be stored in the blockchain in order to be verified by (new) nodes, its size keeps growing as more and more users operate on it. Secondly, blocks' information is not necessarily –intentionally or unintentionally– private, and this may allow attackers to target users' privacy by, for example, tracing their on-chain activities, or in the case of financial applications, by keeping track of coin flows.

While the first of these problems affects *scalability* of the blockchain technology (together with some other consensus-related reasons), *privacy* and *security* issues, instead, may reduce the range of its possible applications.

With no surprise, the research community reacted to these problems by proposing alternative constructions that address one or more of these aspects, with the difficult task of maintaining their implementations accessible to users and their most common devices.

However, the delicacy of some applications, the possible relevant economic incentives involved, and the global-scale implications of security and cryptographic flaws require each new design choice to be motivated by rigorous security analysis and possibly validated by extensive cryptanalysis. Indeed, blockchain as a tool to build secure decentralized networks can benefit modern society by reducing, if not eliminating, the possible impacts of malicious actors. Still, it requires reliable and flexible primitives and protocols to be safely adopted in those applications that will most benefit from it.

Security, scalability, and privacy represent some of the biggest challenges of blockchain technology, and to these aspects, this dissertation is dedicated to a large extent. I will present new designs and cryptanalytic results regarding blockchain cryptographic primitives and propose and cryptanalyze new public-key cryptosystems based

on elliptic curves.

1.3 Thesis Overview

1.3.1 Part I: Accumulators

Accumulator schemes allow aggregating many different elements from a particular set into a short constant-sized digest, called *accumulator value*. Differently than hash functions, accumulators further allow to prove if an element is currently accumulated into a certain accumulator value or not, thanks to special values called *witnesses*.

Some advantages of accumulators are immediately evident: for example, they can be directly applied to implement a white or a black-list, where, by just verifying witnesses with respect to the most updated accumulator value, we can know for several millions of users (but also billions and more) to which of these two lists they currently belong, and with no need to store this information in a database.

In this Part, I present the works I have done with my co-authors on accumulator schemes, aiming at designing an efficient and secure accumulator-based authentication mechanism that possibly is scalable, privacy-friendly, lightweight on the users' side, and suitable to be implemented on the blockchain.

We do so by building upon a combination of previous works, namely Nguyen's positive accumulator [Ngu05], Au *et al.* Dynamic Universal Accumulator [Au+09] and Damgård and Triandopoulos non-membership proof mechanism [DT08], all defined over a prime-order bilinear group. While studying these schemes, we found out that one of the two constructions proposed by Au *et al.* in [Au+09] has serious security weaknesses, which allow a complete takeover of the accumulator. In Chapter 2, we describe such vulnerabilities and we start investigating possible accumulator variants resistant to specific attacks outside Au *et al.* original security model, but relevant in the case the accumulator is used as an authentication mechanism: prevent users from being able to compute witnesses for arbitrary elements, in the case witnesses are issued by a central authority. In our authentication mechanism use-case, this will ensure that users cannot generate on their own valid credentials that were not previously issued by the authority.

In Chapter 3, we define an accumulator scheme in the Accumulator Manager setting based on the above works, but immune to the attacks of Chapter 2. We extend such a scheme by adding support for batch operations, which allow adding and removing elements in batches to and from the accumulator. We further propose and show the security of a public batch witness update protocol to enable users to efficiently update their witnesses by processing some public data distributed on-chain. These new features are designed with privacy in mind: any operations can be executed privately using zero-knowledge protocols, thus protecting users' privacy and making the resulting scheme suitable as a building block of a blockchain-based anonymous credentials system.

The contents of Chapter 2 and Chapter 3 are based, respectively, on [BUV21] and [VB22]. These works were co-authored with Aleksei Udovenko [BUV21] and Alex Biryukov [BUV21; VB22].

1.3.2 Part II: Blockchain Cryptography

In this Part, I present the research I carried out with my co-authors about the security of some cryptographic primitives employed, or considered for adoption, in top blockchain-based cryptocurrencies.

More precisely, in Chapter 4 we detail some attacks targeting Zcash [Zca], one of the most advanced privacy-oriented cryptocurrency that, in contrast to Bitcoin [Nak08], employs encryption, zero-knowledge protocols, and commitment schemes to hide users' transaction information and prevent, as much as possible, on-chain data-mining.

Our attacks, which can be classified in the category of *malicious cryptography* [YY04], focus on the zero-knowledge proof mechanisms and commitment scheme employed by Zcash and exploit malleability of proofs and commitments in order to embed in transactions arbitrary tagging information.

Indeed, at the time of this research, Zcash zero-knowledge proofs were expensive to compute. To allow limited-resources devices, such as smartphones, to transfer coins, developers designed a mechanism where it is possible to delegate proofs generation to untrusted third-party servers and later securely sign them on users' devices to make the corresponding transactions valid.

Within this attack model, we could easily embed 9 bytes of tagging information in an actual Zcash transaction, a possibility that opened to a wide range of users' information leakages, from tracking on-chain activities to the total exposure of transactions amounts.

In Chapter 5, we investigate the security of the Legendre PRF [Dam90], a pseudo-random function proposed at the end of the '80s, that recently gained new attention due to the work of Grassi *et al.* [Gra+16], where it was shown to be very efficient to implement in a *multi-party computation* setting.

This relevant property attracted the interest of Ethereum [Woo14], a blockchain-based cryptocurrency and smart-contract platform, that proposed to use this primitive as the building block of a more efficient *Proof of Custody* consensus mechanism, to be possibly implemented in the new release of the protocol called Ethereum 2.0 [Fei19c].

To build confidence around this re-discovered primitive and attract fresh cryptanalytical interest, the Ethereum Foundation offered some bounties for breaking concrete instances of the Legendre PRF at various security levels. In Chapter 5 we detail a table-based approach that, together with some further optimizations, improved the best attacks for the Legendre PRF known at that time and allowed us to break the first two Ethereum challenge instances of assumed security up to 54 bits.

Besides revising the security of the Legendre PRF, we discuss new attacks for other variants of this PRF described in [Dam90]. These are the higher-degree Legendre PRF, for which we show the existence of a large class of weak keys, the Jacobi PRF, which was erroneously believed to have comparable security to the Legendre PRF but is more efficient to compute, and the Power Residue PRF, which allows a higher PRF bit-throughput than the single bit provided by the original Legendre PRF.

The contents of Chapter 4 are based on [BFV19], written with Daniel Feher and Alex Biryukov, while Chapter 5 is based on [Beu+20], a joint work with Ward Beullens, Tim Beyne and Aleksei Udovenko.

1.3.3 Part III: Public-Key Cryptography

In this Part, I present my line of research on public-key cryptography, with a particular focus on primitives based on elliptic curves.

In Chapter 6, I describe a backdooring procedure for primes, so that when they appear as divisors of a large integer, the latter can be efficiently factored. This technique is based on the theory of Complex Multiplication and generates primes p associated with elliptic curves over $\mathbb{F}_p$ with orders that factor completely over

a certain input factor base. This construction allows a custom variant of Lenstra's Elliptic Curve Factorization Method [Len87] to efficiently factor $N = p \cdot q$, by building a proper complex multiplication elliptic curve over a ring bigger than $\mathbb{Z}_N$ and by executing some XZ-arithmetic on such curve.

Although different alternative backdooring techniques exist under the same attack model, the above prime generation procedure will ultimately allow to efficiently generate (non-maliciously) *semiprimality certificates*. Such certificates are based on a generalization of a result by Goldwasser and Kilian [GK86] and can prove an integer to be the product of exactly two primes, with no need to know, nor disclose, any of its factors.

We will then detail a procedure to compute semiprimality certificates efficiently, and by restating it in a multi-party computation setting, we will be able to generate (certifiable) semiprimes with unknown factorization. In contrast to previous work, we show how to compute in a distributed fashion moduli which are *semiprimes by construction* and thus do not require execution of distributed statistical semiprimality tests, as is common in the other protocols.

The relevance of these results is given by the fact that semiprimes with unknown factorization provide practical unknown-order groups, which, in turn, can be used to implement constructions suitable for decentralized applications. Some examples include: *trapdoorless* accumulators for *stateless* blockchains [BBF19], where blocks' information is accumulated in an accumulator that does not require a trusted central authority; Verifiable Delay Functions (VDFs) [Wes19; Pie19], which are inherently-sequential functions that can be used to instantiate a *proof of sequential work* as alternative to Bitcoin's prone-to-parallelization proof of work; *trustless* zero-knowledge proof systems [BFS20], instantiated using only trapdoor-free parameters.

In Chapter 7, instead, I report the work I have done with my co-author on Supersingular Isogeny Key Encapsulation (SIKE), a post-quantum key encapsulation mechanism whose security is based on the supersingular isogeny problem. In SIKE, two parties independently pick a random secret *isogeny* (that is, a surjective morphism of curves which induces a group homomorphism), and use them to compute, in turn, images of some public points in order to agree on a curve equation, that is the shared secret. Informally, the security of this scheme relies on the fact that it is difficult from the domain and codomain of a random isogeny to retrieve its full definition, which in SIKE would allow recovering the shared secret from parties' exchanged information quickly.

Throughout the Chapter, we will describe a *meet-in-the-middle* approach improved with multiple SIKE-specific optimizations. In these attacks, one party's full isogeny definition is retrieved by iteratively expanding all possible (lower-degrees) isogenies that are compatible with the public domain and codomain of the attacked isogeny until a match *in-the-middle* is found.

Our optimizations allowed us to reduce by a few orders the time and space complexity of practical implementation of these attacks, making it feasible for us to successfully break a reduced-parameters instance of SIKE, proposed by Microsoft and called \$IKEp182, using the University of Luxembourg High-Performance Computing Cluster [Var+14].

The contents of Chapter 6 are based on [Vit21], while Chapter 7 is based on [UV21], a joint work with Aleksei Udovenko.

1.4 Contributions

Conference Proceedings

[Beu+20] Ward Beullens, Tim Beyne, Aleksei Udovenko, and Giuseppe Vitto. "Cryptanalysis of the Legendre PRF and Generalizations". In: *IACR Transactions on Symmetric Cryptology* 2020.1 (2020), pp. 313–330. ISSN: 2519-173X. DOI: 10.13154/tosc.v2020.i1.313-330.

[BFV19] Alex Biryukov, Daniel Feher, and Giuseppe Vitto. "Privacy Aspects and Subliminal Channels in Zcash". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 1813–1830. ISBN: 9781450367479. DOI: 10.1145/3319535.3345663.

[Bir+21] Alex Biryukov, Luan Cardoso dos Santos, Daniel Feher, Vesselin Velichkov, and Giuseppe Vitto. "Automated Truncation of Differential Trails and Trail Clustering in ARX". In: *Selected Areas in Cryptography (to appear)*. Springer International Publishing, 2021. URL: https://eprint.iacr.org/2021/1194.

[BUV21] Alex Biryukov, Aleksei Udovenko, and Giuseppe Vitto. "Cryptanalysis of a Dynamic Universal Accumulator over Bilinear Groups". In: *Topics in Cryptology – CT-RSA 2021*. Ed. by Kenneth G. Paterson. Vol. 12704. Lecture Notes in Computer Science. Reproduced with permission from Springer Nature. Springer, Heidelberg, 2021-05, pp. 276–298. DOI: 10.1007/978-3-030-75539-3_12.

[VB22] Giuseppe Vitto and Alex Biryukov. "Dynamic Universal Accumulator with Batch Update over Bilinear Groups". In: *Topics in Cryptology – CT-RSA 2022*. Ed. by Steven Galbraith. Vol. 13161. Lecture Notes in Computer Science. Reproduced with permission from Springer Nature. Springer, Cham, 2022, pp. 395–426. DOI: 10.1007/978-3-030-95312-6_17.

Unrefereed Publications

[UV21] Aleksei Udovenko and Giuseppe Vitto. *Breaking the $IKEp182 Challenge*. Cryptology ePrint Archive, Report 2021/1421. https://eprint.iacr.org/2021/1421. 2021.

[Vit21] Giuseppe Vitto. *Factoring Primes to Factor Moduli: Backdooring and Distributed Generation of Semiprimes*. Cryptology ePrint Archive, Report 2021/1610. https://eprint.iacr.org/2021/1610. 2021.

Conferences and Talks

I presented the works I co-authored at the following conferences and workshops:

- ACM CCS 2019, London, UK. With Daniel Feher, I presented [BFV19].
- IACR Transactions on Symmetric Cryptology (ToSC) 2020, Virtual. I presented [Beu+20].
- CT-RSA 2021, Virtual. I presented [BUV21].

- Euro S&P 2021: Future of PI Workshop, Virtual. I presented [VB22].
- Selected Areas in Cryptography (SAC) 2021, Virtual. I presented [Bir+21].

Implementations

The source-code of the implementation of the accumulator scheme detailed in Chapter 3, is publicly available on GitHub at:

`https://github.com/cryptolu/accumulator`

The source-code of the prime backdooring procedure and the distributed semiprime generation protocol, both detailed in Chapter 6, can be found, respectively, at:

`https://github.com/cryptolu/primes-backdoor`
`https://github.com/cryptolu/semiprimes`

Part I

Accumulators

In this Part, I will present the research I have done with my co-authors in regards to accumulator schemes defined over bilinear groups.

In Chapter 2, we cryptanalyse the two accumulator variants proposed by Au *et al.* [Au+09], which we call the α-*based* construction and the common reference string-based ($\mathcal{CRS}$-*based*) construction. We will show that when non-membership witnesses are issued according to the α-based construction, an attacker with access to $O(\log p \log\log p)$ witnesses can efficiently recover the secret accumulator parameter α and completely break its security. Further optimizations and different attack scenarios allow reducing the number of required witnesses to $O(\log p)$, together with practical attack complexity. Moreover, we will show that accumulator's collision resistance can be broken if just one of these non-membership witnesses is known to the attacker. We then show how all these attacks for the α-based construction can be easily prevented by using a corrected expression for witnesses instead.

Although outside the original security model assumed by Au *et al.* but motivated by some possible concrete application of the scheme where the Manager must have exclusive rights for issuing witnesses (e.g. white/black list based authentication mechanisms), we will show that if non-membership witnesses are issued using the alternative $\mathcal{CRS}$-based construction and the $\mathcal{CRS}$ is kept secret by the Manager, an attacker accessing multiple witnesses can reconstruct the $\mathcal{CRS}$ and compute witnesses for arbitrary new elements. In particular, if the accumulator is initialized by adding m secret elements, the knowledge of m non-membership witnesses allows succeeding in such attack.

In Chapter 3, instead, we propose a Dynamic Universal Accumulator in the Accumulator Manager setting for bilinear groups which extends Nguyen's positive accumulator [Ngu05], Damgård and Triandopoulos non-membership proof mechanism [DT08] and Au *et al.* [Au+09] Dynamic Universal Accumulator, but immune from the attacks outlined in Chapter 2.

The new features include support for batch addition and deletion operations and a privacy-friendly batch witness update protocol, where the witness update information is the same for all users. Together with a non-interactive zero-knowledge protocol, these will make the proposed scheme suitable as an efficient and scalable Anonymous Credential System, accessible even by low-resource users.

We show the security of the proposed protocol in the Generic Group Model under a (new) generalized version of the t-SDH assumption, and we demonstrate its practical relevance by providing and discussing an implementation realized using state-of-the-art libraries.

Chapter 2

Cryptanalysis of a Au *et al.* Accumulator

2.1 Introduction

A cryptographic accumulator scheme permits aggregating values of a possibly very large set into a short digest, which is commonly referred to as the *accumulator value*. Unlike hash functions, where, similarly, (arbitrary) long data is mapped into a fixed-length digest, accumulator schemes permit to additionally show whenever an element is accumulated or not, thanks to special values called *witnesses*. Depending on the accumulator design, we can have two kinds of witnesses: *membership witnesses*, which permit to show that an element is included in the accumulator, and *non-membership witnesses*, which, on the contrary, permit to show that an element is not included.

Accumulator schemes that support membership witnesses are referred to as *positive* accumulators; the ones that support non-membership witnesses are called *negative*, while the ones that support both are called *universal* accumulators. A common requirement for the accumulator schemes is the ability to change the set of accumulated elements, hence permitting *accumulator updates*: when the accumulator allows to dynamically *add* and *delete* elements, it is said to be a *dynamic accumulator*.

Benaloh and De Mare [Bd94] in 1993 formalized the first accumulator scheme as a time-stamping protocol. Since then, many other accumulator schemes have been proposed, and they play an essential role in various protocols from set membership, authentication to (anonymous) credentials systems and cryptocurrency ledgers. However, there is only a small set of underlying cryptographic assumptions on which such accumulator primitives are based. Currently, three main families of accumulators can be distinguished in literature: schemes designed in groups of unknown order [Bd94; BP97; CL02; LLX07; San99; Lip12; BBF19], schemes designed in groups of known order [Ngu05; DT08; Au+09; CKS09] and hash-based constructions [Mer90; BLL00; BLL02; Cam+08; BC14]. Relevant to this thesis are the schemes belonging to the second of these families, where the considered group is a prime order bilinear group. Moreover, when it comes to Dynamic Universal Accumulators (namely those that support dynamic addition and deletion of members and can maintain both membership and non-membership witnesses), we are down to just a few schemes.

In this Chapter, we cryptanalyse one of the universal schemes proposed for bilinear groups, namely the Dynamic Universal Accumulator by Au *et al.* [Au+09], which is zero-knowledge friendly and stood unscathed for 10 years of public scrutiny. This scheme comes in two variants, which we called the α*-based construction* and the $\mathcal{RS}$*-based construction*, respectively: in the first one, the accumulator manager, who knows the accumulator secret parameter α, keeps the accumulator updated and issues witnesses efficiently to users; in the $\mathcal{RS}$-based construction, instead, accumulator operations are executed by users in a decentralized fashion, by using elements from a special *reference string* (public) set $\mathcal{RS}$. An important remark is that the two construction differs only on how non-membership witnesses are defined and are *interchangeable*: a witness generated with respect to the α-based construction is *valid* for the $\mathcal{RS}$-based construction as well and vice-versa.

For the α-based construction, we show that the non-membership mechanism, designed to allow for more efficiency on the accumulator manager side, has a subtle cryptographic flaw which enables the adversary to efficiently recover the secret of the accumulator manager given just several hundred to few thousand non-membership witnesses (regardless of the number of accumulated elements).

Consequently, the attacker can entirely break the security of the scheme. Moreover, we show that given only *one* non-membership witness generated with this flawed mechanism, it is possible to invalidate the assumed collision resistance property of the accumulator efficiently by creating a membership witness for a non-accumulated element. Despite the presence of a valid security proof in Au *et al.* original paper, this flaw is possible because the provided security reduction covers the non-membership mechanism of the $\mathcal{RS}$-based construction only and it does not take into account the non-membership witness definition given for the α-based construction, which results to be weak.

In the second part of the Chapter we investigate the $\mathcal{RS}$-based variant. Although this construction looks more appropriate for a decentralized setting, the weaknesses we point out for the α-based construction make the latter not implementable, and thus non-membership witnesses should be issued always according to the $\mathcal{RS}$-based construction definition, regardless if the accumulator is managed by a trusted manager

Construction	Scenario	Witnesses	Time	Attack Result
α-*based*	Random-y	$\mathcal{O}(\log p \log\log p)$	$O(\log^2 p)$	Recovery of α
	Random-y	$\mathcal{O}(\log p \log\log p)$	$O((1+\ell/\log\log p)\log^2 p)$	Recovery of α
	Chosen-y	$\mathcal{O}(\log p)$	$O(\ell \log^2 p/\log\log p)$	Recovery of α
	Random-y	1	$\mathcal{O}(1)$	Break Collision Resistance
$\mathcal{RS}$-*based*	Random-y	m	$\mathcal{O}(m^2)$	Issue Witnesses

TABLE 2.1: Time and non-membership witnesses required in our attacks on the Au *et al.* accumulator scheme for both α-based and $\mathcal{RS}$-based construction. In this table, p denotes the order of the underlying bilinear group, m denotes the number of (secret) elements with which the accumulator is initialized, ℓ denotes the number of accumulations that occurred in between the issues of non-membership witnesses. In the $\mathcal{RS}$-based construction, the $\mathcal{RS}$ is unknown to the attacker.

(which can still perform operations efficiently, thanks to the knowledge of the trapdoor α) or publicly by users (through the $\mathcal{RS}$).

Motivated by some concrete applications of the scheme where the Manager must have exclusive rights for issuing witnesses (e.g. white-/black-list based authentication mechanisms), we show that an adversary having access to a sufficient amount of witnesses is able to compute valid witnesses for unauthorized elements even when the accumulator manager keeps secret all the information needed to compute such witnesses, i.e. the $\mathcal{RS}$. In particular, if the accumulator is initialized by adding m secret elements, an attacker that has access to m non-membership witnesses would succeed in reconstructing the $\mathcal{RS}$ and will then become able to issue membership and non-membership witnesses for any accumulated and non-accumulated elements, respectively.

A summary of our attacks with complexities under different scenarios can be found in Table 2.1.

2.1.1 Outline

In Section 2.2 we summarize the notation we will adopt throughout this and the next Chapter. In Section 2.3 we recall both constructions of Au *et al.* Dynamic Universal Accumulator, and we report in Section 2.4 the original security proof and attack model. We then cryptanalyse the α-based construction: in Section 2.5 we show how it is possible to break the assumed collision resistance with just one non-membership witness, while in Section 2.6 we describe our α-recovery Attack, which allows an attacker knowing multiple non-membership witnesses to recover the secret accumulator parameter α. In Section 2.7 we further improve the α-recovery Attack under different attack models, and we discuss the concrete implementation of these attacks in Section 2.8. In Section 2.9 we address another vulnerability affecting the α-based construction, which allows an attacker to recover α with just one non-membership witness, while in Section 2.10 we investigate the security of the alternative $\mathcal{RS}$-based construction under the assumption that users should not be able to issue witnesses independently from the Accumulator Manager, a setting outside the original Au *et al.* security model, but of relevance for next Chapter.

2.2 Notation

In this and in next Chapter, we will analyze accumulator schemes based on prime order bilinear groups $\mathcal{G} = (p, G_1, G_2, G_t, e)$, where the groups G_1, G_2, G_T are of order p and $e : G_1 \times G_2 \to G_T$ is an efficiently computable non-degenerate bilinear map. Following the notation of [GPS08], we say e to be a Type-I pairing if $G_1 = G_2$, while we will call it Type-III pairing if $G_1 \neq G_2$ and there are no efficiently computable isomorphisms between G_1 and G_2.

In the following, we will denote with uppercase Roman letters (e.g. P, V) elements belonging to G_1 and with uppercase Roman letters with a tilde above (e.g. $\tilde{P}, \tilde{Q}$) elements in G_2. The identity points of G_1 and G_2 are denoted with O and $\tilde{O}$, respectively.

Sets are denoted with uppercase letters in calligraphic fonts (e.g. $\mathcal{ACC}, \mathcal{Y}$) while accumulator elements are denoted with (eventually indexed) lowercase Roman letters: y usually denotes the *reference element*, that is the one we take as an example to perform operations, while $y_{\mathcal{S}}$ denotes an element in the set $\mathcal{S}$. Exceptions are the membership and non-membership witness, denoted respectively with w and $\bar{w}$, and the partial non-membership witness d.

Vectors are denoted with Greek capital letters (e.g. Υ, Ω). The vector operation $\langle \Phi, \Psi \rangle$ is the dot product, that is, the sum of the products of the corresponding entries of Φ and Ψ, while $a \circ \Phi$ denotes the usual scalar-vector multiplication where each entry of Φ is multiplied by a.

We also use a convention that sum and the product of a sequence of terms with starting index greater than the ending one are assumed to be equal to $\sum_i^j a_i = 0$ and $\prod_i^j b_i = 1$ when $i > j$.

2.3 Au *et al.* Dynamic Universal Accumulator

In their paper [Au+09], Au and coauthors propose two different constructions for their Dynamic Universal Accumulator, depending on which information is made available to the accumulator managers. The first requires the accumulator's secret parameter α and is suitable for a centralized entity that efficiently updates the accumulator value and issues witnesses to the users. The second, instead, requires a reference string $\mathcal{RS}$ and allows to update the accumulator value and to issue witnesses without learning α, but less efficiently. We will refer to the first one as the *α-based* construction, while we will refer to the latter as the *$\mathcal{RS}$-based* construction.

These two are interchangeable because witnesses can be issued from time to time with one or the other construction. Moreover, we note that all operations done with the reference string $\mathcal{RS}$, can be done more efficiently by using α directly: hence, if the authority which generates α coincides with the accumulator manager, it is more convenient for the latter to use always the secret parameter α to perform operations and thus we will refer to the two constructions mainly to indicate the different defining equations for witnesses (in particular, non-membership witnesses).

2.3.1 The α-based and $\mathcal{RS}$-based constructions

A dynamic universal accumulator consists of 5 main sub-primitives: *Generation*, *Accumulator Update* (Addition, Deletion), *Witness Issuing* (Membership, Non-Membership), *Witness Update* (Membership, Non-Membership), *Witness Verification* (Membership, Non-Membership). In the case of Au *et al.* accumulator scheme, these are detailed as follows, where we differentiate definitions accordingly to the two α-based

and $\mathcal{RS}$-based constructions (if not explicitly stated, each operation refers to both constructions).

Generation. Let $\mathbb{G}$ be a symmetric bilinear group $\mathbb{G} = (p, G_1, G_T, P, e)$ such that $e : G_1 \times G_1 \to G_T$ is a non-degenerate bilinear map, $(G_1, +)$ is an additive group generated by P with identity element $\mathcal{O}$, $(G_T, \cdot)$ is a multiplicative group and $|G_1| = |G_T| = p$ is prime. The secret accumulator parameter α is randomly chosen from $(\mathbb{F}_p)^*$. The set of accumulatable elements is $\mathcal{ACC} = \mathbb{F}_p \setminus \{-\alpha\}$.

- *($\mathcal{RS}$-based construction)* Let t be the maximum number of accumulatable elements. Then the reference string $\mathcal{RS}$ is computed as

$$\mathcal{RS} = \mathcal{RS}_t = \{\, P,\ \alpha P,\ \alpha^2 P,\ \ldots\ ,\ \alpha^t P \,\}$$

Accumulator updates. Accumulator updates consist of single addition and deletion of elements into and from, respectively, the accumulator.

- *(α-based construction)* For any given set $\mathcal{Y}_V \subseteq \mathcal{ACC}$ let $f_V(x) \in \mathbb{F}_p[x]$ represent the polynomial

$$f_V(x) = \prod_{y \in \mathcal{Y}_V} (y + x)$$

  Given the secret accumulator parameter α, we say that an accumulator value $V \in G_1$ accumulates the elements in $\mathcal{Y}_V$ if $V = f_V(\alpha)P$.

  An element $y \in \mathcal{ACC} \setminus \mathcal{Y}_V$ is added to the accumulator value V, by computing $V' = (y + \alpha)V$ and letting $\mathcal{Y}_{V'} = \mathcal{Y}_V \cup \{y\}$. Similarly, an element $y \in \mathcal{Y}_V$ is removed from the accumulator value V, by computing $V' = \frac{1}{(y+\alpha)}V$ and letting $\mathcal{Y}_{V'} = \mathcal{Y}_V \setminus \{y\}$.

- *($\mathcal{RS}$-based construction)* For any given set $\mathcal{Y}_V \subseteq \mathcal{ACC}$ such that $|\mathcal{Y}_V| \leq t$, let $f_V(x) \in \mathbb{F}_p[x]$ represent the polynomial

$$f_V(x) = \prod_{y \in \mathcal{Y}_V} (y + x) = \sum_{i=0}^{|\mathcal{Y}_V|} c_i x^i$$

  Then, the accumulator value V which accumulates the elements in $\mathcal{Y}_V$ is computed using the $\mathcal{RS}$ as $V = \sum_{i=0}^{|\mathcal{Y}_V|} c_i \cdot \alpha^i P$.

Witnesses Issuing. For an accumulated and a non-accumulated element, a corresponding witness (resp. membership, non-membership) has to be issued.

- *(α-based construction)* Given an element $y \in \mathcal{Y}_V$, the *membership witness* $w_{y,V} = C \in G_1$ with respect to the accumulator value V is issued as

$$C = \frac{1}{y + \alpha} V$$

  Given an element $y \in \mathcal{ACC} \setminus \mathcal{Y}_V$, the *non-membership witness* $\bar{w}_{y,V} = (C, d) \in G_1 \times \mathbb{F}_p$ with respect to the accumulator value V is issued[1] as

$$d = \Big(f_V(\alpha) \bmod (y + \alpha) \Big) \bmod p, \qquad C = \frac{f_V(\alpha) - d}{y + \alpha} P$$

[1] We assume that here $f_V(\alpha) = \prod_{y \in \mathcal{Y}_V} (y + \alpha)$ is computed over $\mathbb{Z}$. Alternatively, if this computation is done modulo p, then d would be equal to $f_V(\alpha) \bmod p$ for a large fraction of elements $y \in \mathcal{ACC} \setminus \mathcal{Y}_V$ and α can be easily recovered by factoring $f_V(x) - d$ over $\mathbb{F}_p[x]$.

- ($\mathcal{RS}$-*based construction)* Given an element $y \in \mathcal{Y}_V$, let $c(x) \in \mathbb{F}_p[x]$ be the polynomial such that $f_V(x) = c(x)(y+x)$. Then, the *membership witness* $w_{y,V}$ for y with respect to the accumulator value V is computed using the $\mathcal{RS}$ as $w_{y,V} = c(\alpha)P$.

  Given an element $y \in \mathcal{ACC} \setminus \mathcal{Y}_V$, apply the Euclidean Algorithm to get the polynomial $c(x) \in \mathbb{F}_p[x]$ and the scalar $d \in \mathbb{F}_p$ such that $f_V(x) = c(x)(y+x) + d$. Then, the *non-membership witness* $\bar{w}_{y,V}$ for y with respect to the accumulator value V is computed from the $\mathcal{RS}$ as $w_{y,V} = (c(\alpha)P, d)$.

Witness Update. When the accumulator value changes, users' witnesses are updated accordingly to the following operations:

- **On Addition**: suppose that a certain $y' \in \mathcal{ACC} \setminus \mathcal{Y}_V$ is added into V. Hence the new accumulator value is $V' = (y' + \alpha)V$ and $\mathcal{Y}_{V'} = \mathcal{Y}_V \cup \{y'\}$.

  Then, for any $y \in \mathcal{Y}_V$, $w_{y,V} = C$ is updated with respect to V' by computing

  $$C' = (y' - y)C + V$$

  and letting $w_{y,V'} = C'$.

  If, instead, $y \in \mathcal{ACC} \setminus \mathcal{Y}_V$ with $y \neq y'$, its non-membership witness $\bar{w}_{y,V} = (C, d)$ is updated to $\bar{w}_{y,V'} = (C', d \cdot (y' - y)$, where C' is computed in the same way as in the case of membership witnesses.

- **On Deletion**: suppose that a certain $y' \in \mathcal{Y}_V$ is deleted from V. Hence the new accumulator value is $V' = \frac{1}{y'+\alpha}V$ and $\mathcal{Y}_{V'} = \mathcal{Y}_V \setminus \{y'\}$.

  Then, for any $y \in \mathcal{Y}_V$, $w_{y,V} = C$ is updated with respect to V' by computing

  $$C' = \frac{1}{y' - y}C - \frac{1}{y' - y}V'$$

  and letting $w_{y,V'} = C'$.

  If, instead, $y \in \mathcal{ACC} \setminus \mathcal{Y}_V$, its witness $\bar{w}_{y,V} = (C, d)$ is updated to $\bar{w}_{y,V'} = (C', d \cdot \frac{1}{y'-y}$, where C' is computed in the same way as in the case of membership witnesses.

We note that in both cases, the added or removed element y' has to be public in order to enable other users to update their witnesses.

Verification. A membership witness $w_{y,V} = C$ with respect to the accumulator value V is valid if it verifies the pairing equation $e(C, yP + \alpha P) = e(V, P)$. Similarly, a non-membership witness $\bar{w}_{y,V} = (C, d)$ is valid with respect to V if it verifies $e(C, yP + \alpha P)e(P, P)^d = e(V, P)$.

2.4 Original Security Proof and Attack Scenarios

The security of the above accumulator scheme is intended in terms of *collision resistance*: in [Au+09], this security property is shown under the t-SDH assumption [BB08]. Informally, collision resistance ensures that an adversary has a negligible probability of forging a valid membership witness for a not-accumulated element and, respectively, a non-membership witness for an already accumulated element.

To ease the discussion on the attacks we will detail in the next Sections, we here report both Au *et al.* definition of collision resistance and Boneh and Boyen definition of $t-$SDH assumption.

Definition 2.1. (Collision Resistance [Au+09]) *The Dynamic Universal Accumulator outlined in Section 3.2 is collision resistant if, for any probabilistic polynomial time adversary $\mathcal{A}$ that has access to an oracle $\mathcal{O}$ which returns the accumulator value resulting from the accumulation of the elements of any given input subset of $(\mathbb{F}_p)^*$, the following probabilities*

$$\mathbb{P}\left(\begin{array}{ccc} (\mathbb{G},\alpha,\tilde{Q}) \leftarrow Gen(1^\lambda) & , & (y,C,\mathcal{Y}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathbb{G},\tilde{Q}) \quad : \\ \mathcal{Y} \subset (\mathbb{F}_p)^* & & \wedge\, V = \left(\prod_{y_i\in\mathcal{Y}}(y_i+\alpha)\right)P \ \wedge \\ y \in (\mathbb{F}_p)^* \setminus \mathcal{Y} & & \wedge\, e(C, y\tilde{P}+\tilde{Q}) = e(V,\tilde{P}) \end{array}\right)$$

$$\mathbb{P}\left(\begin{array}{ccc} (\mathbb{G},\alpha,\tilde{Q}) \leftarrow Gen(1^\lambda) & , & (y,C,d,\mathcal{Y}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathbb{G},\tilde{Q}) \quad : \\ \mathcal{Y} \subset (\mathbb{F}_p)^* & & \wedge\, V = \left(\prod_{y_i\in\mathcal{Y}}(y_i+\alpha)\right)P \ \wedge \\ y \in \mathcal{Y} & \wedge & d \neq 0 \qquad\qquad \wedge \\ e(C, y\tilde{P}+\tilde{Q})e(P,\tilde{P})^d = e(V,\tilde{P}) & & \end{array}\right)$$

are both negligible functions in the security parameter λ.

Definition 2.2. ($t-$Strong Diffie-Hellman Assumption [BB08]) *Let $\mathcal{G}$ be a probabilistic polynomial time algorithm that, given a security parameter 1^λ, outputs a bilinear group $\mathbb{G} = (p, G_1, G_2, G_T, P, \tilde{P}, e)$. We say that the $t-$Strong Diffie-Hellman Assumption holds for $\mathcal{G}$ with respect to an $\alpha \leftarrow (\mathbb{F}_p)^*$ if, for any probabilistic polynomial time adversary $\mathcal{A}$ and for every polynomially bounded function $t : \mathbb{Z} \to \mathbb{Z}$, the probability*

$$\mathbb{P}\left(\mathcal{A}(P, \alpha P, \alpha^2 P, ..., \alpha^{t(\lambda)}P, \tilde{P}, \alpha\tilde{P}) = \left(y, \frac{1}{y+\alpha}P\right)\right)$$

is a negligible function in λ for any freely chosen value $y \in \mathbb{F}_p \setminus \{-\alpha\}$.

We can see that in Definition 2.1, the adversary has access to an oracle $\mathcal{O}$ that outputs the accumulator value $V = \left(\prod_{y\in\mathcal{Y}_V}(y+\alpha)\right)P$ for any chosen input set $\mathcal{Y}_V$. Its purpose is to model the information the adversary can eventually get by looking at the published accumulator states. However, in practice, the adversary has no control over the values accumulated by the accumulator manager. This oracle does not make their attacker more powerful when compared to the requirements of the Boneh and Boyen $t-$SDH assumption (where the $\mathcal{RS}_t = \{P, \alpha P, \ldots, \alpha^t P\}$ is directly given to the attacker) due to the following:

Lemma 2.1. *Having access to the oracle $\mathcal{O}$ of Definition 2.1 where sets of size at most t can be queried is equivalent to the knowledge of the set $\mathcal{RS}_t = \{P, \alpha P, ..., \alpha^t P\}$.*

Proof. $(\Rightarrow)$ Let y be a generator of $(\mathbb{F}_p)^*$. Then the polynomials

$$\{1, (y+x), (y+x)\cdot(y^2+x), \ldots, \prod_{i=1}^{t}(y^i+x)\}$$

form a basis for the additive vector space of polynomials in $\mathbb{F}_p[x]$ with degree lower equal t and, hence, for any given $1 \leq i \leq t$, there exists a linear combination of these polynomials that sums up to x^i. It follows that, by iteratively calling $\mathcal{O}$ on the set $\mathcal{Y}_i = \{y, y^2, \ldots, y^i\}$, it is possible to write a linear combination of the $V_{\mathcal{Y}_i}$ values returned which is equal to $\alpha^i P$.

(⇐) Suppose the set $\mathcal{RS}_t$ is known. Then, for any given $\mathcal{Y}_V \subset (\mathbb{F}_p)^*$ with $|\mathcal{Y}_V| \leq t$, using the elements $\mathcal{RS}_t$, it is possible to compute V as

$$V = \left(\prod_{y_i \in \mathcal{Y}_V} (y_i + \alpha) \right) P = \sum_{i=0}^{|\mathcal{Y}|} c_i \cdot (\alpha^i P)$$

□

For completeness, we report Au *et al.* security proof provided in [Au+09, Theorem 2], restated within our notation.

Theorem 2.1. *Let $\mathcal{G}$ be a probabilistic polynomial time algorithm that, given a security parameter 1^λ, outputs a bilinear group $\mathbb{G} = (p, G_1, G_2, G_T, P, \tilde{P}, e)$ and consider an instantiation of the Dynamic Universal Accumulator obtained using $\mathcal{G}$ for the bilinear group generation and $\alpha \in (\mathbb{F}_p)^*$ as the secret accumulator parameter. Then, the accumulator is collision resistant (Definition 2.1) if the $t-$Strong Diffie-Hellman Assumption (Definition 2.2) holds for $\mathcal{G}$ with respect to α.*

Proof. We note that a solution (y, C, d) for $e(C, y\tilde{P} + \tilde{Q})e(P, \tilde{P})^d = e(V, \tilde{P})$ is also a solution for the points equation $(y + \alpha)C + dP = V$. We will then prove the Theorem considering this last equation only, distinguishing between membership and non-membership witnesses.

Membership witnesses. By contradiction, suppose there exists a probabilistic polynomial time adversary $\mathcal{A}$ that, with respect to an (non-trivial) accumulator state $(V, \mathcal{Y}_V)$, outputs with non-negligible probability a membership witness $C \in G_1$ for an element $y \in (\mathbb{F}_p)^* \setminus \mathcal{Y}_V$. It follows that $(y + \alpha)C = V = f_V(\alpha)P$, where $f_V(x) = \prod_{y_i \in \mathcal{Y}_V}(y_i + x)$. Since $y \notin \mathcal{Y}_V$, we have that $(y + \alpha) \nmid f_V(x)$. Using the Polynomial Extended Euclidean algorithm, $\mathcal{A}$ computes $g(x) \in \mathbb{F}_p[x]$ of degree $|\mathcal{Y}_V| - 1$ and $r \in (\mathbb{F}_p)^*$ such that $f_V(x) = g(x) \cdot (y + x) + r$. Therefore, $C = g(\alpha)P + \frac{r}{y+\alpha}P$ and using the $\mathcal{RS} = \{P, \alpha P, \alpha^2 P, ..., \alpha^{q(\lambda)}\}$, with $|\mathcal{Y}_V| \leq q(\lambda)$, can compute $g(x)P$ and hence $\frac{1}{y+\alpha}P = r^{-1}(C - g(\alpha)P)$, contradicting the $t-$SDH assumption.

Non-membership witnesses. Suppose there exists a probabilistic polynomial time adversary $\mathcal{A}$ that with respect to an (non-trivial) accumulator state $(V, \mathcal{Y}_V)$ outputs with a non-negligible probability a non-membership witness $(C, d) \in G_1 \times (\mathbb{F}_p)^*$ for an element $y \in \mathcal{Y}_V$. Then $(y + \alpha)C = f_V(\alpha)P - dP$. Now, since $(y + x)|f_V(x)$ we have that $(y + x) \nmid f_V(x) - d$ for any $d \neq 0$. Thus, similarly as done before, $\mathcal{A}$ uses the Polynomial Extended Euclidean algorithm to compute $g(x) \in \mathbb{F}_p[x]$ of degree $|\mathcal{Y}_V| - 1$ and $r \in (\mathbb{F}_p)^*$ such that $f_V(x) - d = g(x) \cdot (y + x) + r$. Therefore, $C = g(\alpha)P + \frac{r}{y+\alpha}P$ and, using the $\mathcal{RS}$, $\mathcal{A}$ can compute $\frac{1}{y+\alpha}P = r^{-1}(C - g(\alpha)P)$, contradicting the $t-$SDH assumption. □

We note that this proof doesn't differentiate among the α-based and the $\mathcal{RS}$-based construction. However, by assuming $f_V(x) = g(x)(y - x) + d \in \mathbb{F}_p[x]$, which is equivalent to $f(-y) = d \in \mathbb{F}_p$, we infer that non-membership witnesses are defined according to the $\mathcal{RS}$-construction and this proof is referring to the $\mathcal{RS}$-based construction.

This detail is crucial since in the next Sections we will show that the non-membership witness definition of the α-based construction is flawed and allows a probabilistic polynomial-time attacker to recover the secret accumulator parameter α and thus break collision resistance. This flaw is not present in the non-membership witness definition of the $\mathcal{RS}$-based construction −which, in fact, fully satisfy the

above security reduction under the $t-$SDH assumption$-$ and hence the α-based construction can be easily fixed by using, instead, the non-membership witness defining equation of the other $\mathcal{RS}-$based variant. In other words, a *"fixed"* α-based construction will correspond to a slightly more time-efficient (but asymptotically equivalent) version of the $\mathcal{RS}-$based construction, where the $\mathcal{RS}$ is not directly given to the attacker but can be computed in polynomial time (Lemma 2.1).

Motivated by this observation and by concrete applications of the scheme where the attacker cannot arbitrarily query an oracle returning witnesses for any freely chosen element, we show, in Section 2.10, that even when the accumulator manager keeps the $\mathcal{RS}$ secret, the attacker would be able to efficiently recover the latter by accessing few non-membership witnesses, thus making him able to issue membership and non-membership witnesses accordingly to the $\mathcal{RS}-$based defining equations, but not able to break collision resistance for this variant. We remark that this scenario is outside Au *et al.* security model $-$where such $\mathcal{RS}$ is always available to the attacker, which can further obtain witnesses from the oracle$-$ but becomes relevant in all those concrete scenarios where the Manager wishes to have exclusive rights for issuing witnesses (and thus keeps the $\mathcal{RS}$ secret), such us authentication mechanisms where witnesses are used as black-/white-list authentication tokens.

2.5 Breaking Collision Resistance in the α-based Construction

In the α-based construction, the knowledge of a single non-membership witness is enough to break the (assumed) collision resistance property of the accumulator scheme when the polynomial $f_V(x) \in \mathbb{F}_p[x]$ is fully known or, equivalently, the set of all accumulated elements is publicly known (which is typically the case).

In the security reduction provided in [Au+09], it is required that given a non-accumulated element $y \in \mathcal{ACC} \setminus \mathcal{Y}_V$ and its non-membership witness $\bar{w}_{y,V} = (C_y, \tilde{d}_y)$ with respect to the accumulator value V, the element $\tilde{d}_y \in \mathbb{F}_p$ verifies

$$\Big(f_V(x) - \tilde{d}_y \bmod (y+x)\Big) \equiv 0 \pmod p$$

which in turn corresponds to $\tilde{d}_y \equiv f_V(-y) \pmod p$, a condition enforced by the $\mathcal{RS}-$based construction non-membership witness definition.

By using, instead, the defining equation for d_y provided in the α-based construction, the partial non-membership witness for y equals $d_y = \Big(f_V(\alpha) \bmod (y+\alpha)\Big) \bmod p$ and thus

$$d_y \equiv \tilde{d}_y \pmod p \quad \Rightarrow \quad \Big(f_V(-y) \bmod (y+\alpha)\Big) \equiv f_V(-y) \pmod p$$

holds only when $f_V(-y) < y + \alpha$, i.e. with negligible probability if V accumulates more than one element chosen uniformly at random from $\mathbb{F}_p$.

Now, if $d_y \not\equiv \tilde{d}_y \bmod p$, we have $f_V(x) - d_y \not\equiv 0 \bmod (y+x)$, and we can use Euclidean algorithm to find a polynomial $c(x) \in \mathbb{F}_p[x]$ and $r \in \mathbb{F}_p$ such that $f_V(x) - d_y = c(x)(y+x) + r$ in $\mathbb{F}_p[x]$. Then, by recalling that $C_y = \frac{f_V(\alpha)-d_y}{y+\alpha} P$, under the $t-$SDH assumption, the attacker uses the available $\mathcal{RS} = \{P, \alpha P, \ldots, \alpha^t P\}$ to compute $c(\alpha)P$ and obtains a membership witness with respect to V for an arbitrary

non accumulated element y as

$$C_y + \frac{d_y}{r}\Big(C_y - c(\alpha)P \Big) = C_y + \frac{d_y}{r}\left(C_y - C_y - \frac{r}{y+\alpha}P \right) = \frac{f_V(\alpha)}{y+\alpha}P = \frac{1}{y+\alpha}V$$

thus breaking the assumed collision resistance property. We note that this result doesn't invalidate the security proof provided by Au *et al.* in [Au+09]: indeed, the reduction to the t-SDH assumption is shown for (non-membership) witnesses generated accordingly to the $\mathcal{RS}$-based construction only, and thus, collision resistance can be guaranteed only for this latter construction.

We speculate that this flaw comes from the wrong assumption that

$$\Big(f_V(x) \bmod (y+x)\Big) \equiv \Big(f_V(\alpha) \bmod (y+\alpha)\Big) \pmod p$$

which, if true, would have implied security of non-membership witnesses issued accordingly to the α-based construction as well. The authors also declare [Au+09, Section 2.2] that by using the secret accumulator value α, the accumulator manager can compute membership and non-membership witnesses in $O(1)$ time: this clearly cannot be true since, regardless of the variant considered, the evaluation of the polynomial $f_V(x)$ and its reduction modulo a $\sim \log p$-bits integer requires (at least) $O(\deg f_V)$ time.

In the following Sections we will show that within the α-based construction, an attacker can efficiently recover the secret accumulator parameter α by accessing multiple non-membership witnesses, thus making him able to break collision resistance by computing membership witnesses for non-accumulated elements similarly as above, but also non-membership witnesses for accumulated elements.

2.6 The α-Recovery Attack for the α-Based Construction

From now on, we assume the secret parameter α and the accumulator value V along with the set of currently accumulated elements $\mathcal{Y}_V$ to be fixed.

The following attack on the α-based construction consists of two phases: the retrieval of the value $f_V(\alpha) \in \mathbb{Z}$ used to compute non-membership witnesses modulo many small primes, and the full recovery of the accumulator secret parameter α.

2.6.1 Recovering $f_V(\alpha)$

Let $d_y = \Big(f_V(\alpha) \bmod (y+\alpha) \Big) \bmod p$ be a partial non-membership witness with respect to V for a certain element $y \in \mathcal{ACC} \setminus \mathcal{Y}_V$, and let $\tilde{d}_y$ denote the integer $f_V(\alpha) \bmod (y+\alpha)$. We then have $d_y = \tilde{d}_y \bmod p$, and we are interested in how often d_y equals $\tilde{d}_y$ as integers. Attacker benefits from the cases when $y+\alpha < p$, since the reduction modulo p does nothing and $d_y = \tilde{d}_y$ for all such y.

The worst case happens when α is maximal, i.e. $\alpha = p-1$. Indeed, in this case, if $y = 0$ then $y+\alpha < p$ and $d_y = \tilde{d}_y$ with probability 1; if instead $y > 0$ and $y \neq p-\alpha = 1$ the probability that $d_y = \tilde{d}_y$ is $\frac{p}{y+\alpha}$ and, hence, is minimal when compared to smaller values of α. Thus, with $\alpha = p-1$ the probability that d_y equals $\tilde{d}_y$ as integers ranges from 1 (when $y = 0$) to almost $1/2$ (when $y = p-1$). Assuming that y is sampled uniformly at random, we can obtain the following lower bound on

the probability (for arbitrary α):

$$\begin{aligned}\mathop{\mathbb{P}}_{\substack{y\in\{0,\dots,p-1\}\\ y\neq p-\alpha\\ f_V(\alpha)\in\mathbb{Z}}}(d_y=\tilde{d}_y) &\geq \frac{1}{p-1}\left(1+p\sum_{\tilde{y}=2}^{p-1}\frac{1}{\tilde{y}+p-1}\right)\\ &= \frac{p}{p-1}\left(\sum_{i=1}^{2p-2}\frac{1}{i}-\sum_{i=1}^{p-1}\frac{1}{i}\right)=\frac{p}{p-1}\left(H_{2p-2}-H_{p-1}\right)\\ &= \left(1+\frac{1}{p-1}\right)\cdot\left(\ln 2-\frac{1}{4(p-1)}+o\left(p^{-1}\right)\right)\\ &= \ln 2+\frac{4\ln 2-1}{4(p-1)}+o(p^{-1})\\ &> \ln 2. \end{aligned} \tag{2.1}$$

where H_n denotes the $n-$th Harmonic number, and the last inequality holds for all values of p used in practice.

Let us assume that $q|(y+\alpha)$ for a small prime $q\in\mathbb{Z}$ such that $q\ll y+\alpha$. If $d_y=\tilde{d}_y$ we have $f_V(\alpha)\equiv d_y \pmod q$ with probability 1, otherwise it happens with probability 0 since then $f_V(\alpha)\equiv d_y+p \pmod q$.

If instead $q\nmid(y+\alpha)$, we can assume $d_y \bmod q$ to be random in $\mathbb{Z}/q\mathbb{Z}$ and thus $f_V(\alpha)\equiv d_y \pmod q$ happens with probability close to $\frac{1}{q}$.

More formally,

$$\mathbb{P}\Big(f_V(\alpha)\equiv d_y \pmod q\Big)>\ln 2\cdot\frac{1}{q}+\frac{q-1}{q^2}=\frac{(\ln 2+1)q-1}{q^2} \tag{2.2}$$

while for any other $c\in\mathbb{Z}/q\mathbb{Z}$ such that $c\not\equiv d_y \pmod q$ we have

$$\mathbb{P}\Big(f_V(\alpha)\equiv c \pmod q\Big)<(1-\ln 2)\cdot\frac{1}{q}+\frac{q-1}{q^2}=\frac{(2-\ln 2)q-1}{q^2} \tag{2.3}$$

In other words, the value $d_y \bmod q$ has a higher chance to be equal to $f_V(\alpha) \bmod q$ compared to any other value in $\mathbb{Z}/q\mathbb{Z}$.

We will use this fact to deduce $f_V(\alpha)$ modulo many different small primes. More precisely, suppose that an attacker has access to the elements $y_1,\dots,y_n$ together with the respective partial non-membership witnesses

$$d_{y_i}\equiv\Big(f_V(\alpha) \mod (y_i+\alpha)\Big) \bmod p$$

If q is a small prime and n is sufficiently large (in Subsection 2.6.3 we provide a rigorous analysis), $f_V(\alpha) \bmod q$ can be deduced by simply looking at the most frequent value among

$$d_{y_1} \bmod q,\ \dots\ ,\ d_{y_n} \bmod q$$

Once we compute $f_V(\alpha)$ modulo many different small primes $q_1,\dots,q_k$ such that $q_1\cdot\ldots\cdot q_k>p$, we can proceed with the next phase of the attack: the full recovery of the secret parameter α.

2.6.2 Recovering α

If the discrete logarithm of any accumulator value is successfully retrieved modulo many different small primes whose product is greater than p, α can be recovered with

(virtually) no additional partial non-membership witnesses. The main observation we will exploit is the following:

Observation 2.1. *Let q be an integer and let $y \in \mathcal{ACC} \setminus \mathcal{Y}_V$ be a non-accumulated element such that its partial non-membership witness with respect to V satisfies $d_y = \tilde{d}_y$. Then $d_y \not\equiv f_V(\alpha) \pmod q$ implies that $q \nmid (y + \alpha)$, or, equivalently, $\alpha \not\equiv -y \pmod q$.*

From Equation 2.1, it follows that for any given $q \in \mathbb{Z}$ and non-accumulated element y such that $(f_V(\alpha) - d_y) \not\equiv 0 \pmod q$, we have

$$\mathbb{P}\Big(\alpha \not\equiv -y \pmod q \,|\, f_V(\alpha) \not\equiv d_y \pmod q\Big) > 1 - \frac{(1-\ln 2)q}{q^2 - (1+\ln 2)q + 1} \approx 1 - \frac{1-\ln 2}{q}$$

By considering all available non-membership witnesses, if q is small and n is sufficiently larger than q (as we detail in Subsection 2.6.3), we can deduce $\alpha \bmod q$ as the element in $\mathbb{Z}/q\mathbb{Z}$ which is the least frequent −or not occurring at all− among the residues

$$-y_{i_1} \bmod q\ ,\ \ \ldots\ \ ,\ -y_{i_j} \bmod q$$

such that $(f_V(\alpha) - d_{y_{i_k}}) \not\equiv 0 \bmod q$ for all $k = 1, \ldots, j$.

It follows that, if $q_1, \ldots, q_k$ are small primes such that $q_1 \cdot \ldots \cdot q_k > p$, from the values $f_V(\alpha) \bmod q_i$ (computed according to Subsection 2.6.1) and the values $\alpha \bmod q_i$ with $i \in [1, k]$, then $\alpha \in \mathbb{Z}$ can be easily obtained thanks to the Chinese Remainder Theorem.

2.6.3 Estimating the minimum number of witnesses needed

We now give an asymptotic estimate of the minimum number of non-membership witnesses required to succeed with a high probability both phases of the above attack. We will use the multiplicative Chernoff bound, which we briefly recall.

Theorem 2.2. (Chernoff Bound) *Let $X_1, \ldots, X_n$ be independent random variables taking values in $\{0, 1\}$ and let $X = X_1 + \ldots + X_n$. Then, for any $\delta > 0$*

$$\mathbb{P}\Big(\, X \le (1-\delta)\mathbb{E}[X] \,\Big) \le e^{-\frac{\delta^2 \mu}{2}} \qquad 0 \le \delta \le 1$$

$$\mathbb{P}\Big(\, X \ge (1+\delta)\mathbb{E}[X] \,\Big) \le e^{-\frac{\delta^2 \mu}{2+\delta}} \qquad 0 \le \delta$$

Proof. See [MU05, Theorem 4.4, Theorem 4.5]. □

Our analysis will proceed as follows: first, we introduce two random variables to model, for a given small prime q, the behaviour of the values $f_V(\alpha) \bmod q$. Then, we will use Chernoff bound to firstly estimate the probability of wrongly guessing $f_V(\alpha) \bmod q$, and then deduce a value for n so that such probability is minimized for all primes q considered in the attack.

Let $q \in \mathbb{Z}$ be a fixed prime and let X_g be a random variable which counts the number of times $f_V(\alpha) \bmod q$ is among the values $d_1 \bmod q, \ldots, d_n \bmod q$. Similarly, let X_b be a random variable which counts the number of times a certain residue $t \in \mathbb{Z}/q\mathbb{Z}$ not equal to $f_V(\alpha) \bmod q$ is among the values $d_1 \bmod q$, $\ldots$, $d_n \bmod q$. Then

$$\mathbb{E}[X_g] = n \cdot \frac{(\ln 2 + 1)q - 1}{q^2} \approx (\ln 2 + 1)\frac{n}{q}$$

$$\mathbb{E}[X_b] = n \cdot \frac{(2-\ln 2)q - 1}{q^2} \approx (2-\ln 2)\frac{n}{q}$$

By applying Theorem 2.2, we can estimate the probability that X_g and X_b crosses $\frac{\mathbb{E}[X_g]+\mathbb{E}[X_b]}{2} = \frac{3n}{2q}$ as

$$\mathbb{P}\left(X_g \leq \frac{3n}{2q}\right) = \mathbb{P}\left(X_g \leq \left(1 - \frac{2\ln 2 - 1}{2\ln 2 + 2}\right)\mathbb{E}[X_g]\right) < e^{-\frac{n}{91q}} \doteq e_{q,g}$$

$$\mathbb{P}\left(X_b \geq \frac{3n}{2q}\right) = \mathbb{P}\left(X_b \geq \left(1 + \frac{2\ln 2 - 1}{4 - 2\ln 2}\right)\mathbb{E}[X_b]\right) < e^{-\frac{n}{76q}} \doteq e_{q,b}$$

and we minimize these inequalities by requiring that

$$1 - (1 - e_{q,g})(1 - e_{q,b})^{q-1} \approx e_{q,g} + (q-1)e_{q,b} \doteq s_q$$

is small for each prime q considered in this attack phase. Thus, if $q = max(q_1, \ldots, q_k)$, we can bound the sum

$$\sum_{i=1}^{k} s_{q_i} \leq q s_q = q(e^{-\frac{n}{91q}} + (q-1)e^{-\frac{n}{76q}}) \approx e^{-\frac{n}{91q}+\log q} + e^{-\frac{n}{76q}+2\log q}$$

which we make small by taking $n = O(q \log q)$.

In order to apply the Chinese Remainder Theorem for the full recovery of α we need that $q_1 \cdot \ldots \cdot q_k > p$. If $q_1, \ldots, q_k$ are chosen to be the first k primes, we can use an estimation for the first Chebyshev function growth rate to obtain $\ln(q_1 \cdot \ldots \cdot q_k) = (1 + o(1)) \cdot k \ln k \sim q_k$ by Prime Number Theorem and thus $q_k > \ln p$. We then conclude that

$$n = O(\log p \log\log p)$$

non-membership witnesses are enough to recover $f_V(\alpha) \bmod q_1 \cdot \ldots \cdot q_k$ with high probability.

We note that by using Chernoff bound in order to estimate the minimum number of witnesses needed to recover α, it can be shown, similarly as done above for $f_V(\alpha)$, that $O(\log p \log\log p)$ non-membership witnesses are enough to identify with high probability $\alpha \bmod q_1 \cdot \ldots \cdot q_k = \alpha$.

The time complexity is dominated by

$$(\#\text{ primes } q) \times (\#\text{ witnesses}) = O\left(\frac{\log p}{\log\log p}\right) \times O(\log p \log\log p)$$

which is equal to $O(\log^2 p)$.

2.7 Improvements to the α-Recovery Attack

We can improve the α-Recovery Attack outlined in Section 2.6 by detailing variants under two different attack scenarios, depending on whether the attacker has access to non-membership witnesses for *random-y* or *chosen-y*.[2] These improvements will further reduce the number of non-membership witnesses needed to fully recover the secret accumulator parameter α to a small multiple of $\log p$.

[2]We observe that according to Definition 2.1, the attacker has access to an oracle which returns witnesses for any *chosen-y*. However, in concrete instances of the accumulator scheme, an attacker might have access only to witnesses for *random* values y.

The main idea behind the improved attack is to keep removing wrong candidates for $\alpha \bmod q$ for small primes q (*sieving*), until only the correct one is left. As in the previous attack, the full value of α is reconstructed using the Chinese Remainder Theorem.

2.7.1 Collecting Witnesses Issued at Different States

In the α-Recovery Attack, $O(\log p \log\log p)$ non-membership witnesses issued with respect to the same accumulator value V are needed in order to fully recover α. In the following attacks we drop this condition and allow non-membership witnesses to be issued with respect to different accumulator values $f_1(\alpha)P = V_1, \ldots, f_\ell(\alpha)P = V_\ell$, but we require that no deletions occur between the accumulator states V_1 and V_ℓ. In this case, since the sequence of elements added must be public to permit witness updates, we have that the polynomial functions $g_{i,j}(x) \in \mathbb{F}_p$ such that $f_j(\alpha) = g_{i,j}(\alpha) f_i(\alpha)$ for all $\alpha \in \mathbb{F}_p$, can be publicly computed for any $i, j \in [1, \ell]$. It follows that, given a small prime q, once $\alpha \bmod q$ and $f_i(\alpha) \bmod q$ for some $i \in [1, \ell]$ are correctly computed, $f_j(\alpha) \bmod q$ can be computed as $g_{i,j}(\alpha) f_i(\alpha) \bmod q$ for any $j \in [1, \ell]$ such that $j > i$.

The requirement that no deletion operation should occur if the collected witnesses were issued at different states comes from the fact that the accumulator can be initialized by accumulating some values which are kept secret by the accumulator manager.

It follows that, whenever the polynomial $f_1(x) \in \mathbb{F}_p$ is publicly known (or, equivalently, the set of all accumulated elements $\mathcal{Y}_{V_1}$) for a certain accumulator value V_1, we can remove the condition that no later deletion operations occur during attack execution, since the knowledge of $\alpha \bmod q$ is enough to compute $f_i(\alpha) \bmod q$ for any $i \in [1, \ell]$. Thus any non-membership witnesses issued from V_1 on can be used to recover α.

2.7.2 Removing reduction modulo p

Under some realistic assumptions, we show that it is possible to eliminate with a high probability the noise given by the reduction modulo p performed by the accumulator manager when he issues a non-membership witness. That is, we recover $\tilde{d}_{y_i} = f_{V_j}(\alpha) \bmod (y_i + \alpha)$ for a large fraction of pairs (y_i, V_j), given the partial non-membership witnesses $d_{y_i} = \Big(f_{V_j}(\alpha) \bmod (y_i + \alpha)\Big) \bmod p$ collected with respect to different accumulator values V_j with $j > 1$.

Aiming at this, we first observe that from the fact that $0 \leq y, \alpha < p$ for any given $y \in \mathcal{ACC} \setminus \mathcal{Y}_V$, the partial non-membership witness d_y for y with respect to V can be expressed in terms of $\tilde{d}_y$ in one of the following way:

(1) $d_y = f_V(\alpha) \bmod (y + \alpha) = \tilde{d}_y$,

(2) $d_y = \Big(f_V(\alpha) \bmod (y + \alpha)\Big) - p = \tilde{d}_y - p$.

Since p is odd, whenever $y + \alpha$ is even, these two cases can be easily distinguished modulo 2: indeed, in the first case $d_y \equiv f_V(\alpha) \pmod 2$, while in the second case $d_y \not\equiv f_V(\alpha) \pmod 2$.

This observation effectively allows to correctly compute $\tilde{d}_y$ half of the times given a correct guess for $\alpha \bmod 2$ and $f_V(\alpha) \bmod 2$. Indeed, given a set of partial non-membership witnesses $d_{y_1}, \ldots, d_{y_n}$ with respect to V, each guess of $\alpha \bmod 2$ and

$f_V(\alpha) \bmod 2$ will split the witnesses in two subsets, namely one where the corresponding elements y_i satisfy $y_i + \alpha \equiv 0 \pmod 2$ (and thus $\tilde{d}_{y_i}$ can be correctly recovered), and the other where this doesn't happen.

