LATTICE-BASED PRIVACY-PRESERVING PROTOCOLS WITH DISTRIBUTED TRUST

PATRICK HOUGH

St. Hugh's College

University of Oxford

A thesis submitted for the degree of

DOCTOR OF PHILOSOPHY

Hilary 2024

To my Mother and Father

ABSTRACT

In readiness for the eventual arrival of quantum computers, there has been much work done on the design of quantum-safe cryptographic primitives. Propelled by the conclusion of the NIST PQC standardization effort, the global tech industry is beginning a transition to the use of public-key and digital signature schemes, based on quantum-safe computational hardness assumptions. In particular, lattice-based designs appear to be a widely favoured choice.

Whilst appetite for shoring up the security of our digital landscape in advance of a quantum era, is self-evident, the preservation of user-privacy in that landscape is also beginning to gain traction. Once again, lattices have emerged to provide the means to marry the often competing notions of privacy an security. Most notably, the advances in zero-knowledge proofs, multi-party computation, and fully-homomorphic encryption suggest a world in which we can enjoy the fruits of digital technology whilst avoiding the abuses of privacy through centralised aggregation of user-data seen in recent years. Nevertheless, research on the development of quantum-safe privacy-preserving protocols has still seen relatively little attention in the literature.

In this thesis we present a number of lattice-based primitives which aim to straddle the demands of post-quantum security and privacy preservation. In particular, our designs exploit the notion of distributed trust to achieve their privacy objectives whilst building upon lattice assumptions for quantum-security.

We propose a game-based security framework for direct anonymous attestation (DAA) of IoT devices containing a trusted-platform module (TPM). Building upon this, we present the first truly efficient lattice-based DDA based on the hardness of problems over structured lattices, achieving efficiency gains of more than an order of magnitude over previous works.

Finally, we present and implement an electronic voting scheme relying on the NTRU lattice assumption. This improves upon the efficiency of the state-of-the-art with communication costs and running times around 2-3 times less.

PUBLICATIONS

Articles fully included in the thesis:

- Jan Camenisch, Manu Drijvers, Patrick Hough, Ali El Kaafarani, Shuichi Katsumata, Anja Lehmann, Khoa Nguyen. DAA Security, Revisited: Property-Based Definitions with UC Endorsement. 2020. In submission to the Journal of Cryptology.
- Patrick Hough, Tjerand Silde, Caroline Sandsbråten. Concrete NTRU security and Advances in Practical Lattice-Based Electronic Voting. 2023. In submission to PETs 2024.
  (Full version at `https://eprint.iacr.org/2023/933`)
- Liqun Chen, Patrick Hough, Nada El Kassem. A More Efficient Lattice-Based DAA. 2024. In submission to Africacrypt 2024.

Other articles:

- Jan Camenisch, Liqun Chen,Rachid El Bansarkhani, Patrick Hough, Ali El Kaafarani, Nada El Kassem. L-DAA: Lattice-Based Direct Anonymous Attestation. 2018.
  (Full version at `https://eprint.iacr.org/2018/401`)
- Jan Camenisch, Liqun Chen,Rachid El Bansarkhani, Patrick Hough, Ali El Kaafarani, Nada El Kassem, Paulo Martins, Leonel Sousa. More efficient, provably-secure direct anonymous attestation from lattices. 2019.
  (Full article at `https://www.sciencedirect.com/science/article/pii/S0167739X19300536?via%3Dihub`)
- Patrick Hough, Tjerand Silde, Caroline Sandsbråten. Distributed Key-Generation for NTRU. 2024.

ACKNOWLEDGEMENTS

In setting out to acknowledge the great many people who have got me to this position, a possibility emerged: this may indeed be the most-read part of my thesis and, for some, the only part they read. Moreover, it can be the only passage which attempts to map the *human* achievement, reach, and cost of completing a doctorate. I don't intend to indulge in such an exercise for the exposition of my own achievement, its impact, or the emotional toll it has taken; the former are for others to judge and the latter 'we leave for future work'. I am, however, indebted to many people for their contributions academically, socially, and administratively.

I begin, as is customary, by thanking my primary supervisor Dr. Ali Kaafarani. I thank him for his early academic involvement in my PhD and in particular for his connecting me with the industry placements at IBM, Zurich and AIST, Tokyo. These were some of the most transformative opportunities and I am truly grateful for his arrangement of them. More recently, I want to thank him for his administrative support, particularly at late notice and knowing the demands he faces in running his company. Perhaps the most impactful advice he gave to me is that 'life if too short to change people'. My subconscious sings these words weekly and they have brought peace in challenging times.

I would also like to thank Prof. Liqun Chen who has acted as my secondary supervisor and as a collaborator. I thank her for her unwavering calm, kindness, and diligence in reading things I send to her. Her relentlessness and organisation has been a great inspiration.

I would like to pay special thanks to Prof. Tom Sanders who has acted as my pastoral supervisor. He has had the privilege of seeing me at moments of my greatest despair, confusion, and loneliness. I owe Tom a great debt of gratitude, not least for lending me use of his office through the pandemic and for being a constant in my supervision.

Those who are familiar with the cryptography group at Oxford university will be familiar with me. That is to say, there is only me. Though a lonely furrow at times, this has given me great reason to visit research groups all over the world. Through these trips I have met some of my greatest friends and collaborators.

I would like to give my special thanks to those at IBM Research, Zurich. The two summers I spent were, by some margin, the happiest time of my

PhD. Firstly, I would like to thank Jan Camenisch and Anja Lehmann for hosting me there and for bestowing on my research trajectory the DAA protocol, which I so love. Jan, I would like to thank for his wry smile, one that seemed to forgive the infancy of my cryptographic powers. Indeed, when I reflect on those I collaborated with at IBM, all were, without exception, incredibly generous with their patience and generosity of ideas. I thank Anja also for her patience, warmth, and ability to speak faster than I could think. I would also like to thank Manu (for his mastery of DAA and dazzling forearms), Patrick 2 (for his confidences and ability to float), Cecilia (for her climbing humiliations), Gregor (for his quiet typing, shared love of cool temperatures, and debilitating humour), Khanh (for gifting me my very own 'lantern'), Anna (for the adventures), Gregory (for his iridescent warmth and fabulous shirts), Tommasso (for his irregular billiards form), Rafael (for his knowledge of lattices and other salad constituents), Nick (for maintaining a cryptographically-capable British front), Lydia (for her recent career guidance), Michael Osborne (for such a warm welcome and oversight of my visits and whose surname always needs to be said to enforce the grandeur of the man), Alexandra Gorski (and her nice dog for their administrative support), Jonathan Bootle (for being such an example of how kindness and humility need not be lost at the top), Jesus (for the generosity of conversation about his family and life and ability to resist overthrowing the tables each time the canteen staff insisted on the messianic pronunciation), Vadim (for his appreciation of my running (from life and otherwise)). Finally I would like to further thank Jan for use of his Speedos, a connection I know will secure my professional success.

I would also like to thank my fantastic collaborators Tjerand Silde and Caroline Sandsbråten at NTNU, Norway. In particular I would like to thank Tjerand for inviting me to visit his beautiful city, and most significantly for giving me a sense that he 'believed in me'. I can't overstate how our meeting began to turn so many personal tides.

I would also like to thank Shuichi Katsumata for his friendship, something that has been, and will always be strong enough to be distinguished from the world of maths.

I would like to thank my office mates for their friendship and camaraderie. In many ways it has been a blessing that our mathematical areas are so distinct; I am so glad I don't study cohomologies.

Of course, the most crucial support one can have throughout a PhD is the love and support of one's friends and family. I have often bemoaned the fact that scholars of humanities can wear their research in flamboyant and

provocative ways; accessible as it is for dinner party conversation, supporting an image of cultivated social insight and political enlightenment. In contrast, leaving the automatic sliding doors of the mathematical institute often feels like the crossing of a precipice, beyond which the value of what I have seen that day evaporates. Nevertheless, I have come to really value this separation. At times, the greatest support through this period has been from those who do not interact with my maths world. By your friendship, you have given me meaning, value and perspective outside the often harsh environment of academia. The privilege of your friendships has shown me where to spend my respect and it will always lie with those who look outwards more than in, who have empathy, humility, and most of all kindness. Whilst too many to name, I give thanks to the friendships of my housemates (all of whom have been through this process themselves whilst still reserving space to support each other), to Henry, Jonny, and George (who I know will always go 'till the end), to Jake (for his companionship in this struggle to self-identify outside of the maths world), to Sandra (for her steadying hand in my most challenging times), to the Zurich hippies, and to David Mancuso.

I give unique thanks to Sophie for she is perhaps the only person to have borne the pressures of my PhD and its existential challenges more greatly than myself. Her unwavering support outshines all absence of support from those professionally obliged to emulate it.

Finally, and without hyperbole, I thank my parents for their profound, long-suffering support, sacrifice, and love. I have no doubt their endurance of supporting me through a life of mathematics, the obstacles of which are unfamiliar to them, has been far greater than my own experience of it. I am forever indebted to them and this thesis stands as a testament to their love.

CONTENTS

NOTATION

FREQUENTLY USED SYMBOLS

$\mathbb{N}$	set of natural numbers $\{1, 2, 3, \ldots\}$
$\mathbb{Z}_n$	ring of integers modulo n
$[n]$	$\{1, 2, \ldots.\}$
λ	security parameter
q	cryptosystem modulus
d	power-of-two, ring dimension
R	ring of integers $\mathbb{Z}[X]/(X^d + 1)$
R_q	ring $\mathbb{Z}_q[X]/(X^d + 1)$
$\tilde{f}$	constant coefficient of a polynomial f
$\mathcal{C}$	challenge space over R_q
D_σ	discreet Gaussian distribution with standard deviation σ

1

INTRODUCTION

Cryptography has historically been concerned with only question: how can two parties communicate in a *confidential* manner, having already agreed on a shared secret key? The arrival of public-key cryptography, ushered in by the key-exchange protocols of Diffie and Hellman [DH76], brought a satisfactory answer to this question. Indeed, the *authenticity* of messages following such a key-exchange could now be guaranteed.

The ecosystem of cryptographic designs propagating from their work has led to, and continues to underpin, much of our modern technological life e.g. through credit cards, passports, emails, smart phones, digital currencies, e-voting, and the HTTPS protocol which secures the internet we all use. The flourishing integration of cryptography with our modern lives has brought immeasurable convenience, functionality, and connectivity.

However, as we will discuss, the assurance of confidentiality has opened the door to the more subtle concern of who we trust as recipients of our secure communications. This is the issue of *privacy*. One of the most powerful solutions in tackling the issue of trust in the digital world is to enforce honest behavior by *distributing* power amongst a group of participants and in so doing, remove the potential for centralized aggregation of user data. Moreover, our commitment to answering the initial desire for confidentiality turns out to be one that needs serious renewal with the potential advent of quantum-computers. The scramble to preserve the initial successes of Diffie and Hellman in the face of this new threat, and the body of work proposing solutions, is called *post-quantum cryptography*. These key themes of privacy, distributed trust, and post-quantum cryptography are the core principles guiding the contributions of this thesis.

PRIVACY. Privacy has long been a cornerstone of our society, well before the current digital age. Yes, despite our total adoption of digital communications, very little is said about the preservation of personal privacy. In part this is due to the somewhat obscured world of cryptography and technology more generally, in which how the products we use on a daily basis actually operate is not well understood by the average user. Another, potentially more sinister contributing factor in this loss of personal privacy, is the monetization and harnessing of user data that is driving the global tech-

nology economy. It is often said that user data is like the 'new oil'. Indeed, this information comprising anything from our name and age, to our voting intentions and love interests is now such a commodity that many of the technologies that we use gather far more information about us than is necessary to perform the service that they provide. Another risk in handing over so much personal data is that it might get into the wrong hands. Indeed, the news of data leaks containing private user information comes daily. In the wrong hands this data can lead to a whole host of criminal activity such as identity theft, financial theft, and blackmail. The endless possibilities when possessing someone's private data are self-evident.

Ironically, cryptography enables many of these technologies we use to exist in the first place. However, cryptography is also keeping pace by providing so called 'privacy-preserving' solutions to these challenges. In a nutshell, we would like to have technologies that provide all of the functionality that they do today without giving over more personal information that is technically necessary for that technology to operate. Owing to the discovery of truly remarkable primitives such as interactive proof systems (ZKPs) [GMR85] an fully-homomorrphic encryption (FHE) [Gen09], we have the tools to do so.

In the face of the monetary business model associated with the collection of personal data, enthusiasm for the uptake of such techniques might seem premature. However, with end-users beginning to value privacy as an integral feature of technology they adopt, companies and developers are now racing to build privacy into their products. In terms of investment, the area of zero-knowledge research and the development of offerings using ZKPs is growing much faster than the cryptocurrency space. This indicates that the fightback for control of our data and communications is well under way.

DISTRIBUTED TRUST. Common to all abuses of digital privacy is the presense of a centralized party who, at some point in the protocol, has access to a complete set of the data that they exploit. From a practical perspective, it would seem that, in order to provide the service a technology offers, having all operations handled by a single omniscient entity would be most practical, if not essential. Again, cryptographic advances provide an alternative paradigm.

Perhaps the most famous example of decentralized technology is a blockchain, in which trust in the correctness of a public ledger is ensured by the distribution of verification and maintenance of the ledger. However, there are many areas in which the idea of distributing trust can provide a technology with

in-built privacy whilst retaining the same functionality of its centralized predecessor. Indeed, new research is fast showing that almost any cryptographic primitive can be done in a distributed or 'threshold' way (in which t-out-of-n parties must act honestly in order to execute a given task). This has prompted the National Institute of Standards and Technology (NIST) to call for the submission of threshold schemes for standardization [NISb]. This exercise seeks to provide distributed analogue schemes for many basic cryptographic tools e.g. encryption, decryption, signatures, key generation, and more.

Beyond distributing cryptography at a software level, there are increasing efforts to distribute trust *within* digital devices through the use of trusted hardware. The key idea here is to use untamperable hardware chips, containing secret key material, as a root-of-trust. Then, by requiring the involvement of such a chip, the execution of a cryptographic protocol is much more resistant to subversion than if all key material and operations were carried out at a software level, where a system is much more vulnerable to compromise. Several industry standardization groups such at the Trusted Computing Group (TCG) have long been developing such hardware with their Trusted Platform Module (TPM) chip estimated to be deployed in over 500 million devices. Similarly, the Fast IDentity Online (FIDO) Alliance seeks to solve the global password management issue by replacing passwords with keys stored in a secure hardware enclave. Similarly, Intel's SGX enclave provide a sanctuary for sensitive code of cryptography key material through the use of.

POST-QUANTUM CRYPTOGRAPHY AND LATTICES. The topic of post-quantum cryptography is one that is fast expanding both in terms of wider interest and necessity. The powers that a quantum computer might give and the implications for cryptography have long been understood. In 1994, Peter W. Shor published a paper [Sho94] that outlines an algorithm which, if run on a quantum computer, would break two of the most ubiquitous problems upon which much of modern cryptography is built; namely prime factorisation and finding discrete logs, both of which are thought to be NP-hard (classically). Shor's algorithm solves these problems in polynomial time, which is considered to be a full breach of any cryptosystem relying on them. Furthermore, in 1996 Lov K. Grover published [Gro96] what became known as Grover's search algorithm, which exploits the properties of quantum mechanics to give a quadratic speedup of database searches. This implies, at the very least, that cryptosystems whose security is not underpinned by the problem of factorisation or the discrete logarithm problem (DLP) would have

to have to be altered to use keys of double the current length.