Checking if $\alpha \bmod 2$ and $f_V(\alpha) \bmod 2$ were actually correct guesses can be done observing how the attacks described in Subsection 2.7.3 and Subsection 2.7.4 (or in Section 2.6 if witnesses are issued with respect to the same accumulator value) behaves with respect to the subset of witnesses that permitted to recover the values $\tilde{d}_{y_i}$. In case of a wrong guess, indeed, it will not be possible to distinguish α and $f_{V_i}(\alpha)$ modulo some different small primes q: in this case, the attack can be stopped, and a new guess should be considered. On the other hand, a correct guess will permit to correctly recover α and $f_{V_i}(\alpha)$ modulo a few more primes q greater than 2. Since, whenever $\alpha \bmod q$ and $f_V(\alpha) \bmod q$ are known, $\tilde{d}_y$ can be correctly recovered, analogously to the modulo 2 case, for all those y such that $y + \alpha$ is divisible by q, this implies that it is possible to iteratively recover more and more correct values $\tilde{d}_{y_i}$ given the initial set of considered witnesses.

Repeating this procedure for small primes q up to r, it allows to recover $\tilde{d}_{y_i}$ for those y_i that are divisible by at least one prime not exceeding r. This fraction tends to $1 - \varphi(r\#)/(r\#)$ as y_i tend to infinity, where φ is the Euler's totient function and $r\#$ denotes the primorial, i.e., the product of all primes not exceeding r. For example, setting $r = 101$ allows to recover $\tilde{d}_{y_i}$ for about 88% of all available witnesses. We conclude that $\tilde{d}_{y_i}$ can be recovered for practically all $i \in [1, n]$.

In the case where witnesses are issued with respect to different accumulator values $V_1, \ldots, V_\ell$, as remarked above, the knowledge of $\alpha \bmod q$ and $f_{V_1}(\alpha) \bmod q$ allows to compute $f_{V_j}(\alpha) \bmod q$ for all V_j with $j > 1$, so the modulo p noise reduction can be easily performed independently on when the witnesses are issued.

2.7.3 The Random-y Sieving Attack

In this scenario, we assume that all elements y_i for which the partial non-membership witnesses d_{y_i} are available to the adversary are sampled uniformly at random from $\mathbb{F}_p$. Furthermore, these witnesses are pre-processed accordingly to the method described above in order to eliminate the noise given by the reduction modulo p.

Recovering α mod q. Let q be a small prime, i.e. $q = O(\log p)$, and let $\mathcal{Y}_\alpha$ be the set containing all pairs $(y_i, \tilde{d}_{y_i})$ such that $y_i + \alpha \equiv 0 \pmod q$ for a certain guess $\alpha \bmod q$. If the latter is guessed wrongly, then the values $\tilde{d}_{y_i}$ modulo q are distributed uniformly and independently from the values $f_{V_i}(\alpha) \bmod q$. On the other hand, if the guess is correct, then $\tilde{d}_{y_i} \equiv f_{V_i}(\alpha) \pmod q$.

Even in the case when $f_{V_1}(\alpha) \bmod q$ is unknown, $f_{V_i}(\alpha) \bmod q$ can be recovered from the first occurrence of y_i in the set $\mathcal{Y}_\alpha$ and verified at all further occurrences, since all $f_{V_j}(\alpha) \bmod q$ can be computed for any $j \geq i$. It follows that we can easily distinguish if a guess for $\alpha \bmod q$ is either correct or not.

The attack succeeds if for every wrong guess $\alpha^\times$ of $\alpha \bmod q$ we observe a contradiction within the pairs in $\mathcal{Y}_{\alpha^\times}$. It's easy to see that if $|\mathcal{Y}_{\alpha^\times}| = t$, the probability to observe at least one contradiction is $1 - 1/q^{t-1}$. Thus, by ensuring a constant number t of elements in $\mathcal{Y}_{\alpha^\times}$ given each $\alpha^\times \neq \alpha \bmod q$ is sufficient to make the probability of false positives negligible. This requires availability of $O(q \log q)$ witnesses in total.

Recovering α. The final step is the same as in the previous attacks: the secret value α is recovered by repeating the process for different small primes q and applying the Chinese Remainder Theorem. Furthermore, if for some primes q there are

multiple candidates of $\alpha \bmod q$, such primes can be omitted from the application of the Chinese Remainder Theorem. In this case, in order to fully recover $\alpha \in \mathbb{Z}$, the maximum prime q that has to be considered must be larger than $\ln p$ by a constant factor. We conclude that $O(q \log q) = O(\log p \log \log p)$ witnesses are sufficient for full recovery of α with overwhelming probability.

The time complexity of the attack is dominated by guessing $\alpha \bmod q$ for each q considered. Note that for a wrong guess of $\alpha \bmod q$, we can expect on average a constant amount of witnesses to check before an inconsistency is observed; this amount is thus enough to identify the correct value. For each such guess, nearly all accumulator states in history have to be considered in order to take into account all additions to the accumulator. However, the non-membership witnesses issued in each state can be classified by guesses of $\alpha \bmod q$ in a single scan for each prime q.

We conclude that the time complexity is dominated by

$$(\#\text{ primes } q) \times (q \text{ guesses of } \alpha \bmod q) \times (\#\text{ of accumulator states})$$

and by classifying all non-membership witnesses for each prime q

$$(\#\text{ primes } q) \times (\#\text{ witnesses})$$

The final complexity is $O((1 + \ell / \log \log p) \log^2 p)$.

2.7.4 The Chosen-y Sieving Attack

If the adversary is allowed to choose the elements y_i for which the partial non-membership witnesses are issued, no matter with respect to which accumulator state, the number of required witnesses can be further reduced by a $\log \log p$ factor.

First, we assume that the adversary chooses the elements y_i non-adaptively, i.e. before the accumulator is initialized. The idea is simply to use consecutive values, that is $y_0 = r$, $y_1 = r + 1, \ldots, y_i = r + i, \ldots$, for some $r \in \mathbb{F}_p$. This choice fills equally all sets $\mathcal{Y}_{\tilde{\alpha}}$ for all $\tilde{\alpha} \in \mathbb{Z}/q\mathbb{Z}$ and small q, where $\tilde{\alpha}$ represents either a correct guess for $\alpha \bmod q$ or a wrong guess $\alpha^{\times}$. As a result, $t = O(q)$ elements are enough to make the size of each set $\mathcal{Y}_{\tilde{\alpha}}$ at least equal to t. The full total number of required non-membership witnesses is then reduced to $O(q) = O(\log p)$. The time complexity then is improved by a factor $\log \log p$ in the case when ℓ is small: $O(\ell \log^2 p / \log \log p)$.

We now consider the case when the adversary can adaptively choose the elements y_i. Note that, on average, we need only $2 + 1/(q-1)$ elements in each set $\mathcal{Y}_{\alpha^{\times}}$ to discard the wrong guess of $\alpha \bmod q$, for all q. The adaptive choice allows to choose y_i such that $(y_i + \alpha^{\times}) \equiv 0 \pmod q$ specifically for those $\alpha^{\times}$ which are not discarded yet. Furthermore, the Chinese Remainder Theorem allows us to simultaneously combine such adaptive queries for all chosen primes q. As a result, approximately $2 \ln p$ witnesses for adaptively chosen elements are sufficient for the full recovery of α. This improves the constant factor of the non-adaptive attack in terms of the number of non-membership witnesses required.

We conclude by observing that, as described at the beginning of this section, non-membership witnesses can be issued with respect to different successive accumulator values $V_1, \ldots, V_\ell$, within which no deletion operation occurs. If the value $f_{V_1}(x) \in \mathbb{Z}[x]$ is known to the adversary (or equivalently the set of all accumulated elements in V_1), only $\ln p$ non-membership witnesses issued for adaptively chosen elements are sufficient to recover α. In this case, indeed, instead of verifying uniqueness of elements

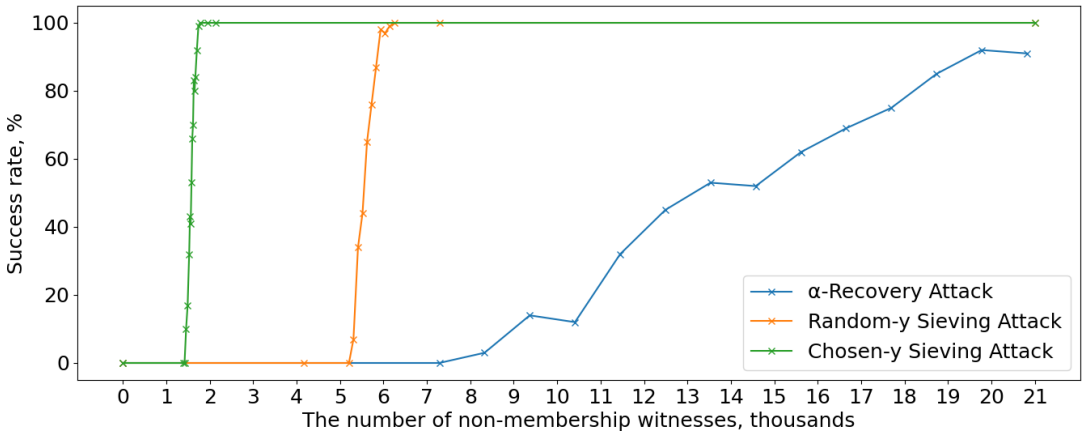


FIGURE 2.1: Attacks experimental success rate as a function of the total number of available witnesses.

in the set $\mathcal{Y}_{\alpha\times}$, we can directly compare our guess to the value $f_{V_j}(\alpha)$ mod q given from $f_{V_1}(\alpha)$, thus requiring $1 + 1/(q-1)$ elements on average.

2.8 Experimental Results

We implemented the *α-Recovery Attack* from Section 2.6 and both the *random-y* and the *non-adaptive chosen-y sieving* attacks from Subsection 2.7.3 and Subsection 2.7.4.

For the verification purpose, we used a random 512-bit prime p, and we measured the success rate of the attacks with respect to the number of available non-membership witnesses. The α-Recovery Attack applies to a single accumulator state, and for the sieving attacks, the number of state changes of the accumulator was ten times less than the number of issued witnesses. The initial state of the accumulator in all attacks was assumed to be secret. Each attack was executed 100 times per analyzed number of available non-membership witnesses. The sieving attacks were considered successful if at most 2^{10} candidates for α were obtained and the correct α was among them. The results are illustrated in Figure 2.1.

The α-Recovery Attack, while being simple, requires a significant amount of witnesses to achieve a high success rate, more than $20000 \approx 10 \ln p \ln \ln p$ witnesses and finishes in less than 5 seconds. The random-y sieving attack achieves an almost full success rate with about $6000 \approx 3 \ln p \ln \ln p$ available witnesses and completes in less than 10 seconds. The chosen-y sieving attack requires less than $2000 \approx 4 \ln p$ witnesses to achieve almost perfect success rate and completes in less than 4 seconds. All timings include the generation of witnesses. The experiments were performed on a laptop with Linux Mint 19.3 OS and an Intel Core i5-10210U CPU clocked at 1.60GHz.

2.9 Weak Non-membership Witnesses

In the α-based construction, non-membership witness definition is affected by another minor design vulnerability: given a non-membership witness $\bar{w}_{y,V} = (C_y, d_y)$ with respect to an accumulator value V, if $d_y \equiv f_V(\alpha) \bmod p$, then $C_y = \mathcal{O}$, i.e. the identity element of G_1.

Those *weak non-membership witnesses* are issued with non-negligible probability in the security parameter λ when only one element is accumulated. Assume, indeed,

that $V = (y' + \alpha)P$ for a certain element $y' \in \mathcal{ACC}$. Then, for any element $y \in \mathcal{ACC}$ such that $y' < y$, the corresponding non-membership witness $\bar{w}_{y,V}$ with respect to V is issued as

$$d_y = \Big(y' + \alpha \bmod (y + \alpha)\Big) \bmod p = (y' + \alpha) \bmod p$$

and thus $C_y = \mathcal{O}$. In this case, as soon as the element y' becomes public (e.g. is removed), the accumulator secret parameter can be easily obtained as $\alpha = (d_y - y') \bmod p$.

2.10 Preventing Witness Forgery in the $\mathcal{RS}$-based Construction

All the attacks we have presented so far are ineffective when witnesses (more precisely, non-membership witnesses) are issued according to the defining equations given for the $\mathcal{RS}$-based construction.

We note that the knowledge of the $\mathcal{RS}$ is functionally equivalent to the knowledge of α when the set of currently accumulated elements is fully known: indeed, besides accumulator updates, the $\mathcal{RS}$ permits to issue both membership and non-membership witnesses for arbitrary elements using the Extended Euclidean Algorithm, with the difference that the knowledge of α permits to break collision-resistance, while the knowledge of the $\mathcal{RS}$ does not. Furthermore, despite what we saw in Section 2.5, witnesses definition in the $\mathcal{RS}$-based construction satisfies the hypothesis for the $t-$SDH security reduction provided by Au *et al.* [Au+09], i.e. collision-resistance is enforced when the $\mathcal{RS}$ is used to issue witnesses.

Depending on the use-case application of the accumulator scheme, the possibility to publicly issue witnesses for arbitrary elements could be undesirable: for example, this is relevant when the accumulator scheme is used as a privacy-preserving authorization mechanism, i.e. an Anonymous Credential System. Suppose, indeed, that in this scenario, the accumulator value V accumulates revoked users' identities and the non-revoked ones authenticate themselves showing the possession of a valid non-membership witness $\bar{w}_{y,V}$ for an identity y, both issued by a trusted Authentication Authority (i.e., the accumulator manager). If an attacker has access to the $\mathcal{RS}$, he will be able to forge a random pair of credentials $(y', w_{y',V})$ and then he could authenticate himself, even if the Authentication Authority never issued the identity y' nor the corresponding witness. This is especially the case when a zero-knowledge protocol is instantiated during users' credentials verification since it is impossible to distinguish between a zero-knowledge proof for an authorized identity y and a proof for the never issued, but valid, identity y'.

In the following, we will investigate the $\mathcal{RS}$-based construction under this scenario, i.e. assuming the accumulator manager to be the only authority allowed to issue witnesses. We stress that resistance to witness forgeries is outside the security model provided by Au *et al.*, where the attacker can generate as many witnesses as he wishes, and the attacks described in the following do not break any security properties assumed for the $\mathcal{RS}-$based construction by the respective authors. The following analysis will, however, be relevant to address the security of an accumulator scheme for bilinear groups built upon Au *et al.* one that we propose in Chapter 3.

In the next two Sections, we will discuss how witness forgery for never-authorized elements can be prevented, namely: a) the manager constructs the set $\mathcal{Y}_V$ of currently accumulated elements in such a way that it is infeasible to fully reconstruct it; b) the reference string $\mathcal{RS}$ is not published, and an attacker cannot reconstruct it.

2.10.1 How to ensure some accumulated elements remain unknown

Given an accumulator value V, assume $\mathcal{Y}_V$ is the union of the disjoint sets $\mathcal{Y}_{V_0}$, whose elements are used exclusively to initialize the accumulator value from P to V_0, and $\mathcal{Y}_{id} = \mathcal{Y}_V \setminus \mathcal{Y}_{V_0}$, the set of currently accumulated elements for which a membership witness have been issued.

Since the elements in $\mathcal{Y}_{id}$ must be public to enable users to update their witnesses[3], the reconstruction of $\mathcal{Y}_V = \mathcal{Y}_{V_0} \cup \mathcal{Y}_{id}$ can be prevented only if $\mathcal{Y}_{V_0}$ remains, at least partially, unknown.

From $\mathcal{Y}_V = \mathcal{Y}_{V_0} \cup \mathcal{Y}_{id}$ and $\mathcal{Y}_{V_0} \cap \mathcal{Y}_{id} = \emptyset$, it follows that the polynomial $f_V(x)$ can be written as

$$f_V(x) = f_0(x) \cdot f_{id}(x) = \prod_{y_i \in \mathcal{Y}_{V_0}} (y_i + x) \prod_{y_j \in \mathcal{Y}_{id}} (y_j + x)$$

When non-membership witnesses are generated according to the $\mathcal{RS}$-construction, as soon as an attacker has access to $|\mathcal{Y}_{V_0}|$ partial non-membership witnesses for the elements $y_1, \ldots, y_{|\mathcal{Y}_{V_0}|}$, i.e.

$$d_{y_i} \equiv f_V(-y_i) \equiv f_0(-y_i) \cdot f_{id}(-y_i) \pmod{p}$$

he will be able to reconstruct the unknown set $\mathcal{Y}_{V_0}$. Indeed, with the knowledge of $\mathcal{Y}_{id}$, the polynomial $f_{id}(x)$ can be easily obtained and it is then possible to compute the $|\mathcal{Y}_{V_0}|$ pairs

$$\Big(-y_i, f_0(-y_i) \Big) = \Big(-y_i, \frac{d_{y_i}}{f_{id}(-y_i)} \Big)$$

With these pairs, the attacker is able to uniquely interpolate, using for example Lagrange interpolation, the monic polynomial $f_0(x) \bmod p$ whose roots are the elements in $\mathcal{Y}_{V_0}$.[4]

The full reconstruction of the set $\mathcal{Y}_V$ can be prevented by initializing the accumulator with several random elements which are more than the total number of issuable non-membership witnesses: this avoids the possibility to interpolate $f_0(x)$, even in the case when the attacker has access to all issued non-membership witnesses.

We note, however, that this approach has some disadvantages. First, the maximum number of issuable non-membership witnesses has to be set at generation time and cannot be increased once the first witness is issued since all further accumulated elements will be public to allow witness updates. When this number is reasonable big, let us say 1 billion, the accumulator manager needs to evaluate at least a 1-billion degree polynomial when issuing any new non-membership witnesses, an operation that becomes more and more expensive as the number of accumulated elements increases. On the other hand, by decreasing it, the accumulator manager can issue the non-membership witnesses less expensively, but only to a smaller set of users.

2.10.2 Recovering the $\mathcal{RS}$

Alternatively to the countermeasure proposed in Subsection 2.10.1, it is natural to wonder if unauthorized witness forgery can be prevented by just keeping the $\mathcal{RS}$ secret from the attacker.

[3] The very first element for which a membership witness is issued can remain unknown if there are no other users who need to update their witnesses. In this case, we assume that this element belongs to $\mathcal{Y}_0$.

[4] Since $f_0(x)$ is monic, only $deg(f_0)$ evaluations are needed to uniquely interpolate it.

We will now show that by executing what we will refer to as the *Witness Forgery Attack* (Attack 1), an attacker that has access to multiple witnesses can successfully recover the $\mathcal{RS}$, even if the accumulator manager keeps it secret.

The main observation on which this attack is based on is that given any partial witness C_y (no matter if it is a membership or a non-membership one) for an element y with respect to the accumulator value V, it can be expressed as $C_y = g_y(\alpha)P$ for a polynomial $g_y(x) \in \mathbb{F}_p[x]$ which depends on y and $f_V(x)$ (i.e. $f_V(x) = g_y(x)(y + x) + d_y$ for some $d_y \in \mathbb{F}_p$).

Assume the attacker has access to $n \geq |\mathcal{Y}_V| = m$ partial non-membership witnesses

$$C_{y_1} = g_1(\alpha)P, \ldots, C_{y_n} = g_n(\alpha)P$$

with respect to V. From Subsection 2.10.1, we know that he is able to fully recover the polynomial $f_V(x)$, and so he can explicitly compute from the elements $y_1, \ldots, y_n$ the n polynomials $g_1(x), \ldots, g_n(x)$ in $\mathbb{F}_p[x]$, each of degree $m-1$. We note that by randomly choosing m out of these n polynomials, they will be linearly independent with probability

$$\frac{1}{p^{m^2}} \cdot \prod_{k=0}^{m-1} (p^m - p^k) = \prod_{k=1}^{m} \left(1 - \frac{1}{p^k}\right) \approx 1$$

and so we assume, without loss of generality, that $g_1(x), \ldots, g_m(x)$ are independent. It follows that for any fixed $i \in [0, \ldots, m-1]$, there exist computable not-all-zero coefficients $a_1, \ldots, a_m \in \mathbb{F}_p$ such that

$$x^i = a_1 g_1(x) + \ldots + a_m g_m(x)$$

and so

$$\alpha^i P = a_1 C_{y_1} + \ldots + a_m C_{y_m}$$

In other words, the partial reference string

$$\mathcal{RS}_m \doteq \{P, \alpha P, \ldots, \alpha^{m-1} P\}$$

can be obtained from these witnesses, which in turn enables the attacker to compute membership and non-membership witnesses with respect to V for any accumulated and non-accumulated element, respectively.

We note that it is more convenient to execute the above attack with respect to the accumulator value V_0 and the polynomial $f_{V_0}(x)$: any non-membership witness for a never added element which is issued with respect to a later accumulator value than V_0, can be iteratively transformed back to a non-membership witness for V_0 by just inverting the non-membership witness update formula outlined in Section 2.3. Once both $f_{V_0}(x)$ and $\mathcal{RS}_{|\mathcal{Y}_{V_0}|}$ are computed, the attacker can issue witnesses with respect to V_0 for elements in and not in $\mathcal{Y}_{V_0}$ and update them with respect to the latest accumulator value as usual. Since it is possible to issue many different non-membership witnesses for V_0, this implies that by updating them, these non-membership witnesses can be used to iteratively expand the previously computed partial reference string $\mathcal{RS}_{|\mathcal{Y}_{V_0}|}$.

More precisely, given an accumulator value V we know that

$$V = \left(\prod_{y_i \in \mathcal{Y}_V \setminus \mathcal{Y}_{V_0}} (y + \alpha) \right) V_0 = f_V(\alpha) P$$

Attack 1 The Witness Forgery Attack

Input: $n \geq |\mathcal{Y}_{V_0}|$ non-membership witnesses for never accumulated elements, the accumulator history (accumulator values and added/removed elements)

Output: a non-membership witness for a non-accumulated element or a membership witness for an accumulated one with respect to V

1: *Un-update* all non-membership witnesses with respect to V_0 inverting witness update formula and using accumulator history.
2: Interpolate the polynomial $f_{V_0}(x) = \prod_{y_i \in \mathcal{Y}_{V_0}} (y_i + x)$ from witnesses.
3: Use Euclidean Algorithm to find $g_i(x)$ and d_{y_i} such that $f_{V_0}(x) = g_i(x)(y_i + x) + d_{y_i}$ for every element y_i, $i = 1, \ldots, n$
4: Use linear algebra to write x^j as a linear combinations of $g_1(x), \ldots, g_n(x)$ for any $j = 0, \ldots, |\mathcal{Y}_{V_0}| - 1$. Obtain $\mathcal{RS}_{|\mathcal{Y}_{V_0}|}$ from witnesses.
5: Use $\mathcal{RS}_{|\mathcal{Y}_{V_0}|}$ and $f_{V_0}(x)$ to issue many different non-membership witnesses with respect to V_0. Update them with respect to V.
6: Use the additional non-membership witnesses issued to expand the reference string to $\mathcal{RS}_{|\mathcal{Y}_V|}$.
7: Issue membership and non-membership witnesses with respect to the accumulator value V.

where $f_V(x)$ can be publicly computed from the published witness update information if the monic polynomial $f_{V_0}(x)$ is recovered by the attacker through interpolation, as outlined in Subsection 2.10.1.

Once the attacker successfully computes $\mathcal{RS}_{|\mathcal{Y}_{V_0}|}$, they use it to issue (a multiple of) $|\mathcal{Y}_V| - |\mathcal{Y}_{V_0}|$ additional non-membership witnesses for random elements with respect to V_0, he updates them with respect to V, and expands its starting set of elements and witnesses. Then, for each element y_i in this bigger set, he computes the corresponding polynomial $g_i(x)$ of degree $deg(f_V) - 1$ such that $f_V(x) = g_i(x)(y_i + x) + d_{y_i}$. At this point and similarly as before, the attacker can explicitly write a linear combination of computable polynomials which equals x^i for any i such that $deg(f_{V_0}) - 1 < i \leq deg(f_V) - 1$, and thus can expand the previously computed $\mathcal{RS}_{deg(f_{V_0})}$ to $\mathcal{RS}_{deg(f_V)}$. In conclusion, an attacker would be able to forge witnesses with respect to the latest accumulator value by accessing only $|\mathcal{Y}_{V_0}|$ non-membership witnesses. The whole attack is summarized in Attack 1.

Similarly as discussed in Subsection 2.10.1, this attack can be prevented if the total number of issued non-membership witnesses is less than $|\mathcal{Y}_{V_0}|$.

2.11 Conclusions

In this Chapter, we cryptanalysed the Dynamic Universal Accumulator scheme proposed by Au *et al.* [Au+09], investigating the security of the two constructions proposed, to which we refer as the α-*based* and the $\mathcal{RS}$-*based* construction.

For the first construction, we have shown several attacks which allow us to recover the accumulator secret parameter α and thus break its collision resistance. More precisely, if p is the order of the underlying bilinear group, an attacker with access to $O(\log p \log\log p)$ non-membership witnesses for random elements will be able to fully recover α, no matter how many elements are accumulated. If, instead, the elements can be chosen by the attacker, the number of required witnesses reduces down to just $O(\log p)$, thus making the attack linear in the size of the accumulator secret α. Furthermore, we showed how accumulator collision resistance could be broken in

the α-based construction given *one* non-membership witness, and we also described another minor design flaw.

For the second, i.e., the $\mathcal{RS}$-based construction, we investigated resistance to witness forgeries under the hypothesis that the accumulator manager has the exclusive right to issue witnesses (as in authentication mechanisms) and thus keeps the $\mathcal{RS}$ private. We have shown that an attacker that has access to multiple witnesses can reconstruct the accumulator manager $\mathcal{RS}$, which would then enable him to compute witnesses for arbitrary elements. In particular, if the accumulator is initialized by accumulating m secret elements, m witnesses suffice to recover the secret $\mathcal{RS}$.

Although we have shown that the α-based construction of Au *et al.* Dynamic Universal Accumulator is insecure, one can still use it by replacing the non-membership witness defining equation with the one provided in the alternative $\mathcal{RS}$-based construction, which ensures collision-resistance instead under the t-SDH assumption. There is one caveat: knowledge of $\mathcal{RS}$ will enable an attacker to issue witnesses for arbitrary elements. If this needs to be avoided, as is the case for authentication mechanisms, then the $\mathcal{RS}$ should be kept secret and the accumulator adequately initialized. Namely, the accumulator manager needs to define an upper limit m to the total number of issuable non-membership witnesses and has to initialize the accumulator by adding $m + 1$ secret elements to prevent Attack 1.

Chapter 3

A Dynamic Universal Accumulator over Bilinear Groups

3.1 Introduction

In the previous Chapter, we have seen that a common requirement for accumulator schemes is the ability to change the set of accumulated elements: when the accumulator allows to add and delete elements dynamically, it is said to be dynamic.

Whenever addition or deletion operations occur for one or several elements (in the latter case, these are called *batch additions and deletions*), already issued witnesses should be updated to be consistent with the new accumulator value. Ideally, this should be done using a short amount of *witness update data* (i.e. whose cost/size is not dependent on the number of elements involved) and with only publicly available information (i.e. without knowledge of any *secret* accumulator parameters). While there are many constructions that satisfy the public update condition, as regards to the update cost, Camacho and Hevia showed in [CH10] an impossibility result to have *batch witness updates* whose update data size is independent of the number of elements involved. More precisely, they showed that for an accumulator state which accumulates n elements, the witness update data size for a batch delete operation

involving m elements could not be less than $\Omega(m \log \frac{n}{m})$, thus requiring at least $\Omega(m)$ operations to update.

In this Chapter, we propose a Dynamic Universal Accumulator in the accumulator manager setting, which supports batch operations and public batch witness updates, as well as privacy-preserving zero-knowledge proof of knowledge for membership and non-membership witnesses. Its features are manifold:

- **Support for Batch Operations:** Starting from Nguyen's positive accumulator [Ngu05] and Au *et al.* [Au+09] and Damgård and Triandopoulos non-membership proof mechanism [DT08], we state a Dynamic Universal Accumulator for bilinear groups in the *accumulator manager* setting (i.e. it is managed by a central authority who knows the accumulator trapdoor) and we extend it to fully support batch addition and deletion operations, as well as membership and non-membership batch witness updates.

- **Batch Witness Update Protocol:** we designed a batch witness update protocol where the batch witness update information published by the Accumulator Manager after a batch operation is the same for all users. This information can be pre-processed by third-party servers in order to allow users to update their witnesses in a constant number of elementary operations, even in the case many batch operations occurred from their last update. This allows the accumulator to be used even when only limited-resource devices (ex. smartphones) are available to users.

- **Optimal Batch Update:** the number of operations needed to batch update witnesses equals the lower bound given by Camacho and Hevia [CH10] in the case of a batch deletion operation. The same complexity holds in the case of either a batch addition operation and a batch addition & deletion operation, where m new elements are added and other m elements are deleted, namely $\mathcal{O}(m)$ update time for a batch witness update information size of $m \log pq$ bits, where p is the size of the underlying bilinear group and q is the bit-size of group elements representations.

- **Security:** we introduce a weaker definition for collision resistance and a new more general definition of $t-$SDH assumption for which we provide in the Generic Group Model a lower bound complexity of a generic algorithm that solves the corresponding hardness problem. We show that our scheme, along with its public batch update protocol and published information is secure under this more general security assumption, and we address the relevant attacks detailed in Chapter 2 by showing that with proper initialization of the accumulator value, a generic algorithm has a negligible probability to compute elements belonging to a particular reference string $\mathcal{RS}$, whose knowledge would allow issuing arbitrary witnesses.

- **Zero-Knowledge Friendly:** zero-knowledge protocols are supported for any operation involving witnesses: we detail an efficient non-interactive zero-knowledge protocol to show ownership of a valid witness, and the batch witness update protocol is designed so that the results of a delegated batch witness update preprocessing can be obtained without letting the third-party servers learn anything except the publicly available data.

- **Implementation:** to show efficiency and its practical relevance, we implemented and benchmarked the proposed accumulator using state-of-the-art libraries for pairing-friendly elliptic curves. Following feedback received from the community,

we briefly report benchmarks of third-party implementations of our scheme, which are already employed in production applications.

Limitations. Our protocol assumes a trusted Accumulator Manager that, by knowing the secret accumulator parameter α, can forge membership and non-membership witnesses at will. In practice, the secret α can be secret-shared among multiple managers, but such scheme's construction and security analysis is left as future work. In case of a batch addition and deletion operation where m elements are added and/or deleted, the batch update data has $O(m)$ size, as in the case for non-batch operations: this is indeed a theoretical lower bound that cannot be improved, although our construction provides better constants, detailed at the end of Subsection 3.4.1, than the non-batch approach. We note, however, that our protocol supports a delegation technique (not possible for non-batch operations) which allows users to safely update witnesses in constant time if third-party servers process, on their behalf, the $O(m)$-sized public witness batch update data published by the Accumulator Manager.

It follows that our accumulator is well suited to be the building block of an *Anonymous Credential System*, which originally motivated this work. In these systems, only the users who were previously authorized by a central authority (the accumulator manager) can use the issued credentials to authenticate to the third-party verifier (e.g. some financial service provider, like a bank or an exchange). They do so by proving ownership of a valid membership or non-membership witnesses in zero-knowledge, depending on whether the accumulator is used as a white- or black-list. Furthermore, doing so anonymously and unlinkably, even if the verifier colludes with the accumulator manager. This could be crucial in many applications given current societal challenges of protecting user privacy on the one hand and government-imposed know-your-customer regulations on the other hand.

Related Works. Relevant to this Chapter are accumulator schemes designed in groups of known order [Ngu05; DT08; Au+09], where the considered group is a prime order bilinear group.

Nguyen in [Ngu05] proposed a dynamic positive accumulator for symmetric bilinear groups, where up to t elements can be accumulated assuming that the $t-$Strong Diffie-Hellman assumption holds in the underlying group. Damgård and Triandopoulos [DT08] extended Nguyen's scheme, under the same security assumptions, to support non-membership proofs, thus defining a *universal* accumulator based on bilinear pairings. Soon after this work, Au *et al.* [Au+09] extended Nguyen's scheme to a universal accumulator by proposing two possible variants: the more efficient α-based construction best suitable when a central authority (the accumulator manager) keeps the accumulator updated, and the alternative more decentralized but less efficient reference string-based construction, whose non-membership witness definition results to be equivalent to Damgård and Triandopoulos' one.

In Chapter 2, although we considered Au *et al.* α-based construction insecure, we concluded that in the presence of an accumulator manager, it is possible to safely use it by replacing the witness defining equations with the one provided in the reference string-based construction (or equivalently, the Damgård and Triandopoulos' construction), with the additional care of properly initializing the accumulator value.

The Dynamic Universal Accumulator obtained by combining Nguyen's positive accumulator and Au et al. and Damgård and Triandopoulos' non-membership witness mechanism will be the starting point of our dynamic universal accumulator scheme,

which we will further extend to support batch operations and a public batch witness update protocol.

3.1.1 Outline

In Section 3.2 we define the underlying dynamic universal accumulator, obtained by combining previous schemes and properly restated in order to address our security model and new features, among which we have the support for batch addition and deletion operations, detailed in Section 3.3. In Section 3.4 we describe the public batch witness update protocol, which allows users to efficiently update their witnesses using information published by the accumulator manager after each batch operation. The security of the resulting scheme is addressed in Section 3.5, while in Section 3.6, we describe how to properly initialize the accumulator so that users will not be able to compute, using the public information available to them, unauthorized witnesses. In Section 3.7 we detail a zero-knowledge proof of knowledge protocol to show knowledge of valid witnesses with respect to a certain accumulator value, while in Section 3.8 we discuss our and community implementations of the resulting extended accumulator scheme.

3.2 A Dynamic Universal Accumulator for Bilinear Groups

We now summarize Nguyen's positive accumulator scheme [Ngu05] (i.e. *Membership Witness, Update and Verification*) extended with the non-membership proof system of Au *et al.* [Au+09] and Damgård and Triandopoulos [DT08] (i.e. *Non-membership Witness, Update and Verification*). We will do so adopting the same notation of Chapter 2, briefly outlined in Section 2.2.

Due to progress in discrete logarithm computations [Adj+15; KW21], which weaken the security of efficient implementable elliptic curves provided with a Type-I pairing, we restate their definitions into a Type-III setting, making it best suitable for efficient and more secure pairing-friendly elliptic curves. In light of this, we will often refer, with a bit of abuse of notation, to elements belonging in the defining groups of the working bilinear group as *points*, thus stressing the fact that in concrete efficient implementations they will correspond to elliptic curve points.

In addition, we introduce new concepts (e.g. *Accumulator States, Epochs*) and parameters (e.g. `batchMax`), to make the accumulator definition coherent with the batch operations and the batch witness update protocol we will describe starting from Section 3.3.

Bilinear Group Generation.[1] Given a security parameter 1^λ, generate a bilinear group $\mathbb{G} = (p, G_1, G_2, G_T, P, \tilde{P}, e)$ where:

- $e : G_1 \times G_2 \to G_T$ is an efficiently computable non-degenerate bilinear map;
- $(G_1, +)$ is an additive group generated by P with identity element $\mathcal{O}$;
- $(G_2, +)$ is an additive group generated by $\tilde{P}$ with identity element $\tilde{\mathcal{O}}$;
- $(G_T, \cdot)$ is an multiplicative group generated by $e(P, \tilde{P})$;
- $|G_1| = |G_2| = |G_T| = p$ is prime;

[1] We refer, for example, to [Ara+13] for more technical details on how these bilinear groups can be efficiently generated and implemented.

Accumulator Parameters. Uniformly sample an $\alpha \in (\mathbb{F}_p)^*$ and consider $\mathcal{ACC} = (\mathbb{F}_p)^* \setminus \{-\alpha\}$ as the domain of accumulatable elements. Moreover, set a bound `batchMax` to the maximum number of batch additions and/or deletions possible in each epoch (we will provide more details in Section 3.3).

The bilinear group $\mathbb{G}$, the bound `batchMax` and the point $\tilde{Q} = \alpha\tilde{P}$ are the accumulator *public parameters* and are available to all accumulator users, while α is the accumulator *secret parameter* and is known only to the accumulator manager.

Accumulator Initialization. Select a set $\mathcal{Y}_{V_0} \subset \mathcal{ACC}$ and let the initial accumulator value to be equal to

$$V_0 = \left(\prod_{y \in \mathcal{Y}_{V_0}} (y + \alpha) \right) P$$

The set $\mathcal{Y}_{V_0}$ is kept secret and its elements are never removed from the accumulator.[2]

Accumulator States and Epochs. An accumulator state is a pair $(V, \mathcal{Y}_V)$ where $V \in G_1$ is the corresponding accumulator value and $\mathcal{Y}_V \subseteq \mathcal{ACC}$ denotes the set of elements accumulated into V (initialization elements excluded). We call *epoch* the period of time during which an accumulator state remains unchanged.

Given an accumulator state $(V, \mathcal{Y}_V)$, the accumulator value V is equal to

$$V = \left(\prod_{y \in \mathcal{Y}_V} (y + \alpha) \right) V_0 = \left(\prod_{y \in \mathcal{Y}_V \cup \mathcal{Y}_{V_0}} (y + \alpha) \right) P$$

and can be computed from $\mathcal{Y}_V$ and V_0 if the secret parameter α is known.

Accumulator Update. The accumulator state $(V, \mathcal{Y}_V)$ changes when one or more elements are added or removed from the accumulator. This can be done using the following single element Addition or Deletion operations.

- **Addition**: if $y \in \mathcal{ACC} \setminus \mathcal{Y}_V$, the element y is added into the accumulator when the accumulator value is updated from V to V' as

  $$V' = (y + \alpha)V$$

  It follows that $\mathcal{Y}_{V'} = \mathcal{Y}_V \cup \{y\}$.

- **Deletion**: if $y \in \mathcal{Y}_V$, the element y is deleted from the accumulator when the accumulator state is updated from V to V' as

  $$V' = \frac{1}{y + \alpha} V$$

  It follows that $\mathcal{Y}_{V'} = \mathcal{Y}_V \setminus \{y\}$.

Membership Witness. Let $(V, \mathcal{Y}_V)$ be an accumulator state and y an element in $\mathcal{ACC}$. Then $w_{y,V}$ is a *membership witness for y with respect to the accumulator value V* if

$$C = \frac{1}{y + \alpha} V$$

[2] Some security properties of the scheme strongly depends on how the elements in $\mathcal{Y}_{V_0}$ are chosen. This is addressed in Section 3.6.

and $w_{y,V} = C$. The accumulator manager issues the membership witness $w_{y,V}$ to a user associated with the element y, in order to permit him to prove that y *is* accumulated into V.[3]

Non-Membership Witness. Let $(V, \mathcal{Y}_V)$ be an accumulator state and y an element in $\mathcal{ACC}$. Then $\bar{w}_{y,V}$ is a *non-membership witness for y with respect to the accumulator state V* if, by letting

$$f_V(x) = \prod_{y_i \in \mathcal{Y}_V \cup \mathcal{Y}_{V_0}} (y_i + x) \quad \in \mathbb{F}_p[x]$$

it holds

$$d = f_V(-y) \bmod p \text{ with } d \neq 0, \qquad C = \frac{f_V(\alpha) - d}{y + \alpha} P$$

and $\bar{w}_{y,V} = (C, d)$. The accumulator manager issues the non-membership witness $\bar{w}_{y,V}$ to a user associated with the element y, in order to permit him to prove that y *is not* accumulated into the accumulator value V.

Witness Update. When accumulator state changes happen, users whose elements are not involved in the corresponding Addition or Deletion operations, have to update their witnesses with respect to the new accumulator state to continue being able to prove statements about their associated elements.

After an accumulator state change, users' membership and non-membership witnesses are updated according to the following operations:

- **On Addition**: suppose the accumulator state changes from $(V, \mathcal{Y}_V)$ to $(V', \mathcal{Y}_{V'})$ as a result of an Addition operation. Hence, for a certain $y' \in \mathcal{ACC} \setminus \mathcal{Y}_V$, $V' = (y' + \alpha)V$ and $\mathcal{Y}_{V'} = \mathcal{Y}_V \cup \{y'\}$.

  Then, for any $y \in \mathcal{Y}_V$ the membership witness $w_{y,V} = C$ is updated with respect to the accumulator state $(V', \mathcal{Y}_{V'})$ by computing

  $$C' = (y' - y)C + V$$

  and letting $w_{y,V'} = C'$, while for any $y \notin \mathcal{Y}_V$, the non-membership $\bar{w}_{y,V} = (C, d)$ is updated, if issued, with respect to $(V', \mathcal{Y}_{V'})$ by computing

  $$C' = (y' - y)C + V, \qquad d' = d \cdot (y' - y)$$

  and letting $\bar{w}_{y,V'} = (C', d')$.

- **On Deletion**: suppose the accumulator state changes from $(V, \mathcal{Y}_V)$ to $(V', \mathcal{Y}_{V'})$ as a result of a Deletion operation. Hence, for a certain $y' \in \mathcal{Y}_V$, $V' = \frac{1}{y'+\alpha} V$ and $\mathcal{Y}_{V'} = \mathcal{Y}_V \setminus \{y'\}$.

  Then, for any $y \in \mathcal{Y}_{V'}$, the membership witness $w_{y,V} = C$ is updated with respect to the accumulator state $(V', \mathcal{Y}_{V'})$ by computing

  $$C' = \frac{1}{y' - y} C - \frac{1}{y' - y} V'$$

[3] When the accumulator is employed as an authentication mechanism, single additions in place of batch operations lack users' privacy and expose to impersonation attacks since the membership witness C would be equal to the previous accumulator state value, while y can be deduced from the public witness update information.

and letting $w_{y,V'} = C'$, while for any $y \notin \mathcal{Y}_{V'}$, the non-membership witness $\bar{w}_{y,V} = (C, d)$ is updated, if issued, with respect to $(V', \mathcal{Y}_{V'})$ by computing

$$C' = \frac{1}{y' - y} C - \frac{1}{y' - y} V', \;\; d' = d \cdot \frac{1}{y' - y}$$

and letting $\bar{w}_{y,V'} = (C', d')$.

Witness Verification. A membership witness $w_y = C$ for an element $y \in \mathcal{ACC}$ is *valid* for the accumulator state $(V, \mathcal{Y}_V)$ if and only if $e(C, y\tilde{P} + \tilde{Q}) = e(V, \tilde{P})$. When w_y is a valid membership witness for the state $(V, \mathcal{Y}_V)$ we assume that $y \in \mathcal{Y}_V$ and hence $w_y = w_{y,V}$.

A non-membership witness $\bar{w}_y = (C, d)$ for an element $y \in \mathcal{ACC}$ is *valid* for the accumulator state $(V, \mathcal{Y}_V)$ if $d \neq 0$ and $e(C, y\tilde{P} + \tilde{Q}) e(P, \tilde{P})^d = e(V, \tilde{P})$. When $\bar{w}_y$ is a valid non-membership witness for the state $(V, \mathcal{Y}_V)$ we assume that $y \notin \mathcal{Y}_V$ and hence $\bar{w}_y = \bar{w}_{y,V}$.

3.3 Adding Support for Batch Operations

We now describe how the Dynamic Universal Accumulator defined in the previous Section can be extended to coherently support batch addition and deletions operations for accumulator updates and user witnesses updates.

We start by defining a family of polynomials which will help us show the correctness of our batch operations compactly with respect to the underlying accumulator scheme.

Batch Polynomials. Given the secret accumulator parameter α and two disjoint sets $\mathcal{A}, \mathcal{D} \subseteq \mathbb{F}_p$ where $\mathcal{A} = \{y_{\mathcal{A},1}, \ldots, y_{\mathcal{A},n}\}$ and $\mathcal{D} = \{y_{\mathcal{D},1}, \ldots, y_{\mathcal{D},m}\}$, we define the following polynomials in $\mathbb{F}_p$:

$$v_{\mathcal{A}}(x) \doteq \sum_{s=1}^{n} \left(\prod_{i=1}^{s-1} (y_{\mathcal{A},i} + \alpha) \prod_{j=s+1}^{n} (y_{\mathcal{A},j} - x) \right)$$

$$v_{\mathcal{D}}(x) \doteq \sum_{s=1}^{m} \left(\prod_{i=1}^{s} (y_{\mathcal{D},i} + \alpha)^{-1} \prod_{j=1}^{s-1} (y_{\mathcal{D},j} - x) \right)$$

$$v_{\mathcal{A},\mathcal{D}}(x) \doteq v_{\mathcal{A}}(x) - v_{\mathcal{D}}(x) \cdot \prod_{i=1}^{n} (y_{\mathcal{A},i} + \alpha)$$

$$d_{\mathcal{A}}(x) \doteq \prod_{t=1}^{n} (y_{\mathcal{A},t} - x) \;, \qquad d_{\mathcal{D}}(x) \doteq \prod_{t=1}^{m} (y_{\mathcal{D},t} - x)$$

Accumulator Batch Update. Several elements are added into or removed from the accumulator using the following Batch Addition and Batch Deletion operations.

- **Batch Addition**: if $\mathcal{A} = \{y_{\mathcal{A},1}, \ldots, y_{\mathcal{A},n}\} \subseteq \mathcal{ACC} \setminus \mathcal{Y}_V$, the elements in $\mathcal{A}$ are *batch added* into the accumulator when the accumulator value is updated from V to V' as

  $$V' = d_{\mathcal{A}}(-\alpha) \cdot V$$

  It follows that $\mathcal{Y}_{V'} = \mathcal{Y}_V \cup \mathcal{A}$.

- **Batch Deletion**: if $\mathcal{D} = \{y_{\mathcal{D},1}, \ldots, y_{\mathcal{D},m}\} \subseteq \mathcal{Y}_V$, the elements in $\mathcal{D}$ are *batch deleted* from the accumulator when the accumulator state is updated from V to V' as
$$V' = \frac{1}{d_{\mathcal{D}}(-\alpha)} \cdot V$$
It follows that $\mathcal{Y}_{V'} = \mathcal{Y}_V \setminus \mathcal{D}$.

- **Batch Addition & Deletion**: if $\mathcal{A} = \{y_{\mathcal{A},1}, \ldots, y_{\mathcal{A},n}\} \subseteq \mathcal{ACC} \setminus \mathcal{Y}_V$, $\mathcal{D} = \{y_{\mathcal{D},1}, \ldots, y_{\mathcal{D},m}\} \subseteq \mathcal{Y}_V$ and $\mathcal{A} \cap \mathcal{D} = \emptyset$, the elements in $\mathcal{A}$ are batch added into the accumulator and the elements in $\mathcal{D}$ are batch deleted from the accumulator when the accumulator state is updated from V to V'' as
$$V'' = \frac{d_{\mathcal{A}}(-\alpha)}{d_{\mathcal{D}}(-\alpha)} \cdot V$$
It follows that $\mathcal{Y}_{V''} = \mathcal{Y}_V \cup \mathcal{A} \setminus \mathcal{D}$.

Batch Witness Update. When a batch addition or deletion changes the accumulator state, users' membership and non-membership witnesses are updated according to the following operation.

- **On Batch Addition:** suppose the accumulator state changes from $(V, \mathcal{Y}_V)$ to $(V', \mathcal{Y}_{V'})$ as a result of an Batch Addition operation. Hence, for certain $\mathcal{A} = \{y_{\mathcal{A},1}, \ldots, y_{\mathcal{A},n}\} \subseteq \mathcal{ACC} \setminus \mathcal{Y}_V$, we have $V' = d_{\mathcal{A}}(-\alpha) \cdot V$ and $\mathcal{Y}_{V'} = \mathcal{Y}_V \cup \mathcal{A}$.
Then, for any $y \in \mathcal{Y}_V$, the membership witness $w_{y,V} = C$ is updated with respect to the accumulator state $(V', \mathcal{Y}_{V'})$ computing
$$C' = d_{\mathcal{A}}(y) \cdot C + v_{\mathcal{A}}(y) \cdot V$$
and letting $w_{y,V'} = C'$. While for any $y \notin \mathcal{Y}_V$, the non-membership witness $\bar{w}_{y,V} = (C, d)$ is updated with respect to $(V', \mathcal{Y}_{V'})$ by computing
$$C' = d_{\mathcal{A}}(y) \cdot C + v_{\mathcal{A}}(y) \cdot V, \qquad d' = d \cdot d_{\mathcal{A}}(y)$$
and letting $\bar{w}_{y,V'} = (C', d')$.

Proof. For the ease of notation, we will denote the elements $y_{\mathcal{A},i}$ with y_i, the accumulator value corresponding to $\left(\prod_{i=1}^{j}(y_i + \alpha)\right) V$ with V_j and, for any $y \in \mathcal{Y}_V$, the intermediate membership witnesses w_{y,V_j} with C_j.
We prove the formula by induction on n, the number of batch added elements:
$\boxed{n = 1}$ We get $C_1 = V + (y_1 - y)C$, the same formula defined for the membership witness update after a single addition operation.
$\boxed{n - 1 \to n}$ For ease of exposition, we denote $v_{\mathcal{A}}(y) = \sum_{s=1}^{n} b_{s,n}$, where
$$b_{s,n} = \prod_{i=1}^{s-1}(y_{\mathcal{A}_i} + \alpha) \prod_{j=s+1}^{n}(y_{\mathcal{A}_j} - y)$$

Using the inductive hypothesis for C_{n-1}, we have

$$\begin{aligned}
C_n &= (y_n - y)C_{n-1} + V_{n-1} \\
&= (y_n - y)\left(\left(\prod_{t=1}^{n-1} y_t - y\right)C + \left(\sum_{s=1}^{n-1}\left(\prod_{i=1}^{s-1}(y_i+\alpha)\prod_{j=s+1}^{n-1}(y_j - y)\right)\right)V\right) + V_{n-1} \\
&= \left(\prod_{t=1}^{n} y_t - y\right)C + \left(\sum_{s=1}^{n-1}\left(\prod_{i=1}^{s-1}(y_i+\alpha)\prod_{j=s+1}^{n}(y_j - y)\right)\right)V + V_{n-1} \\
&= \left(\prod_{t=1}^{n}(y_t - y)\right)C + \left(\sum_{s=1}^{n-1} b_{s,n} + \prod_{t=1}^{n-1}(y_t+\alpha)\right)V \\
&= \left(\prod_{t=1}^{n}(y_t - y)\right)C + \left(\sum_{s=1}^{n} b_{s,n}\right)V
\end{aligned}$$

as required. The induction on d' in the case of non-membership witnesses is straightforward. □

- **On Batch Deletion**: suppose the accumulator state changes from $(V, \mathcal{Y}_V)$ to $(V', \mathcal{Y}_{V'})$ as a result of a Batch Deletion operation. Hence, for certain $\mathcal{D} = \{y_{\mathcal{D},1}, \ldots, y_{\mathcal{D},m}\} \subseteq \mathcal{Y}_V$, we have $V' = \frac{1}{d_{\mathcal{D}}(-\alpha)}V$.

  Then, for any $y \in \mathcal{Y}_{V'}$, the witness $w_{y,V} = C$ is updated with respect to the accumulator state $(V', \mathcal{Y}_{V'})$ computing

$$C' = \frac{1}{d_{\mathcal{D}}(y)}C - \frac{v_{\mathcal{D}}(y)}{d_{\mathcal{D}}(y)}V$$

  and letting $w_{y,V'} = C'$. While for any $y \notin \mathcal{Y}_{V'}$, the non-membership witness $\bar{w}_{y,V} = C$ is updated with respect to $(V', \mathcal{Y}_{V'})$ by computing

$$C' = \frac{1}{d_{\mathcal{D}}(y)} \cdot C - \frac{v_{\mathcal{D}}(y)}{d_{\mathcal{D}}(y)} \cdot V, \qquad d' = d \cdot \frac{1}{d_{\mathcal{D}}(y)}$$

  and letting $\bar{w}_{y,V'} = (C', d')$.

  Proof. Similarly as before, we will denote the elements $y_{\mathcal{D},i}$ with y_i, the accumulator value corresponding to $\left(\prod_{i=1}^{j}(y_i+\alpha)^{-1}\right)V$ with V_j and, for any $y \in \mathcal{Y}_{V'}$, the intermediate membership witnesses w_{y,V_j} with C_j.

  We prove the formula by induction on m, the number of batch deleted elements:
  $\boxed{m = 1}$: We get $C_1 = \frac{1}{y_1 - y}C - \frac{1}{(y_1-y)(y_1+\alpha)}V = \frac{1}{y_1-y}(C - V_1)$, the same formula defined for the membership witness update after a single deletion operation.
  $\boxed{m - 1 \to m}$: Let us denote $v_{\mathcal{D}}(y) = \sum_{s=1}^{m} b_s$, where

$$b_s = \prod_{i=1}^{s}(y_i+\alpha)^{-1}\prod_{j=1}^{s-1}(y_j - y)$$

We then have

$$\begin{aligned} C_m &= \frac{1}{y_m - y}(C_{m-1} - V_m) \\ &= \frac{1}{d_{\mathcal{D}}(y)}C - \frac{1}{d_{\mathcal{D}}(y)} \cdot \left(\sum_{s=1}^{m-1} b_s\right) V - \left(\frac{1}{y_m - y}\prod_{i=1}^{m}(y_i + \alpha)^{-1}\right) V \\ &= \frac{1}{d_{\mathcal{D}}(y)}C - \frac{1}{d_{\mathcal{D}}(y)} \cdot \left(\sum_{s=1}^{m} b_s\right) V \end{aligned}$$

as required. The induction on d' in the case of non-membership witnesses is straightforward. □

- **On Batch Addition & Deletion:** suppose the accumulator state changes from $(V, \mathcal{Y}_V)$ to $(V'', \mathcal{Y}_{V'})$ as a result of a Batch Addition & Deletion operation. Hence for certain disjoint sets $\mathcal{A} = \{y_{\mathcal{A},1}, \ldots, y_{\mathcal{A},n}\} \subseteq \mathcal{ACC} \setminus \mathcal{Y}_V$ and $\mathcal{D} = \{y_{\mathcal{D},1}, \ldots, y_{\mathcal{D},m}\} \subseteq \mathcal{Y}_V$ we have $V'' = \frac{d_{\mathcal{A}}(-\alpha)}{d_{\mathcal{D}}(-\alpha)} \cdot V$.

  Then, for any $y \in \mathcal{Y}_V$, the witness $w_{y,V} = C$ is updated with respect to the accumulator state $(V', \mathcal{Y}_{V'})$ computing

$$C' = \frac{d_{\mathcal{A}}(y)}{d_{\mathcal{D}}(y)} \cdot C + \frac{\nu_{\mathcal{A},\mathcal{D}}(y)}{d_{\mathcal{D}}(y)} \cdot V$$

  and letting $w_{y,V'} = C'$. While for any $y \notin \mathcal{Y}_V$, the non-membership witness $\bar{w}_{y,V} = (C, d)$ is updated with respect to the accumulator state $(V', \mathcal{Y}_{V'})$ by computing

$$C' = \frac{d_{\mathcal{A}}(y)}{d_{\mathcal{D}}(y)} \cdot C + \frac{\nu_{\mathcal{A},\mathcal{D}}(y)}{d_{\mathcal{D}}(y)} \cdot V, \qquad d' = d \cdot \frac{d_{\mathcal{A}}(y)}{d_{\mathcal{D}}(y)}$$

  and letting $\bar{w}_{y,V'} = (C', d')$.

  Proof. Performing a batch addition and then a batch deletion, the membership witness $w_{y,V} = C$ for y with respect to the accumulator value V is iteratively updated to $\bar{w}_{y,V''} = (C'', d'')$ with respect to the updated accumulator value $V'' = \left(\frac{\prod_{i=1}^{n}(y_{\mathcal{A},i}+\alpha)}{\prod_{i=1}^{m}(y_{\mathcal{D},i}+\alpha)}\right) V$ as follows

$$C \quad \xrightarrow{\text{Add}} \quad C' = d_{\mathcal{A}}(y)C + v_{\mathcal{A}}(y)V \quad \xrightarrow{\text{Delete}}$$

$$C'' = \frac{1}{d_{\mathcal{D}}(y)}C' - \frac{v_{\mathcal{D}}(y)}{d_{\mathcal{D}}(y)}V' = \frac{d_{\mathcal{A}}(y)}{d_{\mathcal{D}}(y)}C + \left(\frac{v_{\mathcal{A}}(y)}{d_{\mathcal{D}}(y)} - \frac{v_{\mathcal{D}}(y)}{d_{\mathcal{D}}(y)} \cdot \prod_{i=1}^{n}(y_{\mathcal{A},i} + \alpha)\right) V$$

  where $V' = \prod_{i=1}^{n}(y_{\mathcal{A},i} + \alpha) \cdot V$. The induction on d' in the case of non-membership witnesses is straightforward. □

3.4 The Batch Witness Update Protocol

Users cannot batch update their witnesses directly using the formula defined in the previous Section, since they would need the secret parameter α. However, starting from their definition, the accumulator manager can efficiently compute and publish

some update information (more precisely, the polynomials $d_{\mathcal{A}}(x), d_{\mathcal{D}}(x)$ and a *points vector*) so that users can update their witnesses without requiring or leaking any information related to α. This will allow us to define a batch membership and non-membership witness update protocol for the proposed accumulator scheme.

3.4.1 The Batch Witness Update Information

From now on, we will focus on the Batch Witness Update Addition & Deletion polynomial $v_{\mathcal{A},\mathcal{D}}(x)$ only: indeed, the polynomials $v_{\mathcal{A}}(x)$ and $v_{\mathcal{D}}(x)$ are special cases of this more general one.

We recall that our main goal is to allow users possessing a witness (C, d) for an element y with respect to the accumulator value V to compute the quantities

$$C' = \frac{d_{\mathcal{A}}(y)}{d_{\mathcal{D}}(y)} \cdot C + \frac{\nu_{\mathcal{A},\mathcal{D}}(y)}{d_{\mathcal{D}}(y)} \cdot V \ , \qquad d' = d \cdot \frac{d_{\mathcal{A}}(y)}{d_{\mathcal{D}}(y)}$$

We note that the accumulator manager cannot publish all the polynomials $d_{\mathcal{A}}(x)$, $d_{\mathcal{D}}(x)$ and $v_{\mathcal{A},\mathcal{D}}(x)$, because their coefficients can leak some information related to the secret accumulator parameter α. To give an example, suppose that after a batch addition operation, the accumulator manager publishes the polynomials $v_{\mathcal{A}}(x)$ and $d_{\mathcal{A}}(x)$, defined as above, with $|\mathcal{A}| > 1$. Doing simple algebra, we find that the coefficient of the $(|\mathcal{A}| - 2)-$degree monomial of $v_{\mathcal{A}}(x)$ is equal to $\alpha + \sum_{y_A \in \mathcal{A}} y_A$: by extracting the roots of $d_{\mathcal{A}}(x)$ in $\mathbb{F}_p$ we obtain all the elements in $\mathcal{A}$ and hence the secret parameter α.

Leakages about α can be prevented by requiring the accumulator manager to publish in place of $v_{\mathcal{A},\mathcal{D}}(x)$, the *vector of points*

$$\Omega = \Omega_{\mathcal{A},\mathcal{D},V} = (\ c_0 V,\ c_1 V,\ \ldots\ ,\ c_{\texttt{batchMax}} V\ )$$

where $v_{\mathcal{A},\mathcal{D}}(x) = \sum_{i=0}^{\texttt{batchMax}} c_i x^i$ and $c_i = 0$ if $i > max(|\mathcal{A}|, |\mathcal{D}|)$.

Users can then update their membership witness $w_{y,V} = C$ to $w_{y,V'} = C'$ by first evaluating the two polynomials $d_{\mathcal{A}}(x)$ and $d_{\mathcal{D}}(x)$ in the element y and then computing

$$C' = \frac{d_{\mathcal{A}}(y)}{d_{\mathcal{D}}(y)} \cdot C + \frac{1}{d_{\mathcal{D}}(y)} \cdot \langle \mathrm{Y}_y, \Omega \rangle$$

where $\mathrm{Y}_y = (1, y, y^2, ..., y^{\texttt{batchMax}})$ and $\langle \cdot, \cdot \rangle$ denotes the dot product.

Similarly, a non-membership witness $\bar{w}_{y,V} = (C, d)$ is updated to $\bar{w}_{y,V'} = (C', d')$ by computing

$$C' = \frac{d_{\mathcal{A}}(y)}{d_{\mathcal{D}}(y)} \cdot C + \frac{1}{d_{\mathcal{D}}(y)} \cdot \langle \mathrm{Y}_y, \Omega \rangle \ , \qquad d' = d \cdot \frac{d_{\mathcal{A}}(y)}{d_{\mathcal{D}}(y)}$$

In this scenario, assuming the Discrete Logarithm Problem to be hard in G_1 (a weaker assumption with respect to the generalized $t-$SDH assumption under which accumulator collision resistance is shown in Section 3.5), from the published Ω, $d_{\mathcal{A}}(x)$ and $d_{\mathcal{D}}(x)$ it is only possible, performing roots extraction on the polynomials, to compute the respective sets $\mathcal{A}$ and $\mathcal{D}$ of batch added and batch deleted elements.

It follows that witness update operations can be performed either autonomously by users or by delegating to third-party servers the computation of (some of) the values $\langle \mathrm{Y}_y, \Omega \rangle$, $d_{\mathcal{A}}(y), d_{\mathcal{D}}(y)$. Indeed, since the required updating values are decoupled from users' previous witnesses, third-party servers which are asked to compute such values

with respect to an element y, cannot impersonate the corresponding user, since they do not know any previous valid witness for y.

When the computation of the elements $\langle \mathrm{Y}_y, \Omega \rangle$, $d_{\mathcal{A}}(y), d_{\mathcal{D}}(y)$ is delegated, users are then able to update witnesses in a constant number of elementary operations, i.e., 2 scalar-point multiplication and 1 point addition[4]. Delegation thus allows even resource-constrained devices to be able to keep users' witnesses updated.

As a side note, if third-party servers are *untrusted* (e.g. we want to prevent linkability attacks from subsequent evaluation requests for the same element y), it is possible to use Oblivious Polynomial Evaluation techniques such as [NP06], [Haz15] and [CL01] to delegate the computation of the point $\langle \mathrm{Y}_y, \Omega \rangle = v_{\mathcal{A},\mathcal{D}}(y) \cdot V$ and of the values $d_{\mathcal{A}}(y)$ and $d_{\mathcal{D}}(y)$, in a way that third-parties will not learn anything about y. We note, however, that such protocols have time complexity at least proportional to the degree of the polynomials involved and thus allow users to just save data rather than time, i.e., users are not required to download the public batch witness update data (available instead to third-party servers) and can oblivious evaluate $v_{\mathcal{A},\mathcal{D}}(y) \cdot V$, $d_{\mathcal{A}}(y)$ and $d_{\mathcal{D}}(y)$ with time complexities comparable to standard polynomial evaluations.

Improvements With Respect to Non-Batch Operations. Due to the lower bound showed by Camacho and Hevia in [CH10], in case of a batch addition and deletion operation where m elements are added and/or deleted, the batch update data cannot have size less than $O(m)$ (and thus witnesses cannot be updated in time less than $O(m)$), as in the case for non-batch operations. Our protocol reaches this optimal lower bound and provides better constants compared to the naive approach of iteratively adding and/or deleting each involved element at a time.

In particular, if the bilinear group is implemented using a pairing-friendly elliptic curve over $\mathbb{F}_q$, for m added and deleted elements, the public batch witness information in our protocol would have size $|\Omega| + |d_{\mathcal{A}}(x)| + |d_{\mathcal{D}}(x)|$, i.e. $m \cdot (\log q + 2 \log p)$, while the naive approach consisting in executing m addition operations followed by m deletion operations[5] requires $2m \cdot (\log q + \log p)$ data. Furthermore, as regards time complexities to update witnesses, our protocol requires $m + 2$ scalar-point multiplications, 1 point addition and 2 degree-m polynomials evaluations (possible in $2m$ multiplications and $2m$ additions), while the naive approach requires $2m$ scalar-point multiplications, $2m$ point additions and $2m$ multiplications.