In December 2016, owing to the advances in quantum computing, NIST made a call for submissions of 'quantum-resistant public-key cryptographic algorithms' [NISa]. The deadline for submissions was the 30th, November, 2017. The reader will note that the request is for *public-key* algorithms in particular. This is because symmetric cryptographic standards like AES are thought to be secure against quantum computers with only a slight increase in key size (double). This is largely due to the quadratic speedup that Grover's algorithm provides in attacking such systems. The main focus in developing post-quantum cryptography was on key-encapsulation mechanisms (KEM) (key-exchange) and digital signature schemes since solutions here are most urgently needed.

This process has now concluded with the selection of standards, approved through a long process of scrutiny by experts across academia, government, and industry. The KEM that will be standardized is CRYSTALS-KYBER and three digital signature schemes: CRYSTALS-Dillithium, FALCON, and SPHINCS$^+$. Apart from SPHINCS$^+$, all of these schemes rely on the hardness of computational assumption over structured *lattices*. Indeed, the overwhelming majority of initial submissions were built from lattice assumptions.

Lattices look set to stay in favor amongst options for providing security in a post-quantum age for several reasons. The use of lattice assumptions for cryptography stretches all the way back to Hoffstein, Piper, and Silverman [HPS98a] and their use off the 'NTRU' assumption over module lattices. The hardness of lattice problems has thus enjoyed a robust scrutiny over several decades. Computation over lattices is relatively simple to implement as they mostly involve linear operations e.g. multiplication of vectors and matrices. This has enabled efficient implementations, comparable to ones using classical hardness assumptions. Finally, lattices have given rise to primitives previously unattainable from other hardness assumptions. Perhaps the most famous example of this if fully-homomorphic encryption, which was somewhat of a holy-grain in cryptography before its realization from lattices.

1.1 OUR CONTRIBUTIONS

The areas of privacy, distributed trust, and post-quantum security represent three of the most pressing challenges in creating a safe and morally equitable digital future. Indeed, the rapid advances in each of these fields is encouragement that, while academia may have had their eye on these issues for a while, industry is beginning to take note and crucially the business model to

which it answers is beginning to align with these principles.

However, despite progress being made in each of these areas, research into cryptographic solutions in their *intersection*, has a long way to go. In particular, there has been very little work on privacy-preserving post-quantum designs.

In this thesis, we will focus on two main primitives which are designed with privacy, post-quantum security, and distributed trust at their core; *anonymous attestation* and *distributed electronic voting*. Before describing our contributions to these areas, we give a discussion of the background and related work in these areas.

1.1.1 *Direct Anonymous Attestation*

DAA: AN OVERVIEW. The ever growing ecosystem of devices connected to the internet poses an insurmountable task for security professionals aiming to secure such devices. Acknowledging both the finite number of trained professionals and shortcomings of software solutions, the industry standardisation body, The Trusted Computing Group (TCG), looks to trusted hardware as an axiom of security with its Trusted Platform Module (TPM) chip. In 2004 the TCG standardised the first Direct Anonymous Attestation (DAA) protocol in the TPM 1.2 specification [Tru04].

Now present in more than 500 million devices, and with support for multiple DAA schemes in the TPM 2.0 standardisation [Tru14], DAA is one of the most widely deployed protocols of such complexity. It has been adopted as a basis for EPID [BL11], Intel's recommendation as an industry standard for attestation of trusted systems in the Internet of Things. The standardisation of DAA has also been made by ISO as ISO/IEC 11889 international standards [ISO09, ISO15]. Furthermore, it is under consideration by FIDO for their specification of anonymous attestation [CDE$^+$]. In addition to these standards, DAA has applications for anonymous subscription [LDW$^+$13,KLL$^+$18] and vehicle communication (V2X) [PSFK15,WCG$^+$17].

Direct Anonymous Attestation (DAA) allows a TPM chip, embedded in a host computer, to make attestations about the state of the host system. The TPM is used as a root of trust to create signatures on the integrity of the host machine's hardware and software configuration. Such attestations convince a remote verifier that the platform (host computer and TPM) with whom it communicates is in a healthy state. Crucially, attestations must be anonymous so that a verifier is able to check that a signature originates from a legitimate platform without learning its identity. An additional fea-

ture of DAA is that platforms are able to make signatures linkable so that a verifier knows when two signatures have come from the same anonymous platform. This 'linkability' is steered by the use of a basename where signatures signed using the same basename are linkable whereas those created with a fresh basename are not.

As well as a number of platforms and verifiers, DAA includes an issuer who decides whether a platform can join (obtain signing credentials) or not. In practise this might be a service provider like Google or a government authority for example. Once joined, the platform can sign messages using a credential given by the issuer. These signatures might, for example, allow access to one of the issuer's services. For those familiar with the group signature setting, this process of joining, issuance, and signing (attesting) holds many similarities. However, in DAA the TPM and Host are required to *jointly* create signatures so that neither of them can make valid signatures without the involvement of the other. With the added complexity that the TPM and Host can be in different corruption states this makes defining DAA security a real challenge.

Alarmingly, in spite of wide deployment and a large body of work, from 2004-2016 there existed no sound and comprehensive security model of DAA. Indeed, to date, there exists no such property-based[1] definition and all previous attempts are demonstrably flawed. The shortcomings of DAA security definitions not only represent a theoretical problem but undermine schemes already deployed in practice. In particular, a DAA scheme that allows one to forge attestations (as it does not exclude the 'trivial' TPM credential $(1, 1, 1, 1)$) has even been standardised in ISO/IEC 20008-2 [ISO13, CPS10]. There are several non-trivial challenges in designing security definitions for DAA, however there are two significant obstacles that have, time and again, lead to flaws in security.

One of the most tricky aspects is the notion of TPM 'identity'. Unlike related protocols such as group signatures, where parties posses a certified public/secret key pair, the identity of a TPM is not clearly defined since there is no public key related to the secret key used in signature generation. This has long been the Achilles heel of DAA security models since it is crucial to be able to determine whether a given signature stems from some legitimate TPM. This is particularly important when one wants to prevent the existence of valid signatures that do not come from any legitimate TPM.

1 Note that 'property-based' is often used interchangeably with 'game-based' in the DAA literature.

It is then necessary to be able to determine if a given signature stems from any honest *or corrupt* TPM. The works [BCL08, Che10] skirt around this issue of corrupt TPM identification by assuming that corrupt TPMs already have their secret keys exposed in a public list RogueList. The assumption that a corrupt TPM's signing key is automatically publicly visible in some list is a very strong one. Moreover, a real life adversary has no obligation to publish a TPM's signing key if it were able to obtain one; it may choose simply to create forgeries with it. Furthermore, the works of [BCL08, Che10] do not allow for a notion of correctness in their RogueList process. The most recent work on property-based DAA by Bernard et al. [BFG+11] makes a hidden assumption that all TPM secret keys can be extracted from their signatures. Then a TPM's secret key can be used to determine whether a given signature came from that TPM. Unfortunately this extraction would take exponential time for their accompanying scheme and is not formally defined in their work.

Another aspect of DAA, which adds to the complexity of security models, is that a platform is made up of two entities, a TPM and host, who can be in different corruption states. This calls for extra consideration when defining the unforgeability of DAA since here we require that a corrupt host cannot produce a valid signature without the involvement of an honest TPM. For example, in standard signature schemes, unforgeability is only concerned with preventing forgeries under entirely honest users. In DAA however, an honest TPM who's host is corrupt produces signatures that we also want to be unforgeable. Therefore, our unforgeability definition must also cover these 'mixed' corrupt settings. This has been one of the major sources of complexity for DAA security over the years. One work [BFG+11] tries to combat this complexity by considering the TPM and Host as one party so that both platform components are in the same corruption state at all times. They then argue how to bootstrap this notion of 'Pre-DAA' security to (full) DAA security in which the TPM and Host can be in different corruption states as in the real world. Unfortunately, due to the absence of a full security proof, this lifting argument is only made informally. As shown in [CDL16], the argument for unforgeability cannot be lifted (under the same assumptions) to the full DAA setting. These two subtleties of DAA have been the pitfalls leading to many flawed security models over the years.

In addition to these two intrinsic difficulties with defining DAA security, the most recent work of Bernard et al. [BFG+11] further suffers a trivial security flaw by assuming that the issuer registers its public key with an extractable proof of correctness or that there is a trusted setup; something not

true for their scheme. This permits a corrupt issuer to choose its key in a malformed way, allowing it to win the non-frameability game. Also we observe that their model only permits the adversary a single call to the challenge oracle in their anonymity game allowing for a trivially non-anonymous scheme to be proven secure (see Section 3.3.3). For a comprehensive overview of the flaws in all existing property-based DAA definitions we refer the reader to Section 2.1 of [CDL16].

Owing to apparent difficulties with defining property-based DAA, Camenisch et al. [CDL16] turn to the Universal Composability (UC) model, formulating a DAA functionality to capture security. The UC model [Can01] provides a watertight framework for defining security. One creates an 'ideal functionality', performing the protocol in a perfectly secure way, and then proves a given instantiation indistinguishable from this functionality. The work of [CDL16] stands as the only solid security definition to date and forms the basis of a subsequent line of work [CDL17, CCD+17]. The later papers consider an add-on feature in which some security properties are required to hold even if the TPM is subverted. For the sake of clear presentation our work is in line with the TCG's foundational assumption that TPMs cannot be subverted. We thus refer to the original UC model in [CDL16]. In retrospect, the success of the UC definition over previous property-based ones is unsurprising given the framework's ability to model complex protocols like DAA whose intended use involves composition with other cryptographic protocols. In this way, one could argue that UC is the correct starting point for defining DAA security. Unfortunately, this definition is very hard to work with, even for those familiar with UC, where DAA proofs require as many as 17 game-hops. Consequently, researchers have continued to work with somewhat adapted property-based definitions that are still flawed.

Now that we have a UC based security notion which formally captures all the security notions of DAA, the next big step is to bring back these UC based security notions to the familiar property-based security setting. This would greatly simplify the proofs of DAA security and should form the basis for future work regarding DAA. However, to be absolutely sure that the property-based security definition is solid, unlike prior works, it is desirable to have a mechanism providing a 'sanity-check' that the property-based definition implies the UC-based definition.

Finally, we turn out attention to the status of post-quantum DAA. One of the aims of the Horizon 2020 PROMETHEUS project [H20], is to implement a post-quantum secure TPM chip (FutureTPM) using lattice cryptography.

This would require an efficient lattice-based DAA scheme.

To date, there have been a number off attempts at designing lattice-DAA (LDAA) schemes. The first paper to do so is by El Kaafarani and El Bansarkhani [EE17] who also present a new TAG scheme which provides user-controlled likability. Whilst an important first step, the work does not instantiate their scheme with concrete parameters and the security only provides asymptotic guarantees. Following this paper, El Kassem et al. [ECE+18] propose an LDAA. Unfortunately, this scheme required massive storage and computational resources deeming it unsuitable for implementation on the resource-constrained TPM. The last work in this direction [CKLL19] represents and improvement in the state-of-the-art efficiency. However, the authors do not provide concrete parameters and only estimate the resource costs of their scheme in a rather heuristic and optimistic way. Moreover, signature sizes in their work are estimated to be at least 2MB which is still some way off being considered practical.

DAA: OUR CONTRIBUTIONS. In this thesis we affirmatively close the gap between property-based and UC-based security models of DAA by providing a property-based security model that, after a syntactic re-structuring, implies the UC definition of Camenisch et al. [CDL16]. This provides a meaningful 'sanity-check' that our security model is both sound and comprehensive.

Our first contribution is to define a property-based model of DAA. Our model is based on a completely different design philosophy to other property-based works. Indeed, rather than patching the known flaws in preceding works, we extract the essential security properties of the UC definition, and reformulate them using property-based experiments. In particular, these properties are correctness, anonymity, unforgeability, and non-frameability. Since the UC functionality achieves security through a series of checks and conditions that may be interleaved across multiple interfaces this turns out to be a non-trivial task (For example see Figures 3.4 and 3.5 in Appendix 3.4 for the UC-based security model). Some of these checks may even contribute towards several distinct security properties, and therefore, it is even unclear how many properties the UC-based security model implicitly embeds. We thus consider the reformulation of these properties as separate experiments to be a valuable contribution on its own.

Our second contribution is to show there can be no doubt that our model provides the same security guarantees as the UC definition of [CDL16]. This is done as follows. We begin by considering a scheme, secure in our property-

based model. We then create, via a syntactical re-structuring, a 'wrapper protocol' which allows for a direct proof of indistinguishability between this wrapper protocol and the ideal functionality in the UC framework. Importantly, this wrapping process is merely cosmetic and does not affect the security of the protocol. Indeed, Theorem 3.4.1 provides a 'sanity check' that our model is sound by providing the necessary conditions for the property-based DAA algorithms to imply UC security (when wrapped). This puts a positive end to a long journey of flawed property-based models proposed since 2004.

Finally, we present a novel generic construction who's security can be proven in our model from the security of its building blocks. Notably, our construction allows one to avoid the use of homomorphic commitment schemes and/or non-standard signatures allowing for a two-party signing protocol which makes proofs non-trivial and difficult to verify [EE17, ECE+18]. The accompanying security proof elegantly demonstrates the succinctness of proofs that are possible in the property-based framework.

EFFICIENT LATTICE-BASED DAA. The final piece in our DAA work is the design of an *efficient* lattice-based DAA scheme. With this contribution we hope to meet the requirements of the Horizon 2020 PROMETHEUS project and work towards the deployment of this scheme in the FutureTPM.

The main ideas in our new construction are as follows. We begin by considering the newly introduced 'ISIS_f' assumption [BLNS23]. In the same work, the authors show how its structure lends itself to designing an anonymous credentials scheme. The authors are able to prove knowledge of an issued credential far more efficiently, owing to the linear nature of the verification equation. In particular, this can be seen as a modification of the GPV-style 'hash-and-sign' paradigm [GPV08] but without the need for hash functions whose input-output pairs are notoriously difficult to link with zero-knowledge arguments. We first tranform this framework to use module lattices. Relying M-LWE and MSIS will eventually allow us to set more flexible and favorable parameters. We make a small modification to the NTRU variant of the ISIS_f assumption which allows us to employ the Falcon trapdoor sampling techniques of [CPS+20] in the module setting.

We adopt the new LNP lattice-based zero-knowledge proofs of [LNP22] for the proofs in the join and signing protocols. These represent the most efficient proof system for proving simple lattice relations. The greatest challenge here is in splitting the LNP proof structure so that TPM and HOST *jointly* create the proof together while the TPM's key material remains hidden and the host cannot create valid signatures without the TPM's involvement. We

demonstrate that the LNP proof can indeed be split between two parties in this way so that the secrets of the TPM are not leaked to a potentially corrupt host.
Finally, we instantiate this scheme with concrete parameters that ensure its security under the module learning-with-errors, short-integer-solution, NTRU, and ISIS_f assumptions. Our scheme yields TPM keys of 0.8KB and signatures which are 37.7KB. This represents a 1.5. order of magnitude reduction (53 times) in signatures size and four-fold reduction in TPM key size over the state-of-the-art LDAA scheme [CKLL19].

1.1.2 *Distributed Electronic Voting from NTRU*

E-VOTING: AN OVERVIEW. Regardless of their validity, cries of ballot stuffing, discarding, and filtering continue to dog the democratic process across the globe. Furthermore, paper ballots have many inherent challenges, such as postal delays, family voting and coercion, and the (correct) discarding of incomplete ballots.

In recent years, electronic voting (e-voting) has enjoyed much attention with its adoption in countries such as Estonia and Switzerland, in addition to parts of the population in France and Australia, where it is valued for its potential efficiency and security advantages over traditional paper ballots. In many instances, internet voting or voting through a computer has resulted in a higher voter satisfaction and turnout rate. In addition to the practical advantages of e-voting, these systems allow for end-to-end verification of the voting process. That is, voters can verify that their vote has been submitted, counted, and contributed to the final result. Furthermore, electronic ballots offer a demonstrably faster and cheaper way to implement voting. However, this move to internet voting has come with its own issues.

Estonia was the earliest adopter of e-voting at a national scale, with its introduction in 2005. However, its first attempts were hit by a series of security flaw revelations, and the design has thus evolved through several iterations. In 2019, Switzerland (via its national postal service, Swiss Post) deployed a scheme to ensure voter privacy whilst guaranteeing that only legitimate votes were counted. Only months later, this scheme was shown to have several flaws, allowing undetectable fraud. Despite these obstacles, both countries have reimplemented their e-voting with updated and (hopefully) more secure schemes. With many countries looking to make the transition to e-voting, the long-term security of these systems must be considered with care. Arguably, the most pressing concern for the longevity of cryptographic

protocols is that of postquantum security.

As mentioned, a recent announcement by NIST calls for the proposal of *threshold* crytographic schemes. Such primitives distribute the cryptographic operation in question between n parties. A t-out-of-n threshold is defined whereby, as long as at least t parties cooperate honestly, the protocol will succeed. Such designs provide two crucial security enhancements. Firstly, no single party is ever in control of all of the secret key material, and as such does not have access or influence to the complete set of whatever potentially sensitive data might be processed by the protocol. For example, in a multi-party-computation where users have private inputs, no one learns the private inputs of the other users, yet the computation on all inputs is still possible. Secondly, if $t < n$, at least $t + 1$ users must coordinate maliciously to undermine the security of the process. Owing to the fruits of these processes, we believe that now is the perfect time to weave both these principles of post-quantum security and distributed trust into the design of electronic voting schemes. More generally, there has been much less focus on developing so-called privacy-preserving technologies (PETs) with post-quantum security, within which quantum-resistant e-voting lies. In particular, e-voting concerns the privacy of many millions of voters and the security of the entire democratic process, making its study paramount.

A recent line of works proposes electronic voting schemes achieving long-term security against quantum adversaries from lattice primitives. The earliest works are theoretical constructions based on generic zero-knowledge proofs and fully homomorphic encryption, but we have seen considerable advances, making the schemes suitable for use in real-world elections. The most notable are the schemes by del Pino et al. [dLNS17], Aranha et al. [ABG+21], Farzaliyev et al. [FWK21a], and Aranha et al. [ABGS22], the latter being the most efficient scheme based on (Ring) SIS and (Ring) LWE, designed to fit the commonly used distributed shuffle-and-decrypt-framework implemented in Swiss Post in Switzerland. Unfortunately, there are a number of drawbacks in these works. The paper of del Pino [dLNS17] gives a practical scheme based on homomorphic counting but unfortunately it does not scale well for systems with more complex ballots. A shuffle by [CMM19] was implemented in [FWK21a], however it is less efficient than [ABGS23]. Moreover, [CMM19, FWK21b, HMS21] do not consider the decryption of ballots, which would heavily impact the parameters of the protocols in practice. Finally, [BHM20] gives a fast decryption mix-net, but it cannot achieve universal verifiability and is thus unsuitable for real-world elections. At this point, we note that all voting schemes based on lattices, to date, are based on the

hardness of the ring LWE and SIS assumptions. None, however have considered the potential benefits of using the longest-standing lattice-assumption: NTRU.

The security of RSIS and RLWE is relatively well understood today. The RSIS problem is believed to be hard when the logarithm of the ℓ_2 norm of the secret vector is less than $2\sqrt{d \log_2 q \log_2 \delta}$ for lattice dimension d, modulus q, and root Hermite factor δ [MR09]. Current literature adopts the notion that $\delta = 1.0045$ or smaller gives rise to 128 bits of security. Furthermore, Albrecht et al. have collated a long line of algorithmic cryptanalysis into a publicly available estimator [APS15] used to estimate the hardness of a given RLWE instance. However, the hardness of the NTRU problem is less understood, and the most up-to-date security estimates are either asymptotic or only computed for very particular instances, for example, when the secret vectors are ternary.

At this point we want to motive the interest we take in the NTRU problem for our work. A core building block of privacy-preserving cryptographic primitives is the zero-knowledge proof. These objects allow one to prove knowledge of a secret satisfying some public statement without revealing any more than they know the secret. In the context cryptography, one often needs to prove the so-called 'well-formedness' of an object. This might mean proving knowledge of a plaintext underlying a ciphertext, or proving that some object was computed using the secret key underlying a public key. In all efficient ZK constructions for such lattice statements, the size of the proofs depends on the size of the secret one proves knowledge of. The advantage that NTRU has is that its secret key only comprises a single ring element rather than the two needed for schemes based on LWE (and its variants). Since ZKPs typically dominate the communication cost of any protocol employing them, it makes sense that one hopes to prove knowledge an NTRU key rather than an LWE key pair. This intuition will-turn out to be vindicated in our work but as mentioned, the hardness of the NTRU problem is somewhat less understood than that of LWE.

We briefly recall the NTRU problem [HPS98b]. Let R_q be a polynomial ring of dimension d and modulus q and sample polynomials f and g with coefficients from some discrete gaussian D^d_σ. Informally, the NTRU problem is to recover f and g given h, where $h = g/f \in R_q$.

Intuitively, a key recovery attack (find short f, g given h) is, by definition, to find vectors with small norm in the corresponding *NTRU lattice*, similar to breaking the RSIS problem above. In general, the hardness of lattice problems grows exponentially with the dimension d. However, a more in-

depth analysis of this problem has revealed an algebraic attack against so-called *overstretched* NTRU [ABD16, CJL16]. This attack does not apply to RSIS/RLWE owing to the absence of a *dense sub-lattice* as exists for NTRU lattices, revealing that the NTRU problem gets substantially easier when σ is small and q is much larger than d. Herein, we refer to the point at which this complexity improvement begins as the *fatigue point*. Kirchner and Fouque show [KF17] that the overstretched attack is noticeably more efficient for ternary-coefficient f and g and q (asymptotically) larger than $d^{2.783+o(1)}$. Recent analysis by Ducas and van Woerden [Dv21] decreases the exponent to 2.484, in the asymptotic case, and determines the constant in front of d as 0.004 in the ternary setting.

On the other hand, we have provable hardness bounds for NTRU when σ is large. Stehlé and Steinfeld [SS11] proved that when σ is roughly of size $q^{1/2}$, h is statistically close to uniform, and the NTRU problem enjoys worst-case hardness. This mirrors RLWE, for which the problem gets harder when σ increases towards the square root of the moduli and eventually results in a reduction to worst-case problems over general lattices [Reg05]. While the hardness estimates for RSIS and RLWE secrets of various norms are easily available, an understanding of NTRU's hardness for secret sizes between these two extremes ends here. Denoting by σ_1 the standard deviation for ternary values and $\sigma_{\mathsf{stat}} \approx q^{1/2}$. The natural question to ask is:

> *What is the concrete hardness of the NTRU problem when secrets are sampled with standard deviation σ_{NTRU} for $\sigma_1 < \sigma_{\mathsf{NTRU}} < \sigma_{\mathsf{stat}}$?*

Understanding such behaviour for RSIS and RLWE has already received much attention. NIST-standardised Crystals Dilithium [LDK+20] uses non-ternary secrets of absolute norm 2 or 4 (NIST levels 2 and 4, and 5 resp.) allowing for a fine-tuned security level given a fixed ring-dimension and modulus. Here, ternary-coefficient secrets would lead to larger parameters overall to compensate for the security loss. This provokes the natural question:

> *Can a carefully chosen non-ternary NTRU secret bound lead to more efficient instantiations of lattice-based protocols in practice?*

E-VOTING AND NTRU: OUR CONTRIBUTIONS. We answer both of the above questions comprehensively; we determine a concrete relation for the fatigue point of general NTRU (parametrised not only by d and q, but also by σ) and show how a careful choice of secret size yields optimal parameter selection in practice. We then apply these insights by designing an NTRU-

based e-voting scheme whose efficiency significantly improves the state-of-the art.

We build upon the work of Ducas and van Woerden [Dv21], on NTRU hardness, to analyse the overstretched attack against NTRU when the norm of the secrets grows with respect to the dimension and modulus. We stress that [Dv21] does give an asymptotic fatigue point for general NTRU but only a concrete relation for ternary secrets. Eploying the scripts provided in [Dv21], our analysis shows that when we increase σ, q can be increased with the *square* of this increase before reaching the fatigue point. Concretely, given σ and ring dimension d our experiments suggest a fatigue point q given by the following expression

$$q = 0.0058 \cdot \sigma^2 \cdot d^{\,2.484}.$$

Note, by following a similar asymptotic analysis to that in [Dv21], we confirm that the influence of σ on the fatigue point must indeed manifest only in the leading constant and not in the exponent of d.

To demonstrate the importance of this quadratic relationship, we [Dv21] parameters for the recent blind signature by del Pino and Katsumata [dK22], improving its efficiency as compared to the original scheme which uses ternary secrets.

Having conducted a thorough investigation of the hardness of NTRU, we present a new electronic voting scheme from the RLWE and NTRU assumptions. We follow the framework of Aranha et al. [ABGS23], employing a series of shuffle and decryption nodes to build a distributed and verifiable protocol.

Our first step is to use the NTRU cryptosystem [SS11] throughout. In privacy-preserving protocols, where zero-knowledge proofs (ZKPs) are deployed to verify the honest actions of parties, one wants to minimise the number of relations to be proven in zero-knowledge since ZKPs usually dominate the communication cost. Crucially, NTRU ciphertexts contain only a single ring element and thus give rise to simpler ZKP relations when compared to their two-component RLWE-based counterparts such as the BGV encryption scheme [BGV12] as used in [ABGS23].

Secondly, we observe that by modifying the proof system employed in the verifiability of distributed decryption, we can achieve a more favourable correctness condition. More precisely, we employ an *exact* proof system, removing the 'slack' inherent in the proof used in [ABGS23]. This is used to prove a tighter bound on the 'noise drowning' term necessary for masking decryption key shares. We note that while this makes the proof of distributed

decryption larger, it allows for a less restrictive correctness condition, leading to better global parameters throughout the scheme, more than making up for any communication cost that comes with this modification.

Finally, we call on our analysis of the NTRU problem to choose fine-tuned concrete parameters. Crucially, when choosing the NTRU secret keys, we can drop the ring dimension down to 2048 from 4096 and modulus down to 59 bits from 78 bits as used in [ABGS23] whilst maintaining a 128-bit security level. This would not have been possible without the quadratic nature of the fatigue relation.

Scheme	Ciphertexts	Shuffle	Dist. Dec.	Total
[ABGS23] [KB]	80	370	157	2188
Ours [KB]	15	130	85	875
[ABGS23] [ms]	0.74	261	138	1182
Ours [ms]	0.20	62	328	576

TABLE 1.1: Per vote comparison to [ABGS23] of ciphertexts, shuffle proofs, decryption proofs, and overall with 4 servers. Shuffles are sequential, while decryption is run in parallel.

Overall, we reduce the voting protocol's communication by 2.5× and computation by 2× over [ABGS23], see Section 5.5 for more details. It is interesting to note that, when compared to their classically-secure counterparts, post-quantum secure replacements typically come with a 30× communication cost (e.g. ECDH vs Kyber). Comparing our voting scheme to ElGamal-based schemes often used in practise, we incur a cost of at most 20× in ciphertext size, suggesting that our design may be approaching what can be optimally achieved.

1.2 THESIS ORGANISATION

We begin in Chapter 2 by providing the preliminary background material for tools which are used throughout this thesis. In Chapter 3 we present our new game-based DAA security framework, proving its equivalence to the existing UC definition, and providing a generic construction from standard building blocks to demonstrate the application of this new framework. In Chapter 4 we present our efficient lattice-based DAA scheme based on the ISIS_f assumption and the LNP zero-knowledge proof techniques. In Chapter 5 we present our analysis of the NTRU lattice problem which then paves the

way for our new e-voting scheme. This thesis is concluded by Chapter 6 with some discussions and potential future research directions.

2

PRELIMINARIES

In this chapter, we cover the relevant mathematical and cryptographic background that will be frequently used in this thesis. We begin by introducing notation. Then, we recall some basic mathematical definitions and tools, from linear algebra, probability distributions, and lattices. Finally, we give formal definitions of cyptographic primitives we use and state our security assumptions.

2.1 NOTATION

Let $\mathbb{Z}_n$ be the set of integers modulo n and $[n]$ be the set $\{1, 2, \dots n\}$. We denote by λ the security parameter. All algorithms considered are defined as interactive Turing machines which, unless explicitly defined, receive the security parameter in unary. We use PPT as shorthand for 'probabilistic polynomial time' and when describing the running time of an algorithm $\mathcal{A}$, this should be taken to mean PPT in the security parameter λ. The notation $\mathcal{A}^{(\cdot)}$ means that $\mathcal{A}$ has black-box access to some other algorithm. We denote the joint execution of algorithms $\mathcal{A}$ and $\mathcal{B}$ on private inputs α and β respectively by $(a, b) \leftarrow \langle \mathcal{A}(\alpha), \mathcal{B}(\beta) \rangle$ where $\mathcal{A}$ and $\mathcal{B}$ receive private outputs a and b respectively.

For a set S we use both "$\leftarrow S$" and "$\xleftarrow{\$} S$" to denote the process of sampling uniformly at random from S. For a distribution D, "$\leftarrow D$" denotes sampling according to D and for algorithm $\mathcal{A}$,"$\leftarrow \mathcal{A}$" denotes the (randomized) output of $\mathcal{A}$. The symbol ":=" denotes assignment and "=" is used for logical equality. A function $\nu : \mathbb{N} \to \mathbb{R}^+$ is negligible is for any $c \in \mathbb{N}$, $\lim_{\lambda \to \infty} \nu(\lambda)\lambda^c = 0$. We say that an event, depending on λ, happens with negligible probability if the probability that the event occurs is negligible in λ. Similarly, we say that an event happens with overwhelming probability if the probability that the event does not occur is negligible. We will write negl to denote an arbitrary negligible function and $\mathsf{poly}(\lambda)$ for an arbitrary polynomial in λ. We denote by log and ln logarithms with base 2 and e respectively.

2.2 MATHEMATICAL BACKGROUND

In this section, we lay out the key mathematical structures and notions upon which much of the work in this thesis relies.

2.2.1 *Lattices*

Here, we introduce the background material related to lattices and their integration with cryptography.

The mathematical meaning behind the word 'lattice' is a very tangible one to the idea we commonly associate with it. Its most simple description is a *discrete subgroup of* $\mathbb{R}^n$. For all purposes relating to cryptography and indeed most others, we consider only integral lattices. The definition we use will be the following.

Definition 2.2.1 (Lattice)**.** *Consider the set of vectors* $B := \{\vec{b}_1, ..., \vec{b}_d\} \in \mathbb{R}^n$*. Then the d-dimensional lattice* Λ *generated by* B *is defined as*

$$\Lambda = \mathcal{L}(B) = \left\{ \sum_{i=1}^{d} x_i \vec{b}_i \; ; \; x_i \in \mathbb{Z} \right\}.$$

If the $\vec{b}_i$ *are linearly independent then they are called a basis for* L*.*

Next we make an important distinction between the indices n and d used above.