To summarise, with respect to the naive approach, our protocol provides, approximately, the following improvements:

- $1/4$ reduction in witness update data communication;
- $1/2$ reduction in running time in order to update witnesses.

3.4.2 Batch Witness Update Among Epochs

We now show how the adoption of the points vector $\Omega = \Omega_{\mathcal{A},\mathcal{D},V}$ not only permits the users to batch update their (non-)membership witnesses from the previous accumulator state, but also enables them to directly update from the accumulator state of any older epoch. This feature doesn't force users to permanently keep their witnesses

[4] We note that this is not against the impossibility result of Camacho and Hevia to have batch witness update data size independent from the number of elements added/deleted, since the provided values $\langle \mathrm{Y}_y, \Omega \rangle$, $d_{\mathcal{A}}(y), d_{\mathcal{D}}(y)$ are *per* user and not for *all* users, similarly as any constant-sized (updated) witness is.

[5] Each of these operations send users the element added/deleted and the corresponding updated accumulator value, i.e. $\log q + \log p$ data.

Epoch	Accumulator State	Witness Update Information		
0	$(V_0, \emptyset)$			
1	$(V_1, \mathcal{Y}_{V_1})$	Ω_1	$d_{\mathcal{A}_1}(x)$	$d_{\mathcal{D}_1}(x)$
$\vdots$	$\vdots$		$\vdots$	
i	$(V_i, \mathcal{Y}_{V_i})$	Ω_i	$d_{\mathcal{A}_i}(x)$	$d_{\mathcal{D}_i}(x)$

TABLE 3.1: Data published by the accumulator manager in each epoch.

updated to the latest accumulator state, enabling them to update their witnesses just right before they want to prove statements about the associated element y.

Before showing how this is possible, we extend our notation to associate accumulator and batch witness update data to a specific epoch. Given an epoch $i > 0$, we denote with $(V_i, \mathcal{Y}_{V_i})$ the corresponding accumulator state, where $\mathcal{Y}_{V_1} = \mathcal{A}_1 \setminus \mathcal{D}_1$ and $\mathcal{Y}_{V_i} = \mathcal{Y}_{V_{i-1}} \cup \mathcal{A}_i \setminus \mathcal{D}_i$ for $i > 1$, with $d_{\mathcal{A}_i}(x)$ and $d_{\mathcal{D}_i}(x)$ the addition and deletion batch witness update polynomials, respectively, and with $\Omega_i = \Omega_{\mathcal{A}_i, \mathcal{D}_i, V_i}$. An overview of the data published by the accumulator manager is given in Table 3.1.

We further denote a membership witness w_{y,V_i} for an element y with respect to the accumulator value V_i as $w_{y,V_i} = C_i$ and, similarly, a non-membership witness $\bar{w}_{y,V_i}$ as $\bar{w}_{y,V_i} = (C_i, d_i)$.

Epoch Witnesses Batch Update. A user who owns a valid non-membership witness $\bar{w}_{y,V_i} = (C_i, d_i)$ (resp. a valid membership witness $w_{y,V_i} = C_i$) with respect to the accumulator state $(V_i, \mathcal{Y}_{V_i})$ can update it to $\bar{w}_{y,V_j} = (C_j, d_j)$ (resp. $w_{y,V_j} = C_j$), for any $j > i$, as

$$C_j = \frac{d_{\mathcal{A}_{i\to j}}(y)}{d_{\mathcal{D}_{i\to j}}(y)} \cdot C_i + \frac{1}{d_{\mathcal{D}_{i\to j}}(y)} \cdot \langle\, \mathrm{Y}_y \;,\; \Omega_{i\to j}(y) \,\rangle \;, \qquad d_j = d_i \cdot \frac{d_{\mathcal{A}_{i\to j}}(y)}{d_{\mathcal{D}_{i\to j}}(y)}$$

where

$$d_{\mathcal{A}_{a\to b}}(x) = \prod_{s=a+1}^{b} d_{\mathcal{A}_s}(x) \qquad d_{\mathcal{D}_{a\to b}}(x) = \prod_{s=a+1}^{b} d_{\mathcal{D}_s}(x)$$

$$\Omega_{i\to j}(y) = \sum_{t=i+1}^{j} \Big(d_{\mathcal{D}_{i\to t-1}}(y) \cdot d_{\mathcal{A}_{t\to j}}(y) \Big) \circ \Omega_t$$

Proof. We prove the result by induction on $j > i$.
$\boxed{j = i+1}$ A witness $\bar{w}_{y,V_i} = (C_i, d_i)$ is updated to $\bar{w}_{y,V_{i+1}} = (C_{i+1}, d_{i+1})$ as

$$C_{i+1} = \frac{d_{\mathcal{A}_{i+1}}(y)}{d_{\mathcal{D}_{i+1}}(y)} \cdot C_i + \frac{1}{d_{\mathcal{D}_{i+1}}(y)} \cdot \langle\, \mathrm{Y}_y \;,\; \Omega_{i+1} \,\rangle \;, \qquad d_{i+1} = d_i \cdot \frac{d_{\mathcal{A}_{i+1}}(y)}{d_{\mathcal{D}_{i+1}}(y)}$$

obtaining the same result we get by using the formula for non-membership witnesses batch update.

$\boxed{j \Rightarrow j+1:}$ By inductive hypothesis, we assume the formula holds for C_j. Then

$$\begin{aligned} C_{j+1} =& \frac{d_{\mathcal{A}_{j+1}}(y)}{d_{\mathcal{D}_{j+1}}(y)} \cdot C_j + \frac{1}{d_{\mathcal{D}_{j+1}}(y)} \cdot \langle\, \mathrm{Y}_y \,,\, \Omega_{j+1} \,\rangle \\ =& \frac{d_{\mathcal{A}_{i\to j+1}}(y)}{d_{\mathcal{D}_{i\to j+1}}(y)} \cdot C_i + \frac{1}{d_{\mathcal{D}_{i\to j+1}}(y)} \cdot \left\langle\, \mathrm{Y}_y \,,\, d_{\mathcal{A}_{j+1}}(y) \circ \Omega_{i\to j} \,\right\rangle \\ &+ \frac{1}{d_{\mathcal{D}_{i\to j+1}}(y)} \cdot \langle\, \mathrm{Y}_y \,,\, d_{\mathcal{D}_{i\to j}}(y) \circ \Omega_{j+1} \,\rangle \\ =& \frac{d_{\mathcal{A}_{i\to j+1}}(y)}{d_{\mathcal{D}_{i\to j+1}}(y)} \cdot C_i + \frac{1}{d_{\mathcal{D}_{i\to j+1}}(y)} \cdot \langle\, \mathrm{Y}_y \,,\, \Omega_{i\to j+1} \,\rangle \end{aligned}$$

as required, since

$$\begin{aligned} \Omega_{i\to j+1} &= \sum_{t=i+1}^{j+1} \left(\prod_{h=i+1}^{t-1} d_{\mathcal{D}_h}(y) \prod_{k=t+1}^{j+1} d_{\mathcal{A}_k}(y) \right) \circ \Omega_t \\ &= \left(\sum_{t=i+1}^{j} \left(\prod_{h=i+1}^{t-1} d_{\mathcal{D}_h}(y) \prod_{k=t+1}^{j+1} d_{\mathcal{A}_k}(y) \right) \circ \Omega_t \right) + \left(\prod_{h=i+1}^{j} d_{\mathcal{D}_h}(y) \right) \circ \Omega_{j+1} \\ &= d_{\mathcal{A}_{j+1}}(y) \circ \Omega_{i\to j} + d_{\mathcal{D}_{i\to j}}(y) \circ \Omega_{j+1} \end{aligned}$$

The proof on d_j is straightforward. □

3.5 Security Proofs for the Proposed Protocol

We recall that security of accumulator schemes is usually intended as *collision resistance*: for universal accumulators, this property requires that an adversary forges with negligible probability in the security parameter λ a valid membership witness for a not-accumulated element and, respectively, a non-membership witness for an accumulated element.

Since the outlined Dynamic Universal Accumulator is built on top of Nguyen's positive dynamic accumulator [Ngu05] and Au et al. [Au+09] and Damgård and Triandopoulos' non-membership proof system [DT08], we might be tempted to generalize the security proofs provided in [Ngu05; DT08] (reported in Section 2.4) to show the security of our scheme under the standard t-Strong Diffie-Hellman assumption (Definition 2.2).

However, some technicalities prevent us from doing so straightforwardly: i) in the proposed protocol, the attacker does not necessarily have access to the $\mathcal{RS} = \{P, \alpha P, \ldots, \alpha^t P\}$ (needed in [Ngu05; Au+09; DT08] security reductions, see Theorem 2.1), while he has access to the batch witness update information and valid witnesses, as regular users do; ii) differently than [Ngu05; Au+09; DT08], we allow the accumulator manager to initialize the accumulator value to V_0 by accumulating a certain number of secret values.

To show the security of our proposed accumulator scheme, we then need to provide two slightly more general definitions tailored to the data our attacker would be able to access. We propose the following.

Definition 3.1. (Collision Resistance) *Let $\mathcal{A}$ be a probabilistic polynomial time adversary that has access to an oracle $\mathcal{O}$ which replies to:*

- *"Batch Addition and/or Deletion" queries that batch add non-accumulated and/or delete accumulated elements into/from the accumulator (which is initialized to*

V_0) and return the resulting updated accumulator value and the corresponding public batch witness update data;

- *"Issue Witness" queries that return, for any input element y, its membership witness if y is accumulated, or, if not, its non-membership witness with respect to the latest accumulator value.*

Then, the proposed Dynamic Universal Accumulator is collision resistant if the probability

$$\mathbb{P}\left(\begin{array}{ccc} (\mathbb{G},\alpha,\mathcal{Y}_{V_0},\tilde{Q}) \leftarrow Gen(1^\lambda)\,, & & f(x)=\prod_{y_i\in\mathcal{Y}_{V_0}}(y_i+x), \\ V_0=f(\alpha)\cdot P\,, & & (y,w_y,\bar{w}_y,\mathcal{Y}) \leftarrow \mathcal{A}^{\mathcal{O}}(V_0,\mathbb{G},\tilde{Q}) \quad : \\ \mathcal{Y}\subseteq(\mathbb{F}_p)^* & \wedge & V=\left(\prod_{y_i\in\mathcal{Y}}(y_i+\alpha)\right)\cdot V_0 \quad \wedge \\ \Omega(y,w_y,V,\texttt{membership})=1 & \wedge & \Omega(y,\bar{w}_y,V,\texttt{non-membership})=1 \end{array}\right)$$

is a negligible function in the security parameter λ, where w_y, $\bar{w}_y$ denote a membership and non-membership witness for y, respectively, and $\Omega(y,w,V,\texttt{type})=1$ if and only if w is a valid $\texttt{type}$ witness for y with respect to V.

Proposition 3.1. *Collision Resistance of Definition 3.1 is weaker than Au et al. Collision resistance (Definition 2.1) when* $\deg f>0$*, while it is equivalent if* $\deg f=0$*.*

Proof. In Lemma 2.1 we proved that the oracle $\mathcal{O}$ of Definition 2.1 gives the attacker access to the $\mathcal{RS}_t$ for some $t>0$. By using the Extended Euclidean Algorithm, the set $\mathcal{RS}_t$ further allows the attacker to issue valid membership witnesses for accumulated elements and valid non-membership witnesses for any non-accumulated element, as originally reported in [Au+09]. In other words, an attacker that successfully breaks collision resistance of Definition 2.1, can output in polynomial time using the $\mathcal{RS}_t$ a valid membership witness if he forged a non-membership witness for an accumulated element y, or, similarly, the valid non-membership witness, if he forged a membership witness for a non-accumulated element y. In fact, the two probabilities of Definition 2.1 can be combined by equivalently requiring that

$$\mathbb{P}\left(\begin{array}{ccc} (\mathbb{G},\alpha,\tilde{Q}) \leftarrow \mathrm{Gen}(1^\lambda), & & (y,w_y,\bar{w}_y,\mathcal{Y}) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathbb{G},\tilde{Q}) \quad : \\ \mathcal{Y}\subseteq(\mathbb{F}_p)^* & \wedge & V=\left(\prod_{y_i\in\mathcal{Y}}(y_i+\alpha)\right)\cdot P \quad \wedge \\ \Omega(y,w_y,V,\texttt{membership})=1 & \wedge & \Omega(y,\bar{w}_y,V,\texttt{non-membership})=1 \end{array}\right)$$

is negligible in the security parameter λ.

Since the public batch update information can be computed in polynomial time from the $\mathcal{RS}_t$ (we can compute any element of the form $h(\alpha)P$, where $h(x)\in\mathbb{F}_p$ has degree $\leq t$), it immediately follows that Definition 3.1 is equivalent to Definition 2.1 if $\deg f=0$, i.e. $f=1$ and thus $V_0=P$.

If instead $\deg f>0$, an attacker that breaks collision resistance of Definition 3.1 by outputting a tuple $(y,w_y,\bar{w}_y,\mathcal{Y})$, can, before terminating, query in polynomial time the corresponding oracle $\mathcal{O}$ to get $|\mathcal{Y}_{V_0}|+1$ valid non-membership witnesses for (random) non-accumulated elements, and use Lagrange interpolation to recover from the d-values the polynomial $f(x)\in\mathbb{F}_p$, similarly as done at the beginning of the Witness Forgery Attack outlined in Attack 1. Once the attacker obtains the polynomial $f(x)$, he recovers the set $\mathcal{Y}_{V_0}$ by computing its roots, and can then use the tuple $(y,w_y,\bar{w}_y,\mathcal{Y}\cup\mathcal{Y}_{V_0})$ to break collision resistance of Definition 2.1. □

Definition 3.2. (Generalized $t-$Strong Diffie-Hellman Assumption) *Let $\mathcal{G}$ be a probabilistic polynomial time algorithm that, given a security parameter 1^λ, outputs a bilinear group $\mathbb{G} = (p, G_1, G_2, G_T, P, \tilde{P}, e)$. We say that the generalized $t-$Strong Diffie-Hellman Assumption holds for $\mathcal{G}$ with respect to a uniformly sampled $\alpha \leftarrow (\mathbb{F}_p)^*$ and a non-zero $f(x) \in \mathbb{F}_p[x]$ if, for any probabilistic polynomial time adversary $\mathcal{A}$ and for every polynomially bounded function $t : \mathbb{Z} \to \mathbb{Z}$, the probability*

$$\mathbb{P}\left(\mathcal{A}(P, \alpha f(\alpha)P, \alpha^2 f(\alpha)P, ..., \alpha^{t(\lambda)} f(\alpha)P, \tilde{P}, \alpha\tilde{P}) = \left(y, \frac{1}{y+\alpha}P\right)\right)$$

is a negligible function in λ for any freely chosen value $y \in \mathbb{F}_p \setminus \{-\alpha\}$.

To give confidence in this more general security assumption, we prove the following Theorem which gives a lower bound on the complexity of a generic algorithm that solves the Generalized $t-$SDH Assumption in the Generic Group Model [Sho97].

We briefly recall that in the Generic Group Model [Sho97] elements in the three groups G_1, G_2, G_T are represented with strings given by (random) unique *encoding* functions $\xi_i : G_i \to \{0,1\}^*$. Operations with groups elements (additions, pairings, isomorphism computations $\psi : G_2 \to G_1$) are performed by querying different oracles which communicates with the external world only by using ξ_i-encoding of group elements. In other words, an adversary who interacts with these oracles can only test equality among received encodings in order to understand relations between group elements.

Theorem 3.1. *Let $\mathcal{A}$ be an algorithm that solves the corresponding generalized $t-$SDH problem in the Generic Group Model, making a total of at most q_G queries to the oracles computing the group action in G_1, G_2, G_T, the oracle computing the isomorphism $\psi : G_2 \to G_1$ and the oracle computing the bilinear pairing e. If $\alpha \in (\mathbb{F}_p)^*$ and the encoding functions ξ_1, ξ_2, ξ_T are chosen at random, then the probability ϵ that*

$$\mathcal{A}\left(p, \xi_1(1), \xi_1(f(\alpha)), \xi_1(\alpha \cdot f(\alpha)), \ldots, \xi_1(\alpha^t \cdot f(\alpha)), \xi_2(1), \xi_2(\alpha)\right)$$

outputs $\left(y, \xi_1\left(\frac{1}{y+\alpha}\right)\right)$ with $y \in (\mathbb{F}_p)^*$ is bounded by

$$\epsilon \leq \frac{(\deg f + t) \cdot (q_G + t + 4)^2 + 1}{p}$$

Proof. We will essentially go through the original proof of Boneh and Boyen [BB04] of the generic security of the standard $t-$SDH assumption (Definition 2.2) by slightly readapting it to the definition of the generalized $t-$SDH assumption. The following game setting and query definitions are due to Boneh and Boyen [BB04] as well.

Let $\mathcal{B}$ be an algorithm that maintains three lists of pairs

$$L_j = \{(F_{j,i}, \xi_{j,i}) \ : \ i = 0, \ldots, \tau_j - 1\} \text{ with } j = 1, 2, T$$

where $F_{1,i}$, $F_{2,i}$ and $F_{T,i}$ are polynomials in $\mathbb{F}_p[x]$ verifying $\deg F_{1,i} \leq \deg f + t$, $\deg F_{2,i} \leq t$ and $\deg F_{T,i} \leq 2t$ and such that at step τ of the game $\tau_1 + \tau_2 + \tau_T = \tau + t + 3$. The lists are initialized at step $\tau = 0$ by taking $\tau_1 = t + 1$, $\tau_2 = 2$, $\tau_T = 0$ and letting $F_{1,0} = 1$, $F_{1,i} = x^i \cdot f(x)$ for $0 \leq i \leq t$ and $F_{2,i} = x^i$ with $i = 0, 1$. The corresponding $\xi_{j,i}$ encodings are set to arbitrary distinct strings in $\{0,1\}^*$. $\mathcal{B}$ then starts a game by providing $\mathcal{A}$ the $q + 3$ encodings $\xi_{1,0}, \ldots, \xi_{1,q}, \xi_{2,0}, \xi_{2,1}$ and $\mathcal{A}$'s queries go as follows:

- *Group actions*: given an add (resp. subtract) query and two operands $\xi_{1,i}, \xi_{1,j}$ with $0 \leq i, j < \tau_1$, $\mathcal{B}$ computes $F_{1,\tau_1} \leftarrow F_{1,i} + F_{1,j}$ (resp. $F_{1,i} - F_{1,j}$). If $F_{1,\tau_1} = F_{1,l}$ for some $l < \tau_1$ then $\mathcal{B}$ sets $\xi_{1,\tau_1} = \xi_{1,l}$, otherwise sets ξ_{1,τ_1} to a new distinct string in $\{0,1\}^*$. The pair $(F_{1,\tau_1}, \xi_{1,\tau_1})$ is added in L_1, τ_1 is incremented by 1 and ξ_{1,τ_1} is returned to $\mathcal{A}$. Operations in G_2, G_T are treated similarly.

- *Isomorphism*: given an encoding $\xi_{2,i}$ with $0 \leq i < \tau_2$, $\mathcal{B}$ sets $F_{1,\tau_1} \leftarrow F_{2,i}$. If $F_{1,\tau_1} = F_{1,l}$ for some $l < \tau_1$, then $\mathcal{B}$ sets $\xi_{1,\tau_1} \leftarrow \xi_{1,l}$, otherwise sets ξ_{1,τ_1} to a new distinct string in $\{0,1\}^*$. The pair $(F_{1,\tau_1}, \xi_{1,\tau_1})$ is added in L_1, τ_1 is incremented by 1 and ξ_{1,τ_1} is returned to $\mathcal{A}$.

- *Pairing*: given two operands $\xi_{1,i}, \xi_{2,j}$ with $0 \leq i < \tau_1$ and $0 \leq j < \tau_2$, $\mathcal{B}$ computes the product $F_{T,\tau_T} \rightarrow F_{1,i} \cdot F_{2,j} \in \mathbb{F}_p[x]$. If $F_{T,\tau_T} = F_{T,l}$ for some $l < \tau_T$, then $\mathcal{B}$ sets $\xi_{T,\tau_T} \leftarrow \xi_{T,l}$, otherwise sets ξ_{T,τ_T} to a new distinct string in $\{0,1\}^*$. The pair $(F_{T,\tau_T}, \xi_{T,\tau_T})$ is added in L_T, τ_T is incremented by 1 and ξ_{T,τ_T} is returned to $\mathcal{A}$.

$\mathcal{A}$ terminates and returns to $\mathcal{B}$ a pair $(y, \xi_{1,l})$ with $0 \leq l < \tau_1$. To show correctness of $\mathcal{A}$'s answer, $\mathcal{B}$ considers the corresponding polynomial $F_{1,l}$ in L_1 and computes the polynomial

$$F_{T,*}(x) = F_{1,l} \cdot (F_{2,1} + yF_{2,0}) = F_{1,l} \cdot (x + y) = f(x) \cdot g(x) \cdot (x + y)$$

for a certain polynomial $g(x) \in \mathbb{F}_p[x]$ of degree $\leq t$. If $\mathcal{A}$'s answer is correct, then $F_{T,*}(x) = 1$ (which corresponds to check, in the current framework, that it results to be a correct DDH pair when representing $\xi_{1,l}$ with an element of G_1). Now, unless $\deg F_{T,*} \geq p - 2$ (due to Fermat's Little Theorem), the equation $F_{T,*}(x) - 1 = 0$ admits at most $\deg f + t + 1$ roots in $\mathbb{F}_p$.

At this point, $\mathcal{B}$ chooses a random $x^* \in \mathbb{F}_p$ and his simulation is perfect unless $x^* \leftarrow x$ creates equality relations between simulated elements not revealed to $\mathcal{A}$. Thus the success probability of $\mathcal{A}$ is bounded by the probability that any of the following conditions holds:

1. $F_{1,i}(x^*) - F_{1,j}(x^*) = 0$ for some i, j so that $F_{1,i} \neq F_{1,j}$
2. $F_{2,i}(x^*) - F_{2,j}(x^*) = 0$ for some i, j so that $F_{2,i} \neq F_{2,j}$
3. $F_{T,i}(x^*) - F_{T,j}(x^*) = 0$ for some i, j so that $F_{T,i} \neq F_{T,j}$
4. $f(x^*)g(x^*)(x^* + y) - 1 = 0$

Now since, for some fixed i, j, the polynomial $F_{1,i} - F_{1,j}$ has degree at most $\deg f + t$ while $F_{2,i} - F_{2,j}$ has degree at most t, they vanishes at x^* with probability $(\deg f + t)/p$ and t/p, respectively. Similarly, $F_{T,i} - F_{T,j}$ being a polynomial of degree at most $2t$, vanishes at x^* with probability $2t/p$. As regards $f(x^*)g(x^*)(x^* + y) - 1$, it vanishes at x^* with probability $(\deg f + t + 1)/p$. Hence, by summing these probabilities over all valid pairs (i, j) for the first three cases, $\mathcal{A}$ wins the game with probability

$$\epsilon \leq \binom{\tau_1}{2} \frac{\deg f + t}{p} + \binom{\tau_2}{2} \frac{t}{p} + \binom{\tau_T}{2} \frac{2t}{p} + \frac{\deg f + t + 1}{p}$$

Given that $\tau_1 + \tau_2 + \tau_T \leq q_G + t + 3$, we obtain $\epsilon \leq \frac{(\deg f + t) \cdot (q_G + t + 4)^2 + 1}{p}$

□

We are now ready to prove that breaking collision resistance of our accumulator scheme in the Generic Group Model cannot be easier than breaking the generalized t-SDH assumption:

Theorem 3.2. *Consider a Generic Group Model instance of the Dynamic Universal Accumulator outlined in Section 3.2 equipped with the public Batch Witness Update protocol detailed in Section 3.4. If $|\mathcal{Y}_{V_0}|$ elements are accumulated to initialize the accumulator value, then the probability ϵ that an attacker $\mathcal{A}$ breaks collision resistance of Definition 3.1 in q_G queries to the group oracles, is bounded by*

$$\epsilon \leq \frac{(|\mathcal{Y}_{V_0}| + t) \cdot (q_G + t + 4)^2 + 1}{p}$$

where t is the maximum number of elements allowed to be accumulated simultaneously.

Proof. We refer to the proof of Theorem 3.1 for the definition of the game setting between $\mathcal{A}$ and the accumulator manager and the corresponding notation.

Since $\mathcal{Y}_{V_0}$ contains distinct elements from $\mathbb{F}_p$, then at any epoch all elements in the batch witness update information Upd sent from the Accumulator manager to $\mathcal{A}$ are of the form $\xi_1(\, g(x) \cdot f(x)\,)$ where $f(x) = \prod_{y \in \mathcal{Y}_{V_0}} (y + x)$ and $g(x) \in \mathbb{F}_p[x]$ has degree $\leq t$.

Since any such polynomial $g(x) \cdot f(x)$ can be represented uniquely in the base $\{f(x), xf(x), \ldots, x^t f(x)\}$, we can assume $\mathcal{A}$ to be slightly more powerful by having initial access to all the following encodings:

$$\xi_1(1), \xi_1(f(\alpha)), \xi_1(\alpha \cdot f(\alpha)), \ldots, \xi_1(\alpha^t \cdot f(\alpha)), \xi_2(1), \xi_2(\alpha)$$

We note that accumulator values at different epochs can be obtained in polynomial time with queries to the group action oracle from the remaining update information, i.e. the elements added and deleted. All in all, this corresponds to the information the attacker would have access to under the hypothesis of Theorem 3.1.

Suppose that after q_G queries, $\mathcal{A}$ terminates and returns the tuple $(y, w_y, \bar{w}_y, \mathcal{Y})$ with $w_y = (\xi_{1,i}, 0)$ and $\bar{w}_y = (\xi_{1,j}, d)$. If the answer is correct and breaks the collision resistance property of the accumulator scheme, then the corresponding polynomials $F_{1,i}(x)$, $F_{1,j}(x)$ will satisfy

$$F_{1,i}(x) \cdot (x + y) = F_{1,j}(x) \cdot (x + y) + d$$

Note, that $\mathcal{A}$ can transform the tuple $(y, \xi_{1,i}, \xi_{1,j}, d)$ to the pair $(y, d^{-1}(\xi_{1,i} - \xi_{1,j}))$ by querying the oracles in polynomial time. This pair, if correct, would then solve the generalized $t-$SDH problem since, by letting $F_{T,*} = d^{-1}(F_{1,i} - F_{1,j})$, it holds $F_{T,*}(x) \cdot (x + y) - 1 = 0$ for all $x \in \mathbb{F}_p$. Hence $\mathcal{A}$ would win in q_G queries the game instantiated in the proof of Theorem 3.2 and, as was shown, this cannot be done with probability greater than $\frac{(|\mathcal{Y}_{V_0}|+t)\cdot(q_G+t+4)^2+1}{p}$. □

Relation with previous security assumptions and proofs The proposed Generalized $t-$Strong Diffie-Hellman assumption straightforwardly reduces to the standard t-SDH assumption (Definition 3.2) in the case when $\deg f = 0$, similarly as happens in Proposition 3.1 for our definition of Collision Resistance (Definition 3.1) and Au *et al.* one Definition 2.1.

Thus, when $\deg f = 0$ (without loss of generality, $f(x) = 1$), collision resistance of the scheme can be shown directly under the standard t-SDH assumption of Definition 2.2, without requiring the Generic Group Model and similarly as done in Theorem 2.1.

Generalizing definitions and security proofs to allow for $\deg f > 0$, allows us to address accumulator initialization, which (surprisingly) has a direct connection with the attacker ability to have access to (any element in) the $\mathcal{RS}_t$ (which can be (ab)used to compute witnesses and update the accumulator value), a circumstance that would be against our will to design an accumulator scheme also suited for authentication purposes, where only the accumulator manager can update and issue witnesses and whose construction will be finalized in next Section.

3.6 Accumulator Initialization

Depending on which would be the final application of the proposed accumulator scheme, it might be necessary to prevent the possibility to forge non-membership witnesses for *never authorized* non-accumulated elements, i.e. elements for which the accumulator manager did not issue witnesses.[6] This is relevant, for example, in the cases when the accumulator is used as an authentication mechanism, and accumulated elements represent either white-listed or black-listed users who authenticate with respect to the accumulator value by showing possession of a valid membership or non-membership witness.

Forging witnesseses in the case when the accumulator manager should be the only authorized entity to do so is, in fact, what the *Witness Forgery Attack* Attack 1 outlined in Subsection 2.10.2 does: a set of colluding users who share their non-membership witnesses can recover the (secret) reference-string sets $\mathcal{RS}_t = \{P, \alpha P, ..., \alpha^t P\}_{t>0}$, which would enable them to compute membership and non-membership witnesses with respect to the latest accumulator value. Indeed, the knowledge of the set $\mathcal{RS} = \mathcal{RS}_t$ results to be functionally equivalent to the knowledge of α: it is possible to either update the accumulator value (see Lemma 2.1) or issue valid membership and non-membership witnesses (see the reference string $\mathcal{RS}$-based construction in Section 2.3).

The Witness Forgery Attack is possible as long as the number of colluding users is equal or greater to the number of elements added to initialize the accumulator value. The countermeasure we propose in Section 2.10 consists in setting an upper limit `NMWitnessesMax` to the total number of issuable non-membership witnesses and initialize the accumulator by adding at least `NMWitnessesMax` $+ 1$ secret elements.

This will clearly prevent the reconstruction of the sets $\mathcal{RS}_s$. However, in our protocol, the attackers have access in each epoch to the witness update information (see Table 3.1), which in principle could help circumvent the fact that they will not be able to collect and share enough non-membership witnesses or can be used to directly compute some elements in $\mathcal{RS}_s$.

We will show that this is indeed possible. However, we will prove that a generic algorithm in the Generic Group Model would compute any element in $\mathcal{RS}_t$ with negligible probability by just carefully choosing a few of the elements added to initialize the accumulator value.

[6] Membership witnesses for new elements would require an accumulator value update, an operation that we could assume to be executed by the accumulator manager that has exclusive access to the public register containing the current accumulator value. When issuing non-membership witnesses, instead, the accumulator value remains unchanged.

We start by introducing some theoretical results. The purpose of the following Proposition is to show some properties on elements that have particular multiplicative orders in the group $(\mathbb{F}_p)^*$. These properties will be useful to prove the subsequent Theorem 3.3, which will give us sufficient conditions on the elements we need to add to prevent the reconstruction of the $\mathcal{RS}$ from the publicly available information. Thus, initializing the accumulator with $\texttt{NMWitnessesMax} + 1$ random elements, where some of them satisfies the hypothesis of Theorem 3.3, will ultimately prevent any, even partial, successful execution of the Witness Forgery Attack (Attack 1).

Proposition 3.2. *Let $p \in \mathbb{N}$ be a prime such that $p-1 = p_1^{e_1} \cdot \ldots \cdot p_n^{e_n}$ factorizes as the product of $n > 1$ powers of distinct primes $p_i \in \mathbb{N}$. Let $f(x) \in \mathbb{F}_p[x]$ be a polynomial with $n \leq m < p-1$ distinct non-zero roots $x_1, \ldots, x_m \in \mathbb{F}_p$ such that the multiplicative order in $(\mathbb{F}_p)^*$ of x_i, for $1 \leq i \leq n$, is $p_i^{e_i}$. Then*

i. The least $k > 0$ for which there exists $z \in \mathbb{F}_p$ and $g(x) \in \mathbb{F}_p[x]$ such that $g(x)f(x) \equiv x^k - z \mod p$ is $k = p-1$.

ii. The degree of the minimal-degree non-constant monomial of $f(x)$ is s with $0 < s < m$.

Proof. Suppose there exists $z \in \mathbb{F}_p$ and $g(x) \in \mathbb{F}_p[x]$ such that

$$g(x)f(x) \equiv x^k - z \mod p$$

Then, each root $x_1, \ldots, x_m$ of $f(x)$ must be a root for $x^k - z$ in $\mathbb{F}_p$, that is

$$x_1^k \equiv \cdots \equiv x_t^k \equiv z \mod p \tag{3.1}$$

Since, by hypothesis $(\mathbb{F}_p)^* \simeq \mathbb{Z}/(p-1)\mathbb{Z} \simeq \langle x_1 \rangle \times \cdots \times \langle x_n \rangle$ with $n > 1$ we have $z \in \bigcap_{i=1}^m \langle x_i \rangle \leq \bigcap_{i=1}^n \langle x_i \rangle = \langle 1 \rangle$. Hence a solution to Equation 3.1 exists only if $z \equiv 1$ and the least k for which it holds is $k = lcm(\mathrm{ord}(x_1), \ldots, \mathrm{ord}(x_m)) \geq lcm(\mathrm{ord}(x_1), \ldots, \mathrm{ord}(x_n)) = p-1$. Since $k \leq p-1$, we have $k = p-1$.

It follows that as long as $m < p-1$, there are no $z \in \mathbb{F}_p$ such that $f(x) \equiv x^t - z \mod p$. Hence the degree of the minimal-degree non-constant monomial of $f(x)$ is s with $0 < s < m$. □

Theorem 3.3. *Let $p \in \mathbb{N}$ be a prime such that $p-1 = p_1^{e_1} \cdot \ldots \cdot p_n^{e_n}$ factorizes as the product of $n > 1$ powers of distinct primes $p_i \in \mathbb{N}$. Let $f(x) \in \mathbb{F}_p[x]$ be a polynomial with $n \leq m < p-1$ distinct non-zero roots $x_1, \ldots, x_l \in \mathbb{F}_p$ such that the multiplicative order in $(\mathbb{F}_p)^*$ of x_i, for $1 \leq i \leq n$, is $p_i^{e_i}$.*

Let $(\mathcal{V}, +)$ be the vector space of polynomials with degree lower equal $p-2$ and $\mathcal{B} = \{1, x, \ldots, x^{p-2}\}$ its $(p-1)-$dimensional canonical basis. Then, for every $1 \leq k < p-1$

$$rank\left(\begin{bmatrix} 1 \\ f(x) \\ xf(x) \\ \vdots \\ x^{p-m-2}f(x) \\ x^k \end{bmatrix}_{\mathcal{B}}\right) = p - m + 1$$

Proof. The rank is maximum when the row vectors are linearly independent in $\mathcal{V}$, that is for any $a_0, \ldots, a_{p-m-2}, b, c \in \mathbb{F}_p$ such that

$$\left(\sum_{j=0}^{p-m-2} a_j x^j f(x)\right) + bx^k + c = 0 \tag{3.2}$$

we have $a_0 \equiv \cdots \equiv a_{p-m-2} \equiv b \equiv c \equiv 0$.

We will prove the statement by exhaustion on the values of k.

$\boxed{1 \leq k < m:}$ The dependence relation (Equation 3.2) can be rewritten as

$$g(x)f(x) = -bx^k - c$$

where $g(x) = \sum_{j=0}^{p-m-2} a_j x^j$. By hypothesis $f(x)$ has m different roots, while $-bx^k - c$ can have at most $k < m$ distinct roots. The equation then holds only if both sides are equal to the 0 polynomial, that is $-bx^i - c = 0$ and $g(x) = 0$. This implies $a_0 \equiv \cdots \equiv a_{p-m-2} \equiv 0$ and $b \equiv c \equiv 0$ because the elements $\{1, x, \ldots, x^{p-m-2}\}$ are linearly independent vectors of $\mathcal{V}$.

$\boxed{k = m:}$ In this case the dependence relation (Equation 3.2) can be rewritten as

$$g(x)f(x) = -bx^m - c$$

with $g(x)$ defined as in the previous case. By hypothesis $f(x)$ has m distinct roots, while the right side can have at most m distinct roots. This implies that $g(x) = g(0) = a_0$ is a constant polynomial or, equivalently, that $a_1 \equiv \cdots \equiv a_{p-m-2} \equiv 0$. Suppose by contradiction that $a_0 \neq 0$, then $f(x) = -a_0^{-1}bx^m - a_0^{-1}c$ is a contradiction since, by Proposition 3.2, the degree of the minimal-degree non-constant monomial of $f(x)$ is s with $s \neq m$ and $s > 0$. Hence $a_0 \equiv 0$, and then $-bx^m - c = 0$ which implies $b \equiv c \equiv 0$.

$\boxed{m < k < p-1:}$ Let $k = m + k'$ with $1 \leq k' \leq p - m - 2$. From

$$g(x)f(x) = -bx^{m+k'} - c$$

it follows that $deg(g) \leq k'$ and then, if $k' < p - m - 2$, we have $a_{k'+1} \equiv \cdots \equiv a_{p-m-2} \equiv 0$.

Assume, by contradiction, $b \neq 0$ and $c \equiv 0$. In this case the dependence relation becomes $g(x)f(x) = -bx^{m+k'}$, but the right side has only 0 as root while the left side has, by hypothesis, at least t non-zero distinct roots. This implies, similarly as before, that $g(x) = 0$ and $b \equiv 0$, a contradiction.

Let us therefore assume $b \neq 0$ and $c \neq 0$. In this case the dependence relation can be rewritten as

$$g'(x)f(x) = x^{m+k'} - z$$

where $g'(x) = (-b)^{-1}g(x)$ and $z = (-b)^{-1}c \neq 0$.

If, by contradiction, $g'(x) \neq 0$, then, by Proposition 3.2, the least value for $m + k'$ such that the dependence relation holds is $m + k' = p - 1$, that is $k' = p - m - 1$, a contradiction to $1 \leq k' \leq p - m - 2$. Hence $g'(x) = 0$, which in turn implies $b \equiv c \equiv 0$, a contradiction to our assumption $b \neq 0$.

It follows that $b \equiv 0$ and then $g(x)f(x) = -c$. Since by hypothesis f has m distinct roots, this equation holds only if $c = 0$ and $g(x) = 0$, which, similarly as before, implies $a_0 \equiv \cdots \equiv a_{k'} \equiv b \equiv c \equiv 0$. □

Corollary 3.1. *Let $f(x) \in \mathbb{F}_p[x]$ be a polynomial satisfying the hypothesis of Theorem 3.3 and consider a Generic Group Model instance of the proposed Dynamic Universal Accumulator equipped with the public Batch Witness Update protocol. If the accumulator value is initialized by adding all the roots in $\mathbb{F}_p$ of $f(x)$, then the probability ϵ that an attacker $\mathcal{A}$ outputs in q_G queries to the group oracles the value $\xi_1\left(\alpha^k\right)$ for any $1 \leq k < p-1$ is bounded by*

$$\epsilon \leq \frac{(\deg f + t) \cdot (q_G + t + 3)^2}{p}$$

where t is the maximum number of elements allowed to be accumulated simultaneously.

Proof. The proof proceeds similarly as done in the proof of Theorem 3.2. The only difference is the condition checked by the accumulator manager to ensure correctness of $\mathcal{A}$'s output value ($\xi_{1,l}$). This new check corresponds to verifying the polynomial equation $F_{1,l} - x^k = 0$ where $F_{1,l} = g(x) \cdot f(x)$ with $\deg g \leq t$. The accumulator manager then chooses a random value $x^* \in \mathbb{F}_p$: by Theorem 3.3 the equation $F_{1,l} - x^k = 0$ vanishes in x^* with probability 0 for any $1 \leq k < p-1$, thus $\mathcal{A}$ wins the game with non-zero probability only if some of the following conditions holds:

1. $F_{1,i}(x^*) - F_{1,j}(x^*) = 0$ for some i, j so that $F_{1,i} \neq F_{1,j}$
2. $F_{2,i}(x^*) - F_{2,j}(x^*) = 0$ for some i, j so that $F_{2,i} \neq F_{2,j}$
3. $F_{T,i}(x^*) - F_{T,j}(x^*) = 0$ for some i, j so that $F_{T,i} \neq F_{T,j}$

From this we conclude,in a similar way as done at the end of the proof of Theorem 3.2, that $\mathcal{A}$ wins the game with the accumulator manager with a probability $\epsilon \leq \frac{(\deg f+t)\cdot(q_G+t+3)^2}{p}$.

□

We are now ready to define the Accumulator Initialization procedure for our protocol explicitly:

Accumulator Initialization Set an upper limit `NMWitnessesMax` to the total number of issuable non-membership witnesses. Assume p is such that $p-1 = p_1^{e_1} \cdot \ldots \cdot p_n^{e_n}$ factorizes as the product of $n > 1$ powers of distinct primes p_i and consider n elements $x_1, \ldots, x_n \in \mathbb{F}_p$ such that the multiplicative order in $(\mathbb{F}_p)^*$ of x_i is $p_i^{e_i}$, for $1 \leq i \leq n$. Then, the accumulator manager sets

$$\mathcal{Y}_{V_0} = \{x_1, \ldots, x_n\} \cup \{\texttt{NMWitnessesMax} - n + 1 \text{ random elements in } \mathcal{ACC}\}$$

so that $|\mathcal{Y}_{V_0}| = \texttt{NMWitnessesMax} + 1$ and defines the corresponding initialization polynomial as $f_0(x) = \prod_{x_i \in \mathcal{Y}_{V_0}} (x - x_i)$, where $V_0 = f_0(\alpha)P$. He then publishes $(V_0, \emptyset)$, the accumulator state at epoch 0, and keeps secret and never deletes the elements in $\mathcal{Y}_{V_0}$.

We note that as soon as an epoch changes, the accumulator manager publishes the corresponding Batch Witnesses Update information: at epoch 1, for example, this corresponds to the new state $(V_1, \mathcal{Y}_{V_1})$, the updating vector Ω_1 and the polynomials $d_{\mathcal{A}_1}(x)$ and $d_{\mathcal{D}_1}(x)$. At this point, the polynomial

$$f_{V_1}(x) = f_0(x) \cdot f_1(x) = \prod_{y_i \in \mathcal{Y}_{V_0}} (y_i + x) \cdot \prod_{y_j \in \mathcal{Y}_{V_1}} (y_j + x)$$

has $|\mathcal{Y}_{V_0}| + |\mathcal{Y}_{V_1}|$ distinct non-zero roots, n of which are $x_1, \ldots, x_n$, and is such that $V_1 = f_{V_1}(\alpha)P$. Even if it is possible to obtain from Ω_1, $\mathcal{A}_1$ and $\mathcal{D}_1$ all the values $\alpha^k V_1$ for $1 \leq k < p - |\mathcal{Y}_{V_0}| - 1$ (we relax the condition $k \leq \texttt{batchMax}$) we still are under the hypothesis of Theorem 3.3 and Corollary 3.1, which ensure the infeasibility to obtain any element of the $\mathcal{RS}$. This reasoning can be easily generalized to any subsequent epoch.

One more question might arise from all these considerations: is it possible to obtain some elements in the $\mathcal{RS}$ combining the vectors Ω_i coming from different epochs? When the accumulator is initialized as described in Theorem 3.3, the answer is *no*. To show this, consider, without loss of generality, the m vectors $\Omega_1, \ldots, \Omega_m$, where the j-entry of any Ω_i is of the form $c_j V_i$. Hence, a linear combination with coefficients $a_{i,j} \in \mathbb{F}_p$ of entries of these vectors can be written as

$$\sum_{i=1}^{m} \sum_{j=0}^{|\texttt{batchMax}|} a_{i,j} c_j V_i = \left(\sum_{i=1}^{m} \sum_{j=0}^{|\texttt{batchMax}|} a_{i,j} c_j f_i(\alpha) \right) \cdot f_0(\alpha) P = g(\alpha) \cdot V_0$$

where $g(x) = \sum_{i=1}^{m} \sum_{j=0}^{\texttt{batchMax}} a_{i,j} c_j f_i(x)$. In other words, in the luckiest situation, what we can obtain combining all these vectors is a *"basis"* made of elements of the form $\alpha^k f_0(\alpha) P = \alpha^k V_0$ with $1 \leq k \leq \texttt{batchMax}$ which, as we already discussed, does not permit to obtain any element in $\mathcal{RS}$.

3.7 Zero-Knowledge Proof of Knowledge

We now explicitly show how an interactive zero-knowledge protocol can be instantiated between a Prover and a Verifier to prove the ownership of a valid non-membership witness $\bar{w}_{y,V}$ for y with respect to the accumulator state $(V, \mathcal{Y}_V)$ (the corresponding protocol to show ownership of a valid membership witness $w_{y,V}$ is similar and will not be discussed). Although different NIZK protocols for bilinear equations verification can be adopted for this purpose (e.g. Groth-Sahai [GS08]), we chose to detail a construction that doesn't need a trusted setup (to avoid extra storage needs on users' side), and that can be easily implemented in order to provide a reference for our benchmarks.

To this end, we will then extend the zero-knowledge proof of knowledge protocol defined by Boneh *et al.* in [BB04], which proves under the Decision Linear Diffie-Hellman assumption the knowledge of a pair (y, C) such that $(y + \alpha)C = V$, in order to support tuples (y, C, d) which verify $(y + \alpha)C + dP = V$.

However, for non-membership witnesses, we need to ensure further $d \neq 0$ or, equivalently in G_1, that has a multiplicative inverse. In this regard, we will then consider, for a random generator $K \in G_1$ and random $a, b \in \mathbb{F}_p$, the Pedersen commitments $E_d = dP + aK$ and $E_{d^{-1}} = d^{-1}P + bK$ for d and d^{-1}, respectively. Noticing that $P = dE_{d^{-1}} - dbK$, we then extend the protocol applying EQ-composition to the factor d among the values E_d and P, thus showing that $E_{d^{-1}}$ is a Pedersen commitment to the multiplicative inverse of the committed value in E_d.

Under the Random Oracle Model, we can make such proof of knowledge non-interactive and full zero-knowledge by applying the Fiat-Shamir heuristic [FS87]. We will do so by using a heuristic variant adopted by Boneh *et al.* in [BB04] in order to reduce Prover's proof size. We assume that calls to the random oracle can be concretely realized through evaluations to a cryptographic hash function $H : \{0,1\}^* \to \mathbb{F}_p$.

Security proofs. By reporting the relative security proofs, we will substantially replicate the original results of the respective authors: we, therefore, refer to [BB04]

and [Sch20, Ex. 5.3.4] for the completeness, soundness and (special) honest-verifier zero-knowledgeness security proofs of Boneh et al. protocol and EQ-composition for multiplicative-inverse relation, respectively.

3.7.1 The NIZK Protocol

After a *Setup* phase where parties agree on public parameters, the resulting non-interactive zero-knowledge *Proof of Knowledge* consists of 4 main stages: *Blinding*, *Challenge*, *Response* executed by the Prover, and a proof verification *Verify* executed by the Verifier. These are defined as follows.

Setup. The Prover and Verifier agree on the public values $P, X, Y, Z, K \in G_1$ and $\tilde{P}, \tilde{Q} \in G_2$, where X, Y, Z, K are distinct random generators of G_1.

Proof Of Knowledge. The Prover randomly selects $\sigma, \rho, \tau, \pi \in \mathbb{F}_p$ and computes

$$E_C = C + (\sigma + \rho)Z, \qquad E_d = dP + \tau K, \qquad E_{d^{-1}} = d^{-1}P + \pi K,$$

$$T_\sigma = \sigma X, \qquad T_\rho = \rho Y, \qquad \delta_\sigma = y\sigma, \qquad \delta_\rho = y\rho$$

A non-interactive zero knowledge Proof of Knowledge of values $(y, d, \sigma, \rho, \tau, \pi, \delta_\sigma, \delta_\rho)$ satisfying

$$P = dE_{d^{-1}} - d\pi K, \qquad E_d = dP + \tau K,$$

$$\sigma X = T_\sigma, \qquad \rho Y = T_\rho, \qquad yT_\sigma - \delta_\sigma X = O, \qquad yT_\rho - \delta_\rho Y = O,$$

$$e(E_C, \tilde{P})^y e(Z, \tilde{P})^{-\delta_\sigma - \delta_\rho} e(Z, \tilde{Q})^{-\sigma - \rho} e(K, \tilde{P})^{-\tau} = \frac{e(V, \tilde{P})}{e(E_C, \tilde{Q}) e(E_d, \tilde{P})}$$

is undertaken between Prover and Verifier as follows:

Blinding. The Prover randomly picks r_y, r_u, r_v, r_w, r_σ, r_ρ, r_{δ_σ}, $r_{\delta_\rho} \in \mathbb{F}_p$, computes

$$R_A = r_u P + r_v K, \qquad R_B = r_u E_{d^{-1}} + r_w K,$$

$$R_E = e(E_C, \tilde{P})^{r_y} e(Z, \tilde{P})^{-r_{\delta_\sigma} - r_{\delta_\rho}} e(Z, \tilde{Q})^{-r_\sigma - r_\rho} e(K, \tilde{P})^{-r_v},$$

$$R_\sigma = r_\sigma X, \qquad R_\rho = r_\rho Y, \qquad R_{\delta_\sigma} = r_y T_\sigma - r_{\delta_\sigma} X, \qquad R_{\delta_\rho} = r_y T_\rho - r_{\delta_\rho} Y$$

Challenge. The Prover sets the challenge $c \in \mathbb{F}_p$ to

$$c = H(V, E_C, E_d, E_{d^{-1}}, T_\sigma, T_\rho, R_A, R_B, R_E, R_\sigma, R_\rho, R_{\delta_\sigma}, R_{\delta_\rho})$$

Response. The Prover computes

$$s_y = r_y + cy, \qquad s_u = r_u + cd, \qquad s_v = r_v + c\tau, \qquad s_w = r_w - cd\pi,$$

$$s_\sigma = r_\sigma + c\sigma, \qquad s_\rho = r_\rho + c\rho, \qquad s_{\delta_\sigma} = r_{\delta_\sigma} + c\delta_\sigma, \qquad s_{\delta_\rho} = r_{\delta_\rho} + c\delta_\rho$$

and sends $(E_C, E_d, E_{d^{-1}}, T_\sigma, T_\rho, c, s_y, s_u, s_v, s_w, s_\sigma, s_\rho, s_{\delta_\sigma}, s_{\delta_\rho})$ to the Verifier.

Verify. The Verifier computes

$$R_A = s_u P + s_v K - cE_d, \qquad R_B = s_w K + s_u E_{d^{-1}} - cP,$$

$$R_\sigma = s_\sigma X - cT_\sigma, \qquad R_\rho = s_\rho Y - cT_\rho, \qquad R_{\delta_\sigma} = s_y T_\sigma - s_{\delta_\sigma} X, \qquad R_{\delta_\rho} = s_y T_\rho - s_{\delta_\rho} Y,$$

$$R_E = e(E_C, \tilde{P})^{s_y} \cdot e(Z, \tilde{P})^{-s_{\delta_\sigma} - s_{\delta_\rho}} \cdot e(Z, \tilde{Q})^{-s_\sigma - s_\rho} \cdot e(K, \tilde{P})^{-s_v} \cdot \left(\frac{e(V, \tilde{P})}{e(E_C, \tilde{Q}) e(E_d, \tilde{P})} \right)^{-c}$$

and accepts if $c = H(V, E_C, E_d, E_{d^{-1}}, T_\sigma, T_\rho, R_A, R_B, R_E, R_\sigma, R_\rho, R_{\delta_\sigma}, R_{\delta_\rho})$.

3.7.2 Complexity Analysis

Within this protocol, zero-knowledge proofs for valid non-membership witnesses consists of 5 elements in G_1 and 11 elements in $\mathbb{F}_p$, while they consists of 3 elements in G_1 and 6 elements in $\mathbb{F}_p$ in the case of membership witnesses. Thus, if elliptic curves are used to concretely implement the underlying bilinear group (in particular, we assume elements in G_1 are elliptic curve points over $\mathbb{F}_q$) and elliptic curve points compression is used, zero-knowledge non-membership and membership proofs can be represented with $5(\log q + 1) + 9 \log p$ bits and $3(\log q + 1) + 6 \log p$ bits, respectively. In our concrete instance (see Section 3.8) this translates to 4926 bits $\approx$ 616 bytes proofs for non-membership witnesses and 3135 bits $\approx$ 392 bytes proofs for membership witnesses.

As regards computational costs, if the quantities $e(Z, \tilde{P})$, $e(Z, \tilde{Q})$, $e(K, \tilde{P})$ and $e(V, \tilde{P})$ are pre-computed and stored by both Prover and Verifier, zero-knowledge proofs of knowledge for non-membership witnesses are computed with 15 scalar-point multiplications in G_1, 7 point additions in G_1, 4 exponentiation in G_T and 1 pairing. We note that the Prover can reduce the cost of evaluating $e(E_C, \tilde{P})$ by computing and storing the value $e(C, \tilde{P})$. Thus, with just 1 pairing per-epoch, the Prover can compute each $e(E_C, \tilde{P})$ as $e(Z, \tilde{P})^{\sigma+\rho} \cdot e(C, \tilde{P})$ with 1 exponentiation and 1 multiplication in G_T. Using this optimization, the cost to compute a proof of knowledge of a membership witness boils down to a total of 9 scalar-point multiplications in G_1, 3 point additions in G_1, 5 exponentiation in G_T and 1 multiplication in G_T.

Similarly, the Verifier needs 16 scalar-point multiplications in G_1, 9 point additions in G_1, 4 exponentiation in G_T and 2 pairings (if computes $e(E_C, \tilde{P})^{s_y} e(E_C, c\tilde{Q})$ as $e(E_C, s_y\tilde{P} + c\tilde{Q})$) to verify a non-membership witness zero-knowledge proof, while he needs 10 scalar-point multiplications in G_1, 5 point additions in G_1, 3 exponentiation in G_T and 1 pairing to verify a zero-knowledge proof of knowledge for a membership witness.

3.8 Implementation

To show the efficiency and its practical relevance, we implemented the proposed accumulator scheme by using the RELIC library [Ara+], which implements the arithmetic of many pairing-friendly elliptic curves. In order to guarantee a security level of 128-bits, we selected the available pairing-friendly Type-III prime order curve *B12-P446*. We then benchmarked the main features of the proposed accumulator, obtaining the following average results:

- **Accumulator Updates**: 0.75 seconds to add $1,000,000$ elements (random elements generation requires 1.46s); 0.48 seconds to delete $1,000,000$ elements.
- **Witness Issuing**: 1.9 milliseconds to issue a membership witness; 229.5 milliseconds for a non-membership witness ($1,000,000$ elements accumulated).
- **Witness Verification**: 2.2 milliseconds to verify a membership witness; 3.2 milliseconds to verify a non-membership witness.

- **Public Batch Witness Update**: 37.9 seconds to generate batch update data corresponding to a $10,000$ elements batch addition operation; 27.1 seconds for a $10,000$ elements batch deletion operation.
- **Batch Witness Update**: 2.1 seconds to update a membership or non-membership witness after a batch addition or deletion operation of $10,000$ elements.
- **Non-Batch Witness Update**: 4.4 seconds to update a membership or non-membership witness after a batch addition or deletion operation of $10,000$ elements.
- **Zero-Knowledge Proof Creation**: 5.2 milliseconds to create a zero knowledge proof of knowledge of a membership witness; 7.4 milliseconds to create a proof for a non-membership witness.
- **Zero-Knowledge Proof Verification**: 6.5 milliseconds to verify a zero knowledge proof of knowledge of a membership witness; 11.2 milliseconds to verify a proof for a non-membership witness.

These benchmarks came from running our implementation on a standard Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz desktop provided with 8.00GB of RAM and running Ubuntu 18.04 x64. No parallelization was used.

Our implementation can be found on GitHub at:

https://github.com/cryptolu/accumulator/

Feedback from the community. We were contacted by a company that implemented our accumulator (including the Public Batch Witness Update and Zero-Knowledge protocols) as a revocation mechanism for verifiable credentials. Their implementation, public on GitHub [7], is used already in production applications where 10-20 million entries remain accumulated at any given time, and 1000/600 elements are added/deleted, respectively, per day. They reported to us that in their implementation: (i) witness update data generation takes 17s and 99KB/day; (ii) users' witness update after 1 year offline requires 80s and 36MB of update data; (iii) witness updates work on IoT; (iv) using the pairing-friendly elliptic curve BLS12-381, RAM requirements are few megabytes. They added that our scheme "solves their scaling problem" compared to Hyperledger-Indy[8] implementation of [CKS09], which would take hours/day and larger proof sizes.

3.9 Conclusions

In this Chapter, we presented a Dynamic Universal Accumulator in the Accumulator Manager setting over bilinear groups, which supports batch operations and batch membership and non-membership public witness updates.

The proposed accumulator extends previous schemes by adding batch operations, enabling users to update witnesses in optimal time. Furthermore, since batch update data is designed to be decoupled from users' witnesses, our protocol permits (privacy-preserving) witness updates delegation, thus enabling lightweight users to keep their witnesses updated with a constant number of elementary operations.

We then showed in the Generic Group Model its security in terms of collision resistance by introducing a more general version of the $t-$SDH assumption, for which

[7] https://github.com/mikelodder7/accumulator-rs

[8] https://www.hyperledger.org/use/hyperledger-indy

we give an upper bound complexity for a generic algorithm that solves the corresponding problem. We further showed how to initialize the accumulator to be safe from an attack which would allow forging witnesses for non-authorized elements, an essential requirement in the case the accumulator scheme is used as an authentication mechanism under the accumulator manager authority.

We then described how to instantiate a zero-knowledge proof of ownership of a valid witness for a given accumulator state, and we implemented the accumulator logic along with batch operations, the public witness update protocol, and the zero-knowledge proof mechanism to show its practical relevance as an efficient and scalable privacy-preserving authentication mechanism.

Part II

Blockchain Cryptography

In this Part, I will present my works with my co-authors on the cryptanalysis of primitives adopted, or considered for adoption, by top blockchain-based cryptocurrencies.

In Chapter 4, we will focus on the privacy-oriented cryptocurrency Zcash [Zca], by showing the existence of multiple *subliminal channels* in both the zk-SNARK protocol and commitment scheme used for creating *shielded* transactions, that is transactions were the sender, the recipients and the amounts transferred are all hidden.

Under our attack model, justified by some design choice developers made in order to allow limited-resources devices to create such private transactions efficiently, an attacker can embed certain tagging information up to the channel capacity, with the consequence of being the only one able to track on-chain users' activities or reveal the contents of a shielded transaction.

We will show the practicality of our attacks by illustrating an efficient 70-bits channel in Zcash Sapling, which we used to embed an all-zero message in a valid on-chain shielded transaction, and we will discuss some possible countermeasures that can be efficiently implemented on users' side.

In Chapter 5, instead, we will cryptanalyze the Legendre PRF, a pseudo-random function proposed by Damgård [Dam90] relying on the conjectured pseudorandomness properties of the Legendre symbol with a hidden shift. This function was recently suggested as an efficient PRF for multi-party computation purposes by Grassi *et al.* [Gra+16], and is being considered as the building block of a Proof of Custody mechanism to be eventually implemented in the upcoming Ethereum 2.0 blockchain [Fei19c].

Thorought the Chapter, we will describe an improved key-recovery attack on the Legendre PRF, where we reduce the time complexity from $\mathcal{O}(p \log p / M)$ to $\mathcal{O}(p \log^2 p / M^2)$ Legendre symbol evaluations when $M \leq \sqrt[4]{p \log^2 p}$ queries to the PRF are available to the attacker. These improvements allowed us to break three concrete instances of the PRF proposed by the Ethereum foundation and whose cryptanalysis will be discussed.

We will then generalize our attack to target the higher-degree variant of the Legendre PRF, for which we show existence of a large class of weak keys, and we will provide the first security analysis of two additional generalizations of the Legendre PRF originally proposed by Damgård in the PRG setting, namely the Jacobi PRF and the power residue PRF.

Chapter 4

Subliminal Channels in Zcash

4.1 Introduction

Cash-like privacy is one of the critical properties to be implemented in modern blockchain-based cryptocurrencies. Bitcoin [Nak08], with its pseudonymous transactions, while initially believed to offer payment privacy, was shown to suffer from transaction graph analysis and linkability issues [Mei+13; RH13], mainly due to the public nature of its ledger. To address these aspects, there has been a rise of *privacy-preserving cryptocurrencies* such as Dash [Das], Monero [Mon], Zcash [Zca], each using different privacy-enhancing technologies. The simplest one is used by the blockchain Dash, which uses so-called masternodes with built-in mixers to provide privacy. Monero's ring signatures provide another form of transaction mixing, which does not require central nodes while also hiding the values of transactions. These ring signatures are a cryptographic primitive which provides a valid signature for a group of n users where only one user has to use his secret key together with the public keys of the other members. It follows that, externally, it is impossible to tell which group member signed the message. For transactions, that means that multiple outputs can be spent simultaneously without the ability to distinguish which signer used a secret key and which ones the public keys.

The last promising technology that is in use for privacy-preserving blockchains is *zk-SNARK*, which stands for Zero-Knowledge Succinct Non-interactive ARguments of Knowledge. This technology offers provable security and requires a *Common Reference String* ($\mathcal{CRS}$), which is the central trusted piece of data. In the case of the Zcash blockchain, the $\mathcal{CRS}$ was generated once at the launch of the chain in a distributed multi-party computation (MPC) with trusted peers. Zcash launched on 28 October 2016 as the first commercial release of zk-SNARKs. Zcash itself is based on Bitcoin-style UTXO (unspent transaction output) system of tracking coins and has a public and a private portion. The public part works the same way as in Bitcoin, while the private part uses zk-SNARKs proofs, and transactions that use such zk-SNARKs are called *shielded* transactions. One of the problems with such zk-SNARK technology was that, in the first version, the creation of a proof took 40 seconds and 1.5GB of memory, so for usability reasons, this blockchain kept transparent Bitcoin-style transactions as default. In October 2018, Zcash added a new and much faster zk-SNARK protocol called Sapling [Zca18] in which a proof creation takes 3 seconds and requires only 40MB of memory.

In this Chapter, we will describe some transaction-tagging attacks for the Zcash Sapling protocol, based on subliminal channels [Sim85; Sim94; YY04] and which can be used to weaken users privacy. With a subliminal channel, an attacker can *reveal* b bits of arbitrary information out of a cryptosystem, using system parameters that were not originally designed to exchange such information. This means that subliminal channels, when they exist, can become a hidden part of the cryptosystem, and if the attacker controls the affected parameters, he can freely decide whether to send a secret message or not: in particular, such hidden communication can be used to reveal secret keys or user IDs. Even worse, to maintain the confidentiality of the subliminal channel, the attacker can easily use encryption to permit only some receivers that know a pre-shared secret to retrieve the plaintext message, while the others cannot even detect if a message was sent. While subliminal channels can be present in many cryptosystems, it is clear that privacy-focused applications need to be aware of their existence and properties, due to severe consequences for user privacy (and coin fungibility, in the case of cryptocurrencies).

In Zcash Sapling we discovered three different subliminal channels: the *Inner* and the *Outer Subliminal Channels*, which can be used assuming that the zk-SNARK proof generation mechanism (but not the secret master key!) is under the control of the attacker, and the *Pedersen Subliminal Channel*, which requires the control of the randomness source in the commitment scheme adopted to hide values of shielded inputs and outputs. Such subliminal channels can be exploited, for example, by closed-source lightweight wallets which are used on mobile devices. Another possible attack vector is represented by (delegated) computation servers (or hardware devices) used for zero-knowledge proof generation.

Alongside the descriptions of these subliminal channels, we provide an example of how an attacker can exploit them to permit external entities, that we will refer to as *Subliminal Verifiers*, to distinguish transactions created by the same user and to de-commit/reveal all shielded transaction amounts. We will then discuss countermeasures to prevent the exchange of *subliminal messages*, or alternative methods to disrupt their contents.