Definition 2.2.2 (Rank)**.** *Let* $L \subseteq \mathbb{Z}^n$ *be a lattice. The rank* d *of* L *is exactly the dimension of the subspace* $Span(L) \subseteq \mathbb{R}^n$*. If* $n = d$*, we say that* L *has full rank.*

We make a few assumptions about this quantity going forward. We assume unless otherwise stated that our lattices have full rank. Furthermore, we deal only with bases since any generating set which is linearly independent may be transformed into a basis. It therefore follows that (in most cases) we will have $d = n$ is the number of basis vectors.

LATTICE BASES. Bases are worth taking a moment to consider carefully. Clearly a basis B defines a lattice Λ but it is not clear a priori that the converse is true, that is whether a lattice Λ defines its basis or not. We ignore the one-dimensional case since there is always a choice of two trivial

bases. Consider, for dimension $d \geq 2$, two $d \times d$ basis matrices B and B' whose column vectors are $\vec{b}_1, ..., \vec{b}_d$ and $\vec{b}'_1, ..., \vec{b}'_d$ respectively. These two bases span Λ exactly when all columns of B' can be expressed as integral linear combinations of the columns of B and vice-versa. This is equivalent to

$$BC = B' \text{ and } B = B'C',$$

where the entries c_{ij} and c'_{ij} of C and C' respectively are integers. This implies that C and C' are inverses of each other and are thus unimodular due to their integrality. We have that two bases span the same lattice if and only if they are related by a unimodular matrix as above. Since there are infinitely many such matrices, we have that there are infinitely many bases for each lattice.

Next we define an important invariant for a given lattice.

Definition 2.2.3 (Volume)**.** *The volume of a lattice Λ is defined by*

$$vol\,(L) = \sqrt{\det B^\top B} = |\det B|,$$

where B is a basis for Λ.

IMPORTANT CONSTANTS. In addition to the volume of a lattice, there are several other quantities associated with a given lattice which will prove crucial. Most importantly for our applications, we have the notion of successive minima.

Definition 2.2.4 (Successive minima)**.** *Let L be a lattice. The i-th successive minima is defined as*

$$\lambda_i(L) = \min\{R\,;\, \text{span}\,(L \cap \mathcal{B}(R)) \geq i\}.$$

Here $\mathcal{B}(R)$ is the ball of radius R centred around the origin. Note that, by definition, λ_1 is the length of the shortest vector in the lattice and $\lambda_1 \leq \lambda_2 \leq ...$. We now consider, given a dimension d, how λ_1 might scale as the volume of the lattice does. Consider the scaled lattice tL for some $t \in \mathbb{R}$. Now,

$$\text{vol}(tL) = |\det(t\vec{B})| = |t|^d |\det(\vec{B})|,$$

which suggests that λ_1 scales with the d-th root of the volume. This observation leads us very naturally to examine the following quantity.

Definition 2.2.5 (Hermite Constant)**.** *The Hermite constant in dimension d is*

$$\gamma_d = \sup_L \frac{\lambda_1(L)^2}{\text{vol}(L)^{\frac{2}{d}}}.$$

Here the supremum is taken over all full-rank lattices $L \in \mathbb{R}^n$. It is more common to consider $\sqrt{\gamma_d}$ and it is perhaps more intuitive to use the inequality

$$\lambda_1(L) \leq \sqrt{\gamma_d} \operatorname{vol}(L)^{\frac{1}{d}}. \tag{2.1}$$

This gives us our first upper bound on the length of a lattice's shortest vector. It has been show that there exists a real lattice L that achieves the bound in (2.1). It is also very difficult to calculate the value of γ_d and has only been computed for $1 \leq d \leq 8$ and $d = 24$. We can, however bound γ_d as follows.

Theorem 2.2.6 (Hermite's inequality)**.** *For dimensions $d \geq 2$, we have*

$$\gamma_d \leq \gamma_2^{d-1} = \left(\frac{4}{3}\right)^{\frac{d-1}{2}}. \tag{2.2}$$

2.2.2 *Gaussian Distribution on Lattices*

We now describe one of the most ubiquitous probability distributions used for sampling lattice points in cryptography. We begin by defining a Gaussian function on $\mathbb{R}$ centered at $\vec{v} \in \mathbb{R}^d$ with parameter σ as:

$$\rho_{\vec{v},\sigma}(\vec{x}) := \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^d \exp\left(-\frac{\|\vec{x}-\vec{v}\|_2^2}{2\sigma^2}\right).$$

We omit the subscript $\vec{v}$ when $\vec{v} = \vec{0}$.

Now we define the *discreet* Gaussian distribution over $\mathbb{Z}^d$ centered at $\vec{v} \in \mathbb{Z}^d$ with standard deviation σ as follows:

$$D_{\vec{v},\sigma}(\vec{x}) := \frac{\rho_{\vec{v},\sigma}(\vec{x})}{\rho_\sigma(\mathbb{Z}^d)}.$$

We recall the following tail bounds from [Lyu12, Lemma 4] which we often use to bound the length of elements sampled from a Gaussian distribution.

Lemma 2.2.7. *Let $d, k > 1$, $r > 0$ and $\vec{v} \in \mathbb{R}^d$. Then*

1. $\Pr_{\vec{z} \leftarrow D_\sigma^d}[\|\vec{z}\|_\infty > k\sigma] \leq 2e^{\frac{-k^2}{2}}$
2. $\Pr_{\vec{z} \leftarrow D_\sigma^d}[\|\vec{z}\|_2 > k\sigma\sqrt{d}] \leq k^d e^{\frac{m}{2}(1-k^2)}$
3. $\Pr_{\vec{z} \leftarrow D_\sigma^d}[|\langle \vec{z}, \vec{v}\rangle| > r] \leq 2e^{-\frac{r^2}{2\|\vec{v}\|_2^2\sigma^2}}$.

Definition 2.2.8 (q-ary Lattice)**.** *A lattice $\Lambda \in \mathbb{Z}^n$ is called q-ary (or modular) if there exists an integer $q < \mathrm{vol}(\Lambda)$ such that $q\mathbb{Z}^n \subseteq \Lambda$.*

Let us take a moment to unpick this definition. The statement $q\mathbb{Z}^n \subseteq \Lambda$ means that $qe_i \in \Lambda$ for all i. This implies that, given a vector $\vec{x} = (x_1, ..., x_n) \in \Lambda$, the vector $\vec{x} \mod q = (\lfloor x_{1_q} \rfloor, ..., \lfloor x_{n_q} \rfloor)$ is also in the lattice, since it is obtained by subtracting the correct multiple of qe_i for each component. We now define two modular lattices which will take centre stage in may lattice problems and cryptosystems that we will examine. Let $m, d \in \mathbb{N}$. Let $A \in \mathbb{Z}^{d \times m}$ be a matrix and let

$$\Lambda_q^{\perp}(A) = \{\vec{x} \in \mathbb{Z}^m \,:\, A\vec{x} = 0 \mod q\}, \tag{2.3}$$

$$\Lambda_q(A) = \{\vec{y} \in \mathbb{Z}^m \,:\, y = A^{\top}\vec{y}' \mod q \text{ for some } \vec{y}' \in \mathbb{Z}^d\}. \tag{2.4}$$

It is easy to see that these both define lattices.

2.2.3 *Power-of-Two Cyclotomic Rings*

Let d be a power-of-two and $K = \mathbb{Q}[X]/(X^d + 1)$ be the $2d$-th cylotomiic field. In this thesis, all constructions will use the ring of integers of K which we denote by $R := \mathbb{Z}[x]/(x^d + 1)$. We further consider $R_q = \mathbb{Z}_q[x]/\phi$, where q is a prime. Elements in both rings are polynomials of degree at most $d - 1$, with those in the latter ring having coefficients between $-(q-1)/2$ and $(q-1)/2$. We denote elements of $\mathbb{Z}$ and R by lower-case letters, vectors in R^k by bold lower-case letters, and matrices in $R^{(k \times \ell)}$ by bold upper-case letters. In this thesis, all vectors will be *column* vectors. When sampling elements of R, we use $a \leftarrow D_\sigma$ to mean that the coefficient vector of $a \in R_q$ is sampled from $D_{\mathbb{Z}^d,\sigma}$.

NORMS. For $\mathbf{a} \in R^k$, we define the L_α norm of $\mathbf{a}$ as $\|\mathbf{a}\|_\alpha := \|\vec{\mathbf{a}}\|_\alpha$. Finally, we define the set $S_\nu := \{s \in R \,:\, \|s\|_\infty \leq \nu\}$ for $\nu \in \mathbb{R}$.

COEFFICIENT EMBEDDING AND ROTATION MATRICES. We define $\Phi : R_q \to (\mathbb{Z}_q^d)^{\top}$ to be the map that sends a polynomial $a \in R_q$ to its (column) coefficient vector. We define $\mathsf{Rot} : R_q \to \mathbb{Z}_q^{d \times d}$ to be the map that sends a polynomial $a \in R_q$ to a matrix whose i'th column is $\Phi(a \cdot X^i \mod (X^d + 1))$. It is easily checked that, over the rings defined above, $\mathsf{Rot}(a)\Phi(b) = \Phi(a \cdot b)$ and that this notation extends naturally to vectors and matrices of polynomials. We will often use $\vec{a}$ in place of $\Phi(a)$ to ease presentation.

GALOIS AUTOMORPHISMS. Let $\mathsf{Aut}R := \{\sigma(_)i \,:\, i \in \mathbb{Z}_{2d}^{\times}\}$ be the automorphism group of R where each automorphism $\sigma(_)i : R \to R$ is defined by $\sigma(()X) = X^i$. For a vector $\mathbf{x} = (x_1, \ldots, x_k) \in R^k$ and any $\sigma(\in)\mathsf{Aut}(R)$, denote $\sigma(()\mathbf{x}) = (\sigma(()x_1), \ldots, \sigma(()x_k))$ (and similarly for matrices over R).

2.3 CRYPTOGRAPHIC DEFINITIONS

2.3.1 *Provable Security*

With the dawn of modern cryptography came the use of advanced mathematics to design protocols enabling ways of communicating that were previously not possible. Perhaps of greater importance, this influx of mathematics and its practitioners brought *provable security*. No more would the perceived robustness of a cryptographic scheme be underpinned merely by the inability of ones adversaries to break it. Provably security brought quantifiable guarantees on the difficulty of breaking a given protocol's security.

The rigour and completeness of a security proof are comparable to those of a traditional mathematical proof however proofs of security have a slightly different structure. At the heart of any security proof of a cryptographic protocol lies the following idea: *If an (efficient) adversary $\mathcal{A}$ is able to break the security of the protocol, one can use $\mathcal{A}$ as a subroutine to (efficiently) solve some computationally hard problem.* The meaning of terms like 'efficient' and 'hard' need to be well defined but the core idea of all security proofs is the same; since such 'hard' problems are computationally intractable, there cannot exists such an adversary that can break our scheme (in any reasonable time).

We now look at the two main generic models for proving security of a scheme.

GAME-BASED SECURITY The game-based model of security is the most well known and widely used of all security models. The idea is quite simple. On conception of a task that needs to be achieved in some secure fashion one defines all of the security properties that need to be satisfied in order to securely carry out that task. For example, one might require of a digital signature scheme that it is non-frameable meaning that no adversary can create a valid signature on some message that appears to have been created by some honest party who never signed that message. Such properties are defined using security 'games'. These are usually two-player interactions between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. The game specifies a goal condition

that the adversary is trying to achieve (like correctly guessing a particular piece of hidden information).

Let us consider the eavesdropping indistinguishably experiment for the encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ ad described in [**?**].

The eavesdropping indistinguishably experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\Pi,\mathcal{A}}(n)$:

1. The adversary $\mathcal{A}$ is given input 1^n and outputs a pair of messages m_0, m_1 of the same length.
2. A key k is generated by running $\mathsf{Gen}(1^n)$, and a random bit $b \leftarrow \{0,1\}$ is chosen. A ciphertext $c \leftarrow \mathsf{Enc}_k(m_b)$ is computed and given to $\mathcal{A}$. We call c the challenge ciphertext.
3. $\mathcal{A}$ outputs the bit b^*.
4. The output of the experiment is defined to be 1 if $b = b^*$ and 0 otherwise. If $\mathsf{PrivK}^{\mathsf{eav}}_{\Pi,\mathcal{A}}(n) = 1$ we say that $\mathcal{A}$ succeeded.

The motivation for formally defining such games is that we can explicitly control what powers the adversary has and what we define to be a break of security. Furthermore, writing the game down in such an algorithmic way enables us to more easily show how an adversary winning the defined game can be used to solve some computationally hard problem.

This model of defining security is ubiquitous in cryptography literature and is, largely, the go-to technique for proving security of schemes.

COMPOSABLE (SIMULATION) SECURITY In this section we review what is a relatively new paradigm for proving security of cryptographic protocols. Its modern formulation can be dated back to work of Ran Canetti in [Can01] however it emerged from simulation-based methods first seen in [GMS87]. Canetti proposes a framework that allows for describing arbitrary protocols in a unified and systematic way. Furthermore, a protocol's security is preserved even under composition with other protocols through an operation called *universal composition*. The power of this approach is that it allows for analysis of highly complex cryptographic constructions by considering its simpler building blocks. More precisely, the security of such protocols is preserved even in composition with an unbounded number of adversarially controlled protocols. The motivation for such a framework lies in the uncertainty around the security of protocols used in unpredictable

and complex environment such as modern communication networks.

Traditionally, when defining security for a cryptographic protocol, one tends to consider only a single execution of the protocol in question. This has its obvious advantages, allowing for more simple and concise statements of the security requirements and its analysis. Although this has long been the conventional paradigm, there have been shown protocols in many corners of cryptography where parallel and concurrent composition of these protocols does not respect the closure of original security notions. For a number of such examples, we refer the reader to the introduction of [Can01]. The universal composition (UC) framework aims to foresee such problems with previous security models by ensuring that security is preserved under a general *composition operation* of protocols. Moreover, it guarantees security in arbitrary protocol environments, even those that have not been explicitly considered.

We begin by introducing the model of computation with which we describe the cryptographic protocols to be considered. In its current familiar form this framework was first presented by Canetti in [Can01]. We work with the most up to date formalisation of these ideas as described in the updated version of [Can01] from 2013 (an overview of these updates can be found in Appendix B of the new paper). Whilst more rigorous considerations can be given to the complexity and resources available to the interactive Turing machines which form the building blocks of our computational model, the definitions presented here will be more than sufficient for this introduction. Figure 2.1 represents the state of the art model of execution protocol in the UC framework and the one we will consult for security proofs. Again, this is taken from Canetti's updated version of [Can01] (2013).

Having described the basic model of computation, we will examine the notion of protocol emulation. This idea is what allows us to assert that a protocol achieves the effective behaviour for the idealised process that we wish our protocol to have. This is done by showing that no adversary interacting with the real-world protocol can gain any more information than a simulated one interacting with an idealized version of the protocol. Crucially, the job of distinguishing between adversaries in these two worlds is carried out by a third entity; the environment. The environment is tasked with observing the adversary acting in the real world and the simulated one in the ideal world and then deciding which world it has indeed been observing (without prior knowledge). It may use any strategy or resources available to it (which we lay

out formally), in order to 'play' with the world it has been challenged with. In order that a protocol 'realizes' the ideal functionality of the process it is designed to achieve, the environment mustn't be able to distinguish which world it is in. Moreover, this must be the case for any real-world adversary that one could define.