4.1.1 Outline

In Section 4.2 we provide a brief background on Zcash and how shielded transactions are computed. In Section 4.3 we detail the Inner and Outer Subliminal channel

we found in the zk-SNARK proof system mechanism used to ensure the correctness of private Zcash transactions, while in Section 4.4 we detail the Pedersen Subliminal Channel, which affects, instead, the employed commitment scheme. We discuss possible transaction-tagging and information leakage attacks, and we show their practicality in Section 4.5 by embedding 9 bytes in a real Zcash Sapling shielded transaction, reported in Section 4.6.

4.2 Preliminaries

4.2.1 Introduction to Zcash

Zcash is a privacy preserving blockchain based on Zerocash [Ben+14], which uses practical zero-knowledge proofs called zk-SNARKs. Zcash has a public open part, which exactly mimics Bitcoin's ledger and is based on unspent transaction outputs (UTXO), and a parallel hidden part, which employs zero-knowledge proofs and value commitments in order to transfer coins anonymously. In Zcash, transactions that use such zero-knowledge proofs are called *shielded* transactions, in contrast to *transparent* transactions, which happen on the public side of the blockchain.

Zcash was launched on 28 October 2016 with the release called *Sprout*. This version's zk-SNARK proof mechanism took 40 seconds and required 1.5GB of memory. On 29 October 2018 the first major update to Zcash's proof system called *Sapling* was deployed. This update reduced the time to create a zk-SNARK to 3 seconds, while the memory requirement was reduced to 40MB. Sprout and Sapling addresses and transactions use completely different elliptic curves and proof protocols, which makes them incompatible with each other. In this Chapter we will address only Sapling shielded transactions.

Coins in Zcash are referred to as ZEC, while the smallest possible value is called Zatoshi, where 1 ZEC = 10^8 Zatoshi[1]. Zcash uses a UTXO based ledger both in its public and shielded setting. In order to create a new output with a specific value, one must consume previously unspent outputs, where the value sum of these has to be larger or equal to the value sum of the desired new outputs. The only transaction that can create new value without consuming a previously unspent output is the so-called coinbase transaction, which is always the first transaction in a block. Its base value is fixed by the blockchain protocol, and miners can add an extra value equal to the sum of fees of all transactions present in the block they are mining. In a transaction, if the sum of the values from the consumed outputs is larger than the new outputs, the difference can be claimed in the coinbase transaction by the miner as the transaction fee. In the rest of the Chapter we will refer to the consumed or spent outputs as inputs of a transaction, while to the newly created unspent outputs simply as the outputs of a transaction. Every output is connected to a public key which is also called *address* (transparent or shielded). The output can be only spent with the address's corresponding private key.

4.2.2 Sapling Transaction Layout

To clarify the context in which the subliminal channels found work and how they can be exploited to exchange subliminal messages, we will now briefly describe how transactions are created and stored on the Zcash blockchain.

The input of a Zcash Sapling transaction consists of a sequence of *Spend Descriptions* and *Transparent Inputs*, while its output consists of a sequence of *Output*

[1] At the moment of writing 1 ZEC is exchanged at about 200 USD.

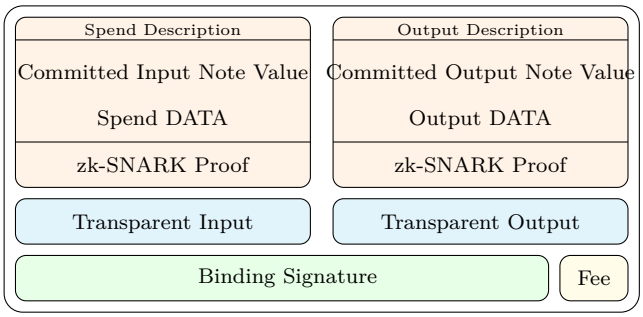


FIGURE 4.1: Sapling Transaction Layout

Descriptions and *Transparent Outputs*. It is up to the user to create fully shielded transactions consisting of only Spend and Output Descriptions, fully transparent ones or transactions that result from a combination of them.

An example of a Sapling transaction is shown in Figure 4.1, where one Spend Description, one Transparent Input, one Output Description and one Transparent Output are included. This simplified layout will guide our description through the main elements of a transaction. While a Spend Description refers to a previous transaction note, which is spendable only if the corresponding *spending key* is known, an Output Description corresponds to a new one.

Each note contains a *note value* which accounts for the total amount of ZEC involved. While in Transparent Inputs/Outputs the note value is publicly readable, in the case of Spend and Output Descriptions, it is hidden in the form of a Pedersen Commitment [Ped92]. Note value commitments are included, along with some other *DATA*, in the respective descriptions they refer to and allow - thanks to a Binding Signature - to publicly verify the total transaction balancing value.

Each description is finalized by appending a zk-SNARK proof that assures value commitment integrity, spend authority (as in the case for Spend Descriptions), double-spending prevention and other protocol coherence requisites.

4.2.3 Subliminal Channels and Attack Scenarios

The Zcash Sapling protocol introduced many new features. Among these, it implements the *"Decoupled Spend Authority"* that, quoting from the official pre-release note[2], enables *"enterprises [to] perform an inexpensive signature step in a trusted environment while allowing another computer, not trusted with the spending key, to construct the proof. Additionally, hardware wallets can support shielded addresses by allowing the connected computer to construct the proof without exposing the spending key to that machine"*. We can consider the following two scenarios to motivate why this could be a security and privacy issue. In the first one, the zk-SNARK proof generation is delegated to a computation server (or hardware) that can surreptitiously embed extra tagging information in the generated proofs. Whenever it happens, we have a subliminal channel that we will refer to as *Inner Subliminal Channel*. In the second scenario, lightweight closed source wallets can embed subliminal information into already generated valid zk-SNARK proofs exploiting their malleability. We will refer to such channels as *Outer Subliminal Channel*.

In the Zcash Sapling protocol, we found subliminal channels that exist in both the zk-SNARK proof generation mechanism and the adopted commitment scheme. More precisely, we found an Inner Subliminal Channel and an Outer Subliminal Channel,

[2] https://z.cash/blog/whats-new-in-sapling/

described in the following Sections related to the currently implemented zk-SNARK scheme. We further found another channel that we called Pedersen Subliminal Channel, which refers to the Pedersen Commitment scheme used.

For each type of subliminal channel found, the techniques to embed a subliminal message are similar: for this reason, we chose to describe in detail the embedding of a subliminal message and its retrieval only for the subliminal channels found during and after the zk-SNARK proof generation.

After describing the computational complexity of the proposed embedding procedures, a concrete attack scenario is discussed, and some countermeasures are proposed to prevent the use of such subliminal channels. A brief discussion on the effectiveness and efficiency of these channels to tag real Zcash transactions will then follow.

4.3 The zk-SNARK Subliminal Channels

We now show how Provers compliant with the Sapling zk-SNARK proof generation protocol can exchange extra b-bits of information with protocol-compliant Verifiers. Since these bits are exchanged by using parameters employed during the proof generation that were not intentionally designed for communications, we will refer to these communication channels as subliminal channels [Sim83; Sim94; YY04] and to the exchanged messages as subliminal messages.

We will describe in detail two different constructions: the *Inner Subliminal Channel* and the *Outer Subliminal Channel*. In the Inner one, the message is embedded in the zk-SNARK proof during its generation, while with the Outer Subliminal Channel a subliminal message is embedded in an already generated valid proof before it is signed.

When these subliminal channels are used, the resulting proofs will be valid and indistinguishable from any other valid proof for the same statements if some *auxiliary* information remains unknown to the Verifier.

4.3.1 Groth's NIZK argument

In order to set out the definitions and notations employed in the construction of the proposed subliminal channels, we briefly recall Groth's NIZK argument [Gro16] for arithmetic circuit satisfiability currently adopted by Zcash Sapling. The setting is as follows:

- Consider a relation generator $\mathcal{R}$ that returns relations of the form

$$R = (p, G_1, G_2, G_T, e, g, h, l, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^{m}, t(X))$$

  where (p, G_1, G_2, G_T, e) is a bilinear group of order p and g, h are generators, respectively, of G_1 and G_2.

  The relation defines a language of statements $(a_1, ..., a_l) \in \mathbb{Z}_p^l$ and witnesses $(a_{l+1}, ..., a_m) \in \mathbb{Z}_p^{m-1}$ such that with $a_0 = 1$

$$\sum_{i=0}^{m} a_i u_i(X) \cdot \sum_{i=0}^{m} a_i v_i(X) = \sum_{i=0}^{m} a_i w_i(X) + h(X)t(X)$$

  for some degree $n-2$ polynomial $h(x)$.

A Prover that generates a proof π for statement $(a_1, ..., a_l)$ should be able to convince any Verifier that he knows the corresponding witness $(a_{l+1}, ..., a_m)$ without revealing any information related to the witness.

In Groth's scheme, both proof generation and verification are done in the *Common Reference String model*: the respective *Setup*, *Prover* and *Verifier* procedures are defined as follows.

- $\boldsymbol{\sigma \leftarrow \mathbf{Setup}(R)}$ **:** pick $\alpha, \beta, \gamma, \delta \leftarrow \mathbb{Z}_p^*$ and compute

$$\sigma_1 = \left(g^{\alpha}, g^{\beta}, g^{\delta}, \left\{ g^{x^i} \right\}_{i=0}^{n-1}, \left\{ g^{\frac{x^i t(x)}{\delta}} \right\}_{i=0}^{n-2}, \left\{ g^{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}} \right\}_{i=0}^{l} \right)$$

$$\sigma_2 = \left(h^{\beta}, h^{\gamma}, h^{\delta}, \left\{ h^{x^i} \right\}_{i=0}^{n-1} \right)$$

  The tuple $\sigma = (\sigma_1, \sigma_2)$ is referred to as *Common Reference String*.

- $\boldsymbol{\pi \leftarrow \mathbf{Prover}(R, \sigma, (a_1, ..., a_m))}$ **:** pick $r, s \leftarrow \mathbb{Z}_p$ and compute $\pi = (A, B, C)$ where

$$A = g^{\alpha + \sum_{i=0}^{m} a_i u_i(x) + r\delta}, \qquad B = h^{\beta + \sum_{i=0}^{m} a_i v_i(x) + s\delta},$$

$$\hat{C} = g^{\frac{\sum_{i=l+1}^{m} a_i (\beta u_i(x) + \alpha v_i(x) + w_i(x)) + h(x)t(x)}{\delta}},$$

$$\hat{B} = g^{\beta + \sum_{i=0}^{m} a_i v_i(x) + s\delta}, \qquad C = \hat{C} \cdot A^s \cdot \hat{B}^r \cdot g^{-rs\delta}$$

- $\boldsymbol{\mathbf{0/1} \leftarrow \mathbf{Verifier}(R, \sigma, (a_1, ..., a_l), \pi)}$ **:** compute

$$T = g^{\frac{\sum_{i=0}^{l} a_i (\beta u_i(x) + \alpha v_i(x) + w_i(x))}{\gamma}}$$

  Parse $\pi = (A, B, C)$, and accept the proof if and only if

$$e(A, B) = e(g^{\alpha}, h^{\beta}) e(T, h^{\gamma}) e(C, h^{\delta})$$

4.3.2 The Inner Subliminal Channel

Suppose that a Prover, hereafter called *"Subliminal Prover"*, wishes to send b-bits of information to a Verifier, that from now on we will refer to as *"Subliminal Verifier"*. A Subliminal Prover cannot directly communicate to a Subliminal Verifier, because otherwise, there is no need to use subliminal channels. When a message is embedded, the Prover does not know when it will be recovered, and he cannot warn the Verifier to recover the message from a specific proof rather than another.

To overcome these limitations, in our constructions, the Subliminal Prover and Verifier share some secret auxiliary information, denoted as aux, that permits the Verifier to distinguish a proof with a subliminal message from an honest random proof generated for the same statements.

We let $\omega : \{0,1\}^* \rightarrow \{0,1\}^b$ and $\xi : \{0,1\}^* \rightarrow \{0,1\}^{\frac{b}{2}}$ be two uniformly distributed polynomial-time computable functions. The abstract purpose of ω is to obtain the message to embed from the statements of the proof, while ξ acts as a message extractor, that is it recovers parts of the embedded message from the proof. To both bind a subliminal message to some specific information and to permit Subliminal Verifiers to recognize proofs with subliminal messages embedded, we require that both ω and ξ take as input the auxiliary information aux.

In short, for statement $(a_1, ..., a_l)$, we let

$$M = \omega(aux, a_1, .., a_l)$$

be the message that the Subliminal Prover would like to send to a Subliminal Verifier, assuming they both know ω, ξ and aux.

As opposed to the Subliminal Prover, the Subliminal Verifier takes as input a set $\Omega = \{\omega_i\}_{i \in I}$, for some index set I. Each function ω_i corresponds to a different method the Subliminal Prover can use to obtain the subliminal message, especially when the Prover uses a unique $\omega = \omega_i$ associated to a user.

In order to embed and recover the subliminal message M, the Prover and Verifier procedures are therefore modified as follows:

- $\boldsymbol{\pi \leftarrow \text{SubliminalProver}(R, \sigma, (a_1, ..., a_m), \omega, \xi, aux)}$ **:**
  - Compute $M = \omega(aux, a_1, .., a_l)$ and denote with m_1, m_2 the first and the last $\frac{b}{2}$ bits of M, respectively.
  - Randomly select $r \leftarrow \mathbb{Z}_p$ until

$$\xi\Big(aux, a_1, ..., a_l, \Big(g^{\alpha + \sum_{i=0}^{m} a_i u_i(x)}\Big) \cdot \Big(g^{\delta}\Big)^r\Big) = m_1$$

  - Randomly select $s \leftarrow \mathbb{Z}_p$ until

$$\xi\Big(aux, a_1, ..., a_l, \Big(h^{\beta + \sum_{i=0}^{m} a_i v_i(x)}\Big) \cdot \Big(h^{\delta}\Big)^s\Big) = m_2$$

  - With the r, s obtained compute $\pi = (A, B, C)$, where

$$A = g^{\alpha + \sum_{i=0}^{m} a_i u_i(x) + r\delta} \qquad B = h^{\beta + \sum_{i=0}^{m} a_i v_i(x) + s\delta}$$

$$\hat{C} = g^{\frac{\sum_{i=l+1}^{m} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + h(x)t(x)}{\delta}}$$

$$\hat{B} = g^{\beta + \sum_{i=0}^{m} a_i v_i(x) + s\delta} \qquad C = \hat{C} \cdot A^s \cdot \hat{B}^r \cdot g^{-rs\delta}$$

- $\boldsymbol{(0/1, M) \leftarrow \text{SubliminalVerifier}(R, \sigma, \pi, (a_1, ..., a_l), \Omega, \xi, aux)}$ **:**
  - Parse $\pi = (A, B, C)$, and let

$$m_1 = \xi(aux, a_1, ..., a_l, A) \qquad m_2 = \xi(aux, a_1, ..., a_l, B)$$

  - If exists an $\omega \in \Omega$ such that $\omega(aux, a_1, ..., a_l) = (m_1 || m_2)$, recover M as

$$M = (m_1 || m_2)$$

    Otherwise, set $M = \perp$.
  - Return $(\text{ Verifier}(R, \sigma, (a_1, ..., a_l), \pi)\text{ , } M\text{ })$

4.3.3 The Outer Subliminal Channel

In Groth's original scheme and in its corresponding Zcash Sapling implementation, a user that possesses, for a certain statement, a valid proof $\pi = (A, B, C)$ is able to transform it into a different, but still valid, proof $\pi' = (A', B', C')$ for the same statement.

The proof can be transformed using the following transformations, which exploit the multiplicative nature of the proof structure:

- Select random $\tilde{r} \in \mathbb{Z}_p$ and set

$$A' = A^{\tilde{r}}\,, \qquad B' = B^{\frac{1}{\tilde{r}}}\,, \qquad C' = C$$

- Select random $\tilde{s} \in \mathbb{Z}_p$ and set

$$A' = A\,, \qquad B' = B \cdot (h^{\delta})^{\tilde{s}}\,, \qquad C' = A^{\tilde{s}} \cdot C$$

- Select random $\tilde{r}, \tilde{s} \in \mathbb{Z}_p$ and set

$$A' = A^{\tilde{r}}\,, \qquad B' = B^{\frac{1}{\tilde{r}}} \cdot (h^{\delta})^{\frac{\tilde{s}}{\tilde{r}}}\,, \qquad C' = A^{\tilde{s}} \cdot C$$

It is straightforward to see that if $\pi = (A, B, C)$ is accepted by a Verifier, then $\pi' = (A', B', C')$ will be accepted as well: indeed, considering the last transformation (the other two are special cases of this), we have

$$\begin{aligned}
e(A', B') &= e(g^{\alpha}, h^{\beta})e(T, h^{\gamma})e(C', h^{\delta}) &\Leftrightarrow\\
e(A^{\tilde{r}}, B^{\frac{1}{\tilde{r}}} \cdot (h^{\delta})^{\frac{\tilde{s}}{\tilde{r}}}) &= e(g^{\alpha}, h^{\beta})e(T, h^{\gamma})e(A^{\tilde{s}} \cdot C, h^{\delta}) &\Leftrightarrow\\
e(A, B)e(A^{\tilde{s}}, h^{\delta}) &= e(g^{\alpha}, h^{\beta})e(T, h^{\gamma})e(A^{\tilde{s}}, h^{\delta})e(C, h^{\delta}) &\Leftrightarrow\\
e(A, B) &= e(g^{\alpha}, h^{\beta})e(T, h^{\gamma})e(C, h^{\delta})
\end{aligned}$$

With similar techniques as employed in the Inner Subliminal Channel, we show how these three transformations can be used to embed a message into a valid proof π, thus creating another subliminal channel that we will refer to as *"Outer Subliminal Channel"*. We will demonstrate it using the third proof transformation, but the construction easily generalizes the other two transformations. Here we can also take advantage of parallel computation since the randomized group elements $A' = A^{\tilde{r}}$ and $C' = A^{\tilde{s}} \cdot C$ can be computed independently.

We let again $\omega : \{0,1\}^* \to \{0,1\}^b$ and $\xi : \{0,1\}^* \to \{0,1\}^{\frac{b}{2}}$ be two uniformly distributed polynomial-time computable functions and for statement $(a_1, ..., a_l)$ and some auxiliary information aux, we let $M = \omega(aux, a_1, .., a_l)$ be the subliminal message a Subliminal Prover would like to embed into a proof π for statement $(a_1, .., a_l)$.

The new SubliminalProver and SubliminalVerifier procedures are as follows:

- $\boldsymbol{\pi \leftarrow \mathbf{SubliminalProver}(R, \sigma, (a_1, ..., a_m), \omega, \xi, aux)}$ **:**
  - $\tilde{\pi} = (\tilde{A}, \tilde{B}, \tilde{C}) \leftarrow \text{Prover}(R, \sigma, (a_1, ..., a_m))$.
  - Compute $M = \omega(aux, a_1, .., a_l)$ and denote with m_1, m_2 the first and the last $\frac{b}{2}$ bits of M, respectively.
  - Randomly select $r \leftarrow \mathbb{Z}_p$ until

$$\xi(aux, a_1, .., a_l, \tilde{A}^r) = m_1$$

  - Randomly select $s \leftarrow \mathbb{Z}_p$ until

$$\xi(aux, a_1, .., a_l, \tilde{A}^s \cdot C) = m_2$$

  - With the r, s obtained compute $\pi = (A, B, C)$, where

$$A = \tilde{A}^r\,, \qquad B = \tilde{B}^{\frac{1}{r}} \cdot (h^{\delta})^{\frac{s}{r}}\,, \qquad C = \tilde{A}^s \cdot C$$

- **$(0/1, M) \leftarrow$ SubliminalVerifier$(R, \sigma, \pi, (a_1, ..., a_l), \Omega, \xi, aux)$:**
  - Parse $\pi = (A, B, C)$, and let
  $$m_1 = \xi(aux, a_1, ..., a_l, A)\,, \qquad m_2 = \xi(aux, a_1, ..., a_l, C)$$
  - If exists an $\omega \in \Omega$ such that $\omega(aux, a_1, ..., a_l) = (m_1||m_2)$, recover M as
  $$M = (m_1||m_2)$$
  Otherwise, set $M = \perp$.
  - Return $(\text{ Verifier}(R, \sigma, (a_1, ..., a_l), \pi)\,,\, M\,)$

4.3.4 Computational Complexity

We will now discuss the average number of operations required to successfully embed a b-bits message M in a valid proof π, using the Inner or the Outer Subliminal Channel, respectively. Both estimations are done assuming that ξ is uniformly distributed on $\{0,1\}^{\frac{b}{2}}$.

For the Inner Subliminal Channel, a b-bits message can be embedded with an average cost of $O(2^{\frac{b}{2}})$ parallelizable step operations, where each step consists of a scalar-point multiplication, a point addition and one ξ-evaluation. More precisely, $O(2^{\frac{b}{2}})$ step operations are needed to find r and, similarly, $O(2^{\frac{b}{2}})$ operations are needed to find s, such that the resulting A, B successfully embed the first and the second half, respectively, of the subliminal message M. These complexities derive directly from the hypothesis that ξ is uniformly distributed and that both G_1 and G_2 are cyclic groups of order p with $\log_2(p) >> \frac{b}{2}$. To give a more concrete example, let us fix a random value $M \in \{0,1\}^{\frac{b}{2}}$ and let, as in the case for r, $f(i) = \left(g^{\alpha+\sum_{i=0}^{m} a_i u_i(x)}\right) \cdot \left(g^{\delta}\right)^i$. Then, varying $i \in [0, 2^{\frac{b}{2}+1} - 1]$, $f(i)$ will take distinct values and hence, with high probability, $\xi(aux, a_1, ..., a_l, f(i))$ will take all possible values in the interval $[0, 2^{\frac{b}{2}+1} - 1]$. This means that there is a $j \in [0, 2^{\frac{b}{2}+1} - 1]$ such that $\xi(aux, a_1, ..., a_l, f(j)) = m$ and to find it, an average number of $\frac{2^{\frac{b}{2}+1}-1}{2} \approx 2^{\frac{b}{2}}$ $\xi \circ f$-evaluations are needed, corresponding to an average of $O(2^{\frac{b}{2}})$ step operations overall.

Each step operation cost can be further improved to consists only of a point addition and one ξ-evaluation: for example, in the case of r, the Subliminal Prover can iteratively compute the values

$$A_i = \begin{cases} g^{\alpha+\sum_{i=0}^{m} a_i u_i(x)} & \text{if } i = 0 \\ A_{i-1} \cdot g^{\delta} & \text{otherwise} \end{cases}$$

until a preimage A_r for m_1 with respect to ξ is found.

With similar arguments, it is easy to prove that, similarly as in the Inner Subliminal Channel, embedding a b-bits message using the Outer Subliminal Channel requires an average number of $O(2^{\frac{b}{2}})$ point additions and ξ-evaluations.

Note that for both proposed channels, each value r and s can be searched in parallel: hence, if 2^c computation units are available, M could be embedded with an average number of $O(2^{\frac{b}{2}-c})$ point additions and ξ-evaluations per unit.

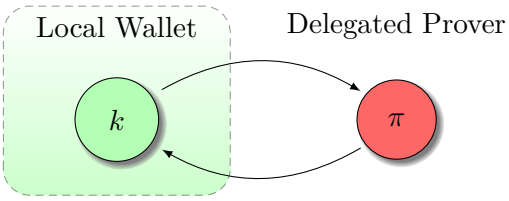


FIGURE 4.2: Delegated proof π signed with user's secret master key k.

4.3.5 Adversary Assumptions

In a lightweight wallet scenario, where proof generation is delegated to a third-party Prover, subliminal channels could be maliciously exploited to share a certain amount of information related to users, thus permitting a wide variety of different activities such as proof fingerprinting, user-tracing, leak of transaction data and so on. Thus at stake is privacy and security of the user funds as well as fungibility of the affected coin.

In our use case, a Subliminal Prover could embed in all generated proofs unique tracing information related to the user that requests the proof computation, and later share with some Subliminal Verifiers a certain auxiliary information aux that permits them to perform tracing analysis only to a particular subset of users the Prover authorizes them to look at.

This is a realistic setting for an adversary in our case as well: at the time of writing, the Zcash developers were working on implementing delegated proofs, where only the secret key and basic signing is done locally, while the rest of the proof generation is delegated to another system. In this setting, as shown in Figure 4.2, the adversary would not learn the secret master key of the target (since it is stored only in the wallet) but could still reveal, using one of the proposed subliminal channels, some transaction-tagging information.

Furthermore, to remain undetectable, the attacker wishes to hide its malicious activities and reduce the number of interactions with its target and the environment to a minimal level: for example, the attacker will not send any, even if encrypted, message to the external world indicating that he is adopting a subliminal proving mechanism on the target's machine, as this would be easily detectable by network traffic analysis. Within this scenario, our adversary would attack the proving system once, for example, when the software is installed, and will not require any further direct interaction as long as the proving system remains malicious.

Aiming at fingerprinting proofs in order to permit later tracing activities, a Subliminal Prover could proceed according to the following steps:

- Associate to each user U a unique random n-bit key k_U. Let $K = \{k_U\}_U$ be the set of all user keys.
- For $t \in \mathbb{N}^+$ let H_t be a t-bits cryptographic hash functions. For example, if $t \leq 512$, $H_t(x)$ could be defined as the last t-bits of $BLAKE2b\text{-}512(x)$.
- Let $E(M, k)$ be an easily computable algorithm that takes a plaintext M as input, a key k and returns an output of b-bits. Depending on the security property required for the subliminal channel, E can differently model a Message Authentication Code, a Block Cipher, a Public Key Encryption scheme and others. The following description indeed, easily generalizes to all these cases.

- When the user U requests a proof π for statement $(a_1, ..., a_l)$, the Subliminal Prover defines

$$aux = \{E, k_U, b, H_t\}$$
$$\xi(aux, a_1, .., a_l, x) = H_{\frac{b}{2}}(x)$$
$$\omega(aux, a_1, ..., a_l) = E(a_1 \,||\, ... \,||\, a_l, k_U)$$

  and executes

$$\pi \leftarrow \text{SubliminalProver}(R, \sigma, (a_1, ..., a_m), \omega, \xi, aux)$$

Consequently, the Subliminal Verifier can link proofs to users in the following way:

- The Subliminal Verifier receives from the Subliminal Prover the set

$$aux = \{E, Trace, b, H_t\}$$

  where $Trace \subseteq K$ is the subset of keys associated to users he wishes to trace.

- The Subliminal Verifier defines

$$\xi(aux, a_1, .., a_l, x) = H_{\frac{b}{2}}(x)$$

  and let Ω

$$\begin{aligned}\Omega &= \{\omega_{k_U}(aux, a_1, ..., a_l)\}_{k_U \in Trace} \\ &= \{E(a_1 \,||\, ... \,||\, a_l \,, k_U)\}_{k_U \in Trace}\end{aligned}$$

  be the set of functions ω_{k_U} indexed over the user keys k_U in $Trace$.

  He then executes

$$(0/1, M) \leftarrow \text{SubliminalVerifier}(R, \sigma, \pi, (a_1, ..., a_l), \Omega, \xi, aux)$$

- If the recovered message $M = (m_1 \,||\, m_2) \neq \perp$, the Subliminal Verifier associates π to the user U such that, for a certain $\omega_{k_U} \in \Omega$, it holds $\omega_{k_U}(aux, a_1, ..., a_l) = M$.

It is straightforward to see how this construction can be easily generalized to allow different embedding techniques and other malicious activities such as transaction data leaks and more.

4.3.6 Countermeasures

If ω and ξ have some cryptographic properties, in principle, it should not be possible without the auxiliary information aux to distinguish a random proof from one with a subliminal message embedded (even if their definitions become public). Therefore, as could be the case for a lightweight wallet scenario, if a user delegates heavy cryptographic computations to a third-party entity as a Prover, he cannot know, looking at the generated proof π, if the Prover embedded a subliminal message or not, no matter what subliminal channel the Prover might have used.

To eliminate any potentially embedded subliminal message, the user should further randomize Prover's proof $\pi = (A, B, C)$ using, for example, one of the three proof transformations discussed at the beginning of the Outer Subliminal Channel Section.

Unfortunately, the most expensive transformation

$$A' = A^{\tilde{r}}\,, \qquad B' = B^{\frac{1}{\tilde{r}}} \cdot (h^{\delta})^{\frac{\tilde{s}}{\tilde{r}}}\,, \qquad C' = A^{\tilde{s}} \cdot C$$

is the only one that assures the user to fully disrupt all the (eventually) embedded b-bits. Clearly, the computation of this further transformation should not be delegated to a third-party, since this could, in turn, use the Outer Subliminal Channel to embed its own subliminal message.

Alternatively, if two valid proofs π_1 and π_2 for the same statement are available to the user, it is possible to combine them into a new valid proof π in a way that would disrupt a subliminal message, especially if π_1 and π_2 come from different third-party Provers. Our proof-combination method requires, however, that the user chooses at least one of the two randomnesses r, s that the third-party Provers will employ during the proof generation.

We therefore assume that the user possesses two proofs $\pi_1 = (A_1, B_1, C_1)$ and $\pi_2 = (A_2, B_2, C_2)$ for the same statement $(a_1, ..., a_l)$ generated accordingly to the chosen randomnesses s_1 and s_2, respectively.

He can then compute a new proof π for the same statement as

$$\pi = (A, B, C) = \left(\sqrt{A_1 \cdot A_2}, \sqrt{B_1 \cdot B_2}, \sqrt{C_1 \cdot C_2} \cdot \left(\frac{A_1}{A_2}\right)^{\frac{s_1 - s_2}{4}} \right)$$

To see that π is still valid, we explicitly write the computations, as reported in Groth's original scheme, that generate the group elements of both proofs π_1 and π_2:

$$A_1 = g^{\alpha + \sum_{i=0}^{m} a_i u_i(x) + r_1 \delta} \qquad B_1 = h^{\beta + \sum_{i=0}^{m} a_i v_i(x) + s_1 \delta}$$

$$A_2 = g^{\alpha + \sum_{i=0}^{m} a_i u_i(x) + r_2 \delta} \qquad B_2 = h^{\beta + \sum_{i=0}^{m} a_i v_i(x) + s_2 \delta}$$

$$\hat{C} = g^{\frac{\sum_{i=l+1}^{m} a_i (\beta u_i(x) + \alpha v_i(x) + w_i(x)) + h(x)t(x)}{\delta}}$$

$$\hat{B}_1 = g^{\beta + \sum_{i=0}^{m} a_i v_i(x) + s_1 \delta} \qquad C_1 = \hat{C} \cdot A_1^{s_1} \cdot \hat{B}_1^{r_1} \cdot g^{-r_1 s_1 \delta}$$

$$\hat{B}_2 = g^{\beta + \sum_{i=0}^{m} a_i v_i(x) + s_2 \delta} \qquad C_2 = \hat{C} \cdot A_2^{s_2} \cdot \hat{B}_2^{r_2} \cdot g^{-r_2 s_2 \delta}$$

Hence, using the above formula, the resulting group elements A, B, C of π are

$$A = g^{\alpha + \sum_{i=0}^{m} a_i u_i(x) + \frac{r_1 + r_2}{2} \delta} \qquad B = h^{\beta + \sum_{i=0}^{m} a_i v_i(x) + \frac{s_1 + s_2}{2} \delta}$$

$$\hat{B} = g^{\beta + \sum_{i=0}^{m} a_i v_i(x) + \frac{s_1 + s_2}{2} \delta} \qquad \left(\frac{A_1}{A_2}\right)^{\frac{s_1 - s_2}{4}} = g^{\frac{(r_1 - r_2)(s_1 - s_2)}{4} \delta}$$

$$\begin{aligned} C &= \hat{C} \cdot A^{\frac{s_1 + s_2}{2}} \cdot \hat{B}^{\frac{r_1 + r_2}{2}} \cdot g^{-\frac{r_1 s_1 + r_2 s_2}{2} \delta} \cdot g^{\frac{(r_1 - r_2)(s_1 - s_2)}{4} \delta} \\ &= \hat{C} \cdot A^{\frac{s_1 + s_2}{2}} \cdot \hat{B}^{\frac{r_1 + r_2}{2}} \cdot g^{-\frac{r_1 + r_2}{2} \cdot \frac{s_1 + s_2}{2} \delta} \end{aligned}$$

If we look closely, we notice that π is a proof for statement $(a_1, ..., a_l)$ whose elements A, B, C are generated using randomnesses $s = \frac{s_1 + s_2}{2}$ and $r = \frac{r_1 + r_2}{2}$ and hence it is valid by construction.

If, on the contrary, π is not accepted as valid, but both π_1 and π_2 are, it means that at least one between the randomnesses s_1 and s_2 is not the one that was effectively used during the proofs generation: the user can easily spot (in a non-lightweight

setting) which Prover cheated, recomputing the group elements B_1 and B_2 from the Common Reference String and checking which of these is not equal to the one provided in the proofs π_1 and π_2.

All these considerations imply that, even in a lightweight scenario, a suspicious user has to perform some extra elliptic curve arithmetic on its device to prevent embedding of subliminal messages.

At the same time, to permit users to not blindly trust third-party Provers, the implementation of a delegated proof generation mechanism should either admit further randomization of proofs, or take into consideration the combination of multiple proofs at the cost of extra $|p|-$bits of information exchanged between delegated Provers and the user.

Other than this, Trusted Execution Environments, like SGX, could help light clients to mitigate trust issues in the proof delegation.

Systematic adoption of countermeasures to avoid embedding of subliminal messages is essential and not limited only to the prevention of the malicious activities we have seen so far. Unfortunately, subliminal channels pose a potential issue related to Zcash fungibility: if miners are incentivized by some entity to mine first transactions with a particular subliminal message embedded, and this fact, in turn, encourages users to consciously embed such message to obtain a quicker transaction approval, then this gives rise to a certain disparity between Zcash coins (more precisely, transactions). Moreover, in this hypothetical scenario, users that refuse to embed a subliminal message in their transactions should pay a higher fee to get the same level of priority, a win-win situation for miners.

4.4 The Pedersen Subliminal Channel

In the Pedersen Commitment scheme, given a cyclic group G of order p and two random generators g, h of G for which the discrete logarithm $\log_g(h)$ is unknown, c is said to be a commitment of $v \in \mathbb{Z}_p$ for a randomly chosen randomness $r \in \mathbb{Z}_p$, if $c = g^v h^r$.

One of the main reason why the Pedersen Commitment is employed is because it is additively homomorphic, that is, given any two commitments $c_1 = g^{v_1} h^{r_1}$ and $c_2 = g^{v_2} h^{r_2}$ for values v_1 and v_2, respectively, their product $c = c_1 c_2 = g^{v_1+v2} h^{r_1+r_2}$ is a commitment to the sum $v_1 + v_2$ with randomness $r_1 + r_2$.

The Pedersen Commitment homomorphic property combined with the adoption of a *binding signature*, that binds the note commitment values to the total balancing value, permits the signer to convince everyone that he is able to open the commitment

$$c = g^{\sum_i v_{i,IN} - \sum_j v_{j,OUT}} h^{\sum_i r_{i,IN} - \sum_j r_{j,OUT}}$$

given by the product of all input committed values $g^{v_{i,IN}} h^{r_{i,IN}}$ divided by the product of all shielded output committed values $g^{v_{j,OUT}} h^{r_{j,OUT}}$, without revealing any of the $v_{i,IN}, v_{j,OUT}$. Since in a Sapling transaction, in turn, c is a commitment to the resulting transparent value change, this scheme assures at the same time both confidentially and coherence of a shielded transaction.

Note that the randomness makes any commitment for a value v indistinguishable from a random element in G: this remark suggests that a Subliminal Signer, or an attacker that controls the Signer randomness source, using similar techniques we have seen in previous Sections, could embed a b-bits subliminal message in any commitments, carefully choosing the randomnesses until the resulting committed value

satisfies some desired properties related to the message he wishes to embed. We will refer to this new subliminal channel to as *Pedersen Subliminal Channel*.

Other than that, if the Signer randomness source is under control of a malicious party (i.e. it runs a malicious version of the software), with the knowledge of r, a Subliminal Verifier could partially open a commitment c of v to $g^v = ch^{-r}$: in Zcash Sapling, v is the number of Zatoshi corresponding to the current shielded note, that is an integer in the interval $[0, 2.1 \cdot 10^{15}]$.

Since in our scenario v ranges in this bounded interval and in practice most transactions have note values that lie in the first part of such interval, with the partially opened commitment g^v the Subliminal Verifier could easily mount a rainbow table attack [Oec03].

Just to give an example, suppose he disposes of 4TB $\approx 8 \cdot (4 \cdot 10^{12})$ bits of storage memory. Since the Jubjub curve used in Zcash Sapling, has a base field of 381-bits, the attacker can then store $\approx 2^{36}$ compressed elliptic curve points. Thus, he computes (eventually in parallel) and stores all the elements g^v for $v \in [0, 2^{36}]$, that is, note values up to $\approx 840 \cdot 10^8$ Zatoshi. Then, with this table and for note values lower than this bound, it is straightforward for him to find the discrete logarithm of g^v, thus revealing the shielded value.

It remains to find a way for the attacker to make the Subliminal Verifier aware that a transaction is *weak*, in the sense that the randomness used during commitment generation is deterministic for both attacker and Verifier if some auxiliary information is known to the latter: note that, in fact, in our adversary assumptions, the attacker (i.e. the malicious commitment mechanism) could not directly communicate with the external world once the target is attacked.

To get through this, the attacker could embed in some elements of the transaction a *marker* subliminal message, easily recognizable only if some auxiliary information is known, that makes the Subliminal Verifier aware that the current transaction is weak.

At first glance, this seems a stronger assumption about our attacker: he has to control both the randomness source in the commitment scheme and the proof generation process if he wishes to use, for example, one among the Inner Subliminal Channel or the Outer one, to embed a subliminal message.

Actually, this extra requirement is not necessary: since all shielded input/output values are committed, and the output values minus the input values have to be equal to the public balancing value, by only controlling the randomness source of the commitment scheme, the attacker could embed in only one commitment a subliminal marker message using the Pedersen Subliminal Channel, hence permitting the Subliminal Verifier to become aware that the transaction is weak and that he can proceed to the full de-commitment of the shielded values; the value of the commitment that embeds the marker message can then be easily computed from the balancing value, once all the others commitments are successfully de-committed using a rainbow table attack.

4.5 Implementation Results

We implemented the Inner Subliminal Channel and the Pedersen Subliminal Channel in the Zcash official wallet (v. 2.0.5-2), and we successfully embedded 9 bytes in a fully shielded transaction with 1 shielded input and 2 shielded outputs (a typical payment transaction with returned change), i.e. 2 bytes in each proof and 1 byte in each committed value.

The proof generation time is proportional to the number of inputs since each proof is generated independently. We ran our implementation on a standard Intel(R) Core(TM) i7-3770 CPU 3.40GHz desktop provided with 8GB of RAM and running Ubuntu 16.4 x64.

The proof generation time took on average 3.0087s, compared to the average time of 2.8412s needed for a random proof generation. This is just a 6% increase in proof time, barely noticeable, especially when using hardware wallets.

Considering that we did not implement any algorithmic optimization, this shows that subliminal channels are a compelling means for tagging transactions, and it would be prudent from the user's perspective to implement countermeasures described in Subsection 4.3.6.

4.6 Example of a Tagged Transaction

To provide a concrete example, we used our implementation to tag a transaction using the proposed subliminal channels. All proof elements A and B and values commitment have their last byte set to $0x00$, which correspond to the subliminal message.

The embedded data can be seen by hex-decoding our tagged Zcash transaction to the JSON format. The resulting JSON can be found in Listing 4.1. We embedded different parts of the subiliminal message in the following transaction fields:

- the commitment value `cv`, where 1 byte of the subliminal message is embedded in its last byte, as the result of applying the Pedersen Subliminal Channel (Section 4.4);
- the zk-SNARK proof value `proof`, where 2 bytes of the subliminal message are embedded in the 48th and 144th byte, respectively, using the Inner Subliminal Channel (Subsection 4.3.2).

Our test tagged transaction contains two shielded inputs and one shielded output, and for the ease of readability, we have marked in blue the subliminal message zero bytes. The transaction can be found on the Zcash testnet with a confirmed status and can be verified using the hash of the transaction, reported is the `txid` JSON field of Listing 4.1.

4.7 Conclusion

In this Chapter we have studied the existence and exploitability of subliminal channels in the privacy-oriented cryptocurrency Zcash.

We found three subliminal channels in the cryptographic primitives adopted for creating shielded transactions. These channels can allow a malicious prover to embed tagging information about the user into each transaction, thus invalidating the purpose of the zk-SNARK information hiding and other privacy features the Zcash blockchain technology provides (e.g., value hiding, fungibility, etc.). In particular, we detailed the Inner and Outer Subliminal Channel, which affect the zk-SNARK proof generation mechanism, and the Pedersen Subliminal Channel, which instead affects the commitment scheme used to hide transaction values. We discussed different concrete attack scenarios, and we showed the practicality of our attacks by tagging, with a complexity overhead of just 6%, a Zcash transaction with 9 bytes of information. Countermeasures against these attacks were also discussed.

LISTING 4.1: A valid Zcash Sapling transaction containing the subliminal message 0000000000000000 (highlighted in magenta).


```
"txid": "20ffc99e4e590688b465773ab7034d0055ef7d849d21320c10671253ed0
         db49c",
"overwintered": true,
"version": 4,
"versiongroupid": "892f2085",
"locktime": 0, "expiryheight": 501319,
"vin": [], "vout": [],
"vjoinsplit": [],
"valueBalance": 0.00000000,
"vShieldedSpend": [
    "cv": "840de77de2ccce945cf5e605b6e3b3fe34ac1210adb5288555ac151f3
           88c3200",
    "anchor": "19f636b14e7a7983ba89a14f6c03b5cfe540b867f8c6fe718e8e7
               51fadbf3880"
    "nullifier": "80dc569355b7eab9c101c7dba7a266982ac9b10c64e113c09a
                  44f0959e78bc4d",
    "rk": "c21c2c6db475d2a4ef8eb3f6219112a339ffb5e3d6303eabf017f9345
           da0ff84",
    "proof": "a3ab169ed20718a175e421bde1609e8f94e7f65930ce8d5a93459e
              164d614285b4dbd9aebbd31492f057f23d9bbb5a008632c4b7e9d8
              33181f9bb14b7261e27e7eafc03aa9b6d3374bd9d2169bdcec21f1
              143bf1f79ea3ec49e33765648e289010acd6a3b9dabb5e5421a237
              bba50ca88ef446f877eb87e0ee2e50907023bf9232ce72df4c4081
              873fa42c61188af700a6b76af9d28ddd9df1b026996407073162e2
              92ac301eff406a3a4aecfcb35cd801090ec7957f95fbe3d01702e3
              c09417",
    "spendAuthSig": "b69c1f7f0f7891775cbd1b7bf6ef9335d88e2c31b6dc69c
                     61624c5940db4a7720054847954fe934646ed9c1a5b768a
                     c12395f37b58eb308c3485fb5b6437e808"
],
"vShieldedOutput": [
    "cv": "e74a704a0190b634e6eaaade90dbac40f2794f3d0821a67c5b90c6fc5
           6d10500 ",
    "cmu": "1201a0844ab631ab418f429fc3d2ec64e5bafab4afa858d6a29a6b03
            02a1a8de",
    "ephemeralKey": "f0d7d39f7a748b725108004402d9dcccc095a1c89c4522f
                     d7f46551adec0babf",
    "outCiphertext": "300a380831c98423665517289347d58edc46d019240e33
                      bd824916061eff80c2745e9f22ccbec2cef151191420bd
                      21f911dbb5eaa5ec4cd09adaa09658a69a337b6a28cf38
                      3edbfbfd01789f9911a9a1",
    "proof": "b6868e9d4a8ad6677cdd8e5e0a53f5b07fe6bfa246833855e6f2b7
              139e4b3b6c46c46c7afe8163806f34f8dd1c98d000087186b68c8a4
              311ec81602290bc2f2e13d4bbde46d9baf0ba80ca3bb3986d581c3
              d8cb85fa8956541ec74c32ea7f2afb0d3d839fc8fc4ede4314ff71
              553307d2046a30ce3e6910f9477fb89f1353a04d126797d1a1d2c2
              eda973da208eb3a20095235a7dd26608ca6d3cd6607ec18bf37fb8
              5aae651ad49df523ddd1165ae896a2c8dde70e3a8ebeb4507e4d2a
              f5b186",
```

```
    "cv": "129841260f65ced05c33ffd720ff6d98afdec8f15d1be5c3c6be14181
           716d600",
    "cmu": "399e239a682b1e072f04b81bf5fff37c06b57c23499bd7394d03fae9
            61ffddcb",
    "ephemeralKey": "ef61fd9ca8b37702190ba953f259d931ab0099087d3fe4f
                     8b20cdbb24cb4f657",
    "outCiphertext": "206491a259a6b7fe9d49dafa4eddc2e8d1b700d0a06dd5
                      b58e4d69227f798049c7ccddfbb1be7cb285e75560f629
                      277ae29eaf47410fac0a57ba705a20a4d926cbd8443ad8
                      688ec02a563159de96dc21",
    "proof": "80716af531c0c68b555d5fef665c83cbbe3ad1134ae876dcc188ba
              138af7e4553c2854c6c32412e1191c9b82f1a6b300a50c737ffb62
              723a4b56c1bfaef84c8fafd0fe825c11a13b215ede04074f9e5946
              0a472fbc2035e546021f5b884118900dde46804895ce78c2b49b9d
              8d20b050c3ac6c7a969636855825707971e5b63cedb0ecc144300c
              6eea725a7067d38f00b3357681f9d25c2100f1989992e3ec2236f1
              8f7e1acd32b750bb25c31eedde336a5ad67acfbb540f009d35d867
              7e2d7b"
    ],
"bindingSig": "ea99c5d837508db138cc2d6388d990a43454cd4ccb444587276c5
               23b326205a466e017a30eb0081b1b75405c86332f2d3ee2ecc107
               7ac21ce413ca5e2294d306",
"blockhash": "0018a360262860c4059793b72695785a2276ed71192ec39b06af6a
              ac3b28ad44",
"confirmations": 1089,
"time": 1558901286,
"blocktime": 1558901286
```

Chapter 5

Cryptanalysis of the Legendre PRF and its Generalizations

5.1 Introduction

The Legendre symbol is a multiplicative function modulo an odd prime number p that assigns to an element $a \in \mathbb{F}_p$ the value $1, 0$ or -1 depending on whether or not a is a square. Specifically,

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } a = b^2 \text{ for some } b \in \mathbb{F}_p^\times \ , \\ 0 & \text{if } a = 0 \ , \\ -1 & \text{otherwise .} \end{cases}$$

The distribution of Legendre symbols has been a subject of study for number theorists at least since the early 1900s [Ala96; Ste98; Jac06; Dav31; Dav39]. In particular, it follows from the Weil bound [Wei48] that the number of occurrences of a fixed pattern of l nonzero Legendre symbols among the integers $1, 2, \ldots, p-1$ modulo p is

$$\frac{p}{2^l} + \mathcal{O}(\sqrt{p}) \, ,$$

as $p \to \infty$. In other words, the distribution of fixed length substrings of Legendre symbols converges to the uniform distribution.

In 1988, Damgård [Dam90] conjectured pseudorandom properties of the sequence

$$\left(\frac{k}{p}\right), \left(\frac{k+1}{p}\right), \left(\frac{k+2}{p}\right), \ldots ,$$

where k has been sampled from $\mathbb{F}_p$ uniformly at random. He proposed to use this construction as a pseudorandom number generator. More recently, Grassi *et al.* [Gra+16] have proposed the same construction as a candidate pseudorandom function and have shown that it can be evaluated very efficiently in the secure multiparty computation setting. Concretely, the *Legendre pseudorandom function* $L_k(x)$ is defined by mapping the Legendre symbol with a secret shift k to $\{0, 1\}$:

$$L_k(x) = \left\lfloor \frac{1}{2} \left(1 - \left(\frac{k+x}{p} \right) \right) \right\rfloor ,$$

where p is a public prime number.

Damgård's work additionally considers several generalizations of the Legendre Pseudo Random Generator (PRG) that could be more efficient and/or more secure. One of these is to replace the Legendre symbols above with Jacobi symbols. In this case, the public modulus n is taken to be a product $\prod_i p_i$ of odd primes. We recall that the Jacobi symbol of $a \in \mathbb{F}_p$ is defined as

$$\left(\frac{a}{n}\right) = \prod_i \left(\frac{a}{p_i}\right).$$

In his work, Damgård argues that Jacobi symbols are more secure by showing that the Jacobi generator is *strongly unpredictable* if the Legendre generator is *weakly unpredictable* (more details in Section 5.6). He further notes that calculating Jacobi symbols is more efficient since computing requires computing Legendre symbols modulo each of the smaller prime factors of the modulus. A second generalization proposed by Damgård consists in using higher power residue symbols instead of quadratic residue symbols. Concretely, for a prime p with $p \equiv 1 \bmod r$, he proposes to use the r-th power residue symbol $a \mapsto a^{(p-1)/r} \bmod p$. This potentially increases the throughput

of the PRF, because we now obtain $\log_2 r$ bits of output per PRF call, rather than a single bit.

Very recently, the Legendre PRF was proposed to be used in the Ethereum 2.0 proof-of-custody mechanism [Fei19b]. In this context, several cryptanalysis bounties were announced by the Ethereum foundation during the CRYPTO 2019 rump session [Fei19a]. Among the proposed challenges, there are concrete instances of the Legendre PRF with expected security levels ranging from 44 to 128 bits of security. For each instance, 2^{20} sequential output bits are given, and the goal is to recover the secret key.

Despite the longevity of Damgård's pseudorandomness conjecture and the recent surge of application-oriented interest in the Legendre PRF, relatively few cryptanalytic results are available. It is known that, given quantum query access to L_k, the key k can be recovered with a single query to L_k and in quantum polynomial time [DH00]. No subexponential attacks are known in the classical setting or the setting where a quantum adversary is only allowed to query L_k classically.

The best cryptanalytic results in the classical setting are due to Khovratovich [Kho19], who gives a memoryless birthday-bound attack. His attack recovers the key with a computational cost of $\mathcal{O}(\sqrt{p}\log p)$ Legendre symbol evaluations when given $\sqrt{p}\log p$ queries to L_k. Khovratovich also considers a higher-degree variant of the Legendre PRF, where the output is computed as the Legendre symbol of a secret polynomial in the input. Similar to the Jacobi symbol generalization, the higher-degree Legendre PRF potentially offers security and efficiency benefits.

5.1.1 Outline

In this Chapter we will cryptanalyse the Legendre PRF by improving upon Khovratovich's attacks on the one hand and by providing the first security analysis of the Jacobi and power residue symbol generalizations on the other hand. The main improvement stems from the fact that, differently than earlier work, we exploit the multiplicative property of the Legendre symbol in order to reduce attack complexity. The practicality of the new attacks is demonstrated by our solution to the first three concrete Legendre PRF challenges proposed by the Ethereum foundation [Fei19b]. These were expected to correspond to a security level of 44 and 54 bits, but our attacks imply that the actual security levels for these challenges are significantly lower.

After introducing the necessary preliminaries in Section 5.2, we show how the Khovratovich attack can be significantly improved in the low-data setting. In particular, for $M \leq \sqrt[4]{p}$ queries, the attack described in Section 5.3 recovers the key with a time-complexity of $\mathcal{O}(p\log^2 p/M^2)$ Legendre symbol evaluations and a memory cost of $\mathcal{O}(M^2)$. In Section 5.4, the attack from Section 5.3 is generalized to the higher-degree case. As before, this amounts to a significant improvement in the low-data setting. In addition, for $d \geq 3$ and with $M = p$ queries, we gain a factor of p in time complexity compared to Khovratovich's results. Furthermore, in Section 5.4, a large class of weak keys for the higher-degree Legendre PRF is shown to exist. For keys in this class, key-recovery requires roughly $\mathcal{O}(p^{\lceil d/2\rceil}d\log p)$ operations with only $d\lceil\log p\rceil$ queries to the PRF. This attack requires $\mathcal{O}(p^{\lfloor d/2\rfloor}d\log p)$ bits of memory, but trade-offs are available using Van Oorschot-Wiener golden collision search [vW99]. We also provide a reduction to the unique k-XOR problem, which results in further time-memory trade-offs.

The first of Damgård's generalizations is discussed in Section 5.6. Specifically, it will be shown that the Jacobi PRF can be broken with cost proportional to the cost of breaking the Legendre PRF for each of the prime factors of the modulus separately.

	Reference	Queries	Time	Memory
Legendre PRF	Randomized [Kho19]	$\log p$	$\ell p \log p$	$\log p$
	Khovratovich [Kho19]	$\sqrt{p} \log p$	$\ell \sqrt{p} \log p$	$\log p$
	Subsection 5.3.1	M	$M + \ell p \log p / M$	$M \log p$
	Subsection 5.3.3	M	$M^2 + \ell p \log^2 p / M^2$	M^2
	Subsection 5.3.4	M	$M^2 + p \log^2 p / M^2$	$M^2 / \log p$
Degree $d \geq 2$ Legendre PRF	Randomized [Kho19]	$\log p$	$\ell p^d d \log p$	$d \log p$
	Khovratovich [Kho19]	p	$\ell p^{d-1} d \log p$	$d \log p$
	Section 5.4	M	$M^2 + \ell p^d d^2 \log^2 p / M^2$	M^2
	Section 5.5	$d \log p$	$p^{\lceil d/2 \rceil} d \log p$	$p^{\lfloor d/2 \rfloor} d \log p$
r-th power-residue PRF	Subsection 5.7.2	M	$M^2 + sp \log^2 p / (M^2 \log^2 r)$	$M^2 \log r$
	Subsection 5.7.3	M	$M + sp \log^2 p / (Mr \log^2 r)$	$M \log r$

TABLE 5.1: Query, time and memory requirements of previous and new attacks on the Legendre PRF. The reported time and memory values are asymptotic upper bounds ($\mathcal{O}$-notation) and assume a machine with word size $\Theta(\log p)$, ℓ and s denote the time-complexity of computing a Legendre and power residue symbol respectively. The attack strategy for composite moduli from Section 5.6 can be combined with any of the attacks in this table.

The power residue symbol generalization is analyzed in Section 5.7. Besides the straightforward generalization of the attack from Section 5.3 to the r-th power residue symbol PRF, we additionally provide a more efficient attack for the case where r is large. A summary of our main results can be found in Table 5.1.

Finally, concrete implementation results are provided in Section 5.8. We report on the specific amount of time and memory necessary to solve the first three Legendre PRF challenges of the Ethereum foundation. These results showcase the practical relevance of our attacks.

5.2 Preliminaries

In this Section we formally introduce the Legendre PRF and some related useful notation used in our attacks. Furthermore, in Subsection 5.2.2, we recall how Legendre and power residue symbols can be computed efficiently, and we briefly report Khovratovich's attacks on the Legendre PRF and its higher degree variant in Subsection 5.2.3 and Subsection 5.2.4.

5.2.1 The Legendre PRF

Definition 5.1 (Legendre function)**.** *For a given odd prime p, we consider the function*

$$\begin{aligned} \mathcal{L} : \mathbb{F}_p &\to \mathbb{F}_2 \\ x &\mapsto \left\lfloor \frac{1}{2} \left(1 - \left(\frac{x}{p} \right) \right) \right\rfloor \end{aligned}$$

which maps quadratic residues modulo p to $0 \in \mathbb{F}_2$ and quadratic non-residues to $1 \in \mathbb{F}_2$.

Definition 5.2 (Legendre PRF)**.** *Let p be an odd prime and let d be a positive integer. The degree d-Legendre PRF over $\mathbb{F}_p$ is a family of functions $L_k : \mathbb{F}_p \to \mathbb{F}_2$ such that, for each key $k \in \mathbb{F}_p^d$,*

$$L_k(x) = \mathcal{L}\left(x^d + \sum_{i=0}^{d-1} k_{i+1}\, x^i\right).$$

Remark 5.1. *For any given field $\mathbb{F}_p$, the Legendre symbol is* multiplicative*, i.e.*

$$\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right) \quad \text{for all } a, b \in \mathbb{F}_p.$$

In terms of the Legendre function $\mathcal{L}$, multiplication of inputs corresponds to addition in $\mathbb{F}_2$ of the respective images. Indeed

$$\mathcal{L}(ab) = \mathcal{L}(a) \oplus \mathcal{L}(b) \quad \text{for all } a, b \in \mathbb{F}_p^\times,$$

where $\oplus$ denotes addition in $\mathbb{F}_2$.

In our analysis, we will often consider sequential evaluations of a given degree d Legendre PRF L_k starting from a point a with an additive or multiplicative step b. We call such vectors *L-sequences*.

Definition 5.3 (L-sequences)**.** *Let p be an odd prime, m a positive integer and $a, b \in \mathbb{F}_p$. For a given L_k over $\mathbb{F}_p$, we define the* arithmetic L-sequence of length m with starting point a and stride b *as the $\mathbb{F}_2^m$-vector*

$$L_k(a + b\,[m]) := (L_k(a), L_k(a+b), \ldots, L_k(a+(m-1)b))$$

Similarly, we define the geometric L-sequence of length m with starting point a and common ratio b *as the $\mathbb{F}_2^m$-vector*

$$L_k(a \cdot b^{[m]}) := (L_k(a),\, L_k(a \cdot b),\, \ldots,\, L_k(a \cdot b^{m-1}))$$

To justify the correctness of our attack, the following property of L_k will be assumed.

Assumption 5.1. *Let p be an odd prime and d a positive integer. Let $m = d\lceil \log p \rceil$. For all $k \in \mathbb{F}_p^d$, then as $p \to \infty$, there exist at most $\mathcal{O}(1)$ keys $k' \in \mathbb{F}_p^d$ such that $L_{k'}([m]) = L_k([m])$.*

5.2.2 Evaluating Legendre and Power Residue Symbols

Using the law of quadratic reciprocity, which says that for distinct odd prime integers p and q we have

$$\left(\frac{p}{q}\right)\left(\frac{q}{p}\right) = (-1)^{\frac{p-1}{2}\frac{q-1}{2}},$$

Legendre symbols (and more generally Jacobi symbols) can be computed at essentially the same cost as a GCD computation. Using the Euclidean algorithm, the cost of a Legendre symbol computation is $\mathcal{O}(\log p)$ arithmetic operations, or $\mathcal{O}(\log^2 p \log\log p)$ bit operations. Brent and Zimmerman [BZ10] give an asymptotically better algorithm with complexity $\mathcal{O}(\log p \log^2 \log p)$. Power residue symbols can be computed via modular exponentiation in time $\mathcal{O}(\log p \log(p/r) \log\log p)$. In the remainder of this Chapter, we will often refer to the cost of an algorithm in terms of the number of Legendre symbol computations or power residue symbol computations.

5.2.3 Previous Attacks on the Linear Legendre PRF

Khovratovich [Kho19] describes a chosen plaintext attack for the linear Legendre PRF L_k that recovers $k \in \mathbb{F}_p$ with $\mathcal{O}(\sqrt{p}\log p)$ queries to L_k. The attack is based on a memoryless collision search between two specific functions and can be briefly summarized as follows.

Let $m = \lceil \log p \rceil$ and consider the functions $x \mapsto L_k(x + [m])$ and $x \mapsto L_0(x + [m])$. Note that the L-sequence $L_k(x + [m])$ is available by querying the Legendre PRF, whereas $L_0(x + [m])$ does not depend on k. By Assumption 5.1, a collision between $x \mapsto L_k(x + [m])$ and $x \mapsto L_0(x + [m])$ yields k with high probability. Indeed, let $a, b \in \mathbb{F}_p$ be such that $L_k(a + [m]) = L_0(b + [m])$. We have

$$L_0(a + k + [m]) = L_0(b + [m]).$$

and , by Assumption 5.1, the number of superfluous candidates for k satisfying the above equality is expected to be at most $\mathcal{O}(1)$.

Collisions between $x \mapsto L_k(x + [m])$ and $x \mapsto L_0(x + [m])$ can be found with a generic memoryless collision search method [MOM92; vW94] in $\mathcal{O}(\sqrt{p})$ evaluations of both functions. Since computing each L-sequence requires $m = \mathcal{O}(\log p)$ calls to L_k, the overall complexity sums up to $\mathcal{O}(\sqrt{p}\log p)$ queries to L_k and L_0. More generally, if only M queries to L_k are allowed, a collision can be found with $\mathcal{O}(p\log^2 p/M)$ queries to L_0. This will be discussed in detail in Subsection 5.3.1.

We note that Khovratovich's original attack builds sequences of length m using arbitrary evaluations of the Legendre function L_k, rather than consecutive ones. This difference does not affect the overall attack complexity, but by using L-sequences, we will be able, in Section 5.3, to reduce attack data complexity.

5.2.4 Previous Attacks on the Higher-Degree Legendre PRF

Khovratovich [Kho19] also presents a generalization of the chosen plaintext attack from Subsection 5.2.3 to the quadratic case and, ultimately, to arbitrary degrees.

Let $k = (k_1, k_2) \in \mathbb{F}_p^2$ and consider the associated quadratic Legendre PRF L_k. Choose any $r \in (\mathbb{F}_p)^*$. From the multiplicative property of the Legendre symbol we get that for any $a \in \mathbb{F}_p$ and $j \in \mathbb{Z}$,

$$L_{(r^{2j}\,k_1, r^j\,k_2)}(a) = \mathcal{L}(r^{2j}) \oplus L_{(k_1,k_2)}(ar^{-j}) = L_k(ar^{-j}),^{1}$$

since r^{2j} is clearly a quadratic residue modulo p. Let $m = 2\lceil \log p \rceil$. If we find a $k' \in \mathbb{F}_p^2$ and a $j \in \mathbb{Z}$ such that

$$L_{k'}(r \cdot r^{[m]}) = L_k(r^{1-j} \cdot r^{[m]}),$$

then we successfully recover k by letting $k_1 = k_1' r^{-2j}$ and $k_2 = k_2' r^{-j}$. As for the linear case, such a collision can be found memorylessly with $\mathcal{O}(p)$ queries to L_k and $\mathcal{O}(p)$ Legendre symbol computations.

For the general case, let us consider the degree-d Legendre PRF L_k. Similarly to the quadratic case, we have for each $a \in \mathbb{F}_p$ and $j \in \mathbb{Z}$ that

$$L_{k_1 r^{dj},\, k_2 r^{(d-1)j},\, \ldots,\, k_d r^j}(a) = l(r^{dj}) \oplus L_k(ar^{-j}).$$

[1] This equation, and many other equations in this paper, only holds if none of the involved Legendre symbols evaluate to zero. Since this does not pose a problem in practice we choose to ignore this issue for notational convenience.

By guessing the coefficients $k_3, \ldots, k_d$, it is possible to attack the remaining coefficients k_1 and k_2 using geometric L-sequences of length $d\lceil \log p \rceil$, similarly as done in the quadratic case. It follows that k can be recovered using $\mathcal{O}(p^{d-2} \cdot p \cdot d \log p) = \mathcal{O}(p^{d-1} d \log p)$ Legendre symbol evaluations, given $\mathcal{O}(p)$ queries to L_k.

5.3 Improved Attack on the Linear Legendre PRF

In this Section, we show how Khovratovich's attack (Subsection 5.2.3) on the Legendre PRF can be improved when the total number of available queries is less than $\sqrt{p}$. Although our method requires additional memory in its simplest form, we discuss several techniques to reduce memory requirements while keeping the same overall time complexity.

5.3.1 Table-Based Collision Search

We first transform the attack by Khovratovich into a table-based collision search. Aiming at this, let M be the allowed number of queries to the oracle L_k, where $\log p \ll M < \sqrt{p}$ and let us set $m = \lceil \log p \rceil$ and $\tilde{M} = M - m + 1$. The attack then proceeds as follows:

1. Store in a table $\mathcal{T}$ the pairs $(L_k(a + [m]), a)$ for all $a \in \{0, \ldots, \tilde{M} - 1\}$.
2. Sample b uniformly at random from $\mathbb{F}_p$ until $(L_0(b + [m]), a) \in \mathcal{T}$ for some $a \in \{0, \ldots, \tilde{M} - 1\}$. For each a corresponding to such collision, a candidate key[2] $\tilde{k}$ is recovered as $\tilde{k} = b - a$. Candidate keys $\tilde{k}$ can be tested for correctness by comparing one or more entries of $\mathcal{T}$ with the corresponding arithmetic L-sequences with starting point $\tilde{k}$.

Regarding the time and memory complexity of this attack, we note that the first step requires M queries to L_k, from which we obtain $\tilde{M}$ arithmetic L-sequences that are stored using $\mathcal{O}(M \log p)$ memory. The second step requires $\mathcal{O}(p \log p / M)$ evaluations of the Legendre symbol, and no additional memory is needed. Hence, the overall computational cost of the attack is $\mathcal{O}(M + p \log p / M)$: this variant of the attack already reduces the query and time complexities by a $\log p$ factor compared to the memoryless collision search, although a significant amount of memory is employed.

We note that the above attack can be made deterministic by choosing $b \in \{0, \ldots, \lfloor p/\tilde{M} \rfloor\}$ and considering the sequences $v = L_0(b\tilde{M} + [m])$ in the second step of the attack. Indeed, it is easy to see that for any $k \in \mathbb{F}_p$, the arithmetic L-sequence at offset $\tilde{M} \lceil k/\tilde{M} \rceil$ will be computed in both steps of the attack and the correct key is guaranteed to be recovered after at most $\mathcal{O}(M + p \log p / M)$ Legendre symbol evaluations.

5.3.2 Expanding the Number of L-Sequences

We now show that the table $\mathcal{T}$ can be expanded without increasing the number of queries M. The key idea is to exploit the *multiplicative* property of the Legendre symbol.

Lemma 5.1. *Let m be a positive integer and $k \in \mathbb{F}_p$. For any $b \in (\mathbb{F}_p)^*$ and $a \in \mathbb{F}_p$ it holds that*

$$L_{k/b}(a/b + [m]) = (\mathcal{L}(b), \ldots, \mathcal{L}(b)) \oplus L_k(a + b[m]).$$

[2] By Assumption 5.1, the number of candidate keys would be at most $\mathcal{O}(1)$

Proof. Immediate by the multiplicative property of $\mathcal{L}$. □

Lemma 5.2. *Let $k \in \mathbb{F}_p$ and $m \leq M$ positive integers. Then from the arithmetic L-sequence $L_k([M])$, it is possible to extract $\approx M^2/m$ arithmetic L-sequences of the form $L_{k/b}(a/b + [m])$ for distinct pairs $(a, b) \in \mathbb{F}_p \times (\mathbb{F}_p)^*$.*

Proof. Let b a positive integer such that $b \leq \lfloor M/m \rfloor$. By Lemma 5.1, we get

$$L_k(a + b[m]) = (\mathcal{L}(b), \ldots, \mathcal{L}(b)) \oplus L_{k/b}(a/b + [m])$$

for any $a \in [0, M - bm + 1)$, thus each b yields a total of $M - bm + 1$ L-sequences of length m. Moreover, since $L_k(a - b[m])$ is equal to the sequence

$$L_k(a - b(m-1) + b[m]) = L_k(a' + b[m])$$

written in reverse order, we can consider negative values for b too, thus doubling the total number of sequences. Hence, the total number of arithmetic L-sequences of length m that can be extracted from $L_k([M])$ equals

$$2 \sum_{b=1}^{\lfloor M/m \rfloor} (M - bm + 1) \approx \frac{2M^2}{m} - m \sum_{b=1}^{M/m} b \approx \frac{2M^2}{m} - \frac{M^2}{m} = \frac{M^2}{m}$$

□

5.3.3 An Improved Table-Based Collision Search.

We will now use the observations from Subsection 5.3.2 to improve the table-based collision search from Subsection 5.3.1. As before, let M be the allowed number of queries to the oracle L_k, where $\log p \ll M < \sqrt{p}$ and let us set $m = \lceil \log p \rceil$. The improved attack proceeds as follows:

1. Query the sequence $L_k([M])$ and from this extract $\approx M^2/m$ sequences of the form $L_{k/b}(a/b + [m])$, as ensured by Lemma 5.2. Store all of the triples $(L_{k/b}(a/b + [m]), a, b)$ in a table $\mathcal{T}$.

2. Sample c uniformly at random from $\mathbb{F}_p$ until $(L_0(c + [m]), a, b) \in \mathcal{T}$ for some a and b. For each pair (a, b) corresponding to such collision, a candidate key $\tilde{k}$ is recovered as $\tilde{k} = bc - a$. As before, the correctness of candidate keys $\tilde{k}$ can easily be verified.

The first step of the attack requires M queries to L_k and $\approx M/m$ Legendre symbol evaluations. Storing the table $\mathcal{T}$ requires $\mathcal{O}(M^2)$ memory. In the second phase, an average of $\approx mp/M^2$ samples must be tested before a collision is found. Hence, the computational cost of this step is dominated by $\mathcal{O}(pm^2/M^2)$ Legendre symbol evaluations.

It follows that the overall cost of the attack is dominated by the extraction of $\mathcal{O}(M^2/m)$ sequences, the evaluation of $\mathcal{O}(M/m + p \log^2 p/M^2)$ Legendre symbols and a memory requirement of $\mathcal{O}(M^2)$. For $M < \sqrt{p}$, this is always an improvement over the attack from Subsection 5.3.1 – possibly after discarding some of the data.

5.3.4 Additional Optimizations

This Section describes several additional optimizations that further reduce the time and memory complexity of the attack by a factor $\Omega(\log p)$.

Using consecutive values of c. The second step of the attack from Subsection 5.3.3 can be optimized by choosing consecutive values of c rather than uniform random samples. This approach allows us to reuse most of the Legendre symbol computations since, for example, $L_0(c + [m])$ and $L_0(c + 1 + [m])$ overlap almost completely. A priori, this allows reducing the number of Legendre symbol computations by a factor of $\Omega(m)$. However, there is an important caveat: since the guesses for c are not independent, the expected number of iterations of the second step is no longer pm/M^2. To see why this is the case, recall that for any c, the algorithm will output the correct key k if there exists $(*, a, b) \in \mathcal{T}$ such that $k = bc - a$. Since the table contains an entry $(*, a, b)$ for all sufficiently small values of a and b, it is clear that if the table contains $(*, a, b)$ such that $k = bc - a$ it is likely to also contain $(*, a' = a + b, b)$ such that $k = b(c + 1) - a'$. Therefore, if c is a good guess, then $c + 1$ is also likely to be a good guess. Since the "good" values of c are clustered together in groups of size $\mathcal{O}(m)$, we expect the required number of iterations to be $\mathcal{O}(pm^2/M^2)$, which means that the factor $\Omega(m)$ that we saved by using consecutive guesses for c is lost again. However, we can still use this idea to reduce the memory complexity of the algorithm: by only storing one entry $(*, a, b)$ for each cluster of good c's, i.e. we only store the triples $(*, a, b)$ such that $|a| < |b|$, the size of the table can be reduced by a factor of $\Omega(m)$ without impacting the time complexity of the attack.

Expanding the number of L-Sequences in the second step. The idea outlined in Subsection 5.3.2 can be used to create new L-sequences from those computed during the second step of the attack. Indeed, after computing a large number of $w = \Omega(m)$ consecutive Legendre symbols $L_0(c + [w])$, it is possible to extract $\Omega(w^2/m^2)$ arithmetic subsequences of the form $L_0(c + c' + d[m])$ such that $|c'| < |d|$, with no need to compute additional Legendre symbols. Using the property that

$$L_0(c + c' + d[m]) = L_0((c + c')/d + [m]) \oplus L_0(d)$$

we can then do $\Omega(w^2/m^2)$ table lookups. Asymptotically, this allows to amortize away the cost of computing Legendre symbols, so the time complexity is dominated by the extraction of $\mathcal{O}(pm^2/M^2)$ subsequences rather than by the computation of $\mathcal{O}(pm^2/M^2)$ Legendre symbols.

Not storing reverse sequences. Since the sequence $a + b[m]$ is just the reverse of the sequence $a + b(m - 1) - b[m]$, there is some redundancy in the lookup table. Indeed, for each entry $(s, a, b) \in \mathcal{T}$, the reverse sequence corresponding to the entry $(s', a + b(m - 1), -b)$ is also stored. If instead, we only store either the sequence or its reverse (e.g. by storing the lexicographically smallest sequence), then the memory requirements are reduced by a factor of two without affecting the overall time-complexity just by looking up either the sequence $L_0(c + [m])$ *or* its reverse in $\mathcal{T}$, depending which comes first lexicographically.

5.4 Application to the Higher-Degree Legendre PRF

In this Section we generalize the attack described in Section 5.3 to Legendre PRFs of degree $d > 1$, showing how it is possible to expand the number of L-sequences in the higher-degree setting as well.

5.4.1 Expanding the Number of L-Sequences

In order to generalize Lemma 5.2, we need to extend Lemma 5.1 to the higher-degree case: this is the object of the following Lemma.

Lemma 5.3. *For any positive integer m, $b \in (\mathbb{F}_p)^*$ and $a \in \mathbb{F}_p$, there exists an invertible affine transformation $T_{a,b}$ such that, for any $k \in \mathbb{F}_p^d$,*

$$L_{T_{a,b}(k)}([m]) = (\mathcal{L}(b^d), \ldots, \mathcal{L}(b^d)) \oplus L_k(a + b[m]).$$

Moreover, for any choice of $(a, b) \in \mathbb{F}_p \times (\mathbb{F}_p)^*$, the transformation $T_{a,b}$ can be efficiently computed.

Proof. Lef f be the monic degree d polynomial with coefficient vector k, and let $T_{a,b}(k)$ be the coefficient vector of the monic polynomial $f(a + bx)/b^d$. Then, by the multiplicative property of the Legendre symbol, we have that

$$L_{T_{a,b}(k)}([m]) = (\mathcal{L}(b^d), \ldots, \mathcal{L}(b^d)) \oplus L_k(a + b[m]).$$

Furthermore, it is not hard to see that $T_{a,b}$ is invertible, affine and that it can be computed efficiently. □

Lemma 5.4. *Let $k \in \mathbb{F}_p^d$ and $m \leq M$ positive integers. Then from the arithmetic L-sequence $L_k([M])$, it is possible to extract $\approx M^2/m$ arithmetic L-sequences of the form $L_{k'}([m])$ with k' as defined in Lemma 5.3 for distinct pairs $(a, b) \in \mathbb{F}_p \times (\mathbb{F}_p)^*$.*

Proof. The proof is entirely analogous to that of Lemma 5.2. □

5.4.2 An Improved Table-Based Collision Search

The attack proceeds essentially in the same way as described in Subsection 5.3.3 for the linear case. Let M be the allowed number of consecutive queries to the oracle L_k and let $m = d\lceil \log p \rceil$. The attack comprises the following steps:

1. Query the sequence $L_k([M])$ and extract $\approx M^2/m$ sequences of the form $L_{k'}([m])$ from it, accordingly to Lemma 5.4. Store all of the triples $(L_{k'}([m]), a, b)$ in a table $\mathcal{T}$.

2. Sample k' uniformly at random from $\mathbb{F}_p^d$ until $(L_{k'}([m]), a, b) \in \mathcal{T}$ for some a and b. For each pair (a, b) corresponding to such collision, a candidate key $\tilde{k}$ can be recovered from k, a and b as in Lemma 5.3. By Assumption 5.1, the number of candidate keys is at most $\mathcal{O}(1)$, and, as before, the correctness of candidate keys can easily be verified.

As in Subsection 5.3.3, the computational cost of the first step is dominated by the extraction of $\mathcal{O}(M^2/m)$ sequences. For the second step, at most $\mathcal{O}(p^d\, m^2/M^2)$ Legendre symbols are expected to be evaluated. Hence, the total computational cost of the attack consists of $\mathcal{O}(M^2/m)$ sequence extractions and $\mathcal{O}(p^d\, d^2 \log^2 p/M^2)$ Legendre symbol evaluations, and a total of $\mathcal{O}(M^2)$ memory.

For $d \geq 3$, the time complexity is minimized when $M = p$ and consists of $\mathcal{O}(p^{d-2} d^2 \log^2 p)$ Legendre symbol evaluations. Hence, we gain a factor of p in time relative to the attacks by Khovratovich [Kho19].

5.5 Weak Keys in the Higher-Degree Legendre PRF

This section exhibits a large class of weak keys for the higher-degree Legendre PRF. Our attacks are based on the observation that for some keys, the corresponding monic polynomial factors as a product of polynomials of lower degree.

5.5.1 A Birthday-Bound Attack for Some Keys

Consider the Legendre PRF of degree $d \geq 2$ over $\mathbb{F}_p$ for a prime p. Recall that the key $k \in \mathbb{F}_p^d$ of the PRF corresponds to the monic polynomial $f(x) = x^d + \sum_{i=0}^{d-1} k_{i+1} x^i \in \mathbb{F}_p[x]$. The attack in this section is based on the observation that, with high probability, the polynomial f has a factor of degree $t = \lfloor d/2 \rfloor$. In this case, there exist two monic polynomials $g, h \in \mathbb{F}_p[x]$ with $\deg g = t$ and $\deg h = d - t$ such that $f = gh$.

Assume that we are given the outputs of the PRF on $m = d\lceil \log p \rceil$ arbitrary inputs, for example the sequence $L_k([m])$. Then, by the multiplicative property of the Legendre symbol[3],

$$L_k([m]) = \mathcal{L}(g([m])) \oplus \mathcal{L}(h([m]))$$

Hence, the problem of finding the secret key $k \in \mathbb{F}_p^d$ reduces to a simple collision search, which we can summarize as follows:

1. Query the sequence $L_k([m])$ from the PRF. For each monic polynomial g of degree t, store the pair $(L_k([m]) \oplus \mathcal{L}(g([m])), g)$ in a table $\mathcal{T}$.

2. Sample monic polynomials h of degree $d - t$ until $(\mathcal{L}(h([m])), g) \in \mathcal{T}$, for some monic polynomial g of degree t. For each such g, recover a candidate key from the coefficients of gh. By Assumption 5.1, the number of candidate keys will be at most $\mathcal{O}(1)$.

For $t = \lfloor d/2 \rfloor$, this attack requires $\mathcal{O}(p^{\lfloor d/2 \rfloor} d \log p)$ bits of memory, its time complexity is dominated by $\mathcal{O}(p^{\lceil d/2 \rceil} d \log p)$ operations and the attack requires only $m = \mathcal{O}(d \log p)$ queries to the PRF (we assume that all Legendre symbols modulo p are pre-computed in $O(p)$ operations).

Using Van Oorschot-Wiener golden collision search [vW94], an improved time-memory trade-off can be obtained: given M bits of memory, the key can be recovered with a time-complexity of $\mathcal{O}(d \log p \sqrt{p^{3d/2}/M})$ Legendre symbol evaluations.

Even if the polynomial f does not have a factor of degree exactly $\lfloor d/2 \rfloor$, it might still have a factor of large degree $t < \lfloor d/2 \rfloor$. In this case, the same strategy results in an attack with time complexity $\mathcal{O}(p^{d-t} d \log p)$ and memory complexity $\mathcal{O}(p^t d \log p)$. This gives a trade-off between more efficient attacks on a smaller fraction of keys (when t is large) or less efficient attacks on a larger fraction of the keys (when t is small). This trade-off is illustrated in Figure 5.1. The figure shows the time-complexity of the attack for the desired fraction of attackable keys. The construction of Figure 5.1 is based on the following fact [Tao15]: the fraction of monic degree-d polynomials whose factorization has exactly c_i monic irreducible factors of degree i is $1/\prod_{i=1}^{d} c_i!\, i^{c_i}$ as $p \to \infty$. By summing these probabilities over all integer partitions of d that allow a $(t, d-t)$ split, we obtain the probability that a uniformly random key is weak.

[3] For convenience, we extend our notation for arithmetic L-sequences (Definition 5.3) to arbitrary functions on $\mathbb{F}_p$. In particular, $\mathcal{L}(g([m])) = (\mathcal{L}(g(0)), \ldots, \mathcal{L}(g(m-1)))$.

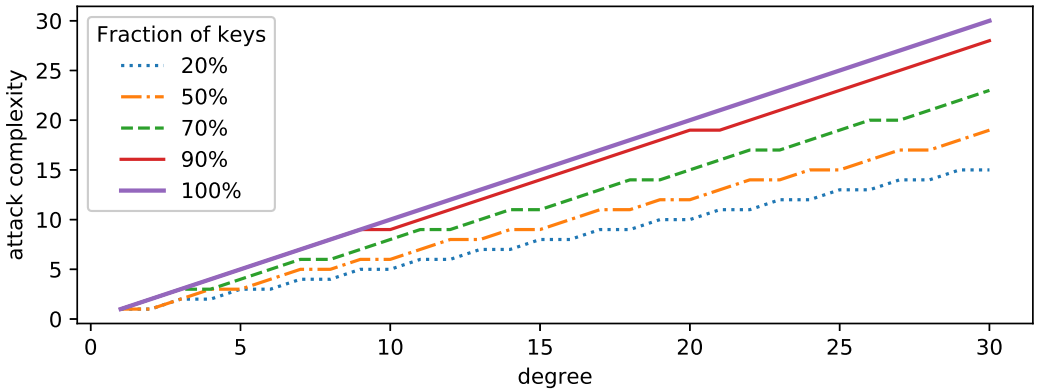


FIGURE 5.1: The complexity of the attack, measured as a power of p, as a function of the degree of f and the desired fraction of keys we want to attack.

We conclude that if the key is chosen uniformly at random, the higher-degree Legendre PRF has security only up to the birthday bound. To completely prevent this class of attacks, one can choose the key k such that the corresponding polynomial f is irreducible.

5.5.2 Reduction to the Unique k-XOR Problem

More generally, the secret polynomial could factor into k polynomials of degree roughly d/k. For example, if d is divisible by k and $f = \prod_{i=1}^{k} f_i$ with $\deg f_i = d/k$, we have

$$L_k([m]) = \bigoplus_{i=1}^{k} \mathcal{L}(f_i([m])).$$

That is, it suffices to find a solution to a variant of the k-XOR problem. Specifically, since each list has length $p^{d/k}$, a unique solution is expected. This makes Wagner's approach [Wag02] inapplicable, but some improvements over the attack in Subsection 5.5.1 are nevertheless possible.

In particular, for $k = 4$, the algorithm of Chose, Joux and Mitton [CJM02] leads to a time complexity $\tilde{\mathcal{O}}(p^{d/2})$ with only $\tilde{\mathcal{O}}(p^{d/4})$ memory. Corresponding time-memory trade-offs can also be obtained.

Finally, we mention that there exist asymptotically better quantum algorithms for the unique k-XOR problem. Bernstein *et al.* [Ber+13] give an $\widetilde{\mathcal{O}}(p^{0.3d})$ algorithm requiring $\widetilde{\mathcal{O}}(p^{0.2n})$ quantum-accessible quantum memory for $k = 4$. For any $k \geq 3$, Naya-Plasencia and Schrottenloher [NS20] give algorithms running in time $\widetilde{\mathcal{O}}(p^{\beta_k d})$ where $\beta_k = (k + \lceil k/5 \rceil)/(4k)$ using $\widetilde{\mathcal{O}}(p^{0.2n})$ quantum-accessible quantum memory. For $k = 3$, there is an algorithm using $\widetilde{\mathcal{O}}(p^{d/3})$ time and $\widetilde{\mathcal{O}}(p^{d/3})$ quantum-accessible *classical* memory.

5.6 Jacobi Symbol PRF

The Jacobi pseudorandom generator was proposed by Damgård [Dam90] as a variation to the Legendre PRG. As discussed by Damgård [Dam90, §5], it is potentially more efficient because it can be computed as the exclusive-or of several Legendre PRGs with a relatively small modulus. In addition, Damgård showed that if the

Legendre generator is weakly unpredictable, the Jacobi generator is strongly unpredictable. A generator is defined to be weakly unpredictable if, for all polynomials f, there exist only finitely many integers $m \geq 0$ such that the next output bit in a sequence of length m can be predicted with probability greater than $1 - 1/f(m)$. Similarly, the generator is strongly unpredictable if the probability of successful prediction exceeds $1/2 + 1/f(m)$ for only finitely many m. For a more formal definition, see [Dam90, Section 3] and references therein.

Whereas the unpredictability result of Damgård could be regarded as a positive result related to the security of the Jacobi PRF, it remains inconclusive concerning its concrete security. Indeed, strong unpredictability is a weaker property than PRF-security, and, in addition, it is only an asymptotic notion of security.

The cost of a key-recovery attack on the Jacobi PRF is at least the cost of attacking a Legendre PRF corresponding to a prime factor of the modulus. In fact, a chosen-plaintext key-recovery attack on the Jacobi PRF, which nearly attains this lower bound, is possible, and the Jacobi PRF offers little benefit over the Legendre PRF for most purposes.

To see this, let $n = \prod_{i=1}^{m} p_i$ with $p_1, \ldots, p_m$ distinct odd primes. We note that it may be assumed that the prime factors of n are distinct, since

$$\left(\frac{x+k}{n}\right) = \left(\frac{x+k}{\prod_{i=1}^{m} p_i^{e_i}}\right) = \prod_{\substack{i=1 \\ e_i \text{ odd}}}^{m} \left(\frac{x+k}{p_i}\right).$$

The attack is based on the following observation: let $\lambda_j = \prod_{\substack{i=1 \\ i \neq j}}^{m} p_i$ and denote the inverse of λ_j modulo p_j by λ'_j, then

$$\left(\frac{\lambda_j\, x + k}{n}\right) = \prod_{i=1}^{m} \left(\frac{\lambda_j\, x + k}{p_i}\right) = \left(\frac{\lambda_j}{p_j}\right) \left(\frac{k}{n/p_j}\right) \left(\frac{x + \lambda'_j\, k}{p_j}\right)$$

Hence, in the chosen-plaintext setting, the key-recovery attack on the Legendre PRF from Section 5.3 can be used to recover the key modulo p_j. The factor $\left(\frac{k}{n/p_j}\right)$ is not known to the attacker, but it is constant, so the cost of the attack is increased by a factor of at most two. Given the value of the key modulo for each prime factor of n, the Chinese remainder theorem yields the value of the key modulo n. Hence, key recovery for the Jacobi symbol costs at most $\mathcal{O}(mM^2 + \sum_{i=1}^{m} p_i \log^2 p_i / M^2)$ Legendre symbol evaluations. The same strategy applies to the higher-degree case and can also be combined with the attacks in Section 5.7 below. Note that a distinguishing attack on the Jacobi PRF reduces to a distinguishing attack on the Legendre PRF corresponding to the smallest prime factor of the modulus.

5.7 Attacks on the Power Residue PRF

The MPC protocol of Grassi *et al.* [Gra+16] for computing the Legendre PRF requires only three rounds of communication, which makes the Legendre PRF superior among the other PRF constructions investigated by Grassi *et al.* in terms of latency. However, since the Legendre PRF only produces one bit of output, it compares less favorably in terms of throughput than, e.g. MiMC [Alb+16], a block cipher that outputs full field elements.

To mitigate this limitation of the Legendre PRF, we can, as proposed by Damgård [Dam90], consider higher power residue symbols rather than quadratic residue symbols. If r divides $p-1$, the *r-th power residue symbol* of $x \in \mathbb{F}_p$ is defined as

$$\left(\frac{x}{p}\right)_r := x^{\frac{p-1}{r}} \bmod p.$$

Jointly computing r-th power residue symbols in the MPC setting can be done at essentially the same cost as computing Legendre symbols with the advantage that $\log r$ bit outputs are produced instead. Therefore, this modification can significantly increase the throughput of the Legendre PRF at essentially no cost – keeping in mind that r should not be too large since the corresponding power residue PRF might lose its security (e.g. $r = p-1$). In this Section we detail a key-recovery attack for the corresponding *r-th power residue PRF*, with time complexity $\mathcal{O}(p \log^2 p/(Mr \log^2 r))$, given $M \leq \sqrt{p}$ queries to the PRF.

5.7.1 Power Residue PRF

By generalising the Legendre function and the Legendre PRF to higher power residues, we obtain the following definitions:

Definition 5.4 (r-th power residue function)**.** *Let p be a prime so that $p \equiv 1 \pmod r$ for a certain integer $r > 0$ and let g be a generator of $(\mathbb{F}_p)^*$. We define the r-th power residue function $l^{(r)} : \mathbb{F}_p \to \mathbb{Z}_r$ as*

$$l^{(r)}(a) = \begin{cases} s \text{ if } a \not\equiv 0 \bmod p \text{ and } a/g^s \text{ is an } r\text{-th power mod } p \\ 0 \text{ if } a \equiv 0 \bmod p \end{cases}$$

Definition 5.5 (r-th power residue PRF)**.** *Let p be a prime congruent to 1 modulo r. The* power residue PRF *over $\mathbb{F}_p$ is a family of functions $L_k^{(r)} : \mathbb{F}_p \to \mathbb{Z}_r$ such that for each $k \in \mathbb{F}_p$,*

$$L_k^{(r)}(x) = \mathcal{L}^{(r)}(k+x).$$

5.7.2 Generalising our Attacks to the r-th Power Residue PRF

The attacks described in Section 5.3 and Section 5.4 do not use any properties of the Legendre symbol other than its multiplicativity. Therefore, they trivially generalize to any multiplicative function with a hidden shift, including the r-th power residue function.

Unlike the quadratic case, the r-th power residue function can take r distinct values, so it suffices to consider L-sequences of length $\log p/\log r$. It follows that a straightforward generalization of our attack to r-th power residue Legendre PRFs requires $\mathcal{O}(p \log^2 p/(M^2 \log^2 r))$ power residue symbol evaluations and $\mathcal{O}(M^2 \log r)$ memory. However, for large values of r, a better attack exists that we detail in the next Section.

5.7.3 Attacks for Large r

We first describe a simple attack on the linear r-th power residue Legendre PRF that requires $\mathcal{O}(p/r)$ power residue symbol evaluations. In the following, we denote the

subgroup of $(p-1)/r$-th roots of unity of $(\mathbb{F}_p)^*$ by $\mathbb{G}$. That is,

$$\mathbb{G} = \{x \in (\mathbb{F}_p)^* \mid x^{(p-1)/r} = 1\}.$$

Remark that $\mathbb{G}$ is generated by g^r, where g is any generator of $(\mathbb{F}_p)^*$.

By querying $L_k^{(r)}(0)$, the attacker immediately learns $\mathcal{L}^{(r)}(k)$, the power residue symbol of $k \in \mathbb{F}_p$. We observe that this single query already narrows down the set of possible values for k to at most $(p-1)/r$ elements of $\mathbb{F}_p$. Indeed, from Definition 5.4, k is contained in the coset $g^s\mathbb{G}$, where g is any generator of $(\mathbb{F}_p)^*$ and s is equal to $\mathcal{L}^{(r)}(k)$. Therefore, an attacker can go through all of these elements and check each candidate. Since, on average, only $\mathcal{O}(1)$ power residue symbols must be computed to check the validity of a candidate key, the attack requires $\mathcal{O}(p/r)$ power residue symbols evaluations. The attack requires a generator g, which can be precomputed in probabilistic subexponential time by factoring $p-1$.

We now explain a more general attack, similar to the table-based collision search from Subsection 5.3.1, that requires $\mathcal{O}(p \log^2 p/(Mr \log^2 r))$ power residue symbol evaluations and $\mathcal{O}(M \log r)$ memory. A speed-up of a factor r can be obtained by querying the PRF at more carefully chosen arithmetic L-sequences. By letting $m = \lceil \log p / \log r \rceil$ and $M < p/r$, the attack proceeds as follows:

1. For M/m distinct values $a \in \mathbb{G}$, store each pair $(L_k^{(r)}(a[m]), a)$ in a table $\mathcal{T}$. Furthermore, query the PRF to get the value $s = L_k^{(r)}(0)$.

2. Sample x uniformly at random from the coset $g^s\mathbb{G}$ until $(L_0^{(r)}(x + [m]), a) \in \mathcal{T}$ for some value a. For each entry $(L_0^{(r)}(x + [m]), a) \in \mathcal{T}$ corresponding to such collision, a candidate key is recovered as $\tilde{k} = xa$. Again, by (a variant of) Assumption 5.1, the number of such candidate keys will be at most $\mathcal{O}(1)$.

The first step of the above attack uses $M = m \cdot (M/m)$ queries to $L_k^{(r)}$ and needs $\mathcal{O}(M \log r)$ memory to store the table $\mathcal{T}$. The key k is found when, in the second step, the attacker samples an x such that k/x is one of the a-values stored in the table. On average, $|\mathbb{G}|/(M/m) = \mathcal{O}(pm/(Mr))$ iterations of the second step are required in order to find a candidate key. Since each iteration requires m power residue symbol computations to evaluate $L_0^{(r)}(x + [m])$, it follows that the total time-complexity of the attack consists of $\mathcal{O}(M)$ storage operations and $\mathcal{O}(pm^2/(Mr)) = \mathcal{O}(p \log^2 p/(Mr \log^2 r))$ power residue symbol evaluations.

5.8 Implementation Results

In this Section we will discuss our implementation of the attack from Subsection 5.3.3, and that we applied to the key recovery challenges proposed by the Ethereum foundation [Fei19b]. Using the attack from Section 5.3, we managed to solve three out of six challenges (including the test instance with a 40-bit prime): a summary of the instance parameters and the time and memory requirements of the attack is given in Table 5.2.

The source code of our implementation is publicly available at

`https://github.com/cryptolu/LegendrePRF`

We compiled our C++ implementation of the attack using Clang 6.0.0 and executed it on a Dell C6420 server with two Intel Xeon Gold 6132 CPUs clocked at 2.6

p	Security level[4] (bits)	Time (core-hours)	Memory/ thread (GiB)	Key
$2^{40}-87$	20	< 0.001	< 1	`4e2dea1f3c`
$2^{64}-59$	44	1.5	3	`90644c931a3fba5`
$2^{74}-35$	54	1500	3	`384f17db02976dcf63d`
$2^{84}-35$	64	2^{21}†	3	
$2^{100}-15$	80	2^{37}†	3	
$2^{148}-167$	128	2^{65}†	3	

TABLE 5.2: Parameters of the concrete challenges proposed by the Ethereum foundation [Fei19b]. For all instances, the first $M = 2^{20}$ consecutive PRF outputs were given. For the first three instances, the running time and peak memory usage is given, for the three hardest instances an estimation of time is provided (marked by †). All experiments were performed on a Dell C6420 server with two Intel Xeon Gold 6132 CPUs clocked at 2.6 GHz and 128 GB of RAM.

GHz (28 cores) and 128 GB of RAM (all the experiments presented in this Section were carried out using the HPC facilities of the University of Luxembourg [Var+14]). The optimizations described in Subsection 5.3.4 allow to significantly reduce the required memory and the number of evaluations of the Legendre symbol. As a result, the table lookups are the bottleneck in our implementation. On average, a single thread required $0.08\,\mu s$ to compute and check a single 64-bit sequence. As discussed below, we expect to compute $p/2^{28}$ sequences on average before the key is recovered. Hence, the required core time to solve a challenge with a prime p and 2^{20} bits of PRF output can be estimated as $p/2^{28} \times 0.08\,\mu s$. The required memory is 1 GB per server and an additional 3 GB per thread. The parameters can be modified to reduce the memory without significantly decreasing the performance.

For the first three instances, we successfully recovered the secret key of the PRF in a timespan close to our estimation. The corresponding keys are given in Table 5.2. The third instance was solved in under two hours using a cluster of 40 nodes with the described configuration. The full attack consisted of two main consecutive steps, detailed in the following Sections: *Processing the PRF output* and *Random Sampling*.

5.8.1 Processing the PRF Output

As a first step, we compute the set $\mathcal{T}$ consisting of all arithmetic sequences extracted from the sequence $L_k([2^{20}])$ given in the challenge. We chose to store sequences of length $m = 64$ since this length provides an acceptable rate of false positives and enables processing sequences as 64-bit words efficiently. As a result, the set $\mathcal{T}$ contains approximately $M^2/(2m^2) = 2^{27}$ of such words-sequences.

A straightforward way to implement a set is by using a hash table, which has a constant amortized time-complexity for membership testing. However, this constant time may be pretty large in practice, especially in the case of large tables, and, in fact, random memory accesses often represents the main bottleneck. In our case, the set $\mathcal{T}$ is never modified after its creation. To exploit this fact, we sort the elements of $\mathcal{T}$, and we store them in an array. We then compute membership queries in batches: first, we collect a large number of membership queries, and we sort them, then, we scan through the two sorted arrays checking for collisions. We note that the most

[4]Expected security level (conservative estimate) prior to this work.

time-consuming aspect of this approach is represented by the sorting step of each batch of membership queries. The described set $\mathcal{T}$ contains 2^{27} 64-bit words, and the corresponding sorted array requires 1GiB of memory, while an extra 1GiB of memory is used to store information required for the key recovery. We note that the set $\mathcal{T}$ and the extra information are shared among all threads used to parallelize the workload of the next step.

5.8.2 Random Sampling

The second and main step of the attack consists of sampling sequences $L_0(c + [m])$ for randomly chosen c, and checking if they collide with an entry of $\mathcal{T}$. Note that the reversed sequence $L_0(c + [m])$ is checked instead if it is lexicographically smaller.

For a uniformly chosen $c \in \mathbb{F}_p$, we compute a long sequence $L_0(c + [t])$ and we extract a large amount of m-bit sequences from it. More precisely, for all $b \in \{1, 2, \ldots, 2^8\}$ and $a \in \{0, 1, \ldots, t - 1 - b(m-1)\}$, we extract $L_0(c + a + b[m])$. The upper bound for b is chosen as 2^8 since it is enough to make the time spent computing Legendre symbols negligible. Furthermore, all these sequences can be computed on the fly by storing only the last sequence per pair (b, a). Indeed, for a large enough $i \in \mathbb{Z}$, after expanding the computed sequence $L_0(c + [i-1])$ by one Legendre symbol $L_0(c + i)$ we obtain a new sequence $L_0(c + i - b(m-1) + b[m])$ for each b. In other words, we obtain 2^8 sequences from each single consequent Legendre symbol computation.

As described above, the computed sequences are accumulated and checked in batches for a collision with the set $\mathcal{T}$. Each batch is sorted using base-2^8 radix sort, and collisions are checked using a linear scan through the sorted batch and the sorted array of $\mathcal{T}$. In the case of a collision, a key candidate is recovered and checked against extra bits from the given PRF output.

We note that this step can be efficiently parallelized, since each thread could start with a uniformly random $a \in \mathbb{F}_p$ and proceed as described above. After a predetermined amount of steps, a new value for a can be chosen to ensure sufficiently uniform coverage of the possible offsets of the sequences.

5.8.3 Concurrent work

Some days after this work first appeared on IACR ePrint Archive, Kaluđerović *et al.* [KKK20] solved the next Legendre PRF challenge. Their attack uses similar ideas to our attack but with an improved complexity of $\mathcal{O}(M^2 / \log p + p \log p \log\log p / M^2)$ operations on a machine with word size $\Theta(\log p)$.

5.9 Conclusions

In this Chapter we presented multiple attacks on the Legendre PRF and its generalizations.

Regarding the Linear Legendre PRF, we proposed an attack of particular interest in the low-data setting. Specifically, given $M \leq \sqrt[4]{p}$ queries, our attack recovers the key using $\mathcal{O}(p \log^2 p / M^2)$ Legendre symbol evaluations, and we demonstrated its practical relevance by solving the first two Legendre PRF challenges set out by the Ethereum foundation [Fei19b].

We have then shown that the techniques used to attack the linear Legendre PRF yield improved attacks on the higher-degree Legendre PRF variant as well. Further attacks on the higher-degree case were given by showing a large class of weak keys

that can be recovered from the PRF output using $\mathcal{O}(p^{\lceil d/2 \rceil} d \log p)$ operations and $\mathcal{O}(p^{\lfloor d/2 \rfloor} d \log p)$ bits of memory. Further improvements to the memory usage, based on a reduction to the unique k-XOR problem, were also discussed. These weak-key attacks can be prevented by choosing a key corresponding to a monic irreducible polynomial.

In addition to the above, we provided the first security analysis of the Jacobi and power-residue generalizations of the Legendre PRF. These extensions were first suggested – for the Legendre pseudorandom generator – at CRYPTO 1988 by Damgård [Dam90]. We demonstrated that the key of a Jacobi PRF can be recovered with time-complexity proportional to the time-complexity of key-recovery on the Legendre PRF for each of the prime factors of the modulus separately a result which eliminates the potential efficiency benefits offered by Jacobi symbols.

We then focused on the r-th Power Residue PRF: the low-data attack for the Linear Legendre PRF equally applies in this setting, but we provide an additional attack that performs better for large power residue symbols. Specifically, for r-th power residue symbols and given $M \leq \sqrt{p}$ queries, our key-recovery attack requires $\mathcal{O}(p \log^2 p / (rM \log^2 r))$ power residue evaluations and $\mathcal{O}(M)$ memory.

Part III

Public Key Cryptography

This part presents my research on public-key primitives based on elliptic curves.

In Chapter 6, I will describe a technique to backdoor a prime factor p of a composite odd integer N, so that an attacker knowing a possibly secret factor base $\mathcal{B}$, can efficiently retrieve p from N.

Such a method builds upon Complex Multiplication theory for elliptic curves in order to generate primes p associated to $\mathcal{B}$-smooth order elliptic curves over $\mathbb{F}_p$. When such primes p divide an integer N, the latter can be efficiently factored using a generalization of Lenstra's Factorization Method [Len87] over rings bigger than $\mathbb{Z}_N$, and with no knowledge other than N and $\mathcal{B}$.

I will then formalize *semiprimality certificates* that, based on a result by Goldwasser and Kilian, allow to prove semiprimality of an integer with no need to reveal any of its factors. I will show how the prime generation procedure can efficiently produce semiprimality certificates, ultimately allowing to sketch a multi-party distributed protocol to generate semiprimes with unknown factorization, particularly relevant in the setting of distributed RSA modulus generation.

I will discuss implementations of all proposed protocols and address the security of semiprimality certificates by showing that semiprimes generated within these methods result at least as secure as random semiprimes of the same size.

In Chapter 7, I will report how, together with my co-author, we successfully broke Microsoft's $IKEp182 challenge [Mic21b]. The latter is a reduced-parameters instance of SIKE (Supersingular Isogeny Key Encapsulation) [Jao+20], an alternative candidate of the ongoing 3$^{\text{rd}}$ round of NIST Post-Quantum Cryptography Standardization process [Nat22], and its security relies on the hardness to find a (smooth-degree) isogeny between two, a *starting* and a *final*, isogenous curves.

Through the Chapter we will detail a meet-in-the-middle attack, improved with multiple SIKE-specific optimizations, to recover a secret 2^e-degree isogeny. On top of an efficient isogeny-tree exploration based on optimal strategies, we will take advantage of a 2-bit leak from the knowledge of the secret isogeny final curve, and an extra 1-bit space-time complexity reduction given by the fact that the starting curve is defined, by specification, over a prime order field.

These, combined with a sort & merge approach and ad-hoc elements encoding that enabled us to use cheaper and more readily available disk-based space in place of RAM, allowed us to break, using the University of Luxembourg High-Performance Computing cluster [Var+14], the $IKEp182 challenge in less than 10 core-years and by using 256TiB of high-performance network storage, a requirement that can be further reduced to 70TiB.

Chapter 6

Backdooring and Distributed Generation of Semiprimes

6.1 Introduction

In this Chapter, we will detail a technique to backdoor a prime factor of an odd composite integer N so that third parties knowing some (secret) auxiliary information, i.e., a factor base, can efficiently retrieve it from N.

More precisely, we will use the *Complex Multiplication* theory to build primes p and elliptic curves over $\mathbb{F}_p$, such that the orders of the latter are $\mathcal{B}$-*smooth* (that is, factor as products of elements in $\mathcal{B}$), and can be explicitly represented as a function of the field characteristic, i.e. $\#E(\mathbb{F}_p) = p + 1 \pm a$ where $4p = a^2 + |D|b^2$ with D a negative *discriminant*. over a ring containing

In the following Sections, we will describe a generation procedure for such primes p so that, with no knowledge of any of the factors of N, we will be able to construct a curve equation $E(\mathbb{Z}_N)$ along with multiple random points P on it, so that the order of $E(\mathbb{F}_p)$ (the modulo p component of $E(\mathbb{Z}_N)$) is $\mathcal{B}$-smooth for a certain factor base $\mathcal{B}$. This will ultimately allow us to efficiently factor N by computing in projective coordinates some multiple $[\prod_{p_i \in \mathcal{B}} p_i] \cdot P$ of the point P, similarly as done in Lenstra's Elliptic-Curve factorization Method (ECM) [Len87]. While, in ECM, random curve equations $E(\mathbb{Z}_N)$ and points $P \in E(\mathbb{Z}_N)$ are generated until a curve of smooth order is found, in our construction, we will be able to deterministically construct from N a $\mathcal{B}$-smooth order elliptic curve in short Weierstrass form defined over a ring containing $\mathbb{Z}_N$, together with multiple points on it.

Although different backdooring techniques exist in literature within our attack model[1], our methods show that large semiprimes N, employed by the RSA encryption scheme [RSA78], can be maliciously generated to be still vulnerable to a generalization of Lenstra's ECM, an attack that is usually not considered of interest for cryptographically sized N due to its negligible success probability. Furthermore, the insights provided by our prime generation procedure will ultimately allow us to sketch a distributed protocol to generate semiprimes.

Recent protocols for the multi-party generation of semiprimes [Che+21; Del+21; Che+20] build on the seminal work of Boneh and Franklin [BF97], by extending it to different security assumptions and adding several algorithmic optimizations. Informally, in these protocols, parties jointly generate some (random) candidate primes p, q, securely multiply them as $N = p \cdot q$ and run a distributed statistical semiprimality test until they are confident enough that N is a semiprime.

In contrast to these, our sketched multi-party protocol outputs integers which are guaranteed to be a product of two unknown primes (and thus there is no need to run a semiprimality test such as the one detailed in [BF97]), thanks to *semiprimality certificates* that can be publicly verified.

Semiprimality certificates are based on a generalization of a theorem by Goldwasser and Kilian [GK86], which provides a sufficient condition for an integer N to have at most m distinct prime factors. Such a condition requires the existence of an elliptic curve over $\mathbb{Z}_N$ so that some points on it have particular small prime order. It follows that when we require an integer N to be the product of two distinct primes, as happens for RSA moduli, such result can be used to ensure semiprimality for N in case we find (or we construct) a semiprimality certificate for it, with the latter consisting in an elliptic curve over $\mathbb{Z}_N$ and some points on it satisfying the theorem assumptions. Remarkably, such certificates prove semiprimality of N with no need to publish nor know any of its factors.

Our semiprime generation protocol uses our prime backdooring procedure to construct elliptic curves with (partially random) orders that satisfy the assumptions of the Goldwasser-Kilian generalized criterion. We then retrieve N from this information, and we test if all conditions of the theorem match: when this happens, N is

[1] A standard one consists in fixing half of the bits of p to a certain secret value c and then use Coppersmith's method [Cop96] to efficiently find a small root of the polynomial $f(x) = 2^{\log_2(p)/2}x + c$ modulo a divisor of $N = p \cdot q$.

a semiprime *by construction* and its semiprimality can be publicly verified from the certificate.

We can then apply standard MPC arithmetic to perform all the computations involved (mainly multiplications and additions, but also scalar-point multiplications that we address with an ad-hoc technique) to make this protocol distributed, and we show its correctness and feasibility by implementing it using state-of-the-art MPC libraries.

Lastly, we analyze if and how knowledge of a semiprimality certificate for N generated within our framework may reduce its bit-security with respect to factorization. We investigate some possible attacks based on generic discrete-logarithm finding algorithms in groups of unknown order, and attack variants that require just a few operations to compute a factor of N if any multiple of the order of the curve provided in the certificate is known. In all cases, semiprimality certificates generated according to our protocol are easier to attack using generic factorization algorithms, and thus we don't expect a decrease in terms of bit-security unless other more specific attacks are found.

We were, however, able to factor a 3599-bits semiprime N, generated by Don Reble [Don05] in 2005 and which remained unfactored until now. Such semiprime comes with a semiprimality certificate that, contrary to ours, includes an extra divisor of the full curve order that makes it vulnerable to one of our attack variants that factors N in just a few operations.

Related Works In a later stage of the (independent) development of the results reported in this Chapter, we found out that Aikawa *et al.* in [ANS19] use similar observations to factor integers N where one of its primes is of the form $4p = 1 + |D|b^2$. These primes, indeed, correspond to anomalous elliptic curves modulo p for which a multiple of the order, i.e., N, is known, thus allowing an attacker to factor N with just 1 scalar-point multiplication. Aikawa *et al.* work extends the applicability of the methods described by Shirase [Shi17], whose manuscript, in turn, is based on Cheng's [Che02a; Che02b] ideas, to a bigger class of *discriminants* D.

In addition to this, Aikawa *et al.* generalize their factoring method for anomalous curves to allow p be of the form $4p = a^2 + |D|b^2$, but requiring associated curves to have smooth order rather than a multiple of N. In contrast to our results, however, they do not provide an actual method to construct such vulnerable elliptic curves efficiently, and the probability to randomly hit any of them exponentially decreases with the size of p. Furthermore, their method works in multiple quotients of the ring $\mathbb{Z}_N[x]$ to carry out arithmetic operations on the constructed curve, while we use a more efficient approach based on XZ-arithmetic.

On a more practical side, Sedlacek *et al.* in [Sed+19] generated millions of RSA keys using different software libraries and hardware and tested them against Cheng's and Shirase's attacks, looking for prime factors p of the form $4p = 1 + |D|b^2$.

As regards semiprimality certificates, to our knowledge, Don Reble, with his short online post [Don05], was the first one to publicly propose the idea of proving semiprimality of an odd integer N, that he calls *interesting semiprimes*, using the Goldwasser-Kilian Theorem. Except for another similar certificate generated by Broadhurst [Dav05] soon after, we did not find any formal treatment of these concepts.

6.1.1 Outline

In Section 6.2 we provide the necessary theoretical background to characterize orders of some elliptic curves in terms of the field characteristic on which are defined, while

in Section 6.3 we provide the first sketch of our prime generation procedure. In Section 6.4 we address some technicalities related to the practical implementation of our idea, which we formalize in Section 6.5 and Section 6.6, where we detail how it is possible to retrieve a factor of a backdoored integer. In Section 6.7 we discuss implementations of both our prime generation procedure and factorization attack, and we detail a full example on a 1024-bits modulus. In Section 6.8 we formalize the concept of a semiprimality certificate for an integer N by generalizing the Goldwasser-Kilian Theorem, and we show how they can be efficiently computed by properly generating the prime factors of N. In Section 6.9 we propose a distributed protocol for generating certifiable semiprimes with unknown factorization, and we discuss how we overcome some practical issues to implement it in practice. Lastly, in Section 6.10, we address the security of semiprimality certificates generated according to our protocol, and we describe how we factored a public 1084-digits semiprime.

6.2 Preliminaries

6.2.1 Curves of Prescribed Order

Our construction relies on the theory of Complex Multiplication (CM) to build elliptic curves of a prescribed order. An elliptic curve E is said to have complex multiplication, if its endomorphism ring $End(E)$ is strictly larger than $\mathbb{Z}$[2]: this is always the case for elliptic curves defined over finite fields, where the endomorphism ring is isomorphic to an order in either a quaternion algebra (in this case, the curve is said to be *supersingular*) or an imaginary quadratic field (in this case, instead, the curve is said to be *ordinary*) [Sil09, V - Theorem 3.1].

A field K is said *quadratic field* if $[K : \mathbb{Q}] = 2$, and discriminants of quadratic fields are said *fundamental*. An integer $d \in \mathbb{Z}$ is a fundamental discriminant if it satisfies either $d \equiv 1 \pmod 4$ and d is square-free, or $d = 4D$ with D square-free and $D \equiv 2, 3 \pmod 4$.

Of interest for this Chapter are ordinary elliptic curves whose endomorphism rings are isomorphic to orders in an imaginary quadratic field.

Definition 6.1 (Order)**.** *For a quadratic field K, let $\mathcal{O}_K$ denote the ring of integers of K. A subring $\mathcal{O} \subset \mathcal{O}_K$ is said to be an order if it is a free $\mathbb{Z}$-module of rank 2 containing an integral basis of K.*

In particular, given a quadratic field K of fundamental discriminant d, we will denote it as $K = \mathbb{Q}(\sqrt{d})$, and an integral basis for it is given by $(1, \omega)$ with $\omega = \frac{d+\sqrt{d}}{2}$, thus $\mathcal{O}_K = \mathbb{Z}[\omega]$.

Every order of a quadratic field is associated to a *discriminant*, characterized by its residuosity modulo 4.

Definition 6.2 (Discriminant)**.** *A $D \in \mathbb{Z}$ is said to be a* discriminant *if D is not a perfect square and $D \equiv 0, 1 \pmod 4$.*

Fundamental discriminants of quadratic fields and discriminants of orders are related by the following result.

Proposition 6.1 ([Coh00, Proposition 5.1.3])**.** *If K is a quadratic field of fundamental discriminant d, then every order $\mathcal{O}$ of K has discriminant $D = df^2$, where $f \in \mathbb{N}$ is the* conductor *of $\mathcal{O}$. Conversely, if D is a discriminant, then D can be written*

[2]The multiplication by $[n] : E \to E$ map which sends $P \mapsto nP$, is an endomorphism of E for any $n \in \mathbb{Z}$.

uniquely as $D = df^2$, with d a a fundamental discriminant, and there exists a unique order $\mathcal{O}$ of $\mathbb{Q}(\sqrt{d})$ of discriminant D.

The following result from CM-theory, also known as *CM Algorithm*, relates specific short Weierstrass curve equations over $\mathbb{F}_p$ to their cardinality, thus providing a practical method for computing elliptic curves over finite fields of a given order.

Theorem 6.1 ([Bro06, Theorem 3.6])**.** *Let D be a square-free negative discriminant not equal to $-3, -4$, and let p be an odd prime so that $4p = a^2 + |D|b^2$ for certain $a, b \in \mathbb{Z}$. Then, the elliptic curve*

$$E(\mathbb{F}_p) \; : \; y^2 = x^3 - 3kc^2x + 2kc^3$$

where

- *j is a root of the Hilbert Class Polynomial $H_D(x) \in \mathbb{Z}[x]$ modulo p,*
- *$k = \frac{j}{j-1728}$,*
- *c is a random non-zero element in $\mathbb{F}_p$*

has either $p + 1 + a$ or $p + 1 - a$ points, depending on the residuosity of c modulo p, and j-invariant equal to j.

Here the Hilbert Class Polynomial $H_D(x)$ is a polynomial with roots exactly the j-invariants of elliptic curves over $\mathbb{C}$ whose endomorphism ring equals an order with discriminant D in an imaginary quadratic field. It can be proven that $H_D(x)$ is irreducible, is defined over $\mathbb{Z}[x]$, and there exist efficient algorithms to compute it: we refer to [Bro06, Section 3.3] for more technical details on this.

We recall that given an elliptic curve $E : y^2 = x^3 + ax + b$, a (quadratic) twist of E over $\mathbb{F}_p$ is given by $\tilde{E} : y^2 = x^3 + c^2ax + c^3b$, where c is a quadratic non-residue modulo p. If $E(\mathbb{F}_p)$ has trace of Frobenius t, then $\tilde{E}(\mathbb{F}_p)$ has trace $-t$ for any chosen quadratic non-residue c, namely $\#E(\mathbb{F}_p) = p + 1 - t$ and $\#\tilde{E}(\mathbb{F}_p) = p + 1 + t$. If instead, c is a square modulo p, then E is isomorphic to $\tilde{E}$ through the change of variable $(x', y') = (x/c, \sqrt{c}y/c^2)$ and hence $\#E(\mathbb{F}_p) = \#\tilde{E}(\mathbb{F}_p)$. It follows that in the statement of Theorem 6.1, the value c chosen determines one of these two different curve twists.

It is possible to characterise the cases $D = -3, -4$ as well, which correspond to curves with j-invariants $j = 0, 1728$, respectively.

Theorem 6.2 ([Bro06, Theorem 3.6])**.** *Let $D \in \{-3, -4\}$ and let p be an odd prime so that $4p = a^2 + |D|b^2$ for some $a, b \in \mathbb{Z}$. Then, the elliptic curve*

$$E(\mathbb{F}_p) : \begin{cases} y^2 = x^3 + c^3 & \text{if } D = -3 \\ y^2 = x^3 + c^2x & \text{if } D = -4 \end{cases}$$

with c a random non-zero element in $\mathbb{F}_p$, has either $p + 1 + a$ or $p + 1 - a$ points, depending on the residuosity of c modulo p, and j-invariant equal to 0 if $D = -3$, or 1728 if $D = -4$.

In the case $D = -4$, we have a more general result that characterize entirely the number of points of the curve $E(\mathbb{F}_p) : y^2 = x^3 - kx$, regardless of $-k$ being a square, as required instead by Theorem 6.2.

We recall that, by Fermat's Theorem on *Sums of Two Squares* [Dud78, Lemma 18.4], an odd prime p can be written as $p = a^2 + b^2$ if and only if $p \equiv 1 \pmod 4$. It

Algorithm 2 Modified Cornacchia Algorithm ([Coh00, Algorithm 1.5.3])

Input: an odd prime p, a negative discriminant D with $|D| < 4p$.
Output: if exists, an integer solution to $x^2 + |D|y^2 = 4p$, otherwise `None`.

1: Compute $k = \left(\frac{D}{p}\right)$. If $k = -1$, return `None`.
2: Compute, e.g. using Shank's algorithm, an x_0 so that $x_0^2 \equiv D \pmod{p}$ and $0 \leq x_0 < p$. If $x_0 \not\equiv D \pmod{2}$, set $x_0 = p - x_0$. Set $a = 2p$, $b = x_0$, $l = \lfloor 2\sqrt{p} \rfloor$.
3: If $b > l$, set $r = a \bmod b$, $a = b$, $b = r$. Go to Step 3.
4: If $|D| \nmid (4p - b^2)$ or $c = (4p - b^2)/|D|$ is not a perfect square, return `None`. Otherwise, return $(x, y) = (b, \sqrt{c})$.

follows that any decomposition of a prime p into a sum of two squares $a^2 + b^2$, is in bijection with a decomposition of $4p$ as $(2a)^2 + |D| \cdot b^2$ with $D = -4$: the following result, in fact, generalizes Theorem 6.2 for the case $D = -4$.

Theorem 6.3 ([Was08, Theorem 4.23])**.** *For an odd prime p, let $k \not\equiv 0 \pmod{p}$ and consider the elliptic curve $E(\mathbb{F}_p) : y^2 = x^3 - kx$. Then*

1. *If $p \equiv 3 \pmod{4}$, E is supersingular and $\#E(\mathbb{F}_p) = p + 1$.*
2. *If $p \equiv 1 \pmod{4}$, let $p = a^2 + b^2$ where a, b are integers with b even and $a + b \equiv 1 \pmod{4}$. Then*

$$\#E(\mathbb{F}_p) = \begin{cases} p + 1 - 2a & \text{if } k \text{ is a fourth power modulo } p \\ p + 1 + 2a & \text{if } k \text{ is a square but not a fourth power modulo } p \\ p + 1 \pm 2b & \text{if } k \text{ is not a square modulo } p \end{cases}$$

6.2.2 Cornacchia's Algorithm

The above characterizations for the orders of elliptic curves over $\mathbb{F}_p$ require the knowledge of a specific decomposition for $4p$, namely $4p = a^2 + |D|b^2$, with D a negative discriminant and $a, b \in \mathbb{Z}$.

In 1908, Giuseppe Cornacchia proposed an algorithm to solve the Diophantine equation $x^2 + dy^2 = p$ with p prime and $0 < d < p$: a proof of correctness of his algorithm can be found, for example, in [Bas04]. It is possible to slightly modify the original Cornacchia's algorithm to find a solution to our case of interest as well, namely finding solutions to the equation $4p = x^2 + |D|y^2$ for a negative D so that $D \equiv 0, 1 \pmod{4}$. For completeness, we report in Algorithm 2 such modified version of his algorithm, taken from [Coh00, Algorithm 1.5.3].

In the following Sections, we will often refer to a pair (a, b) satisfying $4p = a^2 + |D|b^2$ as the *Cornacchia decomposition* of p for the discriminant D.

6.3 The Idea

Given a square-free integer N that decomposes as a product of primes of approximately the same size as $N = p_1 \cdot \ldots \cdot p_n$, and an elliptic curve E defined over $\mathbb{Z}_N$, we clearly have that $E(\mathbb{Z}_N) = E(\mathbb{F}_{p_1}) \times \ldots \times E(\mathbb{F}_{p_n})$ as groups. We are interested in generating primes p so that whenever they divide a square-free integer N as above, we can explicitly write a curve equation E over $\mathbb{Z}_N$ with no need to know any factor of N, and so that $\#E(\mathbb{F}_p)$ is smooth or factors in a chosen factor base $\mathcal{B}$. We will do so using the results and the explicit curve equations provided by Theorem 6.1, Theorem 6.2, Theorem 6.3.

It will then follow that, if we are able to get a (non-trivial) point P on one of such $E(\mathbb{Z}_N)$, by computing $\left(\prod_{p_i \in \mathcal{B}} p_i^{\ell}\right) \cdot P$ for some fixed exponent l, we might reveal a factor of N, similarly as happens in Lenstra's elliptic-curve factorization method [Len87], with probability depending on the size of N and its prime factors. Such probability is high if N is a cryptographically-sized semiprime, a condition we will assume from this point on.

Taking into account all the different sub-cases of the above results, we can sketch a procedure to generate such primes p as follows:

1. Set a factor base $\mathcal{B}$ and a discriminant D.
2. Generate a random integer t (i.e., a candidate for the order of $E(\mathbb{F}_p)$) which factors completely in $\mathcal{B}$.
3. Use Algorithm 2 to find a pair (a, b) so that' $4t = a^2 + |D|b^2$. If no solution exists go to step Step 2.
4. Check if $p = \frac{(a \pm 2)^2 + |D|b^2}{4}$ is prime. If yes, output p. If p is not prime and $D \neq -4$, go to Step 2.
5. If $D = -4$ check if $p = \frac{|D|a^2 + (b \pm 2)^2}{4}$ is prime. If yes, output p, otherwise go to Step 2.

We note that Step 4 is justified from the fact that, from Theorem 6.1, Theorem 6.2, we require the order of the curve $E(\mathbb{F}_p)$ to be $t = p + 1 \pm a$: thus, if $4t = a^2 + |D|b^2$, then $4p = (a \pm 2)^2 + |D|b^2$. Similarly, Step 5 addresses the simmetry induced for the case $D = -4$, where for a prime $p = a^2 + b^2$ we can express $4p$ as either $(2a)^2 + |D| \cdot b^2$ or $|D| \cdot a^2 + (2b)^2$.

Although this procedure looks quite simple, it has some not trivial aspects that need to be addressed when implementing it in practice. In particular, the probability that a random $\mathcal{B}$-smooth integer t so that $4t$ admits a Cornacchia decomposition should not be negligible, and such decomposition should allow generating many candidate primes. Furthermore, when we attack an integer N that has at least one of its factors generated as above, to write a curve equation to work with, we should be able to extract roots of the Hilbert Class Polynomial modulo N, and once we compute the full curve equation $E(\mathbb{Z}_N)$, we should be able to pick, for any choice of the curve twist, random points over it. All these aspects will be addressed in the next Section.

6.4 Addressing Technicalities

6.4.1 Cornacchia's Decompositions for Random Curve Orders

From the *Sum of Two Squares Theorem* [Dud78, Theorem 18.1], we know that an integer can be written as the sum of two squares if it is not divisible by any factor p_i^k, with p_i prime, $p_i \equiv 3 \pmod 4$ and k odd. For the case $D = -4$, this theorem provides an easy condition to generate random curve candidate orders t, since if $t = a^2 + b^2$ then $4t = (2a)^2 + |D|b^2$. Unfortunately, this does not generalize straightforwardly to other values of D, and elements in the factor base $\mathcal{B}$ needs to be carefully chosen if we wish to decompose with non-negligible probability a random $\mathcal{B}$-smooth t using Algorithm 2.

We will exploit the following observation to generate a factor base that will allow us to efficiently generate curve candidate orders along with their Cornacchia's decompositions.

Observation 6.1. *If $p_1, \ldots, p_n$ are positive integers that split in $\mathbb{Z}[\sqrt{D}]$ as $p_i = \pi_i\bar{\pi}_i$, then $p_1 \cdot \ldots \cdot p_n$ splits as $(\pi_1 \cdot \ldots \cdot \pi_n) \cdot (\bar{\pi}_1 \cdot \ldots \cdot \bar{\pi}_n) \doteq \pi \cdot \bar{\pi}$.*

Its main insight is that, when we multiply by its conjugate a given $\pi_i = a + b\sqrt{D} \in \mathbb{Z}[\sqrt{D}]$, we get $p_i = \pi_i \cdot \bar{\pi}_i = a^2 + |D|b^2 \in \mathbb{Z}$, which automatically ensures a Cornacchia decomposition for p_i as $4p_i = (2a)^2 + |D|(2b)^2$.

In fact, we can efficiently compute Cornacchia's decompositions for random $\mathcal{B}$-smooth values, without running Algorithm 2 every time. To show this, we define our factor base $\mathcal{B}$ to contain tuples $(p_i, \pi_i) \in \mathbb{Z} \times \mathbb{Z}[\sqrt{D}]$ so that $p_i = \pi_i \cdot \bar{\pi}_i$. If in Step 2 of our sketched prime generation procedure we compute in parallel the products

$$\prod_j p_{i_j} = a^2 + |D|b^2 \,, \qquad \prod_j \pi_{i_j} = a + b\sqrt{D}$$

from the two coordinates of the latter, we immediately get a solution (a, b) to $\prod_j p_{i_j} = x^2 + |D|y^2$.

In our case of interest, however, we look for a $\mathcal{B}$-smooth value t constrained to a Cornacchia decomposition for $4p$: by expanding their relations, if $p = a^2 + |D|b^2$, then $4p = (2a)^2 + |D|(2b)^2$, and from $t = p + 1 \pm 2a$ we then have $t = (a \pm 1)^2 + |D| \cdot b^2$. It follows, in turn, that if the t generated in Step 2 are of the form $t = a^2 + |D|b^2$, the expression for p in Step 4 can be simplified by checking primality of $p = (a \pm 1)^2 + |D|b^2$ (and $p = |D| \cdot a^2 + (b \pm 1)^2$ in Step 5). Since such p, t values are constrained by $t = p + 1 \pm 2a$ or $t = p + 1 \pm 2b$, for p to be prime we necessarily require t to be even.

To address this, we include in the factor base a pair $(e, v) \in \mathbb{Z} \times \mathbb{Z}[\sqrt{D}]$ where e is even and decomposes in $\mathbb{Z}[\sqrt{D}]$ as $v \cdot \bar{v}$. We can set

$$e = (D \bmod 2) + |D| \,, \qquad v = (D \bmod 2) + \sqrt{D}$$

that is, e is equal to $1 + |D|$ if D is odd or to $|D|$ otherwise.