Protocol emulation (or realization if a protocol emulates an ideal functionality) is the property which allows for modular analysis of cryptographic schemes. In particular, if a protocol π UC-emulates some protocol ϕ which is used as a subroutine of some protocol ρ, then the main UC Composition Theorem 2.3.5 allows us to replace ϕ with π in the protocol ρ (the new overall protocol $\rho^{\phi \to \pi}$ UC emulates the original one ρ). Showing protocol emulation is a very powerful technique but it requires that the ideal functionality, that we hope our real-life protocol achieves, must be carefully specified to give meaningful security results.

THE MODEL OF COMPUTATION. Here we formalise the core model for representing the distributed systems and the interacting computer programs within them. The model is designed to provide both simplicity and generality when used to express the complexity of modern distributed systems.

The fundamental building block of a distributed system is an *algorithm* or *computer program* designed to perform standard one-shot sequential executions. A distributed algorithm (or protocol) may consist of several such programs. Here, we say each program (within the protocol) runs on its own computing device, with its own local input. These programs communicate by exchanging information with one another. To best describe a distributed system of such computing devices, we turn to the notion of an *interactive Turing machine* (ITM). Herein we develop a language that allows us to express instructions and construct protocols in a distributed system. We now define our augmented notion of an interactive Turing machine. So that the reader need not keep referring back to formal definitions, we give a somewhat informal definition.

Definition 2.3.1 (Interactive Turing machine)**.** *This is a Turing machine, M, with the followings additional features.*
Special tapes

- *An identity tape (here 'tape' refers to the usual notion of a readable and/or writeable register) consists of two strings of information. The*

first contains the code of M describing its program, the second is called the identity of M. Together, these two strings for the extended identity *of M. This tape is read-only.*

- *An outgoing message tape holds the messages sent by M together with addressing information for the delivery of the message.*
- *Three externally writeable tapes for holding inputs from other computing devices. These are*
  1. *An input tape for inputs from the'calling programs' or external users outside the cryptographic protocol under consideration. In our protocol execution model, these external inputs will be modelled by an 'environment' ITM.*
  2. *An incoming communications tape, for information coming from other computing devices within the cryptographic protocol under consideration.*
  3. *A subroutine output tape, for outputs of programs that were run as subroutines of the current program.*
- *A one-bit activation tape, representing whether the ITM is currently 'activated'.*

Instructions

- *The* read next message *instruction specifies a tape from input, incoming communication, subroutine output. The read head then jumps to the chosen tape and begins reading that message. (One way to implement this would be to have each message end with a special end-of-message) character.*

There are a few semantics to lay out to formalize our model of the distributed systems. Firstly, we make a distinction between an ITM, which we have considered thus far, and and ITM *instance* (ITI). The latter term represents an instance of a program running on some specific data input whereas an ITM is considered a 'static object'. Secondly, we allow the number of ITIs in our protocol to grow dynamically in an unbounded manner. Finally, the model describing a protocol execution is augmented with a mechanism which regulates the transfer of information between ITI's. We call this the *control function*. One can think of this as containing a set of instructions for which external write information is allowed. This also allows flexibility in designing the execution model for a protocol simply by changing the control

function. Using this new concept of the control function, we define a *system of ITIs* as a pair $S = (I, C)$ where I is an ITM (called the initial ITM), and $C : \{0,1\}^* \to \{allow, disallow\}$ is a control function.

A more rigorous description for the role of the control function and also of the mechanics of machine activating and external writing can be found in Section 3 of [Can01]. For now, we move to a description of protocol execution.

MODEL OF PROTOCOL EXECUTION. When attempting to model any cryptosystem (for the purposes of security), we must consider the different parties involved and the roles that they perform. In this vein, we use ITM's as described above to model the actions of the different parties to be considered. The model of execution is somewhat intuitive by modelling parties in the protocol by ITIs (including an adversary). We make one crucial addition to this model in the form of the *environment machine*. One can think of this a representing all that is external to the current protocol execution. This is crucial to the composition operation since it accounts for possible input from other protocol instances, their adversaries, human users etc.

A model of execution is parametrized by three ITMs: the protocol π to be executed, and environment $\mathcal{E}$ and an adversary $\mathcal{A}$. In the language of the previous subsection, the initial ITM is the environment $\mathcal{E}$. In particular it provides the local inputs of all parties (here we use the word 'party' interchangeably with ITI). To spawn our protocol instance, $\mathcal{E}$ first invokes the adversary $\mathcal{A}$ via instruction from the control function. As the computation proceeds, $\mathcal{E}$ can invoke and pass input to an unbounded number of ITIs, with the only restriction being that they all have the same session identifier SID. More formally, we say that all ITIs in a given execution of a system is an *instance of protocol* π if they all have same code π and the same session id SID. Essentially this ensures that all machines are part of the same (closed) cryptographic protocol under consideration. This allows us to specify which protocol instance an ITI participates in. Furthermore, we instantiate the identity of an ITI with a second string (in addition to the SID) called the *party identifier* in order to differentiate between ITIs within the same protocol instance. The PID indicates the role of the ITI i.e which part of the code π it must perform.

When given an input, a party is activated. It then performs according to its code, given the input supplied to it. It may then write to its output tape and finally it enters a special waiting state which marks the end of its

activation. A protocol execution consists of a series of such activations. The adversary $\mathcal{A}$ may write to any tape of any ITI. If $\mathcal{A}$ writes to the incoming communications tape of some ITI then we say that $\mathcal{A}$ delivers this message. The reason for this terminology is because we ultimately wish to model an untrusted and asynchronous network in which the adversary can observe the communications between parties and has the power to delay the delivery of messages. For this reason one can loosely think of $\mathcal{A}$ delivering all messages between parties within the protocol.

Execution of protocol π with environment $\mathcal{E}$ and adversary $\mathcal{A}$.

Such an execution is a run of a system of ITMs as specified above, with initial ITM $\mathcal{E}$, and a control function that enforces the following restrictions:

1. $\mathcal{E}$ may only *pass inputs* to other parties. The first ITI that $\mathcal{E}$ passes input to is set to be the adversary. The identity of this ITI is set to have special value $id = 1$, and its code is set to be $\mathcal{A}$.
   All other ITIs that $\mathcal{E}$ passes input to are required to be an instance of protocol π. In particular, if the identity contained in an input that $\mathcal{E}$ passes does not correspond to the identity of any ITI in the system then one is created.

2. The adversary $\mathcal{A}$ can write to any tape of any ITI. There are no restrictions on the contents and the sender identities of delivered messages. In particular it may report any information back to $\mathcal{E}$.

3. Any ITI other than $\mathcal{E}$ and $\mathcal{A}$ may send messages to $\mathcal{A}$, and pass inputs and outputs to any ITI other than $\mathcal{E}$ and $\mathcal{A}$. There is one circumstance under which an ITI may pass a message to $\mathcal{E}$: If the extended identity of the target ITI of an external write request coincides with the claimed source identity value id_s in a previous external write request from $\mathcal{E}$, and the target tape is the subroutine output tape, then the control function writes the requested message on the subroutine output tape of $\mathcal{E}$.

FIGURE 2.1: A summary of the model for protocol execution

THE REAL-LIFE PROCESS. Figure 2.1 describes how one attempts to model a protocol execution in the real world. Since our ultimate goal is to consider the security of such protocols when run concurrently with other protocol and positioned within a more complex network of processes, the environment $\mathcal{E}$ is essential for modelling any input to our 'challenge' protocol from this external activity. In essence, the environment is 'prodding' our protocol with any imaginable input and working in league with the adversary by providing this input to $\mathcal{A}$.
As mentioned previously, a protocol execution consists of a series of ITM activations. In order to scrutinise the security of a cryptographic protocol, it makes sense to consider many adversaries covering a range of potential adversarial powers. In particular, we may consider an adversary which corrupts one of the parties. On corruption of a part, $\mathcal{A}$ gains access to all the tapes of that party and can control all future actions of that party. In addition, the environment is notified by the adversary whenever such a corruption occurs. Once the activation of a party halts, the environment is activated. The protocol execution ends when the environment halts. We define the output of the protocol execution to be the environment and without loss of generality assume this output to be a single bit. For consistency with the literature we herein denote the environment by $\mathcal{Z}$.
In order to consider all possible inputs from the environment to our protocol and all possible adversarial behaviours, we consider the ensemble

$$\{\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}(k,z,\vec{r})\}_{k\in\mathbb{N},z,\vec{r}\in\{0,1\}^*}.$$

This represents the set of all possible executions of π with adversary $\mathcal{A}$, environment $\mathcal{Z}$, on security parameter k, input z for $\mathcal{Z}$ and all possible random inputs for $\mathcal{Z}$, $\mathcal{A}$ and the parties of π.

THE IDEAL PROCESS. I. order to assess the security protocol execution in the real-life process, we construct the *ideal-process* execution for comparison. Central to the ideal process is the *ideal functionality*. This is modelled as another ITM and aims to capture the desired functionality of whatever protocol we are examining. Just as in the real-world process we have an environment $\mathcal{Z}$. We now have this new entity; the ideal functionality $\mathcal{F}$ and in addition we have an ideal-process adversary $\mathcal{S}$ (often called the simulator) along with a set of dummy parties mimicking those in the real-world process. These dummy parties essentially act as place-holders for the 'local-subroutines' of the calling parties. Accordingly, they have the same SID and PID as their real-world counterparts and relay all inputs and outputs between the calling ITI and the ideal functionality.

The order of activations is as in the real-life process. Now, however, when a (dummy) party is activated, it simply forwards its input (with which it was activated) to the ideal functionality $\mathcal{F}$. Whenever a dummy party is activated due to delivery of some output from $\mathcal{F}$ it copies this output to its own output tape. In this way, we may think of $\mathcal{F}$ as performing an idealized version of the code that the corresponding real-life party performs. Crucially, the parties do not send messages directly to each other, instead everything must be passed to $\mathcal{F}$ who may then pass on some information to another party. In particular, $\mathcal{S}$ cannot see the inputs and outputs of other parties but $\mathcal{F}$ may send messages to $\mathcal{S}$.
Corruption of parties is again possible by $\mathcal{S}$, in which case $\mathcal{S}$ controls the party's actions. Note that both the environment $\mathcal{Z}$ and $\mathcal{F}$ are notified of corruptions.
As in the real-life process, the execution halts when $\mathcal{Z}$ halts and outputs a single bit.
We may define the analogous ensemble of protocol executions

$$\{\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(k, z, \vec{r})\}_{k\in\mathbb{N},z,\vec{r}\in\{0,1\}^*}.$$

SECURELY REALIZING AN IDEAL FUNCTIONALITY. We say that a protocol ρ securely realizes an ideal functionality $\mathcal{F}$ if for any real-life adversary $\mathcal{A}$ there exists an ideal-process adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell with non-negligible probability whether it is interacting with $\mathcal{A}$ and parties (ITIs) running ρ in the real-life process, or it is interacting with $\mathcal{S}$ and $\mathcal{F}$ in the ideal process.
This means that from the point of view of $\mathcal{Z}$, running protocol ρ is 'just as good' as interacting with an ideal process for $\mathcal{F}$.

Definition 2.3.2 (Secure realization)**.** *Let $\mathcal{F}$ be an ideal functionality and π be a n-party protocol. We say that π securely realises $\mathcal{F}$ if for any adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that for any environment $\mathcal{Z}$ we have*

$$\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx \mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}.$$

Here, '$\approx$' denotes the standard indistinguishably of two binary distribution ensembles. We have also used $\mathrm{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ and $\mathrm{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$ as shorthand for the previously defined ensembles.

A protocol realizing an ideal functionality can be viewed as a particular case of a wider notion called *emulation*.

Definition 2.3.3 (Protocol Emulation)**.** *Let π and ϕ be PPT protocols. We cay that π UC-emulates ϕ is for any PPT adversary $\mathcal{A}$, there exists a PPT adversary $\mathcal{S}$ such that for any PPT environment $\mathcal{Z}$ we have:*

$$\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}.$$

To demonstrate the use of the term 'realize' when talking about ideal functionalities we note that in the case of a protocol π UC-realizing an ideal functionality $\mathcal{F}$, we usually say that π UC-realizes $\mathcal{F}$.

Let us pause for a moment to consider the details of Definition 2.3.2. In order to prove that a protocol securely realizes some ideal functionality, we must show that *for all adversaries $\mathcal{A}$, there exists a simulator $\mathcal{S}$, such that no environment $\mathcal{Z}$ can distinguish as to whether they are in the real-world process or the simulated one.* Thus, one must consider each possible adversary, constructing a simulator for each one which can fool all possible environments. To make our life easier when proving protocol emulation, it is useful to consider 'specialized simulators' as defined in [**?**]. These are simulators that are allowed to depend on the choice of environment and therefore are not required to fool *every* environment. In other words, given an adversary-environment pair $(\mathcal{A}, \mathcal{Z})$, a specialized simulator would satisfy $\text{EXEC}_{\phi,\mathcal{S},\mathcal{Z}} \approx \text{EXEC}_{\pi,\mathcal{A},\mathcal{Z}}$. Furthermore, Canetti proves [**?**] that security with respect to specialized simulators is in fact equivalent to the main definition 2.3.3.

Proposition 2.3.4. *A protocol π UC-emulates protocol ϕ according to Definition 2.3.3 if and only if it UC-emulates ϕ with respect to specialized simulators.*

This makes our life much easier when proving protocol security in the UC framework since we may let the simulator depend on the environment's behaviour.

THE COMPOSITION THEOREM. We now arrive at a result that will allow us to prove the grand notion of composable security that we desire. We consider a protocol ρ which makes subroutine calls to protocol ϕ. We want to show that under certain conditions, we may replace the subroutine protocol ϕ with another subroutine protocol π whilst ensuring that the newly created protocol denotes $\rho^{\phi\to\pi}$ UC-emulates the original protocol ρ. It turns out that this can be achieved when π UC-emulates ϕ.

Before we state the Composition Theorem, we must make one important definition. In order that the result holds we must make one more (reasonable) restriction on the protocols ϕ and π. Namely, they must be *subroutine respecting*. An instance of a protocol π is said the be subroutine respecting if

1. No ITI which is a subsidiary of this instance passes outputs or receives inputs from any ITI which is not a part or subsidiary of this instance.
2. On first activation, each ITI that is or will become a subsidiary of this instance sends a message to the adversary notifying it of its own extended identity (code and id), as well as the code π and the SID of this instance.

Theorem 2.3.5 (Universal composition: General Statement)**.** *Let ρ, π, ϕ be PPT protocols such that π UC-emulates ϕ and both ϕ and π are subroutine respecting. Then protocol $\rho^{\phi \to \pi}$ UC-emulates ρ.*

An obvious corollary, and one we will utilize often is the particular case when π UC-realizes some ideal functionality $\mathcal{F}$. Then $\rho^{\pi/\mathcal{F}}$ UC-emulates protocol ρ.