We note that, regardless of the parity of D, when D is negative $e \equiv 0 \pmod 4$ and the minimum value for such e is 4: hence, for all negative discriminants D, by including such tuple in the factor base $\mathcal{B}$, and by requiring a random $\mathcal{B}$-smooth value $\tilde{t}$ to always have (at least) e as a factor, we can efficiently compute, as above, a Cornacchia's decomposition for the generated curve orders candidates.

In order to be able to exploit all the 4 orders characterization provided by Theorem 6.3, we need to write p as a sum of two squares. If t is generated as above as $t = a^2 + |D|b^2$, then $t = a^2 + (2b)^2 = a^2 + \tilde{b}^2$, and p would then be of the form $p = (a \pm 1)^2 + \tilde{b}^2$ or $p = a^2 + (\tilde{b} \pm 1)^2$. Thus, in the case $D = -4$ we proceed as above, ensuring t to be even, but once $t = a^2 + |D|b^2$ is generated, we *absorb* $\sqrt{|D|} = 2$ into the value $\tilde{b} = 2b$ and we check p accordingly.

6.4.2 Roots of the Hilbert Class Polynomial $H_D(x)$ modulo N

When $D \neq -3, -4$, we need to compute a root of the Hilbert Class Polynomial $H_D(x)$ modulo p, with $p|N$, in order to compute the j-invariant of a curve with desired order (Theorem 6.1). Since, at this point, no factor of N is known, we lift the problem to $\mathbb{Z}_N$ (using the correspondence given by the Chinese Remainder Theorem), and we compute $H_D(x)$ modulo N instead.

We have different possibilities for computing $H_D(x) \in \mathbb{Z}_N[x]$: it is well known that $H_D(x)$ can be expressed as an irreducible polynomial in $\mathbb{Z}[x]$ and we can thus compute it in $\mathbb{Z}[x]$ first, using for example, the techniques outlined in [Bel+08], and then reduce modulo N its coefficients. Unfortunately, this approach quickly

becomes infeasible as $|D|$ increases, since the sizes of the coefficients of $H_D(x) \in \mathbb{Z}[x]$ would grow exponentially. Another possibility consists in computing $H_D(x)$ directly with the coefficients reduced modulo N: Sutherland in [Sut11] describes a method to compute $H_D(x)$ mod N for any integer N, which runs in $O(|D|^{1+\epsilon})$ time and requires $O(|D|^{1/2+\epsilon} \log N)$ space.

Once $H_D(x) \in \mathbb{Z}_N[x]$ is computed, it is generally hard to find a root modulo N without knowing at least a factor of N.

Clearly, when the Hilbert Class Polynomial $H_D(x) \in \mathbb{Z}[x]$ has degree 1, a root modulo N can be trivially computed: there are only 13 negative discriminants D for which this is the case [Cox13, Theorem 7.30], namely

$$D \in \{-3, -4, -7, -8, -11, -12, -16, -19, -27, -28, -43, -67, -163\}$$

When $H_D(x)$ has degree higher than 1, we can get around the problem of explicitly finding a root j modulo N by simply defining our elliptic curve over the ring

$$\mathbb{Z}_N(j) \simeq \mathbb{Z}_N[x]/H_D(x)$$

rather than $\mathbb{Z}_N$. Hence, when using Theorem 6.1, the coefficients of the resulting Weierstrass curve can be easily computed by inverting $j - 1728$, and all arithmetic operations on the curve can be carried on $\mathbb{Z}_N(j)$ similarly as for $\mathbb{Z}_N$: if some inversions fail, then a root j (or a factor of N) can be explicitly computed.

6.4.3 Picking Random Points on Curves over $\mathbb{Z}_N(j)$

Once we obtain a short Weierstrass curve equation E over $\mathbb{Z}_N(j) \simeq \mathbb{Z}_N[x]/H_D(x)$, computed according to either Theorem 6.1, Theorem 6.2 or Theorem 6.3, we need at least one point on it to be able to run our attack against the modulus N.

At first glance, this might look hard since we need to extract square roots modulo N to obtain the Y-coordinate corresponding to some (random) X-coordinate on the curve. Indeed, an oracle returning square roots modulo N can be used to factor N efficiently. We can, however, bypass this problem by adopting, instead, XZ-arithmetic: Bernstein and Lange collected in [BL] many efficient XZ-arithmetic formulas that work in our ring $\mathbb{Z}_N(j)$ as well.

More concretely, for a random X-coordinate $P_X \in \mathbb{Z}_N$, we consider the XZ-point $P = (P_X : 1)$, and we compute $Q = \left(\prod_{p_i \in \mathcal{B}} p_i^{\ell}\right) \cdot P$ using XZ-arithmetic formulas. If $P \in E(\mathbb{Z}_N(j))$, and the order of $E(\mathbb{F}_p)$ is $\mathcal{B}$-smooth, we can then attempt a factorization for N. If, instead, $P \notin E(\mathbb{Z}_N(j))$, the obtained point Q does not help in factoring N: we then need to pick another $P_X \in \mathbb{Z}_N$ and try the above again, until we are confident enough to have picked at least one point on the curve.[3]

When a point P lying on $E(\mathbb{Z}_N(j))$ is effectively picked, we need to check if the corresponding XZ-point $Q = (Q_X, Q_Z)$ gives rise to a non-trivial factorization of N. Given the canonical projection $\pi : \mathbb{Z}_N(j) \to \mathbb{F}_p(j) \times \mathbb{F}_q(j)$, we have that if the order of $E(\mathbb{F}_p)$ is $\mathcal{B}$-smooth, then $\pi(Q_Z) = (0, \tilde{Q}_q)$, that is Q is projected to the identity element $(0 : 0)$ (in XZ-coordinates) of $E(\mathbb{F}_p(j))$: if $\tilde{Q} \neq 0$, we can then reveal a factor for N as follows.

From the fact that $\mathbb{Z}_N(j) \simeq \mathbb{Z}_N[x]/H_D(x)$, we can see the Z-coordinate of Q as a polynomial $Q_Z(x) \in \mathbb{Z}_N[x]$ of degree less than $\deg(H_D(x))$: if j is the j-invariant of $E(\mathbb{F}_p)$, it then must be a root of both $H_D(x)$ and $Q_Z(x)$ modulo p, since, as we

[3] We recall that if an X-coordinate does not lie on a curve, it then lies on its quadratic twist. So, for a random $P_X \in \mathbb{Z}_N$, the probability that the XZ-point $P = (P_X : 1) \in E(\mathbb{Z}_N(j))$ is $\frac{1}{2}$.

already saw, $\pi(Q_Z) = (0, \tilde{Q})_q$. In other words the *resultant*[4] over $\mathbb{F}_p$ of these two polynomials satisfies

$$Res(H_D(x), Q_Z(x)) \equiv 0 \pmod{p}$$

If $\tilde{Q}_q \neq 0$, that is Q is not projected to the identity element of $E(\mathbb{F}_q(j))$, we might then reveal a factor for N as

$$\gcd(\, Res(H_D(x), Q_Z(x)),\, N\,) = p$$

Picking points in affine XY-coordinates It is possible to construct XY-points lying on $E(\mathbb{Z}_N(j))$ with no need to extract square roots modulo N. By taking advantage of the free choice of the twisting coefficient and, for $D = -4$, of the curve parameter, we can explicitly write a solution to the curve equation associated with a certain discriminant. The drawback of this approach is that we will then be able to pick points on a particular curve twist only (if $D \neq -4$). Thus, to backdoor semiprimes successfully, we need to generate primes associated with curves isomorphic only to such reachable twist.

On the other hand, the main advantage of using XY-arithmetic in place of XZ, is that we can use standard formulas to compute points additions, rather than *differential addition formulas*, the only available for XZ-points. Indeed, given two points P, Q in XZ-coordinate, we can compute $P + Q$ only if we know the XZ-coordinates of $P, Q, P - Q$ (hence the adjective *differential*). In contrast, scalar-point multiplications $[k] \cdot P$, the only operation relevant for our attacks, can be executed using Montgomery Ladder by only knowing the XZ-coordinates of P. The possibility to execute additions between arbitrary points is of relevance for sketching distributed protocols to generate semiprimes, based on the prime backdooring idea detailed in previous Sections: we will address this in Section 6.9.

Assuming $N = p \cdot q$, where p is a backdoored prime, we can construct, for each possible discriminant, points in affine XY-coordinates lying on the curves built according to the above Theorems, as follows.

- $\boldsymbol{D \neq -3, -4}$. We have, for a given k, the curve $E(\mathbb{Z}_N(j)) : y^2 = x^3 - 3kc^2x + 2kc^3$. We generate a random $x \in \mathbb{Z}_N(j)$ and we consider the point $P = (x^2, x^3)$: it lies on E if $2c - 3x^2 = 0$, that is if we set $c = \frac{3}{2}x^2$. It follows that with this approach, only one twist can be selected, namely the one given by the residuosity of $\frac{3}{2}$ modulo p.

- $\boldsymbol{D = -3}$. In this case, we work with the curve $E(\mathbb{Z}_N(j)) : y^2 = x^3 + c^3$. If we set $c = 2$, then the points $P = (1, 3)$ and $Q = (2, 3)$ are both points on E. Again, only one twist can be reached, depending on whether $c = 2$ is or is not a quadratic residue modulo p. These two points seems to be all the non-trivial (i.e. points of order not equal $2, 3, 6$) solution to the Diophantine equation $\frac{x^3+c^3}{y^2} = 1$.

- $\boldsymbol{D = -4}$. Here we have $E(\mathbb{Z}_N(j)) : y^2 = x^3 - cx$ for an arbitrary $c \in \mathbb{Z}_N(j)$: we can iteratively hit all the 4 possible orders given by Theorem 6.3 by picking random $x, w \in \mathbb{Z}_N(j)$ and letting $c = x^2 - xw^2$. It follows that $P = (x, xw)$ is on E.

[4] The resultant of two polynomials defined over an integral domain $\mathbb{F}[x]$ is zero if and only if they share a common root in the closure $\overline{\mathbb{F}}$.

An alternative approach, described in [ANS19], allows to work with all discriminants and twists by using standard formulas for XY-coordinates. Given an elliptic curve equation $y^2 = f(x)$, with $\deg f = 3$, we pick a random $P_X \in \mathbb{Z}_N$ and we set $\tau = f(P_X)$. We then consider the quotient of the polynomial ring $\mathbb{Z}_N[x, y]$ given by

$$R_{j,\tau} = \mathbb{Z}_N[x, y]/(H_D(x), y^2 - \tau)$$

and the elliptic curve $E(R_{j,\tau}) : y^2 = f(x)$. It follows that $P = (P_X, y) \in E(R)$ and we can perform scalar-point multiplications and additions over $E(R_{j,\tau})$ as usual, by working with the point P. Indeed, if for a certain scalar $k \in \mathbb{Z}_N$ we have $[k] \cdot P = \mathcal{O}$ in $E(R_{j,\tau})$, then $[k] \cdot P = \mathcal{O}$ in $E(\mathbb{Z}_N)$ if $\sqrt{\tau} \in \mathbb{Z}_N$ and j is a root of $H_D(x)$ modulo N. As happens for random points in XZ-coordinates, we cannot be sure that P_X is the X-coordinate of a point in $E(\mathbb{Z}_N)$, and we then need to select multiple points (and thus work on different rings $R_{j,\tau}$) until we are confident enough to have picked one lying on $E(\mathbb{Z}_N)$. Clearly, the arithmetic in $R_{j,\tau}$ is slower than the one we have in $\mathbb{Z}_N(j)$ or $\mathbb{Z}_N$.

6.5 The Prime Generation Procedure

In light of the considerations outlined in Section 6.4, we can now formally state our prime generation procedure, previously sketched in Section 6.3.

Input: a negative discriminant D, the bitsize of the output prime p, the factor base size $n + 1$.
Output: a b-bits prime p, the factor base $\mathcal{B}$.

1. Set $\mathcal{B} = \emptyset$. Generate n tuples (p_i, v_i), where $p_i \in \mathbb{Z}$ are prime odd integers decomposing as $p_i = a_i^2 + |D|b_i^2$ and $v_i = a_i + b_i\sqrt{D} \in \mathbb{Z}[\sqrt{D}]$, and add them to the factor base $\mathcal{B}$. Only if $D = -4$, ensure that for all $(p_i, v_i) \in \mathcal{B}$, $p_i \equiv 1 \pmod 4$.

2. Set $p_0 = (D \pmod 2) + |D|$ and $v_0 = (D \pmod 2) + \sqrt{D}$, and add (p_0, v_0) to $\mathcal{B}$.

3. Randomly pick elements $(p_1, v_1), \ldots, (p_m, v_m)$ from $\mathcal{B}$, for some $m > 0$, so that

$$b - 2 < \log_2 \left(p_0 \cdot \prod_{i=1}^{m} p_i \right) < b$$

   Set $t = p_0 \cdot p_1 \cdot \ldots \cdot p_n$ and $v = v_0 \cdot v_1 \cdot \ldots \cdot v_n = a + b\sqrt{D} \in \mathbb{Z}[\sqrt{D}]$, for some $a, b \in \mathbb{Z}$.

4. If $D \neq -4$, check if $p = (a \pm 1)^2 + |D|b^2$ is prime. If yes, return $(p, \mathcal{B})$. If p is not prime, go to Step 3.

5. If $D = -4$, check if $p = (a \pm 1)^2 + (2b)^2$ or $p = a^2 + (2b \pm 1)^2$ is prime. If yes, return $(p, \mathcal{B})$, otherwise go to Step 3.

We note that, in Step 1, one can either use Algorithm 2 over some $p_i \in \mathbb{Z}$ in order to compute the corresponding v_i, or we can generate random $a_i, b_i \in \mathbb{Z}$, set $p_i = a_i^2 + |D|b_i^2$ and $v_i = a_i + b_i\sqrt{D}$, and test if such p_i are primes. Although primality for elements in $\mathcal{B}$ is, in general, not required, we experimentally observed that backdoored primes p are generated faster when $\mathcal{B}$ contains only odd primes

and the even element p_0, probably because this choice reduces the probability that p is divisible by small factors, e.g. $2, 3$: we therefore generate $\mathcal{B}$ to contain, besides $p_0 = (D \bmod 2) + |D|$, only odd primes which factors in $\mathbb{Z}[\sqrt{D}]$.[5]

Since the factor base $\mathcal{B}$, once generated, can be used to generate multiple primes p, the running time of the above algorithm is dominated by the search for primes in Step 4. Assuming that the computed values p behave like random b-bits odd integers, we then expect to find a prime after $O(b)$ loops.

6.6 Factoring Backdoored Integers

Let p be a prime generated with respect to a factor base $\mathcal{B}$ and a negative discriminant D, using the prime generation procedure outlined in Section 6.5. If p is a factor of an integer N, we say that N is *backdoored* since, by knowing $\mathcal{B}$ and D, it would be then possible to recover the (secret) factor p from N. We note that in case of semiprimes $N = p \cdot q$, as in the case of RSA moduli, we will obtain a full factorization for N, regardless of the choice of the other prime q.

To recover p from $\mathcal{B}$, D and $N = p \cdot q$, we run the following attack.

Input: a backdoored N, a negative discriminant D, the factor base $\mathcal{B} = \{p_0, \ldots, p_n\}$.
Output: a factor p of N.

1. If $D \neq -3, -4$:
   (a) Compute the Hilbert Class Polynomial $H_D(x) \in \mathbb{Z}[x]$ and let $\mathbb{Z}_N(j) = \mathbb{Z}_N[x]/H_D(x)$;
   (b) Compute $k = \frac{j}{j-1728} \in \mathbb{Z}_N(j)$;
   (c) Pick a random $c \in \mathbb{Z}_N$ and consider the curve $E : y^2 = x^3 - 3kc^2x + 2kc^3$ over $\mathbb{Z}_N(j)$.
2. If $D = -3$, pick a random $c \in \mathbb{Z}_N$ and consider the curve $E : y^2 = x^3 + c^3$ over $\mathbb{Z}_N$.
3. If $D = -4$, pick a random $c \in \mathbb{Z}_N$ and consider the curve $E : y^2 = x^3 - cx$ over $\mathbb{Z}_N$.
4. Pick a random $P_X \in \mathbb{Z}_N$ and set the XZ-point $P = (P_X : 1)$. For $i \in [0, n]$ compute, using XZ-arithmetic formulas over the curve $E(\mathbb{Z}_N(j))$, the point
$$Q = \left(\prod_{p_i \in \mathcal{B}} p_i^{\ell_i} \right) \cdot P$$
where $\ell_i = \left\lfloor \log_{p_i} N \right\rfloor$ or is a fixed constant.
5. Let $Q = (Q_X : Q_Z)$ and consider Q_Z as a polynomial residue in $\mathbb{Z}_N[x]/H_D(x)$. If $\deg H_D = 1$ and $p = \gcd(Q_Z, N) \neq 1, N$, output p. If $\deg H_D > 1$ and $p = \gcd(Res(H_D, Q_Z), N) \neq 1, N$, output p.
6. If $D = -3$ go to Step 2; if $D = -4$ go to Step 3, otherwise go to Step 1c.

[5] This fact should explain the title of this Chapter.

We note that, if the factor base $\mathcal{B}$ used to backdoor such semiprimes N contains all primes less than a certain bound B, then anyone would be able to factor such N, if correctly guesses the discriminant D used. Indeed, as we already discussed in Subsection 6.4.2, D cannot be too big, because, otherwise, the computation of $H_D(x) \in \mathbb{Z}_N(x)$ and the corresponding induced arithmetic in $\mathbb{Z}_N(j)$, would result too expensive to carry out. If, instead, the factor base $\mathcal{B}$ is partially secret and contains, for example, many public small elements and few secret big factors, when backdoored $\mathcal{B}$-smooth curve orders are generated to be multiple of at least one of such secret big factors, then N can be easily factored only by those who fully know $\mathcal{B}$.

Question 6.1. *For a given discriminant D, are backdoored integers N distinguishible from non-backdoored ones when the factor base is (partially) secret?*

6.7 Implementation

We implemented both the prime generation procedure from Section 6.5 and the factorization attack detailed in Section 6.6 in SageMath [The21]. Our implementations is available on GitHub at

`https://github.com/cryptolu/primes-backdoor`

The first script `gen_prime.sage` generates, for a given input negative discriminant D and bitsize b, a random b-bits prime p admitting a Cornacchia's decomposition with respect to D, and so that one among the curve orders built according to either Theorem 6.1, Theorem 6.2, Theorem 6.3 is $\mathcal{B}$-smooth, with $\mathcal{B}$ containing the first suitable primes up to a certain (input) bound. Optionally, this script can output *safe primes* rather than just primes.

The second script is `attack.sage` and takes as input an integer N and a discriminant D. It attempts the attack from Section 6.6 on N, by building a proper elliptic curve $E(\mathbb{Z}_N)$ and by computing $[\prod_{p_i \in \mathcal{B}} p_i^{\ell}] \cdot P$ for a random point $P = (P_X : 1)$ in XZ-coordinates. In order to compute scalar-point multiplications, the implemented XZ-arithmetic uses Montgomery Ladder [Mon87a], differential doubling `xECDBL` and differential addition `xECADD` formulas reported, respectively, in [IT05, Algortihm 3, A.3, A.5]

We employed these two scripts to backdoor and later factor the modulus we report in next Section to show an attack example.

6.7.1 A Full Attack Example

Suppose we have just generated the following 1024-bits RSA modulus

$$
\begin{aligned}
N \;=\; & 10821805528622065889230568696679645000565609385356188057330375777 \\
& 79870852502715750896717380601416332282635416751505870710179577464 \\
& 44011936428240275511212238390668203212834843038096455415137527562 \\
& 23753686005624850655792830578845228701699881266834878524228204494 \\
& 46059872302424866654255328887383184108204782014745221
\end{aligned}
$$

using some shady closed-source software found online. We suspect it has been somehow backdoored and we would like to check it against the attack outlined in Section 6.6.

We start iterating through all possible discriminants up to a certain bound, and we are now working with $D = -107$. We then compute the Hilbert Class Polynomial

$$H_{-107}(x) = x^3 + 129783279616000 \cdot x^2 - 6764523159552000000 \cdot x + 337618789203968000000000$$

we set $\mathbb{Z}_N(j) \simeq \mathbb{Z}_N[x]/H_D(x)$, and we obtain $k = \frac{j}{j-1728} \in \mathbb{Z}_N(j)$ as

$$\begin{aligned} k = {} & 472613932478766669340966844279313193169990661224983092468863164 \\ & 658263076136993132803805737978847428706764681002200815383623942 \\ & 848444530320737151526189038330874037413841705251839051921349352 \\ & 749497051238720669707580258579708435533425339388316585678359757 \\ & 290371999173518938531491168397327757992982621433926 5 \cdot j^2 + \\ & 258353749123922389248704920994134238184684237766667425084465789 \\ & 259491423980136846866193858910647641442977366590032461132868140 \\ & 321600139344333421095604424766542254655451730245674928544209478 \\ & 335747612448949416507130681984905158597071148938192279371583953 \\ & 605339477741539037810199585641157784517018132244846 0 \cdot j + \\ & 921251340899301433133091422967299422144510775713942007364855038 \\ & 982942940744529752320790062724028658140002586224293736924737278 \\ & 005169222018193295962410583564443260916711019326041327795199467 \\ & 542200751242365012754312396131057298816375017611381308687252198 \\ & 375283127733361984532155732573080176612263329313052 1 \end{aligned}$$

We pick a random $c \in \mathbb{Z}_N$, and we define the curve $E(\mathbb{Z}_N(j)) : y^2 = x^3 + Ax + B$, where $A = -3kc^2$ and $B = 2kc^3$. In our case, these values result to be

$$\begin{aligned} A = {} & 327995130978423695949911778178828621193335624428619226678494051 \\ & 318707888740455245704848411357604734944161593606281972065826024 \\ & 658291849294990539631420321202706181596306433878274673417758062 \\ & 582232886810815056929636602581034529682795514610361665304507462 \\ & 953424154071938557120628927683339007148209922702921 8 \cdot j^2 + \\ & 107249083075978240751276263919220465878752147961845607664173475 \\ & 426383970880749919957727022038577172281511589403643080797028145 \\ & 016666779034342327903318877728719080110491797294911829171483273 \\ & 203762143779784374706244554915429186905254336517475532965059602 \\ & 832250693190364379930605663798894935467628957641801 4 \cdot j + \\ & 819160971922140650359819037400073641565902668643711131669462520 \\ & 070780364452384201748612105417995406358643088745166654606993288 \\ & 523934046315308756669815062519925039514142541665858932154029096 \\ & 944489201432328127690468661842228330448470250580291070716536018 \\ & 996600187282021979728700446903734598874041233975421 \end{aligned}$$

$$\begin{aligned} B = {} & 376432816799084515187872402469836997885480396973962586935959720 \\ & 859051489993844340369009542798339328568531548082733106741210278 \\ & 739683928445839760870753245760938270442261414385329319582052805 \\ & 043962557041892207826709216632705035906603638388075427965808304 \\ & 442215510358635714797813179749076921942256339976435 1 \cdot j^2 + \end{aligned}$$

$$
\begin{aligned}
&326265344764730678538644608040399522711425556561667238019644940 3\\
&102410690220980303345333246188191866771155608362353346142265495 9\\
&379288572474313108809468121373358129216608359518792887625368730 4\\
&909544272446933953143704836058382974612419563983493700878157060 2\\
&035560787826699877001403657203557132779168395573 77 \cdot j \ +\\
&201879524150279063899289135904433201494479646292555488453221882 5\\
&062130398301801288156033427731216315886543490632862896772443034 2\\
&332021314758604343303181410828794938158935280297601701497718859 5\\
&963194192471898706544716546040572767120640186238143794256179887 3\\
&822477481110468165631500466054560752894547005649069 9
\end{aligned}
$$

We then generate a random XZ-point P, such as

$$
\begin{aligned}
P \ = \ &(821338627521289352342258992501641854441973747020599835036033473 7\\
&080876753438377728747691738972883214338871755136644335071074897 2\\
&829674213884055186744682533073101226469679418098641072829349126 1\\
&029140374323358189141808979830512831921701394965458585646046199 9\\
&703884598446829478890812625784258260398357177061692 6 : 1)
\end{aligned}
$$

and we compute, using XZ-arithmetic, the point $Q = \left(\prod_{p_i \in \mathcal{B}} p_i^{\ell_i}\right) \cdot P$ (we set $\ell_i = 2$ for all i), using a factor base $\mathcal{B}$ containing all primes p_i of the form $p_i = a_i^2 + |D| b_i^2$, with $0 \leq a_i, b_i < 2^8$ for all $i > 0$, and the element $p_0 = 1 + |D| = 108$, for a total of 3468 elements. The point we obtained is

$$
\begin{aligned}
Q \ = \ &(333486546774833434363989430016044023588688751867285387445940569 6\\
&871929011871127825880719939880900515818694803562077296175156337 6\\
&270699627020501762228657451819904121885847689656173650676028922 2\\
&225633544880373520540585580492811126326091944242005541132773760 4\\
&112320289061101226982531565282146389965688955640469 9 \cdot j^2 \ +\\
&768990649638045539471007347655971602406027931578122973745927075 7\\
&616506833008583862582611334448071824599063655204175095973123384 8\\
&012058381404348134431198252425051250560573126085163220926882023 6\\
&257688149220279076838915118320063892778623624928604439428776432 4\\
&313090534354852013264085421128950995465463715347443 5 \cdot j \ +\\
&434173976803460454968421075911020559985286395293479694907268930 6\\
&788758131453510690971259655887610317060352371378239469718379470 1\\
&993953831068067597209280255295706885774392753656010442644968482 3\\
&455184876742782247286835242833180841022665653992445761968923719 2\\
&048448412914886596104323695646824168404593031366713 4 \ :\\
&618211539561017111651264822865371698429901880962216340832348802 2\\
&374796373632812995398780367381750565749075802819168683372166826 5\\
&198553127961132031704294095209463756691728653769822746319710521 1\\
&985163815416150573121985796044101904858691673691784761775868123 7\\
&353852431023895615553442374888382248599669119196652 8 \cdot j^2 \ +\\
&640200920264136618444571638754781250345180034283207008581478217 2\\
&104565818099915491272074968012291963648399121125602177743509824 2\\
&544756666481623813285991729671734877269407128296465725650516056 4\\
&771911809202319075612205316833579640077609529601021029320266111 4\\
&967692323459569732329897311730819441247818469504254 8 \cdot j \ +
\end{aligned}
$$

628420400008566255947539441090350536835867552567396920232225451659323190728584573361365616423902207967548395330926598218781365113496334139269070887399056726360020627972944444322631792467924055348804177040071434741535302088386805117988777028638124379958729581443707206987266326029167539439551259630750633729 4)

We let $Q = (Q_X, Q_Z)$ and we consider Q_Z as an element in $\mathbb{Z}_N[x]/H_D(x)$, rather than $\mathbb{Z}_N(j)$. We then compute the resultant $r = Res(Q_Z, H_D(x))$, corresponding to

$$\begin{aligned} r = {} & 809355209872163778366488526248967695091073218048618439861345727781611305291018136825668277127163225004186357239784218139537990977555032254069388900963857213765315555138571357726228685587307731320761487508424358064791241796654601648282979492440695041733941015562505031429507249340582508862839515453637586038864 \end{aligned}$$

Unexpectedly, by computing $p = gcd(r, N)$ we obtain

$$\begin{aligned} p = {} & 100698911682728532896824145334441011589619711607218109604252997275000408561553999104080979108065560950825291542049122218726560865150200349165245350130556074 \end{aligned}$$

which confirms our suspicions! The attack leads to full factorization of N as

$$\begin{aligned} N = {} & 100698911682728532896824145334441011589619711607218109604252997275000408561553999104080979108065560950825291542049122218726560865150200349165245350130556074 \cdot 916358501063024091431585450207984096049550676398490431302221858607413434626065849874619277159618017566577515050516933095502483884691758256324683934772908356657751505051693309550248388469175825632468393477290834 \end{aligned}$$

We note that both factors of N are *safe primes*.

6.8 Certifiable Semiprimes

We can use the prime backdooring procedure from Section 6.5 to sketch a *multi-party computation* (MPC) protocol which outputs semiprimes of unknown factorization, a particularly useful application in the setting of distributed generation of RSA moduli. We will take advantage of a generalization of a Theorem by Goldwasser and Kilian, which provides a criterion for an integer N to be semiprime given partial knowledge of the order of an elliptic curve modulo N. When we port this idea into the MPC setting, parties will jointly construct a curve modulo N by backdooring part of its order according to this theorem, and where N, once revealed, can be *certified* to be semiprime, thanks to a semiprimality proof that can be publicly checked without knowing any of the factors of N.

6.8.1 Preliminaries

In this Section we provide the theoretical results to formalize semiprimality *certificates* for odd integers. We start by stating Goldwasser-Kilian Theorem, initially thought of as a tool to prove the primality of a certain integer N.

Theorem 6.4 (Goldwasser-Kilian [GK86])**.** *Let $N > 1$ and let E be an elliptic curve defined over $\mathbb{Z}_N$. Suppose there exist distinct primes $p_1, \ldots, p_k$ and finite points $P_1, \ldots, P_k \in E(\mathbb{Z}_N)$ such that $[p_i] \cdot P_i = \mathcal{O}$ for all $1 \leq i \leq k$ and $\prod_{i=1}^{k} p_i > (\sqrt[4]{N} + 1)^2$. Then N is prime.*

Proof. See [Was08, Theorem 7.3]. □

We can slightly restate Goldwasser-Kilian Theorem to provide a condition that, when true, will ensure N has at most a certain number of distinct prime factors.

Theorem 6.5. *Let $N > 1$ and let E be an elliptic curve defined over $\mathbb{Z}_N$. Suppose there exist distinct primes $p_1, \dots, p_k$ and finite points $P_1, \dots, P_k \in E(\mathbb{Z}_N)$ such that $p_i \cdot P_i = \mathcal{O}$ for all $1 \leq i \leq k$ and $\prod_{i=1}^{k} p_i > \left(\sqrt[2m]{N} + 1\right)^2$. Then N can have at most $m - 1$ distinct prime factors.*

Proof. We will adjust to our needs the proof of Theorem 6.4, that is [Was08, Theorem 7.3]. Let q be a prime factor of N so that $N = q^s \cdot d$ for some $s > 0$ and $\gcd(q, d) = 1$. Since P_i is a finite point in $E(\mathbb{Z}_N) \simeq E(\mathbb{Z}_{q^s}) \times E(\mathbb{Z}_d)$ it is a finite point modulo q^s and, in turn, modulo q as well. Then $p_i \cdot P_i \pmod q = \mathcal{O} \in E(\mathbb{F}_q)$, which implies that $P_i \pmod q$ has order p_i in $E(\mathbb{F}_q)$ for all $1 \leq i \leq k$. It follows that $\prod_{i=1}^{k} p_i \mid \#E(\mathbb{F}_q)$ and by Hasse bound

$$\left(\sqrt[2m]{N} + 1\right)^2 < \prod_{i=1}^{k} p_i \leq \#E(\mathbb{F}_q) < q + 1 + 2\sqrt{q} = (\sqrt{q} + 1)^2$$

So $q > \sqrt[m]{N}$ for any prime q dividing N, hence N cannot have more than $m - 1$ distinct prime factors. □

Corollary 6.1. *Let $N > 1$ be a composite non-square integer and let E be an elliptic curve defined over $\mathbb{Z}_N$. Suppose there exist a prime $s > \left(\sqrt[6]{N} + 1\right)^2$ and a finite point $P \in E(\mathbb{Z}_N)$ so that $[s] \cdot P = \mathcal{O}$ in $E(\mathbb{Z}_N)$. Then N is semiprime.*

We note that in Corollary 6.1 we require s to be prime: its primality can be certified using Theorem 6.4, factoring the orders of elliptic curves built over $\mathbb{Z}_s$ and explicitly construct, when possible, points of prime order satisfying the assumption of Theorem 6.4. It follows that Theorem 6.4 can be used, in turn, to prove the primality of the orders of the points employed to prove the primality of a certain integer: this process ultimately results in a *chain* of primality proofs which reduces the problem of certifying primality of s to certifying primality of (much) smaller integers. Such certificate chains are commonly known as Atkin-Goldwasser-Kilian-Morain primality certificates [GK86; AM93].

6.8.2 Certificates for Semiprimes for which a Factorization in Known

From Corollary 6.1 follows that, if we know (or *we construct!*) a tuple (N, $E(\mathbb{Z}_N)$, P, s) whose elements satisfy its assumptions, then such tuple is, in fact, a *semiprimality certificate* for N.

A naive approach to compute such certificates when the factors of $N = p \cdot q$ are known, consists in generating a random curve $E(\mathbb{Z}_N)$ for which we can factor its order, in turn obtained by computing the orders of the two curves $E(\mathbb{F}_p)$, $E(\mathbb{F}_q)$ using, for example, the Schoof–Elkies–Atkin (SEA) algorithm [Sch95; Elk98; Atk88; Atk99]. Indeed, if $|E(\mathbb{Z}_N)|$ is divisible by a prime $s > (\sqrt[6]{N} + 1)^2$, or by multiple primes p_i so that $\prod_i p_i > (\sqrt[6]{N} + 1)^2$ (Theorem 6.5), we can pick random points $Q \in E(\mathbb{Z}_N)$[6] and check if $P = [|E(\mathbb{Z}_N)|/s] \cdot Q \neq \mathcal{O}$. When this holds, it follows that $[s] \cdot P = \mathcal{O} \in E(\mathbb{Z}_N)$ and thus (N, $E(\mathbb{Z}_N)$, P, s) is a semiprimality certificate for N.

[6] Equivalently, we can work, thanks to the Chinese Remainder Theorem, over $E(\mathbb{F}_p) \times E(\mathbb{F}_q)$.

This approach has two main drawbacks: the first is that partial factoring the order $|E(\mathbb{Z}_N)|$ until enough divisors $\{p_i\}_i$ so that $\prod_i p_i > (\sqrt[6]{N}+1)^2$ are found, is often expensive or even impractical depending on the size of N. Secondly, if a certain p_i divides $|E(\mathbb{F}_p)|$ but not $|E(\mathbb{F}_q)|$, it cannot belong to a semiprimality certificate since any point P_i of order p_i will leak a factor of N, similarly as we describe in Section 6.6: from the condition $[p_i] \cdot P_i = \mathcal{O} \in E(\mathbb{Z}_N)$, we must have P_i to correspond to the identity element of $E(\mathbb{F}_q)$ when its projective coordinates are reduced modulo q, and thus the Z-coordinate of P_i shares a factor with N.

In other words, each element p_i published in a semiprimality certificate should divide *both* orders $|E(\mathbb{F}_p)|$ and $|E(\mathbb{F}_q)|$: on this regards, we note that to reduce certificate sizes, we simply avoid publishing multiple points P_i, each of size at least $2 \log N$, along with their order p_i, and we publish instead a single point P of order $s > (\sqrt[6]{N}+1)^2$ with s prime, as done in the formulation of Corollary 6.1. Indeed, if s is prime, by Cauchy's Theorem we know that both curves $E(\mathbb{F}_p)$ and $E(\mathbb{F}_q)$ have a point of order s, and thus, by Chinese Remainder Theorem, it exists a point P in $E(\mathbb{Z}_N)$ of order s. In fact, $E(\mathbb{Z}_N)$ will contain a subgroup isomorphic to $\mathbb{Z}_s \times \mathbb{Z}_s$ and, to avoid leaking factors of N as showed above, we look for points P of order s in correspondence to elements $(a, b) \in \mathbb{Z}_s \times \mathbb{Z}_s$ with $a, b \neq 0$.

How can we then efficiently generate big certifiable semiprimes? In [Don05; Dav05], we found concrete instances of semiprimality certificates presumably generated according to observations similar to the above[7]. In a private conversation, Reble confirmed to us that his certificate [Don05] for a semiprime N of 1084 digits was generated from its prime factors and added, in regards to its generation: "*I knew the factors. I sought a pair of primes such that the Goldwasser-Kilian test almost worked for that product*". Shortly after [Don05] became public, Broadhurst generated a semiprimality certificate for a 5061-digits integer [Dav05] consisting of a single elliptic curve point P and a 1690-digits prime s, for which a primality proof is known. We note that both [Don05; Dav05] employ the elliptic curve $E(\mathbb{Z}_N) : y^2 = x^3 + A \cdot x$, whose order is characterized by Theorem 6.3.

In a more general fashion, we can efficiently compute semiprimality certificates for an integer N by generating both its prime factors according to the backdooring procedure of Section 6.5, and ensuring that the candidate curve orders are divisible by a prime $s > (\sqrt[6]{N}+1)^2$ which admits a Cornacchia decomposition for the employed D. This can be achieved by slightly change our prime generation procedure, adding such prime s to the factor base $\mathcal{B}$ (Step 1), and requiring t to always be a multiple of s and the even element p_0 (Step 3). From the full knowledge of $|E(\mathbb{F}_p)|$ and $|E(\mathbb{F}_q)|$, where the curve E is retrieved from N as in Section 6.6, we can then randomly pick and re-scale points in $E(\mathbb{Z}_N)$ as above, until we find one of order s (and Z-coordinate not equal to 0 when seen modulo p and modulo q).

6.9 Distributed Computation of Certifiable Semiprimes

In this Section we investigate how we can possible have a multi-party computation protocol to jointly compute an integer N of unknown factorization and a semiprimality certificate for it. Our goal is to port our prime generation procedure, slightly restated as described at the end of Subsection 6.8.2, in a distributed setting, so that parties generate two candidate prime curve orders t_p, t_q for $E(\mathbb{F}_p)$ and $E(\mathbb{F}_q)$, respectively, both divisible by a public prime $s > (\sqrt[6]{N}+1)^2$. Parties then opens

[7] We were not able to find more details about how these certificates were generated, except a short discussion on their validity with respect to (a generalization of) Goldwasser-Kilian Theorem.

$N = p \cdot q$ and multiple points $Q_i = [\frac{t_p t_q}{s^2}] \cdot P_i$ for random $P_i \in E(\mathbb{Z}_N)$. If for some i, we have $Q_i \neq \mathcal{O}$ and $[s] \cdot Q_i = \mathcal{O}$, we then output N and the semiprimality certificate (N, E, s, Q_i).

The whole procedure is sketched as follows, where we employed the notation $[\cdot]$ to denote secret values.

Input. The bitsize `bits`, a negative discriminant D.
Output. A semiprimality certificate for a `bits`-bits integer N.

1. Parties publicly agree on two integers s_a, s_b so that $s = s_a^2 + |D| \cdot s_b^2$ is a prime of `bits`/3 bits.

2. Each party i randomly picks 4 values $[a_{i,1}], [b_{i,1}], [a_{i,2}], [b_{i,2}]$ of `bits`/12 bits each.

3. Parties jointly compute the 4 values:

$$[a_1] = \sum_i [a_{1,i}] \, , \qquad [b_1] = \sum_i [b_{1,i}] \, , \qquad [a_2] = \sum_i [a_{2,i}] \, , \qquad [b_2] = \sum_i [b_{2,i}]$$

4. Parties compute the 4 values

$$\begin{aligned} [p_a] &= s_a \cdot [a_1] + D \cdot s_b \cdot [b_1] \\ [p_b] &= s_a \cdot [b_1] + s_b \cdot [a_1] \\ [q_a] &= s_a \cdot [a_2] + D \cdot s_b \cdot [b_2] \\ [q_b] &= s_a \cdot [b_2] + s_b \cdot [a_2] \end{aligned}$$

5. If $D \neq -4$, for each choice of $v_1, v_2 \in [-1, 1]$, parties jointly open the 4 values

$$N_{v_1,v_2,0} = \left(([p_a] + v_1)^2 + |D| \cdot [p_b]^2\right) \cdot \left(([q_a] + v_2)^2 + |D| \cdot [q_b]^2\right)$$

6. For each choice of $v_1, v_2 \in [-1, 1]$, parties may[8] further open the values

$$N_{v_1,v_2,0} = \left([p_a]^2 + (2 \cdot [p_b] + v_1))^2\right) \cdot \left([q_a]^2 + (2 \cdot [q_b] + v_2)^2\right)$$

$$N_{v_1,v_2,1} = \left(([p_a] + v_1)^2 + |D| \cdot [p_b]^2\right) \cdot \left([q_a]^2 + (2 \cdot [q_b] + v_2)^2\right)$$

$$N_{v_1,v_2,1} = \left([p_a]^2 + (2 \cdot [p_b] + v_1))^2\right) \cdot \left(([q_a] + v_2)^2 + |D| \cdot [q_b]^2\right)$$

7. For each opened value N_{v_1,v_2,v_3}:

   7.1. Let $N_{v_1,v_2,v_3} = N$. If either N is a perfect square, has small prime factors or $s < (\sqrt[6]{N} + 1)^2$, parties skip to the next choice of N_{v_1,v_2,v_3}.

   7.2. Depending on the discriminant D, parties publicly agree on one or multiple curves $\{E_\ell(\mathbb{Z}_N[x]/H_D(x))\}$ along with a point $P_\ell \in E_\ell$ on it[9], until are

[8] Different trade-offs are possible depending on the employed MPC arithmetic protocol, the overall network communication and the number of openings parties execute out of the 16 available.

[9] It may be needed to test multiple random points P_ℓ, depending on the curve arithmetic used, before being sure to have picked at least one point on E_ℓ.

confident enough to have picked a curve representative for each reachable order (cf. Theorem 6.1, Theorem 6.2, Theorem 6.3).

7.3. For each $P_\ell \in E_\ell$, parties jointly compute and open the point

$$Q_\ell = \Big[([a_1]^2 + |D| \cdot [b_1]^2) \cdot ([a_2]^2 + |D| \cdot [b_2]^2)\Big] \cdot P_\ell$$

If $Q_\ell \neq \mathcal{O}$ and $[s] \cdot Q_\ell = \mathcal{O} \in E_\ell$, return the semiprimality certificate $\{N, E_\ell, Q_\ell, s\}$.

8. Go to Step 2.

6.9.1 Practical Considerations

Although there exist many protocols that, depending on the most suitable security scenario, can efficiently perform in MPC the standard arithmetic operations needed in the first part of our sketched protocol, execution of Step 7 is not trivial and may represent the main bottleneck for implementing it in full. Difficulties mainly reside in the opening of the points Q_ℓ, which involves a scalar-point multiplication $[k] \cdot P_\ell$, for a secret k shared among parties.

Although protocols like [ST19; FN20] allow distributed computation of $[k] \cdot P_\ell$ when P_ℓ belongs to a curve defined over a finite field of known characteristic, in our case we work, instead, over a ring (either $\mathbb{Z}_N$ or $\mathbb{Z}_N(j) \simeq \mathbb{Z}_N[x]/H_D(x)$) and the order of P_ℓ is (*and should remain!*) unknown. In other words, these protocols cannot be employed, at least not straightforwardly, and we are not aware of designs that can be used in our case of interest (and which are eventually able to work with XZ-coordinates).

We can however bypass this limitation: instead of directly opening the point $Q_\ell = [k] \cdot P_\ell$, where $k = ([a_1]^2 + |D| \cdot [b_1]^2) \cdot ([a_2]^2 + |D| \cdot [b_2]^2)$, we additively secret share among all parties the secret $[k]$ over a field $\mathbb{F}_r$, with r a public prime much greater than the expected value for k.

Once $[k]$ is additively shared, each party i possesses a share $k_i \in \mathbb{F}_r$ satisfying $(\sum_i k_i) \pmod r = k$. Thus, for each publicly agreed point P_ℓ, party i locally computes $\tilde{P}_\ell = [k_i] \cdot P_\ell$ and publishes this point to other parties. Assuming n parties are involved in the distributed computation, they can then compute in clear $\tilde{Q}_\ell = \sum_{i=1}^{n} \tilde{P}_\ell$ and the set of elliptic curve points

$$\mathcal{Q}_\ell = \{\ \tilde{Q}_\ell,\ \ \tilde{Q}_\ell - [r] \cdot P_\ell,\ \ \tilde{Q}_\ell - [2 \cdot r] \cdot P_\ell,\ \ \ldots,\ \ \tilde{Q}_\ell - [n \cdot r] \cdot P_\ell\ \ \}$$

It follows that the point $Q_\ell = [k] \cdot P_\ell$ is in $\mathcal{Q}_\ell$, and parties can then check the condition $[s] \cdot Q_\ell = \mathcal{O} \in E_\ell$ at the end of Step 7, by checking if $\mathcal{O}$ is in the the set of points $\{\ [s] \cdot \tilde{Q}\ \mid\ \tilde{Q} \in \mathcal{Q}_\ell\ \}$.

There is one last subtlety we need to address: the public computation of the point $\tilde{Q}_\ell = \sum_{i=1}^{n} \tilde{P}_\ell$ cannot be done in XZ-coordinates, since addition formulas are defined in terms of *differential addition* and to compute any sum $P + Q$ we need the X-coordinate of the point $P - Q$, which in our case is unknown.

We can address this problem mainly in two ways. The first approach consists in working over the rings $R_{j,\tau}$ we defined at the end of Section 6.4.3, that, regardless of the discriminant D chosen, allows us to pick random points $P_\ell \in E(R_{j,\tau})$ in XY-coordinates and thus use standard formulas for points addition. Unfortunately, although this approach allows us to generalize the construction to all possible curve twists and negative discriminant D, in practice, it results expensive in terms of ring

arithmetic, and parties need to work on a different ring $R_{j,\tau}$ for each choice of the point P_ℓ in Step 7.

Alternatively, we can explicitly construct random XY-points P_ℓ with coordinates in $\mathbb{Z}_N$ as we detail in Section 6.4.3: although the arithmetic would result faster, within this approach, we will be able to work only with one curve twist for each factor of N when $D \neq -4$, and thus we need, on average, four times as many iterations of the protocol before returning a semiprime certificate. However, the case $D = -4$ suits best our needs: first, the corresponding Hilbert Class Polynomial has degree 1, thus a root j for $H_D(x)$ can be easily obtained modulo N; secondly, in Section 6.4.3, we show how it is possible to explicitly construct points in XY-coordinates over $\mathbb{Z}_N$ for all the curve orders characterized by Theorem 6.3. It follows that when $D = -4$, we can work exclusively over $\mathbb{Z}_N$ and perform point additions with standard affine XY-coordinates formulas.

Assuming the inputs from parties to be random, the algorithm terminates when: i) N is a product of two primes; ii) the random curve modulo N selected matches the curve twists with orders divisible by s; iii) the elliptic curve arithmetic does not fail modulo N. If at each round of the protocol, in Step 5 and Step 6 m different integers are opened in total, then a semiprime is returned in approximately $\frac{1}{m} \cdot (\frac{\texttt{bits}}{2})^2 = O(\texttt{bits}^2)$ rounds execution.

6.9.2 Implementation

To show its practicality, we implemented the protocol outlined in Section 6.9 for the case $D = -4$ in SageMath [The21] and MP-SPDZ [Kel20], a multi-protocol framework for multi-party computations based on SPDZ [Dam+12; Dam+13]. Our implementation can be found at

`https://github.com/cryptolu/semiprimes`

and further algorithmic optimizations are left as future work.

In order to allow parties to open points Q_ℓ in Step 7, we additively secret share the value $k = ([a_1]^2 + |D| \cdot [b_1]^2) \cdot ([a_2]^2 + |D| \cdot [b_2]^2)$ over a prime order field $\mathbb{F}_r$ of bit-size double as N, we compute the set $\mathcal{Q}_\ell$, and we check the value of $[s] \cdot Q_\ell$ as detailed in previous Section.

The SageMath script `generateN.sage` automates the execution of the MP-SPDZ multi-party computation script `semiprimes.mpc`, which performs the required distributed arithmetic operations over parties' secret values, the retrieval of the opened moduli and parties' additive shares, the elliptic curve arithmetic and the final checks on the returned semiprimality certificate.

As a proof of concept, we report a 128-bits semiprime generated, using our implementation, in $1300 \approx \frac{64^2}{4}$ rounds (to reduce communication, we did not implement Step 6). The generation involved two parties, both running on a standard desktop, who communicated using the MP-SPDZ `semi` protocol for semi-honest Oblvious Transfer based computations modulo a prime. The semiprimality certificate returned is

$$\begin{aligned} N &= 46121327044532730890860996866516955982927 33 \\ E(\mathbb{Z}_N) &: y^2 = x^3 + 20921147449173724872153659295373299245369 03 \cdot x \\ Q &= (35792898069199412144322564728728619314297 11 : \\ &\quad 14154437122626629949263497606251710686648 53) \\ s &= 538606233865081 \end{aligned}$$

where, in fact, $N = 556886529430039208669 \cdot 8281997248476609399457$.

6.10 Security of Semiprimality Certificates

In this Section we briefly investigate the security of semiprimality certificates for integers N generated according to Subsection 6.8.2 or Section 6.9, that is containing elliptic curves whose orders modulo each prime factor of N are characterized by either Theorem 6.1, Theorem 6.2 or Theorem 6.3[10]. We will detail and compare three kinds of attacks: a generic point order-finding algorithm based on Baby-step Giant-step algorithm combined with a *twisting attack*, a brute-forcing approach exploiting some leakage provided by semiprimality certificates, and factorization.

6.10.1 Baby-step Giant-step and the Twisting Attack

Since semiprimality certificates (N, E, Q, s) are generated using a variant of the prime generation procedure of Section 6.5, our factorization attack detailed in Section 6.6 will retrieve a factor of $N = p \cdot q$ with high probability, as soon as we compute (a multiple of) the order of a random point P in either $E(\mathbb{F}_p)$ or $E(\mathbb{F}_q)$. While in Section 6.6, we assume the order of at least one of the curves $E(\mathbb{F}_p)$, $E(\mathbb{F}_q)$ to be $\mathcal{B}$-smooth for a certain factor base $\mathcal{B}$, in the case of semiprimality certificates we have that $|E(\mathbb{F}_p)| = s \cdot k_p$ and $|E(\mathbb{F}_q)| = s \cdot k_q$ for certain random integers k_p, k_q.

Thus, as we already discussed in Subsection 6.4.3, given a curve $E(\mathbb{Z}_N) = E(\mathbb{F}_p) \times E(\mathbb{F}_q)$ and a random $P \in E(\mathbb{Z}_N)$ (in XZ-coordinates) on it, the point $\tilde{P} = [s] \cdot P \in E(\mathbb{Z}_N)$ would be such that $[k_p] \cdot \tilde{P} = (\mathcal{O}_p, \tilde{P}_q)$ and $[k_q] \cdot \tilde{P} = (\tilde{P}_p, \mathcal{O}_q)$, where $\mathcal{O}_p$ and $\mathcal{O}_q$ represents the identity elements of $E(\mathbb{F}_p)$ and $E(\mathbb{F}_q)$, respectively. In other words, if we are able to find a multiple of either k_p or k_q, we will be able to factor N as long as $\tilde{P}_q$ or $\tilde{P}_p$, respectively, are non-trivial.

To find such multiple, we can apply the Baby-step Giant-step algorithm to find the order of $\tilde{P} = [s] \cdot P$, where $P \neq Q$ is a random point in $E(\mathbb{Z}_N)$ expressed in XZ-coordinates (as usual, we compute the orders of different points $\tilde{P}$, until we are confident enough to have picked one lying on $E(\mathbb{Z}_N)$).

Assuming $p \approx q$, we have $\ln(k_p) \approx \ln(k_q) \approx \frac{\ln N}{6}$: since $\tilde{P}$ generates a group of order at most $k_p k_q$, we have that a successful execution of the Baby-step Giant-step algorithm would require on average $\exp\left(\frac{\ln N}{6}\right)$ scalar-point multiplications and $\exp\left(\frac{\ln N}{6}\right)$ space to return a (big) divisor k of $k_p k_q$ which, for simplicity, we can safely assume to be $k = k_p k_q$.

However, when we try to compute, for multiple $P \in E(\mathbb{Z}_N) = E(\mathbb{F}_p) \times E(\mathbb{F}_q)$, the point $[k \cdot s] \cdot P$, we obtain $[k \cdot s] \cdot P = (\mathcal{O}_p, \mathcal{O}_q)$, which does not help us in factoring N, unless we factor k itself. Indeed, by factoring k, we will be able to find a multiple $\tilde{k}$ of k_p which is not a multiple of k_q, or vice-versa, and so that $[\tilde{k} \cdot s] \cdot P \neq (\mathcal{O}_p, \mathcal{O}_q)$.

Although factoring k is asymptotically easier than running the Baby-step Giant-step algorithm to find it, we now show a technique which finds with overwhelming probability a factor of N knowing only such k and none of its factors. We call this method *"twisting attack"*. Let $c \in \mathbb{Z}_N$ a random integer so that its Jacobi symbol $\left(\frac{c}{N}\right) = -1$: since $N = p \cdot q$ is a semiprime, it follows that c is a quadratic residue in $\mathbb{F}_p$ and not in $\mathbb{F}_q$, or the opposite. Hence, the quadratic twist $\tilde{E}(\mathbb{Z}_N) = \tilde{E}(\mathbb{F}_p) \times \tilde{E}(\mathbb{F}_q)$ of $E(\mathbb{Z}_N)$ through c would be isomorphic to either $E(\mathbb{F}_p) \times \tilde{E}(\mathbb{F}_q)$ or $\tilde{E}(\mathbb{F}_p) \times E(\mathbb{F}_q)$. It follows that, given its size, s will not divide $|\tilde{E}(\mathbb{F}_p)|$ and $|\tilde{E}(\mathbb{F}_q)|$ with overwhelming

[10] There might exist, indeed, other curve equations for which the order can be characterized in a way that allow efficient generation of semiprimality certificates.

probability and thus, when we pick a random point $P \in \tilde{E}(\mathbb{Z}_N)$, we will have that $[s \cdot k] \cdot P$ is not equal to $\mathcal{O} \in \tilde{E}(\mathbb{Z}_N)$ but decomposes either as $(\mathcal{O}_p, \cdot)$ or $(\cdot, \mathcal{O}_q)$. We can then easily retrieve a non-trivial factor of N by applying the greatest common divisor to the Z-coordinate of $[s \cdot k] \cdot P$.

As a side note, we observe that the twisting attack can be applied to target either *pseudo super anomalous curves* or *super anomalous curves* [KK00], that is elliptic curves $E(\mathbb{Z}_N)$ with order equal N (super anomalous curves further require that if $N = \prod_i p_i$, then each curve $E(\mathbb{F}_{p_i})$ has order p_i, that is *anomalous*). For these curves, indeed, the ring characteristic trivially reveals the full curve order modulo N and hence, by working with twists of $E(\mathbb{Z}_N)$ as above, we can factor N, and directly work over the corresponding sub-curves.

Another example of the applicability of this method is given by [NF19], where a class of parameters for Demytko's Elliptic Curve Cryptosystem [Dem94] is shown to be weak, allowing an attacker to ultimately factor a public RSA modulus N. Here, the authors recover the order of an elliptic curve modulo N, which is factored with Lenstra's ECM to factor N and break the cryptosystem ultimately. With a twisting attack, instead, the knowledge of the curve order would suffice to factor N, allowing the attacks by [NF19] to remain feasible even with bigger key-sizes where the ECM approach becomes unpractical.

6.10.2 Curve Order Leakage Exploitation

Even though the methods of Subsection 6.10.1 will be used in Subsection 6.10.4 to quickly factor Reble's semiprimality certificate [Don05], much better asymptotic alternatives to the Baby-step Giant-step approach are possible for recovering the curve order.

We note, indeed, that a semiprimality certificate (N, E, Q, s) leaks $\frac{11}{12}$ of the bits of the order of the curve $E(\mathbb{Z}_N)$. This immediately follows from Hasse's bound [Sil09, V.I - Theorem 1.1]: by denoting with $\#E(\mathbb{Z}_N)$ the order of $E(\mathbb{Z}_N) \simeq E(\mathbb{F}_p) \times E(\mathbb{F}_q)$, we have that

$$|\#E(\mathbb{Z}_N) - N - 4\sqrt{N} - p - q - 1| \leq 2p\sqrt{q} + 2q\sqrt{p} + 2(\sqrt{p} + \sqrt{q})$$

Since $s \approx \sqrt[3]{N}$, and s^2 divides $\#E(\mathbb{Z}_N)$ by construction, we obtain

$$\left| \frac{\#E(\mathbb{Z}_N)}{s^2} - \frac{N - 4\sqrt{N}}{s^2} \right| \leq \frac{2p\sqrt{q} + 2q\sqrt{p}}{s^2} \approx \sqrt[12]{N}$$

Thus, using the above notation, we have $k_p k_q = \frac{N - 4\sqrt{N}}{s^2} + t$ with $|t| \approx \sqrt[12]{N}$.

In this setting, such t can be found, to the best of our knowledge, either memory-less using a brute-forcing approach or with a time-space trade-off using Baby-step Giant-step. In the case of brute-force, each guess of t is verified by picking multiple points $P \in E(\mathbb{Z}_N)$ and checking if $\left[N - 4\sqrt{N} + s^2 \cdot t\right] \cdot P = \mathcal{O} \in E(\mathbb{Z}_N)$. To use Baby-step Giant-step, instead, we set $R = [4\sqrt{N} - N] \cdot P$ and $Q = [s^2] \cdot P$, constrained by the relation $[t] \cdot Q = R$, and we then search for the discrete-logarithm of R in base Q. Once t is correctly retrieved, we can compute the full order of $E(\mathbb{Z}_N)$ and factor N, similarly, as we do in Subsection 6.10.1.

It follows that the average time complexity of brute-force consists of $\exp\left(\frac{\ln N}{12}\right)$ guesses for t, while the Baby-step Giant-step would require $\exp\left(\frac{\ln N}{24}\right)$ time given $\exp\left(\frac{\ln N}{24}\right)$ space.

6.10.3 Comparison to Factorization

How do these two attacks compare with respect to directly factoring N? The fastest known general-purpose factoring algorithm is the General Number Field Sieve (GNFS) [LL93] which allows to factor an integer N with complexity

$$\exp\left(\left(\sqrt[3]{\frac{64}{9}}+o(1)\right)(\ln N)^{\frac{1}{3}}(\ln\ln N)^{\frac{2}{3}}\right)$$

A semiprimality certificate $(N, E(\mathbb{Z}_N), Q, s)$ generated as in Subsection 6.8.2 or Section 6.9, will have $s \approx \frac{\ln N}{3}$.

The Baby-step Giant-step attack outlined in Subsection 6.10.1 would require, on average, $\exp\left(\frac{\ln N}{6}\right)$ time (assuming constant table look-ups) to find the order of the point $[s] \cdot P$, where P is randomly picked from $E(\mathbb{Z}_N)$. Instead, the curve-order finding method of Subsection 6.10.2 which exploits the leakage provided by the certificate requires a time-space trade-off of $\exp\left(\frac{\ln N}{12}\right)$.

It follows that factoring N using the GNFS factorization algorithm is easier in terms of asymptotic time complexity as long as $N > \approx 2^{3550}$ (under the unrealistic assumption to have access to $\approx 2^{148}$ memory units with constant-time look-ups). Thus, at least with respect to the two attacks outlined above, the structure induced by our semiprimality certificate generation methods does not seem to decrease the bit-security of returned semiprimes when these are of a cryptographic size.

6.10.4 Cryptanalysis of Reble's Semiprimality Certificate

As we already briefly discussed in Subsection 6.8.2, in 2005 Don Reble published a semiprimality certificate for a 1084-digits integer N [Don05]. His certificate consists of a tuple (N, A, s, t, Z) where: i) s is a prime greater than $(\sqrt[6]{N}+1)^2$; ii) t is an integer of size approximately $\sqrt[3]{N}$; iii) $A \in \mathbb{Z}_N$ and $E(\mathbb{Z}_N) : y^2 = x^3 + A \cdot x$; iv) $Z \in E(\mathbb{Z}_N)$ is so that $[2st] \cdot Z = \mathcal{O} \in E(\mathbb{Z}_N)$; v) $[2t] \cdot Z$ and $[st] \cdot Z$ are both not equal to $\mathcal{O}$.

It follows that $Q = [2t] \cdot Z$ is such that $[s] \cdot Q = \mathcal{O}$, and thus N is semiprime. For completeness, we report the relevant values of such certificate:

$$\begin{aligned} N = {} & 2354024638195369096484615970884339820364456147543619758078791036 \\ & 6016835505494573077342501320913482376791383651754310523953671977 \\ & 8522630598306091311778015170184734195517820836859704827368514883 \\ & 2466980967826424129426918703837554365381987202581644055207294439 \\ & 2812834659892991483861033331192664713921736184439296656941684194 \\ & 9144589355450831214521115967827260963610250123042880750137421428 \\ & 7948209489922794049174568735277898089123328514098559487995775109 \\ & 5300647425162891558424879373241151669954799924038445682294400677 \\ & 7458824969177192912226967635528307876492581854466547687556545067 \\ & 7122533240801191691993850537069266814811421303138978077711478017 \\ & 7004871146513516017647370512958483733151403979970908037943150798 \\ & 5695354618849164417252142747095137525007700363418270382794214457 \\ & 6309122358369456491588427467710775884280408394754494151594511691 \\ & 9834042566389996135670147270228034728379156641389487953023534102 \\ & 0154105768197033084151473179374226307186150307934745502893756679 \\ & 4023056085249684891705541856417002502945739752918768387923426402 \\ & 5678162912225114657588289449733450131843630232962354579482 41 \end{aligned}$$

$$
\begin{aligned}
A \;=\;& 54215605853661760575365176565408275413482806064633150544493983184301833635136210016598760938722311811647182817835804813625008105\\
&77739546333144435634891456970207167485885553815621275289423418686965349501417292764025522504282905939725077989940637806955237317\\
&86743151513819101206776709281022382132576840497457739776448272069645008166677142960459788718162659436622279702820662483265681021\\
&80300234905324601952927033377106949538307171616848102159987024513518430482442521180768217033043668300078406219820843085440423824\\
&76767429467054237581896316781460722765264597549719474916836336846696696127030983119227169596167266712055532287744558983685737494\\
&09441340055386769082271805045041156428398245949824817830969066204162023068382112282498719629195618182388184559051503256367664066\\
&57995298287567195324927191929693117348419003384059344367533130406039881394710806359572947380521405825006200952774556404488943147\\
&42558729947025206994334790131727875489866483238383370271553876588656939010395067827348354880845222975715375243269936234055702863\\
&59414942260683044570070721448137577484661026699816478612597
\end{aligned}
$$

$$
\begin{aligned}
s \;=\;& 28928434216210803690615246524734063145625591441015675167162985567756555278474752546432692199987512881223864606750731693606113599430\\
&88760646061779639151531014861564460269548890347219312047804893454433778350500993413376794661644857620913328971255920678525805161\\
&61953301500892518877982770524680448030575538603256133577661891490966933096505500715518252514951552588998 9
\end{aligned}
$$

$$
\begin{aligned}
t \;=\;& 70323610117989232939059249931645122359843269434492413277191595877834782378359181617852483666894698095646842012107118775002443342\\
&24388017710878522065177469249327051428307658369525065414006937495014985645561297450677685295272162306321861055508578946764087780\\
&34790938106697782587142237972655979754214171881757654182542156756291757336604687303080348589058320621801
\end{aligned}
$$

Unfortunately, this certificate is vulnerable to the twisting attack outlined in Subsection 6.10.1, since $k = 4 \cdot t$, due to its size, seems to correspond to $|E(\mathbb{Z}_N)|/s^2$. In fact, if we attempt to compute $[k \cdot s] \cdot P$ for some random point P on the curve twist $\in \tilde{E}(\mathbb{Z}_n)$ given by a c so that $(\frac{c}{N}) = -1$, we quickly obtain the factors p, q of N as:

$$
\begin{aligned}
p \;=\;& 11720990044176604485656482285369732810060846386219272085536405872537609480115115853954545591430322476416636182723320762797474855\\
&55807859184694068616541473864415105593633927158177286722456993852220897470944544069812169358671313601174219473974189972313862484\\
&26021432728107395247210783794849906344922148675372750153154323991759587128290262415063090969439615092699624981254513801117217086\\
&81418118874268076687305476680921066115071942491901610394578884716528402831014158285785393084402661789426579443047023707509873739\\
&48019507352285634402921138312 61
\end{aligned}
$$

$$
\begin{aligned}
q \;=\; & 200838378782254013652474546523992885119436648977370512923301620453807984704796755362473805369481669236335495871721955539823827845656971407635199873386941295582613528886860458159891305742279812\\
& 772280504959629204641128045408565781752248737735981864241505000919240321695702679272316760274629633675852621363062342495323688040923768261759594342438464082084477170767532502558171140259680357\\
& 885849656918196880508244291275258005435663462849789291917798112410948654669762016036504142200427463387254364382680603491377968436127402932630579659919810901 81
\end{aligned}
$$

By computing Cornacchia's decomposition for p, q with respect to $D = -4$ (since E has j-invariant equal to 1728) we can quickly identify, using Theorem 6.3, the correct orders of $Z \in E(\mathbb{F}_p)$ and $Z \in E(\mathbb{F}_q)$, and thus further factor t as $t = t_p \cdot t_q$ with

$$
\begin{aligned}
t_p \;=\; & 202585974003536658329914886728106138969380927557252217111962442871390440379317656774349766705913683870769354946015802483333037813896616490272291024111634551302604335595267654027056 9\\
t_q \;=\; & 347129708578746689361886910971547231806474847120278693278588502230006888376917081833433931070508841051996610596466418096941472788287215322127563236918821757826297680215636403433729
\end{aligned}
$$

6.11 Conclusions

In this Chapter we described a technique to construct primes p for which we can explicitly construct an elliptic curve $E(\mathbb{F}_p)$ such that its order completely factors over a certain factor base. We showed how this method could be used to backdoor one or more factors of an integer N so that an attacker can quickly factor it. To show the practicality of our methods, we implemented our procedures, and we detailed a complete attack example on a 1024-bits RSA modulus, previously backdoored with our implementation and factored in just a few seconds.

We then formalized semiprimality certificates that, based on a result by Goldwasser and Kilian, allow proving semiprimality of an integer with no need to share any of its factors, and we discussed how such certificates could be possibly computed. We described how our prime backdooring procedure could be used to construct prime factors of a semiprime N so that it is easy to compute semiprimality certificates for it, and we ported this construction in the MPC setting by sketching a protocol that allows distributed generation of certifiable semiprimes.

Lastly, we analyzed the security of semiprimality certificates, and we provided different attacks. We concluded that, compared to generic factorization algorithms, the expected bit-security of semiprimes built according to our protocols is the same as the security provided by random semiprimes of the same size.

Chapter 7

Breaking the $IKEp182 Challenge

7.1 Introduction

Under the threat of quantum computers appearing in the near future, public-key cryptography has to evolve to keep modern communication protocols secure. To foster the evolution, NIST organizes a competition for Post-Quantum Cryptography Standardization (PQC) [Nat22]. SIKE [Jao+20] (Supersingular Isogeny Key Encapsulation) is one of the alternative candidates of the ongoing 3rd round. It is based on the SIDH protocol (Supersingular Isogeny Diffie-Hellman) developed by De Feo

and Jao [JD11], following and improving the ideas of the constructions proposed by Rostovtsev and Stolbunov [RS06; Sto10]. *Isogeny*-based cryptography only recently started to develop rapidly.

In particular, for a specially shaped prime p, the security of SIKE relies on the hardness of finding an isogeny between two given supersingular elliptic curves defined over the finite field $\mathbb{F}_{p^2}$. The classic meet-in-the-middle attack (MitM, also known as bidirectional search), applied in the isogeny setting by Galbraith [Gal99], requires $\mathcal{O}(p^{1/4})$ time and memory/storage. Adj, Cervantes-Vazquez, Chi-Domínguez, Menezes and Rodríguez-Henríquez [Adj+19] observed that large amounts of storage are likely impossible to be achieved in practice due to fundamental physical constraints. They thus applied the classic low-memory van Oorschot-Wiener (vOW) golden collision search [vW99] to the isogeny setting by using less memory at the expense of more time, and conjectured that this attack represents the main threat to SIKE. Improved analysis of the application of van Oorschot-Wiener to SIKE with further optimizations was given by Costello, Longa, Naehrig, Renes and Virdia [Cos+20]. Based on this analyses, Longa, Wang and Szefer [LWS21] estimated the real costs of mounting such attack at various security levels, concluded that previous security estimates were conservative, and proposed to revise parameters in order to improve efficiency. For example, they propose to replace SIKEp434 with SIKEp377, which is 40% faster, while still targeting to satisfy NIST Level 1 security requirements.

In order to motivate security analysis of SIKE, Microsoft recently published two challenges [Mic21b] with reduced-size instances of SIKE: $IKEp182 and $IKEp217, with bounties of $5000 and $50 000, respectively. In this Chapter, we will describe how we managed to break the first of these two instances using the HPC facilities of the University of Luxembourg [Var+14]. While the classic security of $IKEp182 via the meet-in-the-middle attack is only about 45 bits, such amount of memory (2^{45} storage units $\geq$ 256TiB) is not trivial to manage efficiently. Nonetheless, we chose to stick to MitM instead of vOW due to the large overheads introduced by the latter, where, for example, a single step requires computing expensive isogenies (which can instead be amortized in MitM), and large penalties are paid to reduce the memory usage. Our implementation is mainly written in SageMath [The21] and C++, using parts of the SIDH library by Microsoft Research [Mic21a].

7.1.1 Our Approach

At the high level, we used the classic meet-in-the-middle approach for solving the isogeny path problem, in which the hardness of SIKE lies. We developed and applied several optimizations, which can be summarized as follows:

- **2-bit leak from the knowledge of the final curve.** In [Cos+20], it was noted that the final curve (i.e., the image of the initial curve through a secret 2^e-isogeny) fully leaks the last 4-isogeny. This effectively reduces the set of j-invariants that can be reached from the final curve by a factor of 4. In addition, we show how to express this reduced set in the same form as the set of j-invariants reached from the initial curve. This simplifies the MitM application to SIKE by unifying the representation of sets arising from the initial and the final curves. In the case of $IKEp182, both sets have 2^{44} elements after applying this and the following optimization.

- **1-bit conjugation-based reduction.** In SIKE, the initial Montgomery curve is $y^2 = x^3 + 6x^2 + x$, and by being defined over $\mathbb{F}_p$, all the curves 2^e-isogenous over $\mathbb{F}_{p^2}$ to it (through SIKE isogenies), have j-invariants which can be grouped

in conjugate pairs. It is thus sufficient to search for a collision of, e.g. the real part of the j-invariants in the middle to halve the set size arising from the initial curve. Recovering the full colliding j-invariant from such partial collision is easy since paths to conjugate elements are element-wise conjugates. This technique was discovered and applied in the vOW setting in [Cos+20].

- **Efficient tree exploration and optimal strategy.** A direct application of meet-in-the-middle with the (optimized) arithmetic from SIKE, would recompute a lot of intermediate steps repeatedly (simply speaking, computing each entry in the middle would require following a full path from the root of a full binary tree to its leaf). However, these computations are not identical and can not be avoided by simply storing some intermediate values. We show how to explore the tree more efficiently and adapt the optimal isogeny evaluation strategy of [DJP11] to this case.

- **Disk-based storage and sorting.** It is much more feasible to obtain and use a large amount of disk-based storage than a similar amount of RAM memory. However, the classic meet-in-the-middle formulation uses a (hash-)table where most queries follow a random access pattern suitable for RAM. When disk storage is used, latency represents the bottleneck of using hash-tables and limits the application of parallelization. To counter this, we follow an alternative approach to implement the MitM attack: we generate the two large j-invariants sets arising from the starting and the final curves, and we intersect them using *sorting* and *merging* techniques, which, instead, require mostly a sequential access pattern.

- **Storage-collision trade-off and compression.** Truncating intermediate entries (j-invariants representations) permits reducing storage requirements at the cost of allowing false-positive collisions. By omitting all the auxiliary information (e.g. the path in the set to an entry), we can reduce the storage further at the cost of an extra recomputation step, where the two sets are recomputed (fully memoryless and in parallel) in order to retrieve the relevant auxiliary information for collisions found in the previous step. Furthermore, the resulting sets become *dense* due to the truncation of entries and can be compressed (when sorted) by storing the differences between successive elements. In our case, we used 64-bit entries, which already at 32GiB of sorted data (2^{32} truncated entries) have the expected difference of about 32 bits. This reduces the total storage requirements down to approximately $2^{44} \times 2 \times 4$ bytes = 128 TiB.

7.1.2 Outline

In Section 7.2 we provide the theoretical background required to state the SIKE key-encapsulation protocol and the security assumptions on which it relies, while in Section 7.3 we describe how a Meet-in-the-Middle approach can be applied to SIKE to recompute one party's secret isogeny and ultimately recover parties' exchanged secret. In Section 7.4 we detail how we can efficiently generate *isogeny trees*, walked during the execution of the Meet-in-the-Middle, that, together with some further optimizations we discuss in Section 7.5, will allow us to reduce by a few orders the overall time and space complexity of the attack. In Section 7.6 we provide full cryptanalysis of Microsoft's $IKEp182 challenge, which we solve by implementing our optimized Meet-in-the-Middle attack, while in Section 7.7 we briefly discuss how our techniques might be applied to solve the next bigger unsolved $IKEp217 challenge.

7.2 Preliminaries

7.2.1 The Supersingular Isogeny Graph

In this Section we will briefly recall some standard algebraic facts relevant to our attack. We start by defining the main subject of this Chapter: *isogenies*.

Definition 7.1. *An isogeny of elliptic curves $\phi : E \to E'$ defined over $\mathbb{F}_q$ is a surjective morphism of curves that induces a group homomorphism $E(\overline{\mathbb{F}}_q) \to E'(\overline{\mathbb{F}}_q)$. When such map exists, E and E' are said to be isogenous over $\mathbb{F}_q$.*

An isogeny of elliptic curves $\phi : E \to E'$ defined over $\mathbb{F}_q$ can be represented as a non-constant rational map fixing the identity, i.e.,

$$\phi : \quad (x, y) \mapsto \left(\frac{a(x)}{c(x)}, \frac{b(x)}{d(x)} y\right)$$

with $a(x), b(x), c(x), d(x) \in \mathbb{F}_q[x]$ and $\gcd(a(x), c(x)) = \gcd(b(x), d(x)) = 1$. The *degree* of ϕ is defined as $\max(\deg a(x), \deg c(x))$ and ϕ is said to be *separable* if $\left(\frac{a(x)}{c(x)}\right)' \neq 0$. For every separable degree-d isogeny $\phi : E \to E'$, there exists a dual degree-d isogeny $\hat{\phi} : E' \to E$ so that the maps $\phi \circ \hat{\phi} = [d]_E$ and $\hat{\phi} \circ \phi = [d]_{E'}$ are the multiplication-by-d endomorphisms on E and E', respectively.

If d is composite, it is possible to decompose a degree-d isogeny, or simply a d-isogeny, into a composition of isogenies of prime order. We note that this property allows, in practice, to compute efficiently high (smooth) degree isogenies. More precisely, if $d = p_0^{e_0} \cdot \ldots \cdot p_n^{e_n}$ and ϕ is a d-isogeny, then there exists p_i-isogenies ϕ_{p_i}, with $i \in [0, n]$ so that

$$\phi = \overbrace{\phi_{p_0} \circ \ldots \circ \phi_{p_0}}^{e_0} \circ \ldots \circ \overbrace{\phi_{p_n} \circ \ldots \circ \phi_{p_n}}^{e_n}$$

It is well known that separable isogenies $\phi : E(\mathbb{F}_q) \to E'(\overline{\mathbb{F}}_p)$ (up to isomorphism) are in bijections with subgroups G of $E(\overline{\mathbb{F}}_p)$ so that $\ker(\phi) = G$ and ϕ is a $|G|$-isogeny: in such case, the curve E' is isomorphic to the group quotient E/G.

In the following, we will consider only separable isogenies over Montgomery elliptic curves.

Definition 7.2 (Montgomery Elliptic Curves)**.** *An elliptic curve over a finite field $\mathbb{F}_q$ is said Montgomery of parameters $A, B \in \mathbb{F}_q$ if it has equation $E_{A,B} : By^2 = x^3 + Ax^2 + x$ with $B(A^2 - 4) \neq 0$.*

The j-invariant of a Montgomery elliptic curve $E_{A,B}(\mathbb{F}_p)$ is equal to $j(E_{A,B}) = \frac{256(A^2-3)^3}{A^2-4}$. Hence, the $\overline{\mathbb{F}}_p$-isomorphism class[1] of $E_{A,B}(\mathbb{F}_p)$ depends only on A^2.

Through a simple change of variables, Montgomery curves $E_{A,B}$ are isomorphic over $\mathbb{F}_q$ to only one of the following two quadratic twists

$$E_{A,B} \simeq \begin{cases} y^2 = x^3 + Ax^2 + x & \text{if } B \in \mathbb{F}_q \text{ is a square} \quad (7.1) \\ uy^2 = x^3 + Ax^2 + x & \text{otherwise} \quad (7.2) \end{cases}$$

where u is a non-square of $\mathbb{F}_q$.

Supersingular elliptic curves (i.e. curves with trace congruent to 0 mod p), have their j-invariant defined over $\mathbb{F}_{p^2}$ [Sil09, V - Theorem 3.1.a]: in fact, any supersingular curve is isomorphic to an elliptic curve defined over $\mathbb{F}_{p^2}$, and we can thus consider

[1] Two elliptic curves are $\overline{\mathbb{F}}_p$-isomorphic if they have the same j-invariant.

only supersingular curves over $\mathbb{F}_{p^2}$. Moreover, the property of being supersingular is invariant under isogeny, and is induced by curves' j-invariants: if there is a supersingular curve with j-invariant equal to j, then j is said to be a *supersingular j-invariant* and all curves having j as j-invariant are supersingular too.

Definition 7.3 (Supersingular Isogeny Graph)**.** *For p, ℓ distinct primes, the degree-ℓ supersingular isogeny graph over $\mathbb{F}_{p^2}$ is the graph where vertexes are curves' representatives of $\mathbb{F}_{p^2}$-isomorphism classes, and two vertexes are connected by an (undirected) edge if and only if there exists a separable ℓ-isogeny between them.*

By Hasse's bound, supersingular curves E over $\mathbb{F}_{p^2}$ have $\#E(\mathbb{F}_{p^2}) = p^2 + 1 - t$ number of points, where the trace t can be equal only to $0, \pm p, \pm 2p$. Since by Tate's Isogeny theorem [Tat66], two curves are isogenous over $\mathbb{F}_q$ if and only if have the same number of points over $\mathbb{F}_q$, it follows that separable ℓ-isogenies over $\mathbb{F}_{p^2}$ partition the ℓ-degree Supersingular Isogeny graph over $\mathbb{F}_{p^2}$ into multiple connected subgraphs, each connecting curves' representatives of same trace.

In [AAM19, Theorem 6] is proved that the two subgraphs associated to traces $2p$ and $-2p$ are isomorphic, which in turn are isomorphic to the ℓ-degree supersingular isogeny graph built considering $\overline{\mathbb{F}}_p$-isomorphism classes instead. This fact suggests that we can equivalently (in terms of security) work in any of these two $O(p)$ size subgraphs induced by supersingular curves with traces in $\pm 2p$ (equivalently, by curves of cardinality $(p \pm 1)^2$), moving between neighbour representatives using ℓ-isogenies. Interestingly, from the fact that for a curve $E(\mathbb{F}_{p^2})$ and an $\ell \nmid p$, we have $E[\ell] \simeq \mathbb{Z}_\ell \times \mathbb{Z}_\ell$ [Sil09, III - Theorem 6.4b], it follows that supersingular curves E belonging to classes in these two subgraphs, decomposes as, depending on E cardinality, $E \simeq \mathbb{Z}_{p\pm1} \times \mathbb{Z}_{p\pm1}$. In particular, curves coincide with their $(p \pm 1)$-torsions, which imply that the latter are $\mathbb{F}_{p^2}$-rational.

By adopting Montgomery curves, it is possible to simplify this setting further. If, for a supersingular j-invariant j_0, we have $j_0 = j(E_{A,1})$, then clearly its quadratic twist satisfies $j_0 = j(E_{A,u})$. However, if we exclusively use the efficient Montgomery x-coordinate only arithmetic [Mon87b; CS18] (which employs the curve A-coefficient only), the twist selected becomes irrelevant[2], since it affects only the y-coordinate, and we can then represent Montgomery curves $E_{A,B}$ simply as E_A.

It follows that, in practice, by using Montgomery curves and x-coordinate arithmetic only, the two isomorphic supersingular isogeny subgraphs corresponding to the traces $\pm 2p$, coincide, and vertexes can be denoted with just supersingular j-invariants rather than with isomorphism class curve representative.

Definition 7.4 (Supersingular Isogeny Graph - Revisited)**.** *For p, ℓ distinct primes, the degree-ℓ supersingular isogeny graph over $\mathbb{F}_{p^2}$ is the graph where vertexes are supersingular j-invariants, and two vertexes (j_1, j_2) are connected by an (undirected) edge if and only if there exists a separable ℓ-isogeny between two Montgomery curves E_{A_1} and E_{A_2} so that $j_1 = j(E_{A_1})$ and $j_2 = j(E_{A_2})$.*

Definition 7.5 ($\mathcal{J}_A^d$-set)**.** *Let E_A be a supersingular Montgomery elliptic curve over $\mathbb{F}_{p^2}$ and let d be so that $\#E_A(\mathbb{F}_{p^2}) = d \cdot r$ with $\gcd(d, r) = 1$. The $\mathcal{J}_A^d$-set is then*

$$\mathcal{J}_A^d = \left\{ j(E_{A'}) \in \mathbb{F}_{p^2} \mid \exists \text{ a separable } d\text{-isogeny } \phi : E_A \to E_{A'} \right\}$$

i.e., the set of j-invariants of curves d-isogenous to E_A.

[2] In [Cos20] it was noted that, in fact, it is possible to build isogeny-based schemes over Montgomery curves which are *"twist-agnostic"*, that is can work independently of curves' quadratic twist chosen.

We note that due to the existence of dual isogenies, edges in an isogeny graph are undirected. In the case of supersingular curves, the degree-ℓ isogeny graph over $\mathbb{F}_{p^2}$ has approximately $\left\lfloor \frac{p+1}{12} \right\rfloor$ vertexes [Sch87, Theorem 4.6] and each vertex has exactly $\ell+1$ neighbours (counting multiplicities), with edges corresponding to an isogeny with kernel being a distinct order-ℓ subgroup of the torsion $\mathbb{Z}_\ell \times \mathbb{Z}_\ell$. In other words, the degree-ℓ supersingular isogeny graph is a connected $(\ell+1)$-regular graph which results to be Ramanujan (see [Piz90; Piz98]). It follows that random length-e walks in the supersingular isogeny graphs correspond to ℓ^e-isogenies between supersingular elliptic curves.

By exploiting the correspondence between order-ℓ kernels and ℓ-isogenies, a walk in the supersingular isogeny graph starting from a curve E, can be expressed in terms of a linear combination of two independent generators of the torsion $E[\ell]$.

Definition 7.6 (Walk)**.** *Let E_0 be a supersingular elliptic curve over $\mathbb{F}_{p^2}$, ℓ a prime distinct from p and let (P_0, Q_0) be two independent generators of $E_0[\ell^e] = \mathbb{Z}_{\ell^e} \times \mathbb{Z}_{\ell^e}$. Two values $a, b \in \mathbb{Z}_{\ell^e}$ not simultaneously divisible by ℓ, define a separable ℓ^e-isogeny $\phi = \phi_{e-1} \circ \ldots \circ \phi_0 : E_0 \to E_e$ over $\mathbb{F}_{p^2}$ (i.e., a walk in the supersingular isogeny graph), where, for $i \in [0, e-1]$, $\phi_i : E_i \to E_{i+1}$ is an ℓ-isogeny with $\ker(\phi_i) = \langle [\ell^{e-1-i}] \cdot ([a]P_i + [b]Q_i) \rangle$ and $(P_{i+1}, Q_{i+1}) = (\phi_i(P_i), \phi_i(Q_i))$. We will often refer to such ϕ as the isogeny arising from $[a]P + [b]Q$.*

Remark 7.1. *If $\ell \nmid a$, then $\langle [a]P + [b]Q \rangle = \langle P + [s]Q \rangle$, with $s = a^{-1}b \in \mathbb{Z}_{\ell^e}$, and such subgroups give rise to ℓ^e distinct isogenies. If instead $a = \ell \cdot c$, kernels can be written as $\langle [s\ell]P + Q \rangle$, with $s = b^{-1}c \in \mathbb{Z}_{\ell^e}$ and there exists at most ℓ^{e-1} such distinct subgroups. This brings the total number of walks that can be traversed from a starting curve E_0 to $\ell^{e-1}(\ell+1)$, which in turn correspond to all walks obtained by iteratively exploring all $\ell+1$ neighbours of E_0 up to depth e (with no backtracking). Kernels of the form $\langle P + [s]Q \rangle$, with $s \in \mathbb{Z}_{\ell^e}$, will be the ones employed by SIKE (Subsection 7.2.2): we note that this choice restricts the possible isogeny-paths that can be walked since only ℓ out of $\ell+1$ neighbours of E_0 can be explored.*

The main observation that ensures correctness of Definition 7.6 is that the order of $\phi_i([a]P_i + [b]Q_i)$ decreases by ℓ with respect to the order of $[a]P_i + [b]Q_i$: indeed, from $\ker(\phi_i) = \langle [\ell^{e-1-i}] \cdot ([a]P_i + [b]Q_i) \rangle$ we must have $[\ell^{e-1-i}] \cdot \phi_i([a]P_i + [b]Q_i) = \mathcal{O}_{E_{i+1}}$ and since P_0, Q_0 both have order ℓ^e, by induction, we can conclude that $\phi_i([a]P_i + [b]Q_i)$ has order ℓ^{e-1-i}.

The difficulty to obtain the scalar s from two curves E and E' isogenous through the ℓ^e-isogeny arising from $P + [s]Q$, is one of the different (formulations of) problems which are believed to be hard in the supersingular isogeny setting.

Problem 7.1 (Path-finding)**.** *Given two supersingular Montgomery curves E_A and $E_{A'}$ over $\mathbb{F}_{p^2}$ so that, for an $\ell \nmid p$ prime and $e > 0$, there exists a separable ℓ^e-isogeny $\phi : E_A \to E_{A'}$ over $\mathbb{F}_{p^2}$ (equivalently $j(E_{A'}) \in \mathcal{J}_A^{\ell^e}$), find a sequence of groups $\{K_i\}_{i \in [1,e]}$ such that*

- *ϕ_i is a separable ℓ-isogeny defined over $\mathbb{F}_{p^2}$ with $\ker(\phi_i) = K_i$;*
- *$\phi = \phi_1 \circ \ldots \circ \phi_e$ up to isomorphism.*

We note that a solution to Problem 7.1 for an ℓ^e-isogeny arising from $P + [s]Q$ can be efficiently mapped bit-by-bit to the corresponding generating secret value s, for example, by checking the visited curves' j-invariants.

7.2.2 The SIKE Protocol

Supersingular Isogeny Key Encapsulation (SIKE) [Jao+20] is a post-quantum key encapsulation mechanism (KEM) based on the difficulty to find a length-e path between two ℓ^e-isogenous elliptic curves (Problem 7.1). It is based on the Supersingular Isogeny Diffie-Hellman (SIDH) [JD11] key exchange.

In SIKE, p has the form $p = 2^{e_A}3^{e_B} - 1$ with $2^{e_A} \approx 3^{e_B}$ and the working field is set to be $\mathbb{F}_{p^2} = \mathbb{F}_p(i) = \mathbb{F}_p[x]/(x^2+1)$. The parameters e_A and e_B are chosen so that the Montgomery curve $E = E_6$ over $\mathbb{F}_{p^2}$ is supersingular with $(p+1)^2$ rational points and torsions $E[\ell_A^{e_A}] = \mathbb{Z}_{\ell_A^{e_A}} \times \mathbb{Z}_{\ell_A^{e_A}} = \langle P_A, Q_A \rangle$ and $E[\ell_B^{e_B}] = \mathbb{Z}_{\ell_B^{e_B}} \times \mathbb{Z}_{\ell_B^{e_B}} = \langle P_B, Q_B \rangle$. To avoid some technicalities introduced by adopting efficient 2-isogeny computation formulas, the order-2 point $(0,0)$ is not allowed to be into any 2-isogeny kernel in a path arising from $P_A + [s]Q_A$ with $s \in \mathbb{Z}_{2^{e_A}}$: thanks to a result of Renes [Ren18, Corollary 2], this is guaranteed by choosing the generators P_A, Q_A of the torsion $E[2^{e_A}]$ so that $[2^{e_A-1}]Q_A = (0,0)$.

Once the public parameters $(p, E(\mathbb{F}_{p^2}), P_A, Q_A, P_B, Q_B)$ are generated, two parties, Alice and Bob, can agree on a common secret as follows:

- Alice picks secret $s_A \leftarrow_\$ \mathbb{Z}_{2^{e_A}}$ and computes the 2^{e_A}-isogeny $\phi_A : E \to E_A$ arising from $\langle P_A + [s_A] \cdot Q_A \rangle$. She then sends to Bob E_A and the points $\phi_A(P_B)$, $\phi_A(Q_B)$.
- Bob picks secret $s_B \leftarrow_\$ \mathbb{Z}_{3^{e_B}}$ and computes the 3^{e_B}-isogeny $\phi_B : E \to E_B$ arising from $\langle P_B + [s_b] \cdot Q_B \rangle$. He then sends to Alice E_B and the points $\phi_B(P_A)$, $\phi_B(Q_A)$.
- Alice computes the 2^{e_A}-isogeny $\phi_{\tilde{A}} : E_B \to E_{BA}$ arising from $\langle \phi_B(P_A) + [s_A] \cdot \phi_B(Q_A) \rangle$ and sets the common secret to $j(E_{BA})$.
- Bob computes the 3^{e_B}-isogeny $\phi_{\tilde{B}} : E_A \to E_{AB}$ arising from $\langle \phi_A(P_B) + [s_B] \cdot \phi_A(Q_B) \rangle$ and sets the common secret to $j(E_{AB})$.

It is easy to see that since separable isogenies correspond to curve quotients, in this setting, they commute, and so $j(E_{BA}) = j(E_{AB})$. For more details and proof of correctness of the above protocol, we refer to [JD11; Jao+20].

7.2.3 Efficient Isogeny Computation

This Section provides an overview of how isogenies, and thus walks in the isogeny graph, can be practically and efficiently computed.

We will focus on ℓ-isogenies with $\ell = 2, 3$, relevant for SIKE and our attacks. Proofs that the following formulas define isogenies can be found, for example, in [CH17; Ren18].

Proposition 7.1 (2-isogeny)**.** *Let $E_{A,B}$ be a Montgomery supersingular elliptic curve over $\mathbb{F}_{p^2}$ with $p \neq 2$ and let $R = (x_R, y_R) \in E(\mathbb{F}_{p^2})$ be an order 2 point not equal to $(0,0)$. Then*

$$\begin{aligned} \phi : E_{A,B} &\longrightarrow E_{A',B'} \\ (x,y) &\longmapsto (f(x), yf'(x)) \end{aligned}$$

with

$$f(x) = x\frac{x \cdot x_r - 1}{x - x_r}$$

is a separable 2*-isogeny between Montgomery elliptic curves with* $\ker(\phi) = \langle R \rangle$ *and* $(A', B') = (2(1 - 2x_r^2), Bx_r)$.

Remark 7.2. *The* 2*-isogeny defined in Proposition 7.1 fixes the point* $(0, 0)$*, and thus cannot belong to its kernel.*

Proposition 7.2 (3-isogeny)**.** *Let* $E_{A,B}$ *be a Montgomery supersingular elliptic curve over* $\mathbb{F}_{p^2}$ *with* $p \neq 2$ *and let* $R = (x_R, y_R) \in E(\mathbb{F}_{p^2})$ *be an order* 3 *point. Then*

$$\begin{aligned} \phi : E_{A,B} &\longrightarrow E_{A',B'} \\ (x, y) &\longmapsto (f(x), yf'(x)) \end{aligned}$$

with

$$f(x) = x\frac{x \cdot (x_R - 1)^2}{(x - x_R)^2}$$

is a separable 3*-isogeny between Montgomery elliptic curves with* $\ker(\phi) = R$ *and* $(A', B') = (-6x_R^3 + Ax_R^2 + 6x_R, Bx_R^2)$.

7.2.4 Walk structure induced by SIKE 2-isogenies

The structure induced by the 2-isogeny formulas adopted by SIKE is relevant for the attack we will outline in the following Sections.

It is easy to see that in the $\mathbb{F}_{p^2}$-isomorphism class of a supersingular j-invariant j_0, we have (at most) 6 distinct Montgomery curves: if $\pm A$ satisfy the equation $j_0 = \frac{256(x^2-3)^3}{x^2-4}$, then also

$$\pm B = \frac{3\tilde{x} + A}{\sqrt{\tilde{x}^2 - 1}} \qquad \pm C = \frac{3\tilde{z} + A}{\sqrt{\tilde{z}^2 - 1}}$$

do, where $\tilde{x}, \tilde{z} = 1/\tilde{x}$ are roots of $x^2 + Ax + 1 = 0$.

When these 6 coefficients are all distinct, a 2-isogeny as in Proposition 7.1 can walk in the supersingular isogeny graph to only 2 of the possible 3 neighbour j-invariants j_1, j_2, j_3, and whose values depend on the A-coefficient of the curve to which we are applying the isogeny.

As already noted in Remark 7.2, by using SIKE 2-isogenies, we cannot have $\langle (0, 0) \rangle$ as kernel: this practically correspond to the fact that if a curve E_B, with $j(E_B) = j_0$, is pushed through a 2-isogeny to $E_{A'}$, then $E_{A'}$ will not be pushed back to E_B by any of the 2-isogeny induced by an order-2 subgroup of $E_{A'}$ distinct from $\langle (0, 0) \rangle$.

In the $\mathbb{F}_{p^2}$-isomorphism class of $E_{A'}$, however, there will be 4 curves $E_{\pm B'}, E_{\pm C'}$ which can be pushed back to a curve in the isomorphism class of E_B (i.e., j_0), but not to the curve E_B itself, because, otherwise, there will be a 2-isogeny that will move E_B back to $E_{A'}$, a circumstance prevented by not allowing $\langle (0, 0) \rangle$ to be an isogeny kernel.

It follows that each of the 4 curves $E_{\pm B'}, E_{\pm C'}$ can be pushed to only one of the two isomorphic curves $E_{\pm A}$[3], which will eventually be pushed further to nodes j_3, j_2 distinct from $j(E_{A'}) = j(E_{\pm B'}) = j(E_{\pm C'}) = j_1$.

This example is illustrated (with same notation) in Figure 7.1.

[3]Since, in SIKE, $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$, the map $(x, y) \mapsto (-x, iy)$ is an isomorphism between E_* and E_{-*}.

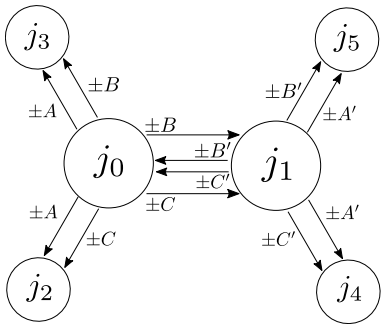


FIGURE 7.1: The different $j-$invariants reached by pushing curve's A-coefficients through 2-isogenies defined by Proposition 7.1. Here $\pm A, \pm B, \pm C$ are the 6 roots satisfying $j_0 = \frac{(x^2-3)^3}{(x^2-4)}$ (resp. $\pm A', \pm B', \pm C'$ and j_1), and define 6 Montgomery curves isomorphic over $\mathbb{F}_{p^2}$. The two edges associated to a certain coefficient represent isogenies with kernels order-2 subgroups not equal to $\langle (0,0) \rangle$.

7.3 The MitM Attack for Solving the Isogeny Path Problem

7.3.1 High-level Description

In this Section we will provide an overview of the meet-in-the-middle attack to solve the path-finding Problem 7.1. In terms of path search on a graph between two nodes, this approach is also known as *bidirectional search*.

In order to find a path of length e between two curves E_A and E_B in the supersingular isogeny graph (i.e., an ℓ^e-isogeny between E_A and E_B), an attacker can explore all length-$\lfloor e/2 \rfloor$ paths starting from E_A and all length-$\lceil e/2 \rceil$ paths starting from E_B (intuitively, this corresponds to exploring the subgraph *spheres* centered in E_A and E_B with radius $\lfloor e/2 \rfloor$ and $\lceil e/2 \rceil$, respectively) looking for a non-trivial intersection: since isogenies are defined up to isomorphisms, we can identify the curve(s) *in-the-middle* by computing their j-invariants.

The full path can then be reconstructed either by iteratively applying the same attack on the two found half-length sub-paths or by simply storing the paths starting in E_A or E_B associated with j-invariant in the middle and concatenating them once a collision is found.

Problem 7.2 (Meet-in-the-Middle)**.** *Given two ℓ^e-isogenous curves $E_A(\mathbb{F}_{p^2})$ and $E_B(\mathbb{F}_{p^2})$ for some $\ell \nmid p$ prime and $e > 0$, the Meet-in-the-Middle (MitM) problem asks to find the intersection*

$$\mathcal{J}_A^{\ell^{\lfloor e/2 \rfloor}} \cap \mathcal{J}_B^{\ell^{\lceil e/2 \rceil}}$$

7.3.2 Application to SIKE

In SIKE, MitM can be applied to attack either Alice's or Bob's public key: indeed, from Alice's public key, we can easily recompute the curve E_A that is 2^{e_A}-isogenous to the starting curve E, and, similarly, Bob's public key reveals the curve E_B that is 3^{e_B}-isogenous to the starting curve E. Explicitly finding the secret isogeny $\phi_A : E \to E_A$ or $\phi_B : E \to E_B$, allows the attacker to reapply it to the other party's public key to ultimately obtain the shared secret key.

As already noted in Remark 7.1, in SIKE not all $(\ell+1)\ell^{e-1}$ isogenies are possible, because isogeny kernels are restricted to the shape $\langle P+[s]Q\rangle$, which excludes in the first ℓ-isogeny step the kernel $\langle[\ell^{e-1}]Q\rangle$, leaving only ℓ^e isogenies.

In Subsection 7.5.1, we show that the isogeny formulas of Subsection 7.2.3 can be used to walk from the curve E_A towards the starting curve E, by moving to an isomorphic curve $E_{A'}$ and defining kernels as $P'+[s']Q'$ with $\langle P',Q'\rangle = E_{A'}[\ell^e]$.

This refines the meet-in-the-middle Problem 7.2 into generating and intersecting the leaves of the two "trees" of j-invariants spanned by *walks* from the bases $(P,Q)\in E(\mathbb{F}_{p^2})$ and $(P',Q')\in E_{A'}(\mathbb{F}_{p^2})$. The meet-in-the-middle trees structure for $\ell=2$ is illustrated in Figure 7.2.

Definition 7.7 (SIKE-tree)**.** *Given a curve E defined over $\mathbb{F}_{p^2}$ and a basis (P,Q) for its torsion $E[\ell^e]$, the* tree *spanned by (P,Q) of depth $d\le e$ is the directed graph consisting of all length-d walks from E arising from $[\ell^{e-d}]\cdot(P+[s]Q)$ with $s\in\mathbb{Z}_{\ell^e}$, i.e. all length-d walks from E excluding those arising from $[\ell\cdot a]P+[b]Q$ for any $a,b\in\mathbb{Z}_{\ell^e}$.*

Remark 7.3. *Since two different kernels may lead to the same image curve, the graph spanned by the ℓ^e-torsion generators (P,Q) may not always correspond to a tree. However, we assume this does not happen for simplicity of analysis, although such cases do not pose a problem in practice.*

In SIKE, a party computes a full ℓ^e-isogeny using an ℓ^e-torsion basis (P,Q). In other circumstances, like in tree computation or in the meet-in-the-middle attack, we need to compute only the initial part of such full walks: thus, to keep Definition 7.6 consistent, such full torsion basis needs to be re-scaled, so that the path length matches the desired one.

For a walk of length i, the re-scaling is done as

$$(P',Q') = ([\ell^{e-i}]P,[\ell^{e-i}]Q)$$

so that all length-i walks arising from $P'+[t]Q'$ with $t\in[0,\ell^i]$ will match the first i steps of length-e walks arising from $P+[s]Q$ with $s\in[0,\ell^e]$.

It follows that, to succeed in a meet-in-the-middle attack, it is crucial to generate trees (more precisely, their leaves) from curves.

Problem 7.3 (Tree generation)**.** *Given a supersingular curve E defined over $\mathbb{F}_{p^2}$ and an ℓ^e-torsion basis (P,Q) for it, compute the set of j-invariants of curves appearing as leaves in the depth-$d\le e$ tree spanned by (P,Q).*

7.4 Tree Generation Strategy

In this Section, we address how it is possible to generate leaves of the tree spanned by some torsion generators.

If in SIKE, a MitM attack relative to one party, e.g. Alice with her full torsion $E[\ell^{e_A}]$, succeeds, then the shared secret can be efficiently computed from the data she exchanged with Bob. It thus suffices to run the attack only on one full torsion and, for the sake of simplicity, we will hereafter address only the case $\ell=2$, i.e. attack $E[2^{e_A}]$. In this setting, we can take advantage of efficient 2-isogeny computation formulas and simpler tree generation and exploration formulation. However, we remark that the following discussion can be generalized to the $E[3^{e_B}]$ torsion as well.

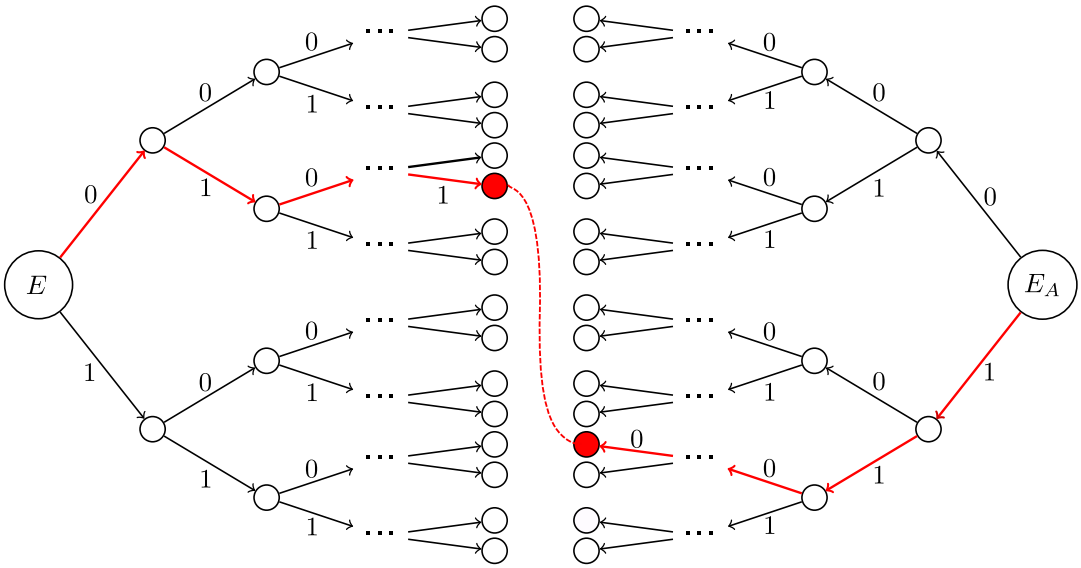


FIGURE 7.2: Example of 2-isogeny trees starting from the two 2^e-isogenous curves E and E_A. Red nodes in the middle denote curves with same j-invariant, whose respective path in the tree (in red) connect E_A to E_B. Edge labels are assigned arbitrarily in order to identify the paths.

A straightforward approach for generating a tree is to enumerate all possible $s \in [0, 2^e - 1]$ and compute the respective isogeny's image curve, similarly as done in SIKE for a given private key s. In fact, such walk computation is performed as a *single step* in the low-memory van Oorschot-Wiener collision search applied to SIKE [Adj+19; Cos+20; LWS21]. However, many intermediate curves will be visited multiple times for different s. More precisely, if two different s_0 and s_1 share the same k least significant bits of their binary representations, then the first k steps in the walks arising from $P + [s_0]Q$ and $P + [s_1]Q$ will be (partially) identical. To better understand the complexity of such a naive approach, a depth-e tree has $2^{e+1} - 1$ nodes and $2^{e+1} - 2$ edges, while here we would walk through $e2^e$ edges, a logarithmic slowdown (in the tree size) with respect to other tree exploration techniques. In addition, lower-depth edges are typically more expensive to compute due to the scalar multiplication required to obtain an order-2 point (e.g. in SIKE isogeny evaluation algorithms), increasing the performance gap further.

We also note that, although s_0 and s_1 share the first k bits, the initial kernel generator points pushed through the two walks differ, and therefore the computations done on the shared sub-walks are not fully identical and cannot be trivially avoided by adopting caching techniques in concrete implementations.

A better and more natural way to explore the tree is through a *depth-first traversal*, which also avoids a large memory footprint.

To better explain how it works, we will label tree nodes with the A_i coefficients of the corresponding curve E_{A_i}. Given a path from the starting node A_0 to the current node A_i, made of a composition of 2-isogenies, we explore the node A_i by generating its 2 children nodes, each identified by an order-2 point on E_{A_i} with nonzero x-coordinate (as we noted in Subsection 7.2.2, the remaining third order-2 point corresponding to $(0, 0)$ is automatically excluded, and this corresponds to the edge pointing backwards from a node towards the root). However, to the best of our knowledge, all known generic ways to iteratively compute coordinates of these order-2 kernel generators in a walk require extracting an expensive square root in $\mathbb{F}_{p^2}$.

Our goal is to avoid this heavy operation while maintaining 2 different order-2 kernels at each step (one for each possible child direction) on the way through the tree exploration. In addition, we want to take advantage of the efficient formulas for 2-isogenies, so we need to ensure that $(0,0)$ never appears as one of the kernel generators, similarly as in [Ren18, Corollary 2].

Our solution is based on a modified isogeny evaluation algorithm from SIKE. For a depth d node A_i, we store a basis P', Q' of $E_{A_i}[2^{e-d}]$, with the constraint that $[2^{e-d-1}]Q' = (0,0)$. The children nodes are then reached by the two isogenies corresponding to the order-2 points $[2^{e-d-1}]P'$ and $[2^{e-d-1}](P'+Q')$, respectively, for which it is ensured that none of them equals $(0,0)$, thus allowing efficient SIKE arithmetic implementation. As scalar multiplication by a large power of 2 is expensive (it is comparable to the field square root cost), we offset such operations by selectively pushing through isogenies few intermediate points, as done in SIKE, with the additional effort of ensuring they form a good basis. Finally, the optimal strategy - a trade-off between the number of point doublings and isogeny evaluations - can be computed using dynamic programming, similarly to how it was done in the SIDH paper by Jao, De Feo and Plût [JD11; DJP11], in order to compute the full walk up to a certain depth. Our general approach is easily parallelizable by distributing subtree generation tasks among available workers.

7.4.1 Maintaining Torsion Basis for Efficient Isogeny Computations

We now describe a method that allows us to maintain, during path traversals, a basis suitable for the efficient arithmetic formulas used by SIKE, i.e., the ones detailed in Subsection 7.2.3.

Proposition 7.3. *Let $A \in \mathbb{F}_{p^2}$ and $e \geq 2$. Let $P, Q \in E_A(\mathbb{F}_{p^2})$ be a basis of $E_A[2^e]$ with $[2^{e-1}]Q = (0,0)$. Then, for a 2-isogeny $\phi : E_A \to E_{A'}$ arising from $[2^{e-1}](P + [s]Q)$ with $s \in [0, 2^e - 1]$,*

1. *if $\ker\phi = \langle [2^{e-1}]P \rangle$, then $P', Q' \in E_{A'}(\mathbb{F}_{p^2})$ is a basis of $E_{A'}[2^{e-1}]$ with $[2^{e-2}]Q' = (0,0)$, where*

$$\begin{aligned} P' &= \phi(P), \\ Q' &= \phi([2]Q); \end{aligned}$$

2. *if $\ker\phi = \langle [2^{e-1}](P+Q) \rangle$, then $P', Q' \in E_{A'}(\mathbb{F}_{p^2})$ is a basis of $E_{A'}[2^{e-1}]$ with $[2^{e-2}]Q' = (0,0)$, where*

$$\begin{aligned} P' &= \phi(P+Q), \\ Q' &= \phi([2]P). \end{aligned}$$

Proof. Since P, Q are distinct generators and both have order 2^e, it follows that the 3 order 2 points $[2^{e-1}]P, [2^{e-1}]Q, [2^{e-1}](P+Q)$ generates the $2+1$ distinct subgroups of $E[2] = \mathbb{Z}_2 \times \mathbb{Z}_2$. Since $[2^{e-1}]Q = (0,0)$, the order-2 point $[2^{e-1}](P + [s]Q)$ appearing as a kernel for ϕ can only be equal to either $[2^{e-1}]P$ or $[2^{e-1}](P+Q)$. If $\ker\phi = \langle [2^{e-1}]P \rangle$ we immediately have $\phi([2^{e-1}]P) = \mathcal{O}_{E_{A'}} = [2^{e-1}]P'$ and since $\phi([2^{e-2}]P) \neq \mathcal{O}_{E_{A'}}$, $P' = \phi(P)$ must then be a generator of $E[2^{e-2}]$. Since 2-isogenies formulas arising from $P + [s]Q$ have the property to fix the point $(0,0)$ (see Remark 7.2), we then have $\phi([2]Q)$ has order 2^{e-1} and is such that $[2^{e-2}]\phi([2]Q) = \phi((0,0)) = (0,0)$.

Similarly, if $\ker\phi = \langle[2^{e-1}](P+Q)\rangle$, then $P' = \phi(P+Q)$ has order 2^{e-1}. It follows that $\phi([2^{e-1}]P) + \phi([2^{e-1}]Q) = \mathcal{O}_{E_{A'}}$, i.e. $[2^{e-2}]Q' = \phi([2^{e-1}]P) = -\phi([2^{e-1}]Q) = (0,0)$.

For P' and Q' to form a basis, we further need to show that $\langle P'\rangle \cap \langle Q'\rangle = \mathcal{O}_{E_{A'}}$. Let us assume, by contradiction, that there exists a non-trivial $R \in \langle P'\rangle \cap \langle Q'\rangle$: we then have, for certain $s, t \neq 0$, that $R = [s]P' = [t]Q'$ and thus $[s]P' - [t]Q' = \mathcal{O}_{E_{A'}}$. If $\ker\phi = \langle[2^{e-1}]P\rangle$, we then have that $[s]P - [t]Q$ is in $\ker\phi$ and thus $[2^{e-1}]P = [s]P - [2t]Q$. Since P, Q form a basis for $E_A[2^e]$, this in turn implies $s = t = 0$, a contradiction. A similar contradiction is reached also for the case $\ker\phi = \langle[2^{e-1}](P+Q)\rangle$. □

Our formulation can be used to straightforward map isogenies used to traverse tree nodes to binary strings: using a bit, we can represent the relation between the kernel used to walk a certain step and the (current) torsion generators (e.g. we associate "0" if $\ker\phi = \left\langle[2^{e-d-1}]\tilde{P}\right\rangle$ and "1" if $\ker\phi = \left\langle[2^{e-d-1}](\tilde{P}+\tilde{Q})\right\rangle$), as illustrated in Figure 7.2.

This allows us to easily reconstruct later from such binary strings[4] the full sequence of j-invariants traversed, which in turn can be easily mapped back to the value s whose walk arising from $P + [s]Q$ traverses the same j-invariants.

7.4.2 Optimal Strategies for the Doubling/Isogeny Evaluation Trade-off

During the evaluation of the isogeny walk arising from $P + [s]Q$, the order-ℓ kernel for the next step can be obtained with a scalar multiplication as $[\ell^{e-1}](P + [s]Q)$. To compute such kernels more efficiently, we can store some intermediate values $[\ell^{e_0-1}](P + [s]Q)$ with $e_0 < e$, and later push all such points through isogenies and scalar multiplications. Indeed, this allows to compute the kernel of the next-step ℓ-isogeny with just $e - 1 - e_0$ point multiplications by ℓ for the maximum e_0 for which $[\ell^{e_0-1}](P + [s]Q)$ is stored, while storing and pushing smaller multiples will be helpful in later steps. It is then clear the relevance of finding good trade-offs between the number of multiplications by ℓ and the number of isogeny evaluations needed to traverse a walk. Indeed, depending on the implementation adopted, these two operations have different costs.

In the extended version of [JD11], i.e.[DJP11], De Feo, Jao and Plût describe how to derive an *optimal evaluation strategy* for the best trade-off between scalar multiplications and isogeny evaluations, using the dynamic programming paradigm.

We now provide a brief overview of how optimal evaluation strategies are found in [DJP11]. Given a $K_0 \in E_A[\ell^e]$, let $\phi_i : E_{A_i} \to E_{A_{i+1}}$, with $i \in [0, e-1]$, be the sequence of isogenies on the length-e walk defined by K_0, and, for $i \in [1, e-1]$, let $K_i = \phi_{i-1}(K_{i-1})$: the goal is to compute $\ker\phi_i = \langle[\ell^{e-1-i}]K_i\rangle$ for all $i \in [0, e-1]$ in a minimum overall cost in terms of scalar multiplications and isogeny evaluations.

Aiming at this, we construct a directed graph with nodes

$$\left\{\ [\ell^i]K_j \ \mid\ j \in [0, e-1], i \in [0, e-1-j]\ \right\},$$

connected by two types of edges, namely:

- *multiplication by* $[\ell]$ *edges* of cost C_{mult}, connecting $[\ell^i]K_j$ to $[\ell^{i+1}]K_j$, for $i + j + 1 \leq e - 1$;

[4] With some abuse of notation, we will often refer to such binary strings as *paths*.

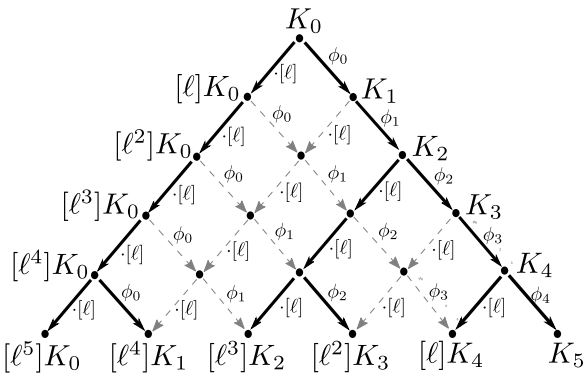


FIGURE 7.3: An example of evaluation strategy graph. Multiplication by $[\ell]$ edges ($\swarrow$) and isogeny evaluation edges ($\searrow$) transform $K_0 \in E[\ell^6]$ to the leaf values $\{[\ell^{6-i-1}]K_i\}_{i\in[0,5]}$, needed to compute the walk arising from K_0. In bold, an optimal evaluation strategy assuming $C_{eval} = 1.5 \cdot C_{mult}$ (verified experimentally).

- *isogeny evaluation edges* of cost C_{eval}, connecting $[\ell^i]K_j$ to $[\ell^i]K_{j+1}$ (through an ℓ-isogeny ϕ_j), for $i + j + 1 \leq e - 1$.

A strategy for evaluating all the $\ker \phi_i = \langle [\ell^{e-1-i}]K_i \rangle$ can then be described by a tree subgraph in this graph, rooted in K_0 and consisting of directed paths towards the goal leaf nodes $[\ell^{e-1-i}]K_i$ for $i \in [0, e-1]$. The cost of a strategy is then the sum of the edges' costs, counting only once edges traversed by multiple paths. It is then clear that the best strategies are those in which paths to leaves overlap as much as possible. An example of such a graph along with an optimal strategy is illustrated in Figure 7.3

In [DJP11], it is shown that there exist minimal-cost strategies with recursive structures. The problem is decomposed into two subproblems: the subgraph induced after following a multiplication edges and the subgraph induced after following $e - a$ isogeny evaluation edges. This is possible because, in paths towards leaves, the order of any two consequent edges can be swapped (if it does not break strategy consistency), since multiplication commutes with isogenies and such swaps do not change the overall strategy cost. An optimal strategy can thus be obtained by evaluating all possible choices of a and solving the induced subproblems recursively. Since the subproblems are fully characterized by their size (and are independent of the root kernel chosen), their solutions can be cached and reused (dynamic programming).

7.4.3 Application to Tree Generation

In order to make path computations faster, we are interested in applying best strategies during tree generation.

Unfortunately, we cannot use them straightforwardly: indeed, the tree generation and a simple isogeny evaluation differ because, in the former, each isogeny evaluation edge creates ℓ new exploration nodes deeper in the tree. However, all the ℓ induced sub-trees differ only by curves and generators, so all can follow the same sub-strategy. Effectively, an isogeny evaluation edge *multiplies* the number of nodes being explored in the isogeny tree by ℓ. To account for this, we can then set the weight of an isogeny evaluation edge ϕ_j to ℓ^{j+1}. At the same time, we assign to multiplication edges $[\ell^i]K_j \to [\ell^{i+1}]K_j$ a weight of ℓ^j, since in this case, the overall number of nodes being explored does not change.

Algorithm 3 Tree generation ($\ell = 2$)

Input: $A_0 \in \mathbb{F}_{p^2}$, (P_0, Q_0) a basis of $E_{A_0}[2^e]$ with $[2^{e-1}]Q_0 = (0, 0)$
Output: j-invariants of curves 2^e-isogenous to E_{A_0} through isogenies with kernel $\langle P_0 + [s]Q_0 \rangle$ for some $s \in [0, 2^e - 1]$
Remark: x-coordinate only arithmetic may be directly implemented (details omitted).

function RECURSE(d, path, A_d, L)
  if $d = e$ **then**
    output $(\text{path}, j(E_{A_d}))$
    return
  $(P, Q, i) \leftarrow \arg\max_{(P,Q,i) \in L} i$
  $(P', Q') \leftarrow ([2^{e-1-i}]P,\ [2^{e-1-i}]Q)$; add tuples $([2^{i'-i}]P,\ [2^{i'-i}]Q, i')$
    to L according to the optimal strategy (depends on d, i')
  for $b \in \{0, 1\}$ **do**
    $K \leftarrow (P' + [b]Q') \in E_{A_d}$
    $(\phi, A_{d+1}) \leftarrow \phi : E_{A_d} \to E_{A_{d+1}}$ is a 2-isogeny with $\ker \phi = \langle K \rangle$
    $L' \leftarrow \emptyset$
    for $(P, Q, i) \in L, i \leq e - 1$ **do**
      if $b = 0$ **then**
        $(P, Q) \leftarrow (\phi(P), \phi([2]Q))$ ▷ Proposition 7.3
      else
        $(P, Q) \leftarrow (\phi(P + Q), \phi([2]P))$ ▷ Proposition 7.3
      $L' \leftarrow L' \cup \{(P, Q, i + 1)\}$
    RECURSE($d + 1$, path$||b$, A_{d+1}, L')

RECURSE(0, (), A_0, $\{(P_0, Q_0, 0)\}$)

Once weights are assigned, the dynamic programming approach can be applied to find the best strategies for these new graphs. However, in contrast to best strategies for single paths, sub-problems are not fully characterized by their size: edge weights depend, indeed, on where we currently are in the strategy graph. Therefore, we have to solve all sub-problems separately.

Alternatively, we can observe that the cost of a sub-problem of height e rooted at $[\ell^i]K_j$ can be obtained by multiplying by ℓ^j the cost of solving a pure instance (i.e., rooted at $[\ell^0]K_0$) of height $e - i - j$. This reduces the dynamic programming dimension back to 1.

7.4.4 Full Algorithm

We sketch the full attack pseudo-code for the case $\ell = 2$ in Algorithm 3.

7.5 Further Optimizations

7.5.1 Final Curve 2-bit Leak

In SIKE, the shared secret key is (computed from) the j-invariant of the image curve E_{AB} of the isogeny resulting from composing Alice and Bob walks in their respective torsions. To allow this, each party publishes intermediate image curves E_A and E_B along with images of others' party torsion basis through their secret isogeny

(Subsection 7.2.2). For example, Alice, who computes her 2^{e_A}-isogeny $\phi_A : E \to E_A$, provides Bob with a basis $(\phi_A(P_B), \phi_A(Q_B))$ for the 3^{e_B}-torsion of E_A, and the coordinates of such basis leak the final curve itself, i.e. the value $A \in \mathbb{F}_{p^2}$. As was further noticed in [Cos+20], the final value A leaks the j-invariant of the curve visited two 2-isogeny steps before reaching the final curve during her walk: more concretely, it can be shown that the order-4 points $\tilde{Q} = (1, \pm\sqrt{A+2})$ lie in the kernel of the dual of the isogeny $\phi : E_6 \to E_A$, and we can thus easily obtain the j-invariant $j' = j(E_{A'})$ of the curve $E_{A'} = E_A / \langle \tilde{Q} \rangle$ visited two steps before the end.

We note, however, that, since j-invariants of Montgomery curves are characterized by A^2, the A-value of the curve effectively visited two steps before the end remains undetermined: indeed, by solving the equation $j' \cdot (A'^2 - 4) - 256(A'^2 - 3)^3 = 0$ we obtain (at most) 6 solutions for A', and all of them correspond to Montgomery curves isomorphic to the curve visited 2 steps before E_A.

We can use 4-isogeny formulas from [Jao+20], in order to detect which coefficients A' can be pushed directly to the final curve E_A through the 2-isogeny formulas from Proposition 7.1. For an order 4 point $(x_k, y_k) \in E_{A'}$ defining the isogeny $\phi : E_{A'} \to E_A$, we have that $A = 4x_k^4 - 2$. It then suffices to check for which of the 6 candidate values for A', the point on $E_{A'}$ with x-coordinate $(-i)^n \cdot \sqrt[4]{\frac{A+2}{4}}$, with $n \in [0, 3]$, has order 4.

Since $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$, the map $(x, y) \mapsto (-x, iy)$ defines an isomorphism between $E_{A'}$ and $E_{-A'}$, thus the order-4 point in $E_{A'}(\mathbb{F}_{p^2})$ with x-coordinate $(-i)^n \sqrt[4]{(A+2)/4}$ and $n \in [0, 3]$, corresponds to an order-4 point in $E_{-A'}(\mathbb{F}_{p^2})$ with x-coordinate $(-i)^m \sqrt[4]{(A+2)/4}$, where $m = n + 2 \pmod 4$.

From Subsection 7.2.4 and Figure 7.1, we know that there are 4 coefficients $\pm B', \pm C'$ out of 6, each corresponding to a curve with j-invariant equal to the j-invariant of the curve visited two step before E_A, that satisfy the final 4-isogeny with respect to E_A: it is then enough to pick any of the 2 remaining coefficients $\pm A'$, and set the updated final curve to $E_{A'}$: in this way, all isogenies from this curve defined as in Proposition 7.1, will (have at least a) walk towards the starting curve E_6. This allows us to save a factor of 4 in the tree generated from the final curve if traversals are matched in-the-middle looking at j-invariants.

Alternative expression for A' Interestingly, by considering the walk structure induced by SIKE 2-isogenies (Subsection 7.2.4), the relation between such A' and the final curve coefficient A is very easy to express.

Lemma 7.1. *Let E_B be a Montgomery supersingular elliptic curve over $\mathbb{F}_{p^2}$ with $p \neq 2$, and let $K_0, K_1 \in E_B(\mathbb{F}_{p^2})$ be two order 2 points distinct from $(0, 0)$. By applying the 2-isogeny formulas from Proposition 7.1 to the groups generated by K_0 and K_1, we obtain, respectively, two isogenies $\phi_1 : E_B \to E_A, \phi_2 : E_B \to E_{A'}$ such that*

$$(A - 2)(A' - 2) = 16.$$

Proof. Let $\tilde{x}, \tilde{z} = 1/\tilde{x}$ be the roots of $x^2 + Bx + 1 = 0$. Then $\tilde{x}, \tilde{z}$ are the x-coordinates of K_0 and K_1. By applying the 2-isogeny formulas from Proposition 7.1 on these two points, we then obtain $A = 2 - 4\tilde{x}^2$ and $A' = 2 - 4\tilde{z}^2$. It then immediately follows that $(A - 2)(A' - 2) = (-4)^2 \cdot \tilde{x}^2 \tilde{z}^2 = 16$. □

Let us now consider the last 3 traversed nodes in Alice's walk, i.e. the j-invariant of $E_{A'}$, followed by a middle node j', and the final $j(E_A)$. Then there exists a $B \in \mathbb{F}_{p^2}$

so that $j(E_B) = j'$, which is pushed, depending on the kernel chosen, through 2-isogenies to $-A$ and A' (cf. Figure 7.1). Note that we use $-A$ instead of A, since otherwise we would select the $\pm B$ from the original path, which only has a backwards edge towards $j(E_{A'})$ (i.e., an isogeny with the kernel $\langle (0, 0\rangle)$. From Lemma 7.1, we then conclude that

$$A' = 2 - \frac{16}{A+2} \quad \text{(two final 2-isogenies)}$$

This equation assumes that the last two steps consist of a sequence of two 2-isogenies. If a 4-isogeny is used instead (as in the case of SIKE challenges), this decomposes, according to specification [Jao+20], into a sequence of two 2-isogenies followed by a sign flip of the pushed curve coefficient. We then need to flip the sign of the coefficient of the final curve to match the assumptions of Lemma 7.1, thus obtaining

$$A' = 2 + \frac{16}{A-2} \quad \text{(one final 4-isogeny)} \tag{7.3}$$

7.5.2 Storing Conjugation Representatives

In SIKE, the starting curve is chosen to be $E_6(\mathbb{F}_{p^2})$, and since $A = 6 \in \mathbb{F}_p$, the Frobenius map $\pi : (x, y) \mapsto (x^p, y^p)$ defines an automorphism for $E_6(\mathbb{F}_{p^2})$. As already noticed in [Cos+20], this implies that for any kernel $\langle R\rangle \subset E_6$, $j(E_6/\langle R\rangle)^p = j(E_6/\pi(\langle R\rangle))$, that is pairs of conjugate kernels give rise to paths to curves having conjugate j-invariants. By recalling that π fixes $\mathbb{F}_p$, from the above we further easily obtain that if E_A is isogenous to E_6, then $E_{\bar{A}}$ is isogenous to E_6 as well, since $\overline{j(E_A)} = j(E_{\bar{A}})$.