2.3.2 *Security Assumptions*

The security of our scheme relies on the well known computational lattices problems Module Learning with Errors (M-LWE), Module Short Integer Solution (MSIS), and Module-NTRU (M-NTRU), defined as follows[1]:

Definition 2.3.6 ($\mathsf{M\text{-}LWE}_{n,m,\chi,q}$ [LS15])**.** *Let n, m be positive integers and χ be a bounded distribution over R_q. The Module-LWE problem then asks an adversary $\mathcal{A}$ to distinguish between the following two cases:*

1. $(\mathbf{A}, \mathbf{As})$ *for public* $\mathbf{A} \leftarrow R_q^{n \times (n+m)}$ *and secret* $\mathbf{s} \leftarrow \chi^{n+m}$,
2. $(\mathbf{A}, \mathbf{b}) \leftarrow R_q^{n \times (n+m)} \times R_q^n$ *where both are sampled uniformly.*

Then $\mathcal{A}$ is said to have advantage ϵ in solving $\mathsf{M\text{-}LWE}_{n,m,\chi,q}$ *if*

$$\Big| \Pr\left[b = 1 \mid \mathbf{A} \leftarrow R_q^{n \times (n+m)};\ \mathbf{s} \leftarrow \chi^{n+m};\ b \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{As})\right] - \Pr\left[b = 1 \mid \mathbf{A} \leftarrow R_q^{n \times (n+m)};\ \mathbf{b} \leftarrow R_q^n;\ b \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{b})\right] \Big| \geq \epsilon.$$

1 We note that for M-LWE we sample a uniform matrix $\mathbf{A}$ over R_q of dimension $n \times (n+m)$ which is equivalent to sampling $\mathbf{A}'$ of dimension $n \times m$ and concatenation it with the the identity matrix $\mathbf{I}_n$ to form $\mathbf{A}$.

Definition 2.3.7 ($\mathsf{MSIS}_{n,m,B,q}$ [LS15])**.** *Let n, m be positive integers and $0 < B \ll q$. Then, given $\mathbf{A} \leftarrow R_q^{n\times m}$, the Module-SIS problem asks an adversary $\mathcal{A}$ to find $\mathbf{z} \in R_q^m$ such that $\mathbf{Az} = \mathbf{0}$ and $0 < \|\mathbf{z}\| \leq B$. $\mathcal{A}$ is said to have advantage ϵ in solving $\mathsf{MSIS}_{n,m,\eta,q}$ if*

$$\Pr\left[0 < \|\mathbf{z}\| \leq B \wedge \mathbf{Az} = \mathbf{0} \mid \mathbf{A} \leftarrow R_q^{n\times m};\, \mathbf{z} \leftarrow \mathcal{A}(\mathbf{A})\right] \geq \epsilon.$$

Definition 2.3.8 ($\mathsf{M\text{-}NTRU}_{n,m,\chi,q}$ [CPS+20])**.** *Let n, m be positive integers, and χ be a bounded distribution over R_q. The Module-NTRU problem then asks an adversary $\mathcal{A}$ to distinguish between the following two cases:*

1. $\mathbf{F}^{-1}\mathbf{G} \in R_q^{n\times m}$ *for secret* $(\mathbf{F}, \mathbf{G}) \leftarrow \chi^{n\times n} \times \chi^{n\times m}$,
2. $\mathbf{H} \in R_q^{n\times m}$ *for uniformly sampled* $\mathbf{H} \leftarrow R_q^{n\times m}$.

Then $\mathcal{A}$ is said to have advantage ϵ in solving $\mathsf{M\text{-}NTRU}_{n,m,\chi,q}$ if

$$\Big|\Pr\left[b = 1 \mid (\mathbf{F}, \mathbf{G}) \leftarrow \chi^{n\times n} \times \chi^{n\times m};\, b \leftarrow \mathcal{A}(\mathbf{F}^{-1}\mathbf{G})\right] - \Pr\left[b = 1 \mid \mathbf{H} \leftarrow R_q^{n\times m};\, b \leftarrow \mathcal{A}(\mathbf{H})\right]\Big| \geq \epsilon.$$

Where we do not require the module variant of the above assumptions but simply their ring variants, we refer to them as R-LWE*,* R-SIS*, and* NTRU *respectively. These correspond to the case when $n = m = 1$.*

2.3.3 *Rejection Sampling*

In lattice-based cryptography in general, and in our zero-knowledge protocols in particular, we would like to output vectors $\mathbf{z} = \mathbf{y} + \mathbf{v}$ such that $\mathbf{z}$ is independent of $\mathbf{v}$, and hence, $\mathbf{v}$ is masked by the vector $\mathbf{y}$. Here, $\mathbf{y}$ is sampled according to a Gaussian distribution $\mathcal{N}_\sigma^k$ with standard deviation σ and we want the output vector $\mathbf{z}$ to be from the same distribution. The procedure is shown in Figure **??**.

Here, $1/M$ is the probability of success, and M is computed as

$$\max \frac{\mathcal{N}_\sigma^k(\mathbf{z})}{\mathcal{N}_{\mathbf{v},\sigma}^k(\mathbf{z})} \leq \exp\left[\frac{24\sigma\|\mathbf{v}\|_2 + \|\mathbf{v}\|_2^2}{2\sigma^2}\right] = M \tag{2.5}$$

where we use the tail bound from **??**, saying that $|\langle \mathbf{z}, \mathbf{v}\rangle| < 12\sigma\|\mathbf{v}\|_2$ with probability at least $1 - 2^{-100}$. Hence, for $\sigma = 11\|\mathbf{v}\|_2$, we get $M \approx 3$. This is the standard way to choose parameters, see e.g. [BLS19]. However, if the procedure is only done once for the vector $\mathbf{v}$, we can decrease the parameters

slightly, to the cost of leaking only one bit of information about $\mathbf{v}$ from given $\mathbf{z}$.

In [LNS21], Lyubashevsky et al. suggest to require that $\langle \mathbf{z}, \mathbf{v} \rangle \geq 0$, and hence, we can set $M = \exp(\|v\|_2/2\sigma^2)$. Then, for $\sigma = 0.675\|\mathbf{v}\|_2$, we get $M \approx 3$. In Section 2.3.3, we use the pre-determined bit b to denote if we only use $\mathbf{v}$ once or not, with the effect of rejecting about half of the vectors before the sampling of uniform value μ in the case $b = 1$ but allowing a smaller standard deviation.

$\underline{\mathsf{Rej}(\mathbf{z}, \mathbf{v}, b, M, \sigma)}$

1. **if** $b = 1$ and $\langle \mathbf{z}, \mathbf{v} \rangle < 0$, **Return** 1
2. $\mu \leftarrow [0, 1)$
3. **if** $\mu > \frac{1}{M} \cdot \exp\left[\frac{-2\langle \mathbf{z}, \mathbf{v} \rangle + \|\mathbf{v}\|_2^2}{2\sigma^2}\right]$, **Return** 1
4. **Return** 1

FIGURE 2.2: Rejection Sampling.

2.3.4 *Public-Key Encryption*

We recall the standard IND-CPA security of a public-key encryption (PKE) scheme.

Definition 2.3.9. *Public-Key Encryption A public-key encryption* Π_{PKE} *over a message space* $\mathcal{M}$ *consists of four algorithms* $\Pi_{\mathsf{PKE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$*:*

- $\mathsf{Setup}(\lambda) \rightarrow \mathsf{pp}$: *On input the security parameter* λ*, it outputs a set of public parameters* pp*.*
- $\mathsf{KeyGen}(\mathsf{pp}) \rightarrow (\mathsf{pk}, \mathsf{sk})$: *On input a public parameter* pp*, it outputs a public and secret key pair* $(\mathsf{pk}, \mathsf{sk})$*.*
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{M}) \rightarrow \mathsf{ctx}$: *On input a public key* pk *and meessage* $\mathsf{M} \in \mathcal{M}$*, it outputs a ciphertext* ctx*.*
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ctx}) \rightarrow \mathsf{M}$ *or* $\perp$: *On input a secret key* sk *and a ciphertext* ctx*, it outputs either* $\mathsf{M} \in \mathcal{M}$ *or a special symbol* $\perp \notin \mathcal{M}$ *indicating failure.*

We will denote by $\mathcal{R}$ the set from which randomness used by the encryption algorithm Enc *is sampled.*

Definition 2.3.10 (Correctness)**.** *We say that a public-key encryption Π_{PKE} is ϵ-correct if for all $\lambda \in \mathbb{N}$, $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, and $\mathsf{M} \leftarrow \mathcal{M}$ we have that*

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, \mathsf{M})) = \mathsf{M}]] \geq 1 - \epsilon,$$

where the probability is taken over the randomness sampled in encryption.

We now define the standard definition of indistinguishability under chosen-plaintext attack.

Definition 2.3.11 (IND-CPA Security)**.** *A* PKE *scheme Π_{PKE} is* IND-CPA *secure if for all $\lambda \in \mathbb{N}$, any PPT adversaries $\mathcal{A}$ has at most a negligible advantage in the following game played against a challenger.*

1. *The challenger runs $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and samples a bit $b \in \{0, 1\}$. The challenger provides $(\mathsf{pp}, \mathsf{pk})$ to $\mathcal{A}$.*
2. *$\mathcal{A}$ then makes a query to the challenger containing a pair of messages $(\mathsf{M}_0, \mathsf{M}_1) \in \mathcal{M}^2$, and the challenger returns $\mathsf{ctx}_b \leftarrow \mathsf{Enc}(\mathsf{pk}, \mathsf{M}_b)$ to $\mathcal{A}$.*
3. *$\mathcal{A}$ outputs a bit $b^* \in \{0, 1\}$. We say $\mathcal{A}$ wins if $b^* = b$.*

The advantage of $\mathcal{A}$ is defined as $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\Pi_{\mathsf{PKE}}}(\mathcal{A}) = |\Pr[\mathcal{A} \text{ wins}] - 1/2]|$.

2.3.5 *Commitments*

We omit the security games for the hiding and binding properties of a commitment scheme as they are well established and standard.

SYNTAX. $\mathsf{Com}(m, r) \to c$: A commitment scheme $\mathsf{Com} : \mathcal{M} \times \mathcal{S} \to \mathcal{C}$ takes as input a message x from the message space $\mathcal{M}$ and an element r of the randomness space $\mathcal{S}$. It outputs a commitment c from the commitment space $\mathcal{C}$.

SECURITY. We say that Com is secure if it is both hiding and binding as defined below.

Definition 2.3.12 (Hiding)**.** *We say that* Com *is statistically hiding (resp. computationally hiding) if, given $m, m' \in X$, a computationally unbounded*

(resp. bounded) adversary is unable to distinguish between the distribution of $\mathsf{Com}(mm, r)$ *and* $\mathsf{Com}(m', r)$ *over random* $r \in \mathcal{S}$ *with more than negligible probability.*

Definition 2.3.13 (Binding)**.** *We say that* Com *is statistically binding (resp. computationally binding) if, given* $m, m' \in \mathcal{M}$*, a computationally unbounded (resp. bounded) adversary is unable to produce* (m, r) *and* (m', r') *such that* $m \neq m'$ *and* $\mathsf{Com}(m, r) = \mathsf{Com}(m', r')$ *with more than negligible probability.*

2.3.6 Σ-*Protocols*

At a high level, a Σ-protocol is an interactive proof system which allows a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that some statement is true. Such protocols are most useful when they allow $\mathcal{P}$ to show that they know a secret or, in the case of cryptography, the solution to some hard mathematical problem. Furthermore, these systems are only interesting when $\mathcal{P}$ wishes to convince $\mathcal{V}$ *without* revealing the secret/solution itself. Indeed, as we shall see, the 'zero-knowledge' property of a Σ-protocol requires that nothing more than the validity of $\mathcal{P}$'s claim is revealed.

Interactive proof systems of this form were first introduced Goldwasser, Micali, and Rackoff in [GMR85] and are now a ubiquitous building block in many privacy-preserving cryptographic constructions. As we shall we, Σ-protocol form the basis for a zero-knowledge-proof.

Let $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^*$ be a polynomial time recognizable relation. For $(x, \mathsf{w}) \in \mathcal{R}$, we call x as the statement and w as the witness. Furthermore, let $\mathcal{L} = \{\, x \mid \exists \text{ s.t. } (x, \mathsf{w}) \in \mathcal{R} \,\}$ be the corresponding **NP** language to the relation $\mathcal{R}$.

A Σ-*protocol* defined for a relation $\mathcal{R}$ is a public-coin three-move interactive protocol between a prover and a verifier. As with many lattice-based Σ-protocols, we define a slightly relaxed variant of the standard Σ-protocol where soundness holds for a wider relation $\mathcal{R}'$ than the relation $\mathcal{R}$ being used in the actual protocol. Below, we provide a definition of a Σ-protocol in the *common reference string* model [Bel20]. An overview is depicted in Figure 2.3. We note that this does not loose generality since most of the results known to hold for Σ-protocols in the plain model hold for Σ-protocols in the CRS model. Moreover, in many cases, Σ-protocols are implicitly defined in the CRS model by, for example, assuming public parameters for a commitment scheme are provided to the prover and verifier.

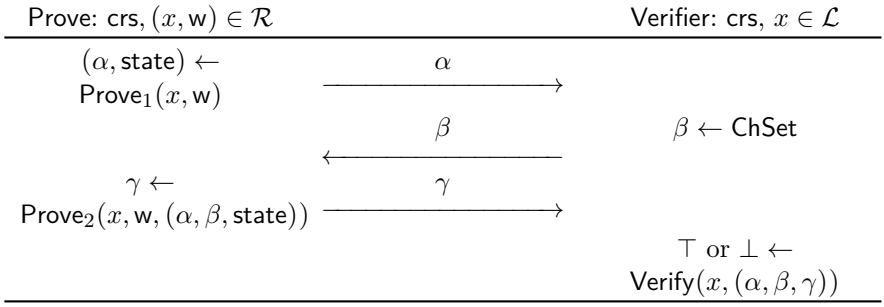

FIGURE 2.3: Σ-protocol in the CRS model. All the algorithms are assumed to be given crs as input.

Definition 2.3.14 (Σ-protocol in the CRS model)**.** *A Σ-protocol in the common reference string (CRS) model for relations $(\mathcal{R}, \mathcal{R}')$ such that $\mathcal{R} \subseteq \mathcal{R}'$ is defined by a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Prove} = (\mathsf{Prove}_1, \mathsf{Prove}_2), \mathsf{Verify})$*, where* Verify *is a deterministic polynomial time algorithm. We assume the relation $\mathcal{R}$ defines the set of all commitments* ComSet*, challenges* ChSet*, and responses* ResSet*. A Σ-protocol in the CRS model proceeds as follows:*

1. *A common reference string is sampled* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$*;*
2. *The prover, on input* crs *and* $(x, \mathsf{w}) \in \mathcal{R}$*, runs* $(\alpha, \mathsf{state}) \leftarrow \mathsf{Prove}_1(\mathsf{crs}, x, \mathsf{w})$ *and returns* $\alpha \in \mathsf{ComSet}$ *to the verifier;*
3. *The verifier then samples a challenge* $\beta \leftarrow \mathsf{ChSet}$ *and returns it to the prover;*
4. *The prover sends a response* $\gamma \leftarrow \mathsf{Prove}_2(\mathsf{crs}, x, \mathsf{w}, (\alpha, \beta, \mathsf{state}))$ *to the verifier, where* $\gamma \in \mathsf{ResSet} \cup \{\bot\}$ *and* $\bot \notin \mathsf{ResSet}$ *is a special symbol indicating failure. Finally, the verifier runs* $\mathsf{Verify}(\mathsf{crs}, x, (\alpha, \beta, \gamma))$ *and outputs* $\top$ *for acceptance and* $\bot$ *for rejection.*

The transcript (α, β, γ) *is called a* valid transcript *if* $\mathsf{Verify}(\mathsf{crs}, x, (\alpha, \beta, \gamma)) \to \top$*.*

For simplicity, we may refer to the above simply as a Σ-protocol when the meaning is clear. We require a Σ-protocol to satisfy several properties. The first property is correctness.

Definition 2.3.15 (Correctness)**.** *A Σ-protocol has correctness error* (δ_0, δ_1) *if for any* $\mathsf{crs} \in \mathsf{Setup}(1^\lambda)$ *and* $(x, \mathsf{w}) \in \mathcal{R}$*, the following holds:*

- *We have* $\Pr[\mathsf{Verify}(\mathsf{crs}, x, (\alpha, \beta, \gamma)) = \top] \geq 1 - \delta_0$*, where the probability is take over the randomness to sample* $(\alpha, \mathsf{state}) \leftarrow \mathsf{Prove}_1(\mathsf{crs}, x, \mathsf{w})$*,* $\beta \leftarrow \mathsf{ChSet}$*, and* $\gamma \leftarrow \mathsf{Prove}_2(\mathsf{crs}, x, \mathsf{w}, (\alpha, \beta, \gamma))$ *conditioning on* $\gamma \neq \perp$*.*
- *The probability that an honestly generated transcript* (α, β, γ) *contains* $\gamma = \perp$ *is bounded by* δ_1*. In particular,* $\Pr[\gamma = \perp] \leq \delta_1$ *where the probability is taken over the random coins of the prover and verifier.*

We define no-abort zero-knowledge, a weak variant of honest-verifier zero-knowledge, for Σ-protocols. For this variant, we only require the transcript to be simulatable with only knowledge of x conditioned on $\gamma \neq \perp$.

Definition 2.3.16 (No-abort honest-verifier zero-knowledge)**.** *Let* $D^{\gamma}_{\mathsf{ts}}(\mathsf{crs}, x, \mathsf{w})$ *be the distribution of* $\mathsf{ts} = (\alpha, \beta, \gamma)$ *from an honest protocol with prover input* $(\mathsf{crs}, x, \mathsf{w})$ *conditioned on* $\gamma \neq \perp$*. Then, we say a* Σ*-protocol is (quantum)* ϵ_{zk}-no-abort honest-verifier zero-knowledge (naHVZK)*, if there exists a PPT algorithm* ZKSim *such that for all* $(x, \mathsf{w}) \in \mathcal{R}$ *and QPT* $\mathcal{A}$*, the advantage* $\mathcal{A}^{\mathsf{naHVZK}}(\mathcal{A})$ *defined below is less than* ϵ_{zk}*:*

$$\mathcal{A}^{\mathsf{naHVZK}}(\mathcal{A}) := \Big| \Pr[\mathsf{ts} \leftarrow D^{\gamma}_{\mathsf{ts}}(\mathsf{crs}, x, \mathsf{w}) : \mathcal{A}(\mathsf{crs}, \mathsf{ts}) \to 1] - \Pr[\beta \leftarrow \mathsf{ChSet}, (\alpha, \gamma) \leftarrow \mathsf{ZKSim}(\mathsf{crs}, x, \beta) : \mathcal{A}(\mathsf{crs}, (\alpha, \beta, \gamma)) \to 1] \Big|,$$

where the probability is taken also over the randomness of $\mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda})$*.*

Definition 2.3.17 (Soundness)**.** *A* Σ*-protocol is* (T, ϵ)*-sound if for every adversary* $\mathcal{A}$ *running in time* $T(\lambda)$ *we have that*

$$\Pr\left[(b = 1 \wedge x \notin \mathcal{L} \;:\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^{\lambda}) \\ (\mathsf{ts}, x) \leftarrow \mathcal{A}(\mathsf{crs}), \\ b \leftarrow \mathsf{Verify}(\mathsf{crs}, x, \mathsf{ts}) \end{array} \right] \leq \epsilon(\lambda).$$

The following property is often useful in proving the soundness of a proof system. In essence, it says that if a prover can respond to two different challenges with, for a fixed initial commitment flow, then one can extract a witness to the statement.

Definition 2.3.18 (Relaxed k-special soundness)**.** *A* Σ*-protocol has* k-special soundness *if there is a deterministic PT algorithm* $\mathsf{Extract}_{\mathsf{ss}}$ *such that given* k *valid transcripts* $(\alpha, \{\beta_i, \gamma_i\}_{i \in [k]})$ *for any* $\mathsf{crs} \in \mathsf{Setup}(1^{\lambda})$ *and statement* $x \in \mathcal{L}$ *with pairwise distinct* β_i*'s, it outputs a witness* w *such that* $(x, \mathsf{w}) \in \mathcal{R}'$*. In case* $\mathcal{R}' = \mathcal{R}$*, we call it "exact" special sound or simply special sound.*

Finally, we note that while Σ-protocol are defined as having three rounds, one can consider interactive proof systems with more, extending the definitions above in a natural way to match this.

2.3.7 *Non-Interactive Zero Knowledge*

Non-iterative zero-knowledge (NIZK) proofs [BFM88, BSMP91] consist of a single proof message sent from a prover to a verifier. This type of zero-knowledge proof has great advantages over interactive proof systems such as Σ-protocols presented in Section 2.3.6. Importantly, no interaction is required from the verifier so that a prover can produce a complete proof without the involvement of the verifier. This means that the prover does no have to wait for the verifier to be 'on-line' when it creates its proof. Furthermore, these types of proof require less 'bandwidth' (communication data size).

As we shall see, the starting point for constructing many NIZKs in the literature is to first create an interactive system which will later be made non-interactive. This will be discussed in more detail in Section 2.3.8, however we must first define the properties we require of a NIZK. It was proven by Goldreich and Oren [GO94] that NIZKs cannot be achieved in the standard model, that it without making additional assumptions about the setting. To circumvent this impossibility result one must look to either the *common reference string* (CRS) model [BFM88] in which both prover and verifier share some common string in addition to the statement being proven, or the *random oracle model* (ROM) [BR93] which assumes the existence of a truly random hash function. Both models are widely used and since we will use both in this thesis we present NIZK definitions which are compatible with both.

Let $\mathcal{R} \subset \{0,1\}^* \times \{0,1\}^*$ be a polynomial time recognizable relation. For $(x, \mathsf{w}) \in \mathcal{R}$, we call x as the statement and w as the witness. Furthermore, let $\mathcal{L} = \{\, x \mid \exists \text{ s.t. } (x, \mathsf{w}) \in \mathcal{R} \,\}$ be the corresponding NP language to the relation $\mathcal{R}$.

Definition 2.3.19 (NIZK in the ROM/CRS model)**.** *A ROM-secure* NIZK *in the CRS model for a relation $\mathcal{R}$ is defined by a tuple of algorithms* (Setup, $\mathsf{Prove}^{\mathsf{H}(\cdot)}, \mathsf{Verify}^{\mathsf{H}(\cdot)}$)*, where* Prove *and* Verify *have access to a cryptographic hash function* H *and* Verify *is a deterministic polynomial time algorithm. A* NIZK *in the CRS/ROM model proceeds as follows.*

1. *A common reference string is sampled* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$*;*

2. *The prover, on input* crs *and* $(x, \mathsf{w}) \in \mathcal{R}$*, returns a proof* π *to the verifier;*

3. *The verifier, on input* (crs, x, π) *returns* 1 *to indicate* accept *and* 0 *for* reject.

We call $(\mathsf{Setup}, \mathsf{Prove}^{\mathsf{H}(\cdot)}, \mathsf{Verify}^{\mathsf{H}(\cdot)})$ a NIZK if it satisfies the properties of correctness, non-interactive zero-knowledge, and knowledge soundness defined as follows. Note that we present the computational versions of these definitions i.e. we do not consider adversaries of unbounded computational resources but only those who run in probabilistic polynomial time.

Definition 2.3.20 (Correctness)**.** *A* NIZK *is correct if for all adversaries* $\mathcal{A}$*,*

$$\Pr\left[(x, \mathsf{w}) \in \mathcal{R} \wedge \mathsf{Verify}^{\mathsf{H}(\cdot)}(\mathsf{crs}, x, \pi) = 1 \;:\; \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ (x, \mathsf{w}) \leftarrow \mathcal{A}(\mathsf{crs}), \\ \pi \leftarrow \mathsf{Prover}^{\mathsf{H}(\cdot)}(\mathsf{crs}, x, \mathsf{w}) \end{array} \right] = 1.$$

Definition 2.3.21 (Non-interactive zero-knowledge)**.** *We say that the proof system* $\mathsf{NIZK} := (\mathsf{Setup}, \mathsf{Prove}^{\mathsf{H}(\cdot)}, \mathsf{Verify}^{\mathsf{H}(\cdot)})$ *is* $\epsilon_{\mathsf{zk}}(\lambda)$*-non-interactive zero-knowledge if there exists a PPT simulator* $\mathsf{ZKSim} = (S_1, S_2)$ *such that for all PPT adversaries* $\mathcal{A}$*, the advantage* $\mathcal{A}^{\mathsf{ZK}}(\mathcal{A})$ *defied below is less that* $\epsilon_{\mathsf{zk}}(\lambda)$*:*

$$\mathcal{A}^{\mathsf{ZK}}_{\mathcal{A}}(\lambda) := \left| \Pr\left[1 \leftarrow \mathcal{A}^{\mathsf{H}, \mathcal{P}}(\mathsf{crs})\right] - \Pr\left[1 \leftarrow \mathcal{A}^{S_1, S_2}(\mathsf{crs})\right] \right|,$$

where $\mathcal{P}$ *and* S_2 *are prover oracles that on input* (x, w) *return* $\perp$ *if* $(x, \mathsf{w}) \notin \mathcal{R}$*, else output* $\mathsf{Prove}^{\mathsf{H}}(\mathsf{crs}, x, \mathsf{w})$ *or* $S_2(\mathsf{crs}, x)$ *respectively. The probability here is taken over the choice of* crs *and of the random oracle* H*, and both* S_1 *and* S_2 *have shared state.*

We now define knowledge soundness. Often called *extractability*, this notion ensures that if a prover can create a valid proof, not only is the statement in the language but they must know a corresponding witness. Note that this implies standard soundness which only asserts that the statement is in the language i.e. existence of a witness.

Definition 2.3.22 (Knowledge-Soundness)**.** *A* NIZK *is knowledge-sound if there exists a PPT extractor algorithm* Ext *such that for any* crs*, given oracle access to any PPT adversary* $\mathcal{A}$ *with*

$$\Pr[(x, \pi) \leftarrow \mathcal{A}^{\mathsf{H}}(\mathsf{crs}) \;:\; \mathsf{Verify}^{\mathsf{H}}(\mathsf{crs}, \pi, x) = 1] \geq \epsilon(\lambda),$$

we have

$$\Pr\left[\begin{array}{c}(x,\pi)\leftarrow\mathcal{A}^{\mathsf{H}}(\mathsf{crs}),\\ \mathsf{w}\leftarrow\mathsf{Ext}(\mathsf{crs},x,\pi,\vec{h})\end{array}\;:\;(x,\mathsf{w})\in\mathcal{R}\right]\geq\frac{\epsilon(\lambda)-\mathsf{negl}(\lambda)}{p(\lambda)},$$

where $\vec{h}$ *are the outputs of* H *and* p *is a PPT algorithm.*

2.3.8 *The Fiat-Shamir Transform*

We now present a key transform which turns a Σ-protocol (see Section 2.3.6) into a non-interactive zero-knowledge proof (NIZK) (see Section 2.3.7). This is one of the most prolific and efficient methods used in the literature to remove interaction from a (interactive) Σ-protocol. In a nutshell, the Fiat-Shamir transform [FS87] removes the interaction by computing the challenge as the hash value of the first message and the statement being proven. The intuition here is that, treating the hash function as a truly random oracle, the prover can still not predict the challenge any better than guessing. This scenario in which we assume that the hash function behaves as a truly random function is what we call the *random-oracle model* (ROM). As noted in Section 2.3.7, one cannot construct NIZKs in the standard (assumption free) model and the ROM is widely used in practice.

In [FKMV12] the authors show that the Fiat-Shamir transform converts a Σ-protocol into a NIZK in the following sense. If the Σ-protocol has the min-entropy property, is honest-verifier zero-knowledge, and has quasi-unique responses then the proof system $(\mathsf{Setup},\mathsf{Prove}^{\mathsf{H}(\cdot)},\mathsf{Verify}^{\mathsf{H}(\cdot)})$ derived from $(\mathsf{Setup},\mathsf{Prove},\mathsf{Verify})$ via the Fiat-Shamir transform is an unbounded non-interactive zero-knowledge and simulation-sound NIZK. The properties are both strictly stronger than th zero-knowledge and knowledge soundness properties defined above.

2.3.9 *Digital Signatures*

SYNTAX. A signature scheme $\mathcal{S}$ is a tuple of three probabilistic polynomial-time algorithms $(\mathsf{S.Kg},\mathsf{S.Sign},\mathsf{S.Ver})$ with the following syntax. .

1. $\mathsf{S.Kg}(1^\lambda)\rightarrow(\mathsf{pk},\mathsf{sk})$: This key generation algorithm takes as input a security parameter λ and outputs a public and private key pair $(\mathsf{pk},\mathsf{sk})$.

2. $\mathsf{S.Sign}(\mathsf{sk},\mathsf{m})\rightarrow\sigma$: This signing algorithm takes as input a private key sk and a message $\mathsf{m}\in\{0,1\}^*$. It outputs a signature σ.

3. $\mathsf{S.Ver}(\mathsf{pk}, \mathsf{m}, \sigma) \to 1/0$: This deterministic verification algorithm takes as input a public key pk, a message m, and a signature σ. It outputs a bit 1 or 0 to indicate accept or reject.

A signature scheme $\mathcal{S} = (\mathsf{S.Kg}, \mathsf{S.Sign}, \mathsf{S.Ver})$ is said to be correct if for every λ, all $(\mathsf{pk}, \mathsf{sk}) \in \mathsf{S.Kg}(1^\lambda)$, and every $\mathsf{m} \in \{0,1\}^*$, it holds that

$$\mathsf{S.Ver}(\mathsf{pk}, \mathsf{m}, \mathsf{S.Sign}(\mathsf{sk}, \mathsf{m})) = 1.$$

SECURITY. We define the following experiment in order to define the security of $\mathcal{S}$.

Definition 2.3.23 (Forgery Experiment $\mathsf{Exp}^{\text{EU-CMA}}_{\mathcal{A},\mathcal{S}}(\lambda)$)**.** *The experiment proceeds as follows.*

1. $\mathsf{S.Kg}(1^\lambda)$ *is run to obtain keys* $(\mathsf{pk}, \mathsf{sk})$.
2. $\mathcal{A}$ *is given* pk *and oracle access to* $\mathsf{S.Sign}(\mathsf{sk}, \cdot)$. $\mathcal{A}$ *then outputs* (m, σ). *Let* $\mathcal{Q}$ *denote the set of signing queries made.*
3. *The output of the experiment is defined to be* 1 *if and only if* $(\mathsf{S.Ver}(\mathsf{pk}, \mathsf{m}, \sigma) = 1) \wedge (\mathsf{m} \notin \mathcal{Q})$.

Definition 2.3.24 (EU-CMA Security)**.** *The signature scheme* $\mathcal{S} = (\mathsf{S.Kg}, \mathsf{S.Sign}, \mathsf{S.Ver})$ *is said to existentially unforgeable under chosen-message attack if for all probabilistic polynomial time adversaries* $\mathcal{A}$*, there exists a negligible function* negl *such that:*

$$\Pr[\mathsf{Exp}^{\text{EU-CMA}}_{\mathcal{A},\mathcal{S}}(\lambda) = 1] \leq \mathsf{negl}(\lambda).$$

3

PROPERTY-BASED DAA SECURITY WITH UC ENDORSEMENT

3.1 INTRODUCTION

In this chapter we present a property-based definition of DAA, prove that it implies the existing definition from [CDL16], and give a novel generic construction, and demonstrate the proficiency of our propose security framework by proving the security of this new construction. For a high-level overview of DAA and existing works, we refer to Section 1.1.1. Since the theoretical framework of this chapter is quite involved, we begin with a detailed technical overview the results.