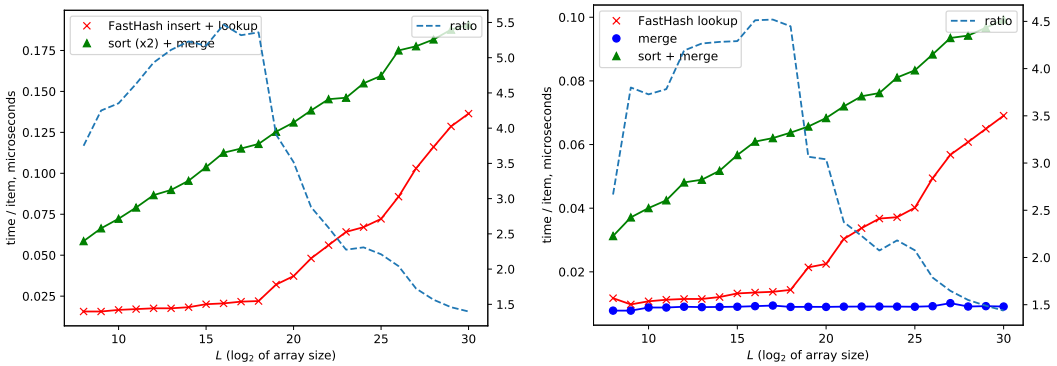
We take advantage of this fact to approximately halve the time complexity of the meet-in-the-middle-attack. It is indeed sufficient to explore non-conjugate subtrees starting from E_6, and store the norm of j-invariants in the middle to detect intersections of j-invariants. It can be shown that at any (non-trivial) depth of the tree expanded from E_6, there exist exactly two curves with their A-coefficients in $\mathbb{F}_p$, while all the other nodes are conjugate pairs (each arising from two conjugate subtrees). This means that at a certain depth $e > 0$, we have exactly $2^{e-1} + 1$ different norms for j-invariants, which can be computed by exploring just $\frac{2^{e-1}+1}{2^e} \approx \frac{1}{2}$ of the tree expanded from E_6.

If the j-invariants found in the intersection during the meet-in-the-middle attack have the same norm but are conjugates, we need to retrieve the correct *"conjugate path"* on the tree from E_6, in order to solve Problem 7.1. From the above properties, the sequence of j-invariants on the path from E_6 to a curve with j-invariant j', is an element-wise conjugate of the sequence towards a curve of j-invariant $\overline{j'}$: it is then enough for a kernel $P + [s]Q$ going to j', to suitably flip bits in s, so that the resulting path walks step-by-step over nodes with conjugate j-invariants.

As a final note, SIKE works in the quadratic extension $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$, so elements can be seen as complex numbers over $\mathbb{F}_p$. Conjugates can be then obtained by simply changing the sign to elements' imaginary parts, and, in place of expensive to compute norms, we can store as a unique representative of a conjugation class just real parts of j-invariants in the middle.

7.5.3 Set Intersection

A standard way to implement the final stage of our meet-in-the-middle attack, i.e., finding elements common to the two trees j-invariant norm datasets, is by using



(A) Intersecting two arrays without preparation

(B) Lookup in a prepared array

FIGURE 7.4: Performance comparison between FastHash and SortMerge over 64-bit integer arrays of total size 2^L.

hash-tables: we fill one of such tables with entries from the first dataset, and we then lookup every element in the second one. In theory, the amortized cost of a hash-table lookup would be $\mathcal{O}(1)$. However, in practice, *random* memory accesses get slower and slower as the table size grows and memory *latency* starts dominating execution time.

An alternative approach is to *sort* the two datasets and perform a linear-time *merge* operation by keeping common elements only, an operation requiring *sequential* memory accesses. The drawback of this approach is that the sorting step has quasi-linear complexity $\mathcal{O}(n \log n)$ in the (biggest) dataset size n, and to complete it, we need memory access patterns that are not necessarily sequential.

In order to compare these two approaches, we implemented a simple hash-set for 64-bit integers with linear scanning and double-sized buffer (i.e., to store n elements, the structure allocates memory for $2n$ elements). In the following, we will refer to such a custom hash-set with the name FastHash. In our experiments, it outperforms the default C++ `unordered_set` (compiled on `g++ 9.3.0`) more than a few times. We then implemented the sort and merge approach (denoted `SortMerge`).

In Figure 7.4, we provide different benchmarks for both FastHash and SortMerge at different array sizes 2^L.

More precisely, in Figure 7.4a we compare the time to intersect two unsorted 64-bit integers arrays, assuming no preprocessing of the input datasets. Here, the FastHash-based approach first inserts all 2^L elements of the first dataset and then performs lookups of the 2^L elements of the second dataset. In SortMerge, instead, the two datasets are first sorted (using C++ `sort()`) and then merged using a two-pointers linear scan. Although FastHash outperforms the sorting approach on up to $2^{30} \approx 10^9$ entries, the advantage ratio decreases quickly from an initial value of 4 (for $L = 8$) to a ratio close to 1 for $L = 30$. In particular, a sharp advantage drop is visible after $L = 18$, which is likely related to the dataset not fitting the CPU cache (experiments were run on an Intel® Core™ i5 10210U 1.6-4.2 GHz).

In Figure 7.4b, we compare the two methods under some allowed precomputations. For FastHash this means that only lookups are counted (insertions are excluded from time computation), while for SortMerge we consider two cases: i) the first dataset is pre-sorted, that is, the timings include sorting the second dataset only and merging (in green); ii) both datasets are pre-sorted, that is the timings include only merging (in blue). We can see that the pure merging cost remains constant for any array size and is negligible compared to both FastHash lookups and sorting.

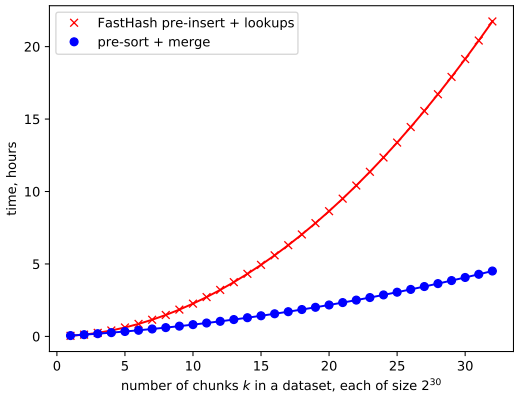


FIGURE 7.5: Performance comparison between FastHash and SortMerge running in parallel over k 64-bit integers chunks, each of size 2^{30} bytes (extrapolated using timings from Figure 7.4b).

Parallelization When dataset sizes are large, efficient parallelization techniques are a requirement. The most straightforward approach for parallelizing intersection finding, consists in splitting the input datasets A and B in k (equally sized) chunks $(A_0, \ldots, A_k)$ and $(B_0, \ldots, B_k)$, and then intersect all k^2 distinct pairs $A_i \cap B_j$ independently in parallel. Clearly, this parallelization comes at the cost of k times more work than standard lookups/merges but can be acceptable if k is small.

An advantage of this approach is that each chunk can be preprocessed independently so that each of the k^2 chunks intersection takes preprocessed data as input. From Figure 7.4 it is clear that already for $k = 2$, SortMerge (which requires an amortized number of 2 sorting and k merges/intersections per chunk) outperforms the FastHash hash-set approach (which requires 1 chunk insertion and k chunks lookups per chunk). In Figure 7.5, we illustrate how these two approaches perform, for different values of k, on k chunks each of size 1GiB.

7.5.4 Storage-Collisions Trade-off and Compression

Real parts (or, in general, norms) of j-invariants belong to $\mathbb{F}_p$: in our meet-in-the-middle attack, each tree has only approximately $2^{e_A/2} \approx p^{1/4}$ leaves. Therefore, the chance of having multiple collisions is negligible.

However, we would like to reduce storage requirements as much as possible. This can be done by reducing the number of bits we use to represent j-invariants while allowing only a reasonable amount of false-positive collisions. More concretely, if we use n bits to represent j-invariants, we then expect to observe approximately $(p^{1/4})^2/2^n$ collisions.

In addition, paths associated with j-invariants in the middle (useful to test full isogenies associated with collisions quickly) may be omitted too. This comes at the cost of an extra (memoryless) tree exploration, required to recover full j-invariants associated with a colliding representation[5] and the respective paths in the trees.

In some cases, it is possible to reduce memory requirements further. Suppose the n-bit j-invariant representations are uniformly distributed. In that case, we can compress *sorted* chunks of 2^m such elements by noticing that two consecutive elements are expected to differ, on average, by 2^{n-m}, i.e., on their least significant $n - m$ bits. We can then store only such reduced differences, reserving 1 *flag* bit in order

[5]Since just a few false-positive collisions are expected, recovering full $2 \log p$-sized j-invariants is useful to immediately detect their correct conjugate branch in the tree expanded from E_6.

to distinguish between a small difference from an n-bit full representation, in case two elements differ by more than 2^{n-m} (in practice, a larger difference is allowed to capture more small elements). This, in fact, reduces memory requirements from $n2^m$ bits to at most $\approx (n-m+1)2^m$ bits, with different implementation-specific wordsize trade-offs in the middle.

We note that by requiring chunks to be sorted, this compression technique goes towards the SortMerge intersection finding approach we detailed in Subsection 7.5.3: indeed, merges can be performed equivalently by using two additional n bits registers, in which we store the n bits values to compare, obtained by iteratively adding differences to each dataset's first uncompressed element.

7.6 Cryptanalysis of the $IKEp182 Challenge

In this Section we will detail how all the above ideas allowed us to break the $IKEp182 challenge [Mic21b], a small parameters specification-compliant SIKE instance generated by Microsoft in a live event during the 3rd NIST PQC Standardization conference.

In $IKEp182, the field characteristic is equal to $p = 2^{91}3^{57} - 1$. According to the specification, we have $\mathbb{F}_{p^2} = \mathbb{F}_p(i) = \mathbb{F}_p[x]/(x^2+1)$, $\#E_6(\mathbb{F}_{p^2}) = (p+1)^2$ and $E_6[2^{91}] = \langle P_A, Q_A \rangle$, $E_6[3^{57}] = \langle P_B, Q_B \rangle$ with

$$
\begin{aligned}
P_A = (&\texttt{0x05a324935a4d7b75024fdc3601fe8b5888cea9f88212b2} + \\
&\texttt{0x02357bdd576772bf2a93e3d680ed7306e16eafc6aff904} \cdot \texttt{i}, \\
&\texttt{0x242a9e09aa8e6995e4fdce9f68e8c2c902154c332de68a} + \\
&\texttt{0x011b23646f8884b7a9faa5159ef13842880ed0f9f43dcd} \cdot \texttt{i}) \\
Q_A = (&\texttt{0x27b8def415bae0506a9607fff7704832151cdcbc93cb22} + \\
&\texttt{0x085c86f386b94b8c413f5e49736f26de95103a9b65f31a} \cdot \texttt{i}, \\
&\texttt{0x16af6790fb0f5cfd0e124033bb7619e2f75a25cae5f42b} + \\
&\texttt{0x172567b99058dd9d5b99ce5ea4bacd685f57c8326011a3} \cdot \texttt{i}) \\
P_B = (&\texttt{0x02ca3bc7e98f88b3ca3239c276eb7a224c51f61bc8c5ed}, \\
&\texttt{0x262a38701d1b61dd8875909ff268a50d912f620db980a1}) \\
Q_B = (&\texttt{0x02dcff7123e2380f552f5bff91da77ae62e9556b866d8f}, \\
&\texttt{0x06aeb7c764aa40913b3fc784d569833d4226cc4a53432f} \cdot \texttt{i})
\end{aligned}
$$

Our meet-in-the-middle attack will target the 2^{91} torsion and will recover Alice's full 91-steps walk in the Supersingular 2-Isogeny Graph. After a quick *Setup*, the full attack will consist of 5 main stages: *Trees Expansions*, *k-way Merge*, *Compression*, *Sieving* and *Final Trees Expansions*.

Setup. In the first step of the SIKE protocol (Subsection 7.2.2), Alice sends to Bob a compressed representation of the points $\phi_A(P_B)$, $\phi_A(Q_B)$, consisting of the following 3 x-coordinates

$$
\begin{aligned}
x_{\phi_A(P_B)} &= \texttt{0x17d02d323c815eee1ec75f1c675609b0bea78064cb8cc1 +} \\
&\quad \texttt{0x12fa80de8027f68c3f780b5bcd519e8205606ac249025d·i} \\
x_{\phi_A(Q_B)} &= \texttt{0x272c54d49af950b0829072753e3525091aaf87085bd7b2 +} \\
&\quad \texttt{0x23efe3c087965a49fcc5161e6453dbe632d7dec90bab12·i} \\
x_{\phi_A(Q_B)-\phi_A(P_B)} &= \texttt{0x22c38abb1427245de1e049408dab87ed9ba54efeb4a4e4 +} \\
&\quad \texttt{0x0c5d768e87a762b6a460b941bcc5537ba0f73ce8b9f955·i}
\end{aligned}
$$

Such representation is justified by the use of efficient implementations which exploits x-only arithmetic: we refer to [Jao+20] for more details.

If we denote the tuple $(x_{\phi_A(P_B)}, x_{\phi_A(Q_B)}, x_{\phi_A(Q_B)-\phi_A(P_B)})$ as (x_P, x_Q, x_{Q-P}) we obtain [CLN16, Section 6] the A coefficient of the Montgomery curve E_A on which the points $\phi_A(P_B)$, $\phi_A(Q_B)$ lie, as

$$
\begin{aligned}
A &= \frac{(1 - x_P x_Q - x_P x_{Q-P} - x_Q x_{Q-P})^2}{4 x_P x_Q x_{Q-P}} - x_P - x_Q - x_{Q-P} \\
&= \texttt{0xc0cbda5ef968048cd2c1b125774f1417125b9b02b6f91 +} \\
&\quad \texttt{0x1e8121a2a60fd266d321bb9db8d9e3111e3095c08e0bc6·i}
\end{aligned}
$$

To take advantage of the final 2-bit leak described in Subsection 7.5.1, we computed the coefficient A' such that $j(E_{A'})$ lies on the (secret) traversed path 2 steps before the final curve, and the SIKE-tree arising from A' does not go towards the final curve E_A. This can be achieved by using (7.3), i.e., $A' = 2 + 16/(A - 2)$, to obtain:

$$
\begin{aligned}
\tilde{A}' &= \texttt{0x164db610b03a9b3c38e59bf29485a60462d1cd9f22d95e +} \\
&\quad \texttt{0x1a8d75d6d0285807042e900df3c2cf74b4eb160d50a92e·i} \\
j(E_{A'}) &= \texttt{0xe48a8271ea06ec4193db09970a23bea55c777ef2fb5be +} \\
&\quad \texttt{0x56910191b4835901ef45e4b857817391ad1213080afa9·i}
\end{aligned}
$$

This Setup phase was implemented in SageMath [The21].

Trees Expansions. We set $A' = \tilde{A}'$, and we proceed by attacking the 89-steps path in the isogeny graph between $j(E_6)$ and $j(E_{A'})$. Note that there may be no path in the SIKE-tree (Definition 7.7) between the exact *curves* (as we might chose a different representative curve), but there must exist a path in the isogeny graph between $j(E_6)$ and $j(E_{A'})$, and the SIKE-trees arising from E_6 and $E_{A'}$ must contain paths following this path by j-invariants (from the opposite endpoints). In order to meet in the middle, we generate in a depth-first manner the SIKE-tree expanded from E_6 (up to the depth 45) and the SIKE-tree expanded from $E_{A'}$ (up to the depth 44), employing the optimal strategies detailed in Subsection 7.4.2.

We note that, as discussed in Subsection 7.5.2, it suffices to explore only half of the conjugate sub-branches of the tree expanded from E_6: this results in an almost equal number of leaves in-the-middle generated from both trees, with a total of $2^{44} + 1$ leaves for the tree expanded from E_6, and 2^{44} leaves for the one expanded from $E_{A'}$.

Once the depth-first generation reaches a leaf, we compute the corresponding j-invariant, and we store the least significant 64 bits of its real part. In our implementation, multiple jobs explore in parallel distinct branches of each tree: when a job collects 2GiB of 64-bit j-invariant representations (which correspond to 2^{28} j-invariants visited), this chunk is sorted in memory, written to disk, and then the job terminates. On the University of Luxembourg High Performance Computing (HPC) facilities [Var+14], each of these jobs took approximately 17 minutes to complete on a single core of an Intel® Xeon® E5-2680v4 @ 2.4GHz with 4GiB of RAM reserved. This sums up to a total of approximately 4.2 core-years and 256TiB of disk space needed to explore both trees and store the truncated j-invariants.

We note that, by utilizing the Merge and Compression earlier, i.e., on the fly after a sufficient amount of chunks is generated, the storage requirement could be reduced to close to 128TiB.

k-way Merge. We employed our custom *k-way merge* implementation optimized for 64-bit unsigned integers, to merge the 2GiB sorted chunks generated from each tree: on a single core of an Intel® Xeon® E5-2680v4 @ 2.4GHz and 4GiB of RAM, we needed approximately 2.5 core hours to merge 256 2GiB chunks into a single 512GiB sorted chunk. We note that to keep memory requirements close to the ones needed to store all j-invariants representations, chunks can be merged in parallel while running the depth-first expansions as soon as enough new 2GiB chunks from a specific tree are generated. The practicality of running multiple such merges in parallel depends, however, on storage architecture, cluster load and maximum disks I/O throughput: on the University of Luxembourg HPC cluster, we were able to run 4 nodes in parallel, running 28 merge jobs each, without degrading too much I/O performances. This merging stage took, overall, approximately 54 core days.

Compression. Since 512GiB chunks contain already 2^{36} 64-bit elements each, at this point we ran single-core jobs to merge 4 chunks directly in compressed form (Subsection 7.5.4), using 32 bits (including 1 flag bit) to encode elements differences. This resulted in a compression factor very close to $\frac{1}{2}$. In the same configuration as above (and under the same limitations), we needed roughly 5 core hours to complete one of such merge-to-compressed jobs (we ran only 2-3 nodes concurrently, each executing 28 such jobs), for a total of 27 core-days to complete all jobs.

We then finally obtained 64 1TiB compressed chunks from each tree, for a total of 128TiB disk space used (all previous sub-chunks were deleted).

Sieving. At this point, we proceed with finding elements shared by chunks from different trees. Since chunks are sorted already, we can use the parallel version of SortMerge with parameter $k = 64$ detailed at the end of Subsection 7.5.3. This stage consists in merging tuples of (compressed) 1TiB chunks and storing only the common elements. When running in a single thread on the full data, this stage only requires a sequential read of the 128TiB of data. However, the heap operations in k-way merge dominate the performance and can not be parallelized. In our implementation, a sieving job consisted in merging at the same time 4 chunks from the first tree with 4 chunks from the second tree, where elements are iteratively decompressed, and only collisions are stored. On a single-core, it took approximately 1.1 core days to complete, for a total of 280 core-days for 256 such jobs. This trade-off results in 2PiB of data read, which is acceptable to allow sufficient parallelization.

We expected and we actually found $16\,777\,119 \approx 2^{44 \cdot 2}/2^{64} = 2^{24} = 16\,777\,216$ 64-bit collisions among the two trees: once such collisions are safely stored, we can delete all the 128 1TiB chunks from previous stages.

Final Trees Expansions. With the collisions just found, we rerun the tree explorations, similarly as in the first stage of the attack, but this time we store only full j-invariants in the middle that have the least 64 bits of their real part matching any of the collisions found, and their paths in the respective trees.

During tree explorations, we regularly check if were found j-invariants from different trees that share the same real part: if yes, we stop tree traversals, and we reconstruct Alice's full walk from E_6 to E_A (and thus her secret) using the paths associated to the matching j-invariants, with special care in case the two result to be conjugate (Subsection 7.5.2).

In our case, the colliding conjugate j-invariants in-the-middle obtained by expanding the trees from E_6 and $E_{A'}$ were, respectively,

$$\begin{aligned} j_0 &= \texttt{0x0008132653e4d53cb9cc0defb36a0141d900adbb128a24f0} + \\ &\quad \texttt{0x0001049f06c78aaed22786dfcff5b202ce3a50429f369b86} \cdot \texttt{i} \\ j_1 &= \texttt{0x0008132653e4d53cb9cc0defb36a0141d900adbb128a24f0} + \\ &\quad \texttt{0x0027910d1a0d795d077f40d1480a4dfd31c5afbd60c96479} \cdot \texttt{i} \end{aligned}$$

Using the (implementation-dependent) path information we stored, we then reconstruct, in linear time, Alice's private key as

$$s_A = \texttt{0x59d64d476da9487be414734}$$

which allowed us to easily compute Alice's and Bob's shared secret from Bob's public key exchanged, as

$$\begin{aligned} j(E_{AB}) = &\texttt{0x7a470546a24124f06f49bcbb855a6e3c1402ba1004bfc} + \\ &\texttt{0x1a88f02557168dd75b64f8407a368aa4ff2bc03121fbaf} \cdot \texttt{i} \end{aligned}$$

whose value is a correct pre-image for the publicly released SHA512 hash of the challenge shared secret [Mic21b].

We found the solution to the challenge after exploring approximately 44% of the tree expanded from E_6 (only conjugate-unique sub-branches) and 63% of the tree expanded from $E_{A'}$ (success probability of $\approx 28\%$).

This brings the total cost of our attack to approximately 8.5 core years and 256TiB of disk memory.

We note, however, that we decided to employ compression only after we started the above attack in order to reduce the amount of not fully parallelizable disk reads needed for the parallel SortMerge. Thus, in fact, the whole attack can be executed in 8.5 core years with just slightly more than 128TiB of disk memory available. The storage requirement can be reduced further by sacrificing parallelization and performing the main steps for a single part of the second tree at a time. In our case, we used 4-chunk groups (4TiB) on each side, so only $64 + 4 = 68$TiB of storage is sufficient for the (less-parallel) attack.

7.7 The $IKEp217 challenge

A natural question is whether the $IKEp217 challenge, the bigger among the toy instances proposed by Microsoft, is practically reachable for attacking using our method.

In $IKEp217, the prime p is equal to $2^{110}3^{67} - 1$, so that $e_A = 110$, leading to 192PiB storage requirement if our attack on $IKEp182 is applied directly and 64-bit j-invariant representations are stored (which may produce a large number of collisions, namely $2^{107-64} = 2^{43}$).

On the University of Luxembourg HPC cluster, the main limitation is the I/O performance, which is about 20GiB/s[6]. Even if arbitrarily storage is available, this maximum throughput limits the time needed to solve the instance since all the data must be read/written at least once. To read the 192PiB of data on the ULHPC cluster, one would need at least 116 days. Since full attack performs several I/O rounds, this would likely take more than a year.

On the other hand, an attacker with a custom highly-parallelized supercomputer may perform the attack much faster. Precise trade-off analysis and parameter selection are left as future work.

7.8 Conclusions

In this Chapter we showed how the Microsoft $IKEp182 challenge can be broken in practice using a Meet-in-the-Middle approach and multiple SIKE-specific optimizations. In particular, we detailed a tree exploration strategy to compute efficiently j-invariants of curves appearing as leaves of the trees expanded by one SIKE party's initial and final curve (i.e., the domain and codomain of a 2^e-isogeny). In doing so, we took advantage of several optimizations, such us computing only one sub-branch per conjugate class for the tree expanded from the initial curve and a 2-step j-invariant leakage in the tree expanded from the final curve, which allowed us to reduce further time and space complexity of our attack by a factor of 8. We then discussed possible approaches to deal with practical challenges arising when implementing such improved MitM attack (e.g., space requirements, parallelization, set intersection, etc.), and we reported the details of running our implementation against the $IKEp182 challenge instance, where we were able to recover the corresponding secret shared successfully.

[6] https://hpc-docs.uni.lu/filesystems/gpfs/

Bibliography

[AAM19] Gora Adj, Omran Ahmadi, and Alfred Menezes. "On isogeny graphs of supersingular elliptic curves over finite fields". In: *Finite Fields and Their Applications* 55 (2019), pp. 268–283. ISSN: 1071-5797. DOI: 10.1016/j.ffa.2018.10.002.

[Adj+15] Gora Adj, Alfred Menezes, Thomaz Oliveira, and Francisco Rodríguez-Henríquez. "Computing Discrete Logarithms in $\mathbb{F}_{3^{6\cdot 137}}$ and $\mathbb{F}_{3^{6\cdot 163}}$ Using Magma". In: *Arithmetic of Finite Fields*. Ed. by Çetin Kaya Koç, Sihem Mesnager, and Erkay Savaş. Cham: Springer International Publishing, 2015, pp. 3–22. ISBN: 978-3-319-16277-5. DOI: 10.1007/978-3-319-16277-5_1.

[Adj+19] Gora Adj, Daniel Cervantes-Vázquez, Jesús-Javier Chi-Domínguez, Alfred Menezes, and Francisco Rodríguez-Henríquez. "On the Cost of Computing Isogenies Between Supersingular Elliptic Curves". In: *SAC 2018: 25th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Carlos Cid and Michael J. Jacobson Jr: vol. 11349. Lecture Notes in Computer Science. Springer, Heidelberg, 2019-08, pp. 322–343. DOI: 10.1007/978-3-030-10970-7_15.

[Ala96] N. S. Aladov. "Sur la distribution des résidus quadratiques et non-quadratiques d'un nombre premier P dans la suite $1, 2, \ldots, P-1$". In: *Matematicheskii Sbornik* 18.1 (1896), pp. 61–75.

[Alb+16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. "MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity". In: *Advances in Cryptology – ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. Lecture Notes in Computer Science. Springer, Heidelberg, 2016-12, pp. 191–219. DOI: 10.1007/978-3-662-53887-6_7.

[AM93] Arthur O. L. Atkin and François Morain. "Elliptic curves and primality proving". In: *Mathematics of computation* 61.203 (1993), pp. 29–68. DOI: 10.1090/S0025-5718-1993-1199989-X.

[ANS19] Yusuke Aikawa, Koji Nuida, and Masaaki Shirase. "Elliptic Curve Method Using Complex Multiplication Method". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 102.1 (2019), pp. 74–80. DOI: 10.1587/transfun.E102.A.74.

[Ara+] Diego F. Aranha, Conrado P. L. Gouvêa, Tobias Markmann, Riad S. Wahby, and Kevin Liao. *RELIC is an Efficient LIbrary for Cryptography*. https://github.com/relic-toolkit/relic.

[Ara+13] Diego F. Aranha, Laura Fuentes-Castañeda, Edward Knapp, Alfred Menezes, and Francisco Rodríguez-Henríquez. "Implementing Pairings at the 192-Bit Security Level". In: *PAIRING 2012: 5th International Conference on Pairing-based Cryptography*. Ed. by Michel Abdalla and Tanja Lange.

Vol. 7708. Lecture Notes in Computer Science. Springer, Heidelberg, 2013-05, pp. 177–195. DOI: 10.1007/978-3-642-36334-4_11.

[Atk88] Arthur O. L. Atkin. "The number of points on an elliptic curve modulo a prime (I)". In: Draft, 1988.

[Atk99] Arthur O. L. Atkin. "The number of points on an elliptic curve modulo a prime (II)". In: Draft, 1999.

[Au+09] Man Ho Au, Patrick P. Tsang, Willy Susilo, and Yi Mu. "Dynamic Universal Accumulators for DDH Groups and Their Application to Attribute-Based Anonymous Credential Systems". In: *Topics in Cryptology – CT-RSA 2009*. Ed. by Marc Fischlin. Vol. 5473. Lecture Notes in Computer Science. Springer, Heidelberg, 2009-04, pp. 295–308. DOI: 10.1007/978-3-642-00862-7_20.

[Bas04] Julius M. Basilla. "On the solution of $x^2 + dy^2 = m$". In: *Proceedings of the Japan Academy, Series A, Mathematical Sciences* 80.5 (2004), pp. 40–41. DOI: 10.3792/pjaa.80.40.

[BB04] Dan Boneh and Xavier Boyen. "Short Signatures Without Random Oracles". In: *Advances in Cryptology – EUROCRYPT 2004*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. Lecture Notes in Computer Science. Springer, Heidelberg, 2004-05, pp. 56–73. DOI: 10.1007/978-3-540-24676-3_4.

[BB08] Dan Boneh and Xavier Boyen. "Short Signatures Without Random Oracles and the SDH Assumption in Bilinear Groups". In: *Journal of Cryptology* 21.2 (2008-04), pp. 149–177. DOI: 10.1007/s00145-007-9005-7.

[BBF19] Dan Boneh, Benedikt Bünz, and Ben Fisch. "Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains". In: *Advances in Cryptology – CRYPTO 2019, Part I*. Ed. by Alexandra Boldyreva and Daniele Micciancio. Vol. 11692. Lecture Notes in Computer Science. Springer, Heidelberg, 2019-08, pp. 561–586. DOI: 10.1007/978-3-030-26948-7_20.

[BC14] Dan Boneh and Henry Corrigan-Gibbs. "Bivariate Polynomials Modulo Composites and Their Applications". In: *Advances in Cryptology – ASIACRYPT 2014, Part I*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. Lecture Notes in Computer Science. Springer, Heidelberg, 2014-12, pp. 42–62. DOI: 10.1007/978-3-662-45611-8_3.

[Bd94] Josh C. Benaloh and Michael de Mare. "One-Way Accumulators: A Decentralized Alternative to Digital Sinatures (Extended Abstract)". In: *Advances in Cryptology – EUROCRYPT'93*. Ed. by Tor Helleseth. Vol. 765. Lecture Notes in Computer Science. Springer, Heidelberg, 1994-05, pp. 274–285. DOI: 10.1007/3-540-48285-7_24.

[Bel+08] Juliana Belding, Reinier M. Bröker, Andreas Enge, and Kristin Lauter. "Computing Hilbert Class Polynomials". In: *Algorithmic Number Theory*. Ed. by Alfred J. van der Poorten and Andreas Stein. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 282–295. ISBN: 978-3-540-79456-1. DOI: 10.1007/978-3-540-79456-1_19.

[Ben+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. "Zerocash: Decentralized Anonymous Payments from Bitcoin". In: *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2014-05, pp. 459–474. DOI: 10.1109/SP.2014.36.

[Ber+13] Daniel J. Bernstein, Stacey Jeffery, Tanja Lange, and Alexander Meurer. "Quantum Algorithms for the Subset-Sum Problem". In: *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013*. Ed. by Philippe Gaborit. Springer, Heidelberg, 2013-06, pp. 16–33. DOI: 10.1007/978-3-642-38616-9_2.

[Beu+20] Ward Beullens, Tim Beyne, Aleksei Udovenko, and Giuseppe Vitto. "Cryptanalysis of the Legendre PRF and Generalizations". In: *IACR Transactions on Symmetric Cryptology* 2020.1 (2020), pp. 313–330. ISSN: 2519-173X. DOI: 10.13154/tosc.v2020.i1.313-330.

[BF97] Dan Boneh and Matthew K. Franklin. "Efficient Generation of Shared RSA Keys (Extended Abstract)". In: *Advances in Cryptology – CRYPTO'97*. Ed. by Burton S. Kaliski Jr. Vol. 1294. Lecture Notes in Computer Science. Springer, Heidelberg, 1997-08, pp. 425–439. DOI: 10.1007/BFb0052253.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. "Transparent SNARKs from DARK Compilers". In: *Advances in Cryptology – EUROCRYPT 2020, Part I*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. Lecture Notes in Computer Science. Springer, Heidelberg, 2020-05, pp. 677–706. DOI: 10.1007/978-3-030-45721-1_24.

[BFV19] Alex Biryukov, Daniel Feher, and Giuseppe Vitto. "Privacy Aspects and Subliminal Channels in Zcash". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 1813–1830. ISBN: 9781450367479. DOI: 10.1145/3319535.3345663.

[Bir+21] Alex Biryukov, Luan Cardoso dos Santos, Daniel Feher, Vesselin Velichkov, and Giuseppe Vitto. "Automated Truncation of Differential Trails and Trail Clustering in ARX". In: *Selected Areas in Cryptography (to appear)*. Springer International Publishing, 2021. URL: https://eprint.iacr.org/2021/1194.

[BL] Daniel J. Bernstein and Tanja Lange. *XZ coordinates for short Weierstrass curves*. https://hyperelliptic.org/EFD/g1p/auto-shortw-xz.html.

[BLL00] Ahto Buldas, Peeter Laud, and Helger Lipmaa. "Accountable Certificate Management Using Undeniable Attestations". In: *ACM CCS 2000: 7th Conference on Computer and Communications Security*. Ed. by Dimitris Gritzalis, Sushil Jajodia, and Pierangela Samarati. ACM Press, 2000-11, pp. 9–17. DOI: 10.1145/352600.352604.

[BLL02] Ahto Buldas, Peeter Laud, and Helger Lipmaa. "Eliminating Counterevidence with Applications to Accountable Certificate Management". In: *Journal of Computer Security* 10.3 (2002-09), pp. 273–296. ISSN: 0926-227X. DOI: 10.3233/JCS-2002-10304.

[BP97] Niko Barić and Birgit Pfitzmann. "Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees". In: *Advances in Cryptology – EUROCRYPT'97*. Ed. by Walter Fumy. Vol. 1233. Lecture Notes in Computer Science. Springer, Heidelberg, 1997-05, pp. 480–494. DOI: 10.1007/3-540-69053-0_33.

[Bro06] Reinier M. Broker. "Constructing elliptic curves of prescribed order". PhD thesis. 2006. URL: https://www.math.leidenuniv.nl/scripties/Broker.pdf.

[BUV21] Alex Biryukov, Aleksei Udovenko, and Giuseppe Vitto. "Cryptanalysis of a Dynamic Universal Accumulator over Bilinear Groups". In: *Topics in Cryptology – CT-RSA 2021*. Ed. by Kenneth G. Paterson. Vol. 12704. Lecture Notes in Computer Science. Reproduced with permission from Springer Nature. Springer, Heidelberg, 2021-05, pp. 276–298. DOI: 10.1007/978-3-030-75539-3_12.

[BZ10] Richard P Brent and Paul Zimmermann. "An $\mathcal{O}(M(n)\log n)$ algorithm for the Jacobi symbol". In: *International Algorithmic Number Theory Symposium*. Springer. 2010, pp. 83–95.

[Cam+08] Philippe Camacho, Alejandro Hevia, Marcos A. Kiwi, and Roberto Opazo. "Strong Accumulators from Collision-Resistant Hashing". In: *ISC 2008: 11th International Conference on Information Security*. Ed. by Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee. Vol. 5222. Lecture Notes in Computer Science. Springer, Heidelberg, 2008-09, pp. 471–486. DOI: 10.1007/s10207-012-0169-2.

[CH10] Philippe Camacho and Alejandro Hevia. "On the Impossibility of Batch Update for Cryptographic Accumulators". In: *Progress in Cryptology - LATINCRYPT 2010: 1st International Conference on Cryptology and Information Security in Latin America*. Ed. by Michel Abdalla and Paulo S. L. M. Barreto. Vol. 6212. Lecture Notes in Computer Science. Springer, Heidelberg, 2010-08, pp. 178–188. DOI: 10.1007/978-3-642-14712-8_11.

[CH17] Craig Costello and Hüseyin Hisil. "A Simple and Compact Algorithm for SIDH with Arbitrary Degree Isogenies". In: *Advances in Cryptology – ASIACRYPT 2017, Part II*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10625. Lecture Notes in Computer Science. Springer, Heidelberg, 2017-12, pp. 303–329. DOI: 10.1007/978-3-319-70697-9_11.

[Che+20] Megan Chen, Ran Cohen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, and abhi shelat. "Multiparty Generation of an RSA Modulus". In: *Advances in Cryptology – CRYPTO 2020, Part III*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12172. Lecture Notes in Computer Science. Springer, Heidelberg, 2020-08, pp. 64–93. DOI: 10.1007/978-3-030-56877-1_3.

[Che+21] M. Chen, C. Hazay, Y. Ishai, Y. Kashnikov, D. Micciancio, T. Riviere, A. Shelat, M. Venkitasubramaniam, and R. Wang. "Diogenes: Lightweight Scalable RSA Modulus Generation with a Dishonest Majority". In: *2021 2021 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, 2021-05, pp. 590–607. DOI: 10.1109/SP40001.2021.00025.

[Che02a] Qi Cheng. *A New Class of Unsafe Primes*. Cryptology ePrint Archive, Report 2002/109. https://eprint.iacr.org/2002/109. 2002.

[Che02b] Qi Cheng. "A New Special-Purpose Factorization Algorithm". In: 2002.

[CJM02] Philippe Chose, Antoine Joux, and Michel Mitton. "Fast Correlation Attacks: An Algorithmic Point of View". In: *Advances in Cryptology – EUROCRYPT 2002*. Ed. by Lars R. Knudsen. Vol. 2332. Lecture Notes in Computer Science. Springer, Heidelberg, 2002-4 / 5, pp. 209–221. DOI: 10.1007/3-540-46035-7_14.

[CKS09] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. "An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials". In: *PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography*. Ed. by Stanislaw Jarecki and Gene Tsudik. Vol. 5443. Lecture Notes in Computer Science. Springer, Heidelberg, 2009-03, pp. 481–500. DOI: 10.1007/978-3-642-00468-1_27.

[CL01] Yan-Cheng Chang and Chi-Jen Lu. "Oblivious Polynomial Evaluation and Oblivious Neural Learning". In: *Advances in Cryptology – ASIACRYPT 2001*. Ed. by Colin Boyd. Vol. 2248. Lecture Notes in Computer Science. Springer, Heidelberg, 2001-12, pp. 369–384. DOI: 10.1007/3-540-45682-1_22.

[CL02] Jan Camenisch and Anna Lysyanskaya. "Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials". In: *Advances in Cryptology – CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. Lecture Notes in Computer Science. Springer, Heidelberg, 2002-08, pp. 61–76. DOI: 10.1007/3-540-45708-9_5.

[CLN16] Craig Costello, Patrick Longa, and Michael Naehrig. "Efficient Algorithms for Supersingular Isogeny Diffie-Hellman". In: *Advances in Cryptology – CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. Lecture Notes in Computer Science. Springer, Heidelberg, 2016-08, pp. 572–601. DOI: 10.1007/978-3-662-53018-4_21.

[Coh00] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Vol. 138. Graduate Texts in Mathematics Ser. Springer Berlin / Heidelberg, 2000, p. 556. ISBN: 9783642081422. DOI: 10.1007/978-3-662-02945-9.

[Cop96] Don Coppersmith. "Finding a Small Root of a Univariate Modular Equation". In: *Advances in Cryptology – EUROCRYPT'96*. Ed. by Ueli M. Maurer. Vol. 1070. Lecture Notes in Computer Science. Springer, Heidelberg, 1996-05, pp. 155–165. DOI: 10.1007/3-540-68339-9_14.

[Cos+20] Craig Costello, Patrick Longa, Michael Naehrig, Joost Renes, and Fernando Virdia. "Improved Classical Cryptanalysis of SIKE in Practice". In: *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II*. Ed. by Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas. Vol. 12111. Lecture Notes in Computer Science. Springer, Heidelberg, 2020-05, pp. 505–534. DOI: 10.1007/978-3-030-45388-6_18.

[Cos20] Craig Costello. "B-SIDH: Supersingular Isogeny Diffie-Hellman Using Twisted Torsion". In: *Advances in Cryptology – ASIACRYPT 2020, Part II*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12492. Lecture Notes in Computer Science. Springer, Heidelberg, 2020-12, pp. 440–463. DOI: 10.1007/978-3-030-64834-3_15.

[Cox13] David A. Cox. *Primes of the Form $x^2 + ny^2$: Fermat, Class Field Theory, and Complex Multiplication: Second Edition*. 2013-01, pp. 1–356. ISBN: 978-1-118-39018-4. DOI: 10.1002/9781118400722.

[CS18] Craig Costello and Benjamin Smith. "Montgomery curves and their arithmetic - The case of large characteristic fields". In: *Journal of Cryptographic Engineering* 8.3 (2018-09), pp. 227–240. DOI: 10.1007/s13389-017-0157-6.

[Dam+12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. "Multiparty Computation from Somewhat Homomorphic Encryption". In: *Advances in Cryptology – CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. Lecture Notes in Computer Science. Springer, Heidelberg, 2012-08, pp. 643–662. DOI: 10.1007/978-3-642-32009-5_38.

[Dam+13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. "Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits". In: *ESORICS 2013: 18th European Symposium on Research in Computer Security*. Ed. by Jason Crampton, Sushil Jajodia, and Keith Mayes. Vol. 8134. Lecture Notes in Computer Science. Springer, Heidelberg, 2013-09, pp. 1–18. DOI: 10.1007/978-3-642-40203-6_1.

[Dam90] Ivan Damgård. "On the Randomness of Legendre and Jacobi Sequences". In: *Advances in Cryptology – CRYPTO'88*. Ed. by Shafi Goldwasser. Vol. 403. Lecture Notes in Computer Science. Springer, Heidelberg, 1990-08, pp. 163–172. DOI: 10.1007/0-387-34799-2_13.

[Das] Dash. *Dash website*. URL: https://www.dash.org/.

[Dav05] David Broadhurst. *A 5061-digit semiprime*. https://web.archive.org/web/20190827000752/http://physics.open.ac.uk/~dbroadhu/cert/semgpch.gp. 2005.

[Dav31] Harold Davenport. "On the distribution of quadratic residues (mod p)". In: *Journal of the London Mathematical Society* 1.1 (1931), pp. 49–54.

[Dav39] Harold Davenport. "On character sums in finite fields". In: *Acta Mathematica* 71.1 (1939), pp. 99–121.

[Del+21] Cyprien Delpech de Saint Guilhem, Eleftheria Makri, Dragos Rotaru, and Titouan Tanguy. "The Return of Eratosthenes: Secure Generation of RSA Moduli Using Distributed Sieving". In: *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. CCS '21. Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, pp. 594–609. ISBN: 9781450384544. DOI: 10.1145/3460120.3484754.

[Dem94] N. Demytko. "A New Elliptic Curve Based Analogue of RSA". In: *Advances in Cryptology – EUROCRYPT'93*. Ed. by Tor Helleseth. Vol. 765. Lecture Notes in Computer Science. Springer, Heidelberg, 1994-05, pp. 40–49. DOI: 10.1007/3-540-48285-7_4.

[DH00] Wim van Dam and Sean Hallgren. "Efficient quantum algorithms for shifted quadratic character problems". In: *arXiv preprint* (2000). URL: https://arxiv.org/abs/quant-ph/0011067.

[DJP11] Luca De Feo, David Jao, and Jérôme Plût. *Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies*. Cryptology ePrint Archive, Report 2011/506. https://eprint.iacr.org/2011/506. 2011.

[Don05] Don Reble. *Interesting Semiprimes*. http://www.graysage.com/djr/isp.txt. 2005.

[DT08] Ivan Damgård and Nikos Triandopoulos. *Supporting Non-membership Proofs with Bilinear-map Accumulators*. Cryptology ePrint Archive, Report 2008/538. https://eprint.iacr.org/2008/538. 2008.

[Dud78] Underwood Dudley. *Elementary Number Theory: Second Edition*. Dover Books on Mathematics. Dover Publications, 1978. ISBN: 9780486469317.

[Elk98] Noam D. Elkies. "Elliptic and modular curves over finite fields and related computational issues". In: *Computational Perspectives on Number Theory: Proceedings of a Conference in Honor of A. O. L. Atkin*. Ed. by D. A. Buell and editors J. T. Teitelbaum. Vol. 7. American Mathematical Society, 1998, pp. 21–76. DOI: 10.1090/amsip/007/03.

[Fei19a] Dankrad Feist. *Cryptanalyzing the Legendre PRF*. Advances in Cryptology – CRYPTO rump session talk. 2019-08.

[Fei19b] Dankrad Feist. *Legendre pseudo-random function*. https://legendreprf.org. 2019.

[Fei19c] Dankrad Feist. *Using the Legendre symbol as a PRF for the Proof of Custody*. https://ethresear.ch/t/using-the-legendre-symbol-as-a-prf-for-the-proof-of-custody. 2019.

[FN20] Brett H. Falk and Daniel Noble. *Secure Computation over Lattices and Elliptic Curves*. Cryptology ePrint Archive, Report 2020/926. https://eprint.iacr.org/2020/926. 2020.

[FS87] Amos Fiat and Adi Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *Advances in Cryptology – CRYPTO'86*. Ed. by Andrew M. Odlyzko. Vol. 263. Lecture Notes in Computer Science. Springer, Heidelberg, 1987-08, pp. 186–194. DOI: 10.1007/3-540-47721-7_12.

[Gal99] Steven D. Galbraith. "Constructing Isogenies between Elliptic Curves Over Finite Fields". In: *LMS Journal of Computation and Mathematics* 2 (1999), pp. 118–138. DOI: 10.1112/S1461157000000097.

[GK86] S Goldwasser and J Kilian. "Almost All Primes Can Be Quickly Certified". In: *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*. STOC '86. Berkeley, California, USA: Association for Computing Machinery, 1986, pp. 316–329. ISBN: 0897911938. DOI: 10.1145/12130.12162.

[GPS08] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. "Pairings for Cryptographers". In: *Discrete Applied Mathematics* 156.16 (2008-09), pp. 3113–3121. ISSN: 0166-218X. DOI: 10.1016/j.dam.2007.12.010.

[Gra+16] Lorenzo Grassi, Christian Rechberger, Dragos Rotaru, Peter Scholl, and Nigel P. Smart. "MPC-Friendly Symmetric Key Primitives". In: *ACM CCS 2016: 23rd Conference on Computer and Communications Security*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM Press, 2016-10, pp. 430–443. DOI: 10.1145/2976749.2978332.

[Gro16] Jens Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: *Advances in Cryptology – EUROCRYPT 2016, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. Lecture Notes in Computer Science. Springer, Heidelberg, 2016-05, pp. 305–326. DOI: 10.1007/978-3-662-49896-5_11.

[GS08] Jens Groth and Amit Sahai. "Efficient Non-interactive Proof Systems for Bilinear Groups". In: *Advances in Cryptology – EUROCRYPT 2008*. Ed. by Nigel P. Smart. Vol. 4965. Lecture Notes in Computer Science. Springer, Heidelberg, 2008-04, pp. 415–432. DOI: 10.1007/978-3-540-78967-3_24.

[Haz15] Carmit Hazay. "Oblivious Polynomial Evaluation and Secure Set-Intersection from Algebraic PRFs". In: *TCC 2015: 12th Theory of Cryptography Conference, Part II*. Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9015. Lecture Notes in Computer Science. Springer, Heidelberg, 2015-03, pp. 90–120. DOI: 10.1007/978-3-662-46497-7_4.

[IT05] Tetsuya Izu and Tsuyoshi Takagi. "Fast Elliptic Curve Multiplications Resistant against Side Channel Attacks". In: *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 88-A.1 (2005), pp. 161–171. DOI: 10.1093/ietfec/E88-A.1.161.

[Jac06] Ernst Erich Jacobsthal. "Anwendungen einer Formel aus der Theorie der quadratischen Reste". PhD thesis. Friedrich-Wilhelms Universität zu Berlin, 1906. Chap. 3. URL: https://edoc.hu-berlin.de/bitstream/handle/18452/739/26850.pdf.

[Jao+20] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca de Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. *Supersingular Isogeny Key Encapsulation*. https://sike.org/files/SIDH-spec.pdf. NIST Post-Quantum Cryptography Round 3 - Alternate Candidate. 2020-10.

[JD11] David Jao and Luca De Feo. "Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies". In: *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*. Ed. by Bo-Yin Yang. Springer, Heidelberg, 2011-11 / 12, pp. 19–34. DOI: 10.1007/978-3-642-25405-5_2.

[Kel20] Marcel Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020. DOI: 10.1145/3372297.3417872.

[Kho19] Dmitry Khovratovich. *Key recovery attacks on the Legendre PRFs within the birthday bound*. Cryptology ePrint Archive, Report 2019/862. https://eprint.iacr.org/2019/862. 2019.

[KK00] Noboru Kunihiro and Kenji Koyama. "Two Discrete Log Algorithms for Super-Anomalous Elliptic Curves and Their Applications". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 83 (2000), pp. 10–16.

[KKK20] Novak Kaluđerović, Thorsten Kleinjung, and Dusan Kostic. *Improved key recovery on the Legendre PRF*. Cryptology ePrint Archive, Report 2020/098. https://eprint.iacr.org/2020/098. 2020.

[KW21] Thorsten Kleinjung and Benjamin Wesolowski. "Discrete logarithms in quasi-polynomial time in finite fields of fixed characteristic". In: *Journal of the American Mathematical Society* (2021-09), p. 44. DOI: 10.1090/jams/985.

[Len87] Hendrik W. Lenstra. "Factoring Integers with Elliptic Curves". In: *Annals of Mathematics* 126.3 (1987), pp. 649–673. DOI: 10.2307/1971363.

[Lip12] Helger Lipmaa. "Secure Accumulators from Euclidean Rings without Trusted Setup". In: *ACNS 12: 10th International Conference on Applied Cryptography and Network Security*. Ed. by Feng Bao, Pierangela Samarati, and Jianying Zhou. Vol. 7341. Lecture Notes in Computer Science. Springer, Heidelberg, 2012-06, pp. 224–240. DOI: 10.1007/978-3-642-31284-7_14.

[LL93] Arjen K. Lenstra and Hendrik .W. Lenstra. *The Development of the Number Field Sieve*. Lecture Notes in Mathematics. Springer, 1993. DOI: 10.1007/BFb0091534.

[LLX07] Jiangtao Li, Ninghui Li, and Rui Xue. "Universal Accumulators with Efficient Nonmembership Proofs". In: *ACNS 07: 5th International Conference on Applied Cryptography and Network Security*. Ed. by Jonathan Katz and Moti Yung. Vol. 4521. Lecture Notes in Computer Science. Springer, Heidelberg, 2007-06, pp. 253–269. DOI: 10.1007/978-3-540-72738-5_17.

[LWS21] Patrick Longa, Wen Wang, and Jakub Szefer. "The Cost to Break SIKE: A Comparative Hardware-Based Analysis with AES and SHA-3". In: *Advances in Cryptology – CRYPTO 2021, Part III*. Ed. by Tal Malkin and Chris Peikert. Vol. 12827. Lecture Notes in Computer Science. Virtual Event: Springer, Heidelberg, 2021-08, pp. 402–431. DOI: 10.1007/978-3-030-84252-9_14.

[Mei+13] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. "A Fistful of Bitcoins: Characterizing Payments Among Men with No Names". In: *Proceedings of the 2013 Conference on Internet Measurement Conference*. IMC '13. Barcelona, Spain: ACM, 2013, pp. 127–140. ISBN: 978-1-4503-1953-9. DOI: 10.1145/2504730.2504747.

[Mer90] Ralph C. Merkle. "A Certified Digital Signature". In: *Advances in Cryptology – CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Springer, Heidelberg, 1990-08, pp. 218–238. DOI: 10.1007/0-387-34805-0_21.

[Mic21a] Microsoft Research. *SIDH v3.4 (C Edition)*. https://github.com/microsoft/PQCrypto-SIDH. 2021.

[Mic21b] Microsoft Research. *SIKE Cryptographic Challenge*. https://www.microsoft.com/en-us/msrc/sike-cryptographic-challenge. 2021.

[MOM92] Hikaru Morita, Kazuo Ohta, and Shoji Miyaguchi. "A Switching Closure Test to Analyze Cryptosystems". In: *Advances in Cryptology – CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Springer, Heidelberg, 1992-08, pp. 183–193. DOI: 10.1007/3-540-46766-1_13.

[Mon] Monero. *Monero website*. URL: https://web.getmonero.org/.

[Mon87a] Peter L. Montgomery. "Speeding the Pollard and elliptic curve methods of factorization". In: *Mathematics of Computation* 48 (1987), pp. 243–264. ISSN: 0025–5718. DOI: 10.1090/S0025-5718-1987-0866113-7.

[Mon87b] Peter L. Montgomery. "Speeding the Pollard and elliptic curve methods of factorization". In: *Mathematics of Computation* 48 (1987), pp. 243–264. DOI: 10.1090/S0025-5718-1987-0866113-7.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. USA: Cambridge University Press, 2005. ISBN: 0521835402. DOI: 10.1017/CBO9780511813603.

[Nak08] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. Tech. rep. 2008. URL: https://bitcoin.org/bitcoin.pdf.

[Nat22] National Institute of Standards and Technology (NIST). *Post-Quantum Cryptography Standardization*. https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization. 2016/2022.

[NF19] Abderrahmane Nitaj and Emmanuel Fouotsa. "A new attack on RSA and Demytko's elliptic curve cryptosystem". In: *Journal of Discrete Mathematical Sciences and Cryptography* 22.3 (2019), pp. 391–409. DOI: 10.1080/09720529.2019.1587827.

[Ngu05] Lan Nguyen. "Accumulators from Bilinear Pairings and Applications". In: *Topics in Cryptology – CT-RSA 2005*. Ed. by Alfred Menezes. Vol. 3376. Lecture Notes in Computer Science. Springer, Heidelberg, 2005-02, pp. 275–292. DOI: 10.1007/978-3-540-30574-3_19.

[NP06] Moni Naor and Benny Pinkas. "Oblivious Polynomial Evaluation". In: *SIAM Journal on Computing* 35.5 (2006), pp. 1254–1281. DOI: 10.1137/S0097539704383633.

[NS20] María Naya-Plasencia and André Schrottenloher. "Optimal Merging in Quantum k-xor and k-xor-sum Algorithms". In: *Advances in Cryptology – EUROCRYPT 2020, Part II*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12106. Lecture Notes in Computer Science. Springer, Heidelberg, 2020-05, pp. 311–340. DOI: 10.1007/978-3-030-45724-2_11.

[Oec03] Philippe Oechslin. "Making a Faster Cryptanalytic Time-Memory Trade-Off". In: *Advances in Cryptology – CRYPTO 2003*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Springer, Heidelberg, 2003-08, pp. 617–630. DOI: 10.1007/978-3-540-45146-4_36.

[Ped92] Torben P. Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Advances in Cryptology – CRYPTO'91*. Ed. by Joan Feigenbaum. Vol. 576. Lecture Notes in Computer Science. Springer, Heidelberg, 1992-08, pp. 129–140. DOI: 10.1007/3-540-46766-1_9.

[Pie19] Krzysztof Pietrzak. "Simple Verifiable Delay Functions". In: *ITCS 2019: 10th Innovations in Theoretical Computer Science Conference*. Ed. by Avrim Blum. Vol. 124. LIPIcs, 2019-01, 60:1–60:15. DOI: 10.4230/LIPIcs.ITCS.2019.60.

[Piz90] Arnold Pizer. "Ramanujan graphs and Hecke operators". In: *Bulletin of the American Mathematical Society* 23 (1990), pp. 127–137. DOI: 10.1090/S0273-0979-1990-15918-X.

[Piz98] Arnold Pizer. "Ramanujan graphs". In: *Computational perspectives on number theory (Chicago, IL, 1995)* 7 (1998), pp. 159–178.

[Ren18] Joost Renes. "Computing Isogenies Between Montgomery Curves Using the Action of (0, 0)". In: *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*. Ed. by Tanja Lange and Rainer Steinwandt. Springer, Heidelberg, 2018, pp. 229–247. DOI: 10.1007/978-3-319-79063-3_11.

[RH13] Fergal Reid and Martin Harrigan. "An Analysis of Anonymity in the Bitcoin System". In: *Security and Privacy in Social Networks*. Ed. by Yaniv Altshuler, Yuval Elovici, Armin B. Cremers, Nadav Aharony, and Alex Pentland. New York, NY: Springer New York, 2013, pp. 197–223. ISBN: 978-1-4614-4139-7. DOI: 10.1007/978-1-4614-4139-7_10.

[RS06] Alexander Rostovtsev and Anton Stolbunov. *Public-Key Cryptosystem Based On Isogenies*. Cryptology ePrint Archive, Report 2006/145. https://eprint.iacr.org/2006/145. 2006.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the Association for Computing Machinery* 21.2 (1978), pp. 120–126. DOI: 10.1145/359340.359342.

[San99] Tomas Sander. "Efficient Accumulators without Trapdoor Extended Abstracts". In: *ICICS 99: 2nd International Conference on Information and Communication Security*. Ed. by Vijay Varadharajan and Yi Mu. Vol. 1726. Lecture Notes in Computer Science. Springer, Heidelberg, 1999-11, pp. 252–262.

[Sch20] Berry Schoenmakers. *Lecture Notes Cryptographic Protocols*. 2020. URL: https://www.win.tue.nl/~berry/CryptographicProtocols/LectureNotes.pdf.

[Sch87] René Schoof. "Nonsingular plane cubic curves over finite fields". In: *Journal of Combinatorial Theory, Series A* 46.2 (1987), pp. 183–211. ISSN: 0097-3165. DOI: 10.1016/0097-3165(87)90003-3.

[Sch95] René Schoof. "Counting points on elliptic curves over finite fields". In: *Journal de Théorie des Nombres de Bordeaux* 7.1 (1995), pp. 219–254. DOI: doi.org/10.5802/jtnb.142.

[Sed+19] Vladimir Sedlacek, Dusan Klinec, Marek Sys, Petr Svenda, and Vashek Matyas. "I Want to Break Square-free: The $4p-1$ Factorization Method and Its RSA Backdoor Viability". In: *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications (ICETE 2019) - Volume 2: SECRYPT,* INSTICC. SciTePress, 2019, pp. 25–36. ISBN: 978-989-758-378-0. DOI: 10.5220/0007786600250036.

[Sha48] Claude E. Shannon. "A mathematical theory of communication". In: *Bell Syst. Tech. J.* 27.3 (1948), pp. 379–423. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[Sha49] C. E. Shannon. "Communication theory of secrecy systems". In: *The Bell System Technical Journal* 28.4 (1949), pp. 656–715. DOI: 10.1002/j.1538-7305.1949.tb00928.x.

[Shi17] Masaaki Shirase. *Condition on composite numbers easily factored with elliptic curve method*. Cryptology ePrint Archive, Report 2017/403. https://eprint.iacr.org/2017/403. 2017.

[Sho97] Victor Shoup. "Lower Bounds for Discrete Logarithms and Related Problems". In: *Advances in Cryptology – EUROCRYPT'97*. Ed. by Walter Fumy. Vol. 1233. Lecture Notes in Computer Science. Springer, Heidelberg, 1997-05, pp. 256–266. DOI: 10.1007/3-540-69053-0_18.

[Sil09] Joseph Silverman. *The Arithmetic of Elliptic Curves*. Vol. 106. 2009-01. ISBN: 978-0-387-09493-9. DOI: 10.1007/978-0-387-09494-6.

[Sim83] Gustavus J. Simmons. "The Prisoners' Problem and the Subliminal Channel". In: *Advances in Cryptology – CRYPTO'83*. Ed. by David Chaum. Plenum Press, New York, USA, 1983, pp. 51–67. DOI: 10.1007/978-1-4684-4730-9_5.

[Sim85] Gustavus J. Simmons. "The Subliminal Channel and Digital Signature". In: *Advances in Cryptology – EUROCRYPT'84*. Ed. by Thomas Beth, Norbert Cot, and Ingemar Ingemarsson. Vol. 209. Lecture Notes in Computer Science. Springer, Heidelberg, 1985-04, pp. 364–378. DOI: 10.1007/3-540-39757-4_25.

[Sim94] Gustavus J. Simmons. "Subliminal Communication is Easy Using the DSA". In: *Advances in Cryptology – EUROCRYPT'93*. Ed. by Tor Helleseth. Vol. 765. Lecture Notes in Computer Science. Springer, Heidelberg, 1994-05, pp. 218–232. DOI: 10.1007/3-540-48285-7_18.

[ST19] Nigel P. Smart and Younes Talibi Alaoui. "Distributing Any Elliptic Curve Based Protocol". In: *17th IMA International Conference on Cryptography and Coding*. Ed. by Martin Albrecht. Vol. 11929. Lecture Notes in Computer Science. Springer, Heidelberg, 2019-12, pp. 342–366. DOI: 10.1007/978-3-030-35199-1_17.

[Ste98] Robert von Sterneck. "Sur la distribution des résidus et des non-résidus quadratiques d'un nombre premier". In: *Matematicheskii Sbornik* 20.2 (1898), pp. 269–284.

[Sto10] Anton Stolbunov. "Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves". In: *Advances in Mathematics of Communications* 4.2 (2010), pp. 215–235. DOI: 10.3934/amc.2010.4.215.

[Sut11] Andrew V. Sutherland. "Computing Hilbert Class Polynomials With The Chinese Remainder Theorem". In: *Mathematics of Computation* 80.273 (2011), pp. 501–538. DOI: 10.1090/S0025-5718-2010-02373-7.

[Tao15] Terence Tao. *Cycles of a random permutation and irreducible factors of a random polynomial*. https://terrytao.wordpress.com/2015/07/15/cycles-of-a-random-permutation-and-irreducible-factors-of-a-random-polynomial/. 2015.

[Tat66] John Tate. "Endomorphisms of abelian varieties over finite fields". In: *Inventiones mathematicae* 2.2 (1966), pp. 134–144. DOI: 10.1007/BF01404549.

[The21] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.4)*. https://www.sagemath.org. 2021.

[UV21] Aleksei Udovenko and Giuseppe Vitto. *Breaking the $IKEp182 Challenge*. Cryptology ePrint Archive, Report 2021/1421. https://eprint.iacr.org/2021/1421. 2021.

[Var+14] Sébastien Varrette, Pascal Bouvry, Hyacinthe Cartiaux, and Fotis Georgatos. "Management of an Academic HPC Cluster: The UL Experience". In: *Proc. of the 2014 Intl. Conf. on High Performance Computing & Simulation (HPCS 2014)*. Bologna, Italy: IEEE, 2014-07, pp. 959–967.

[VB22] Giuseppe Vitto and Alex Biryukov. "Dynamic Universal Accumulator with Batch Update over Bilinear Groups". In: *Topics in Cryptology – CT-RSA 2022*. Ed. by Steven Galbraith. Vol. 13161. Lecture Notes in Computer Science. Reproduced with permission from Springer Nature. Springer, Cham, 2022, pp. 395–426. DOI: 10.1007/978-3-030-95312-6_17.

[Vit21] Giuseppe Vitto. *Factoring Primes to Factor Moduli: Backdooring and Distributed Generation of Semiprimes*. Cryptology ePrint Archive, Report 2021/1610. https://eprint.iacr.org/2021/1610. 2021.

[vW94] Paul C. van Oorschot and Michael J. Wiener. "Parallel Collision Search with Application to Hash Functions and Discrete Logarithms". In: *ACM CCS 94: 2nd Conference on Computer and Communications Security*. Ed. by Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, and Ravi S. Sandhu. ACM Press, 1994-11, pp. 210–218. DOI: 10.1145/191177.191231.

[vW99] Paul C. van Oorschot and Michael J. Wiener. "Parallel Collision Search with Cryptanalytic Applications". In: *Journal of Cryptology* 12.1 (1999-01), pp. 1–28. DOI: 10.1007/PL00003816.

[Wag02] David Wagner. "A Generalized Birthday Problem". In: *Advances in Cryptology – CRYPTO 2002*. Ed. by Moti Yung. Vol. 2442. Lecture Notes in Computer Science. Springer, Heidelberg, 2002-08, pp. 288–303. DOI: 10.1007/3-540-45708-9_19.

[Was08] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography, Second Edition*. 2nd ed. Chapman and Hall/CRC, 2008. ISBN: 9781420071467.

[Wei48] André Weil. "On some exponential sums". In: *Proceedings of the National Academy of Sciences of the United States of America* 34.5 (1948), p. 204.

[Wes19] Benjamin Wesolowski. "Efficient Verifiable Delay Functions". In: *Advances in Cryptology – EUROCRYPT 2019, Part III*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11478. Lecture Notes in Computer Science. Springer, Heidelberg, 2019-05, pp. 379–407. DOI: 10.1007/978-3-030-17659-4_13.

[Woo14] Gavin Wood. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151 (2014), pp. 1–32. URL: https://ethereum.org/.

[YY04] Adam Young and Moti Yung. *Malicious Cryptography: Exposing Cryptovirology*. USA: John Wiley & Sons, Inc., 2004. ISBN: 0764549758.

[Zca] Zcash. *Zcash official website*. URL: https://z.cash/.

[Zca18] Zcash. *Zcash Sapling Release*. 2018-10. URL: https://z.cash/upgrade/sapling/.