3.1.1 *Technical Overview*

COMPREHENSIVE SECURITY DEFINITIONS. In moving to the game-based framework, for each experiment, we aim to extract the same guarantees that the UC-DAA functionality provides for a particular security notion. This is a non-trivial task since in the ideal functionality $\mathcal{F}_{\mathsf{daa}}$, it is often the case that numerous checks across several of its interfaces interleave to achieve a given security notion. Thus, we must unravel $\mathcal{F}_{\mathsf{daa}}$ and split its security guarantees into individual properties. We do this for each of the five properties mentioned above.

In Section 1.1.1 we highlight two main pitfalls with defining property-based DAA security. We now briefly describe the techniques used in this work to overcome each of these obstacles. Interestingly both of these challenges are most acute in the unforgeability game. Here, no adversary should be able to create more unlinkable signatures than he has corrupt TPMs, nor can he produce a signature which identifies to an honest TPM who was not involved in its creation. Crucially, this property should hold even when the platform's host machine is corrupt but the TPM remains honest. Whilst the solutions we present do lead to a sound notion of unforgeability, these techniques have wider reaching benefits for the whole model. However, the unforgeability property best motivates their necessity and exhibits their elegance, so we focus on the unforgeability game below. There are several approaches explored

in the literature to tackle this heavy security notion. We take the most recent property-based model of Bernard et al. [BFG+11] as a motivating example which becomes unstuck in the following ways.

- Due to the complexity introduced by the TPM and host being in different corruption states, it is attractive to first consider an unforgeability game in which both parties are in the same corruption state. One must then provide a 'lifting' argument to bootstrap this weaker notion to a stronger one where the TPM and host can be in different corruption states. However, due to the sensitive nature of the unforgeability notion, such lifting does not exist in general, under the same hardness assumptions. Indeed, this is where the work of [BFG+11] brakes down. Details of the attack on this scheme can be found in Section 2.2 of [CDL16].

- A fundamental feature of the unforgeability game is to ensure that a valid signature can only be generated by a legitimate TPM. In particular this requires that given a signature one must be able to determine whether it has come from any honest *or corrupt* TPM who has 'joined'. This latter case is where the difficulty lies since it is not clear how the challenger can check this against corrupt TPMs for which he doesn't know the secret key. This is exactly due to the absence of a notion of TPM identity since there is no public key associated with a TPM's secret key. [BFG+11] makes some progress toward this problem by using extracted secret keys from join transcripts to identify signatures coming from corrupt TPMs. Unfortunately, the extractability of secret keys from each join transcript is something not assumed in the accompanying scheme nor is formally defined in the security model.

To overcome this first problem of great complexity introduced by the presence of a TPM and host that can be in different corrupt states, we accompany our security experiments with a detailed set of oracles (see Figure A.3) used to simulate honest parties for the adversary. Where interactive protocols are considered, our oracle definitions allow the adversary to make queries on *individual protocol messages* rather than having to complete the full protocol before moving on. For example, the adversary is not obliged to complete a signing protocol with one TPM before beginning another with a different TPM. This is done by adapting the notation used in group signatures [BMW03], [BCC+16] to the DAA setting. The granularity of these oracle definitions is essential for capturing all meaningful adversarial behaviour, something only attempted in [BFG+11], though there it was for the vastly simplified setting where the TPM and host are assumed to be in

the same corruption state. Although this slightly adds to the complexity of oracle definitions, we are rewarded with a much more succinct set of security experiments which we can be sure capture the multitude of corruption settings in DAA. Indeed, our model comprises fewer security experiments than the most recent attempt of Barnard et al. in which a vastly simplified 'pre-DAA' is considered.

To tackle the second issue of TPM identification we import the auxiliary algorithms Ext and $\mathsf{Identify}$ from [CDL16] where they are used informally. We formalise them for the first time, and introduce one extra auxiliary algorithm $\mathsf{Identify}_\mathsf{T}$ along the way to make the formalisation clear. In order to formally argue unforgeability, there are three questions to answer: how can the challenger (playing the role of the issuer) be sure that it is communicating with some TPM during the join protocol, how can it identify TPMs that have joined the system via the join protocol, and how can it identify a signature to a TPM that has joined the system? Algorithms Ext, $\mathsf{Identify}_\mathsf{T}$, and $\mathsf{Identify}$ are defined exactly to answer these questions, respectively. Namely, algorithm Ext is used to extract TPM secret keys from a TPM's communication with an (honest) issuer. Algorithm $\mathsf{Identify}_\mathsf{T}$ is then used to determine whether a given TPM key was used in a given join transcript. Finally, algorithm $\mathsf{Identify}$ can then be used to determine whether a signature was generated using this secret key. In many previous works authors have tended to use more informal notions in their security definitions of DAA. This is perhaps due to DAA's inherent complexity to begin with. More often than not, however, this lack of formality has been at the root of security flaws overshadowing those works. We note the work of Bernard et al. [BFG+11] does not formally define TPM key extraction but assumes this implicitly by requiring an adversary to output keys of corrupt TPMs. This less concrete treatment of extraction leads to a security reduction requiring exponential time (in rewinding). Our work formalises the aforementioned auxiliary algorithms with their necessary interdependence for the first time.

Besides overhauling the unforgeability definition, we make small, but essential alterations to the anonymity and non-frameability games. Here, we allow the adversary to choose the issuer's secret key. Since schemes like EPID do not assume a trusted setup nor require the issuer to register its public key with an extractable proof of correctness, we have to model this correctly. We also allow the adversary multiple calls to the challenge oracle in the anonymity experiment. Failure to do so in [BFG+11] leads to a definition allowing for trivially non-anonymous schemes to be proven secure. This is demonstrated in Appendix 3.3.3.

IMPLYING THE UC DEFINITION. The remaining task is to check that our new definition indeed captures all desirable aspects of DAA security. We give a natural confirmation of this by proving that our model implies the UC-DAA definition of [CDL16].

To build confidence in any property-based definition of DAA, one should be able to claim that it at least provides, in essence, the same security guarantees as the UC-DAA definition of [CDL16]. The use of 'essence' here is a strategic one since clearly no property-based definition can provide the composability guarantees as one in the UC model. Nevertheless, existing works [CLNS17] have successfully shown how a new property-based definition can be re-formulated or 'wrapped' to imply an existing UC definition. Crucially, this process should not affect the security guarantees of the new definition but should merely serve to make a direct comparison between them possible. Indeed, one can view this merely as a cosmetic repackaging of the property-based algorithms to create a UC protocol that can be compared to an existing UC functionality.

Proving that a property-based definition implies one formulated in the UC framework presents a number of challenges. First and foremost, the UC framework [Can01] assumes by default a completely asynchronous network in which only one party can be activated at any given time. As a consequence of this and DAA's interactive protocols, the authors of [CDL16] were forced to split the join and sign interfaces of their ideal functionality each into two phases: a request and proceed phase. Consequently, schemes proven in the UC model must also contain these split phases even though any implementation must eventually combine them to create a single join protocol and a single signing protocol. Indeed, the ability to define intuitive, single join and signing protocols in our new model is very attractive for scheme design and something we consider a valuable and non-trivial contribution. We define a wrapper protocol Π_{DAA} in Section 3.4.1 that matches the compact, intuitive join and sign algorithms of our model to the split interfaces of the UC DAA functionality. Our main result, Theorem 3.4.1, shows that protocol Π_{DAA} realises the UC functionality $\mathcal{F}_{\mathsf{daa}}$ [CDL16]. This is done by a series of game hops where indistinguishably between games is shown by reducing to the assumed security properties of our new model. Thus, this result confirms that our property based model captures the full essence of DAA security and does not suffer from the flaws of previous property-based models.

A NOVEL GENERIC CONSTRUCTION. We present a generic construction of DAA based on standard cryptographic primitives and prove it secure

in our proposed property-based model. Consequently, we present the first generic DAA construction which (when wrapped) provides UC security. The generic construction allows us to view prior constructions of DAA [BFG+11] in a more modular manner, and in particular, it provides better intuition on why the constructions are secure/insecure. This in sharp contrast to prior works that design schemes using involved tools such as homomorphic commitments and/or non-standard signatures allowing for a two-party signing process [BE17, KCB+18]. Since our construction only relies on simple building blocks such as standard commitment schemes and signature schemes, we believe our proofs are much easier to understand and verify.

3.2 PRELIMINARIES

For the presentation of our generic construction in Section 3.5 we will need a number of cryptographic building blocks not yet defined in the preliminary of this thesis. These are an injective one-way function, a linkable-indistinguishable tag, and a signature of knowledge. We define them here.

Secondly, we recall the UC-based definition of DAA formalised by Camenisch, Drijvers, and Lehmann.

3.2.1 *Injective One-Way Function*

Let n and m be integer-valued polynomials of the security parameter λ. A family of functions $\mathcal{F} = \{\mathsf{F}_\lambda : \{0,1\}^n \to \{0,1\}^m\}_{\lambda\in\mathbb{N}}$ is said to be an *injective one-way function* (OWF) if the following condition holds:

1. (Easy to compute:) For any $x \in \{0,1\}^n$, $\mathsf{F}_\lambda(x)$ can be computed in polynomial time.
2. (Injectivity:) For any $x_1, x_2 \in \{0,1\}^n$, $\mathsf{F}(x_1) = \mathsf{F}(x_2)$ implies that $x_1 = x_2$.
3. (Hard to invert:) For every probabilistic polynomial-time algorithm $\mathcal{A}$, we have

$$\Pr[x \leftarrow \{0,1\}^n; y = \mathsf{F}_\lambda(x); x' \leftarrow \mathcal{A}^{\mathsf{F}_\lambda(\cdot)}(y) : \mathsf{F}_\lambda(x') = y] \leq \mathsf{negl}(\lambda),$$

   where the probability is taken over the randomness of sampling x and the randomness used by $\mathcal{A}$.

When the context is clear, we omit the subscript and simply say that F is a injective one-way function.

3.2.2 *Linkable Indistinguishable Tag*

A linkable indistinguishable tag (LIT) compatible with a one-way function F (not necessarily injective) was originally introduced in [BFG+11] in the context of DAA. Formally, we define LIT with respect to F as follows.

SYNTAX. A collision-resistant LIT scheme $\mathcal{T}$ with message space $\{0,1\}^n$ with respect to a function $\mathsf{F} : \{0,1\}^n \to \{0,1\}^m$ is a tuple of algorithms $(\mathsf{T.Kg}, \mathsf{Tag}, \mathsf{T.Link}, \mathsf{T.Check})$ defined as follows.

- $\mathsf{T.Kg}(1^\lambda) \to x$: The key generation algorithm takes as input a security parameter λ and outputs a secret key x.
- $\mathsf{Tag}(x, \mathsf{m}) \to \mathsf{v}$: The tag generation algorithm takes as input a secret key x and a message m, and outputs a tag v.
- $\mathsf{T.Link}(\mathsf{v}_1, \mathsf{v}_2) \to 1/0$: The deterministic linking algorithm takes as input two tags v_1, v_2, and outputs 1 or 0 to indicate accept or reject, respectively.
- $\mathsf{T.Check}(\mathsf{v}, x, \mathsf{m}) \to 1/0$: The deterministic check algorithm takes as input a tag v, a secret key x and a message m, and outputs 1 or 0 to indicate accept or reject, respectively.

CORRECTNESS. For all $\lambda \in \mathbb{N}$, $x \in \mathsf{T.Kg}(1^\lambda)$, and $\mathsf{m}_0, \mathsf{m}_1 \in \{0,1\}^n$, if we run $\mathsf{v}_b \leftarrow \mathsf{Tag}(x, \mathsf{m}_b)$ for $b \in \{0,1\}$, then we have $1 \leftarrow \mathsf{T.Check}(\mathsf{v}, x_b, \mathsf{m}_b)$ for $b \in \{0,1\}$ with probability one. Furthermore, if $\mathsf{m}_0 = \mathsf{m}_1$ then we have $1 \leftarrow \mathsf{T.Link}(\mathsf{v}_0, \mathsf{v}_1)$ with probability 1. Moreover for any v_0 and v_1, we require the linking algorithm to be symmetric with respect to its input: $\mathsf{T.Link}(\mathsf{v}_0, \mathsf{v}_1) = \mathsf{T.Link}(\mathsf{v}_1, \mathsf{v}_0)$.

SECURITY OF FUNCTION F. We require one-wayness of function F to hold in the presence of tags. Specifically, we require the following.

Definition 3.2.1 (One-Wayness for F)**.** *Consider LIT scheme* LIT *with respect to a function* F*. We say the function* F *is one-way with respect to LIT if for every probabilistic polynomial-time algorithm* $\mathcal{A}$ *which queries the* $\mathsf{Tag}(x, \cdot)$ *oracle only once on the same input message, there exists a negligible function* negl *such that*

$$\Pr[x' \leftarrow \mathcal{A}^{\mathsf{F}(\cdot), \mathsf{Tag}(x,\cdot)}(y) : \mathsf{F}(x') = y] \leq \mathsf{negl}(\lambda),$$

where $y = \mathsf{F}(x)$ *for* $x \leftarrow \mathsf{T.Kg}(1^\lambda)$.

SECURITY OF LIT. There are three properties that we wish our LIT to have with respect to function F; indistinguishability, linkability, and collision resistance. These three properties are formalized by the games in Figure 3.1. Informally, these properties capture the following scenarios.

- *Indistinguishably with respect to* F ensures that for some fixed key, given a tag, no adversary can tell whether a new tag on a message of his choice was generated under the fixed key or not.

- *Linkability* ensures that no adversary can create two tags which link but were not created using the same key.

- *Collision resistance* ensures that, no adversary can come up with two different secret keys that would create a same valid tag on the same message. Note that if we assume $\mathsf{T.Link}(\mathsf{v},\mathsf{v}) = 1$ for all possibly maliciously generated tag v, then collision resistance implies linkability.

Definition 3.2.2 (Security of LIT)**.** *A LIT scheme* LIT *with respect to a function* F *is said to satisfy indistinguishably with respect to* F*, linkability, and collision resistance if for all probabilistic polynomial-time adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that*

- $|\Pr[\mathsf{Exp}^{\text{OWF-IND-0}}_{\mathcal{A},\mathsf{LIT}}(\lambda) = 1]| - |\Pr[\mathsf{Exp}^{\text{OWF-IND-1}}_{\mathcal{A},\mathsf{LIT}}(\lambda) = 1]| \leq \mathsf{negl}(\lambda)$,
- $\Pr[\mathsf{Exp}^{\text{LINK}}_{\mathcal{A},\mathsf{LIT}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$,
- $\Pr[\mathsf{Exp}^{\text{COLL-RES}}_{\mathcal{A},\mathsf{LIT}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$.

3.2.3 *Signatures of Knowledge*

We define signatures of knowledge, often referred to as signature-based proofs of knowledge (SPK). These are essentially specialised forms of non-interactive zero-knowledge proofs: if a party P can generate a valid signature of knowledge on any message m for a statement x in a language L, that should mean that, firstly, the statement is true, and secondly, P knows a witness for that statement.

$\mathsf{SPK.Setup}(\lambda) \rightarrow (\mathsf{sp})$: This setup algorithm takes as input a security parameter λ and outputs the system parameters sp.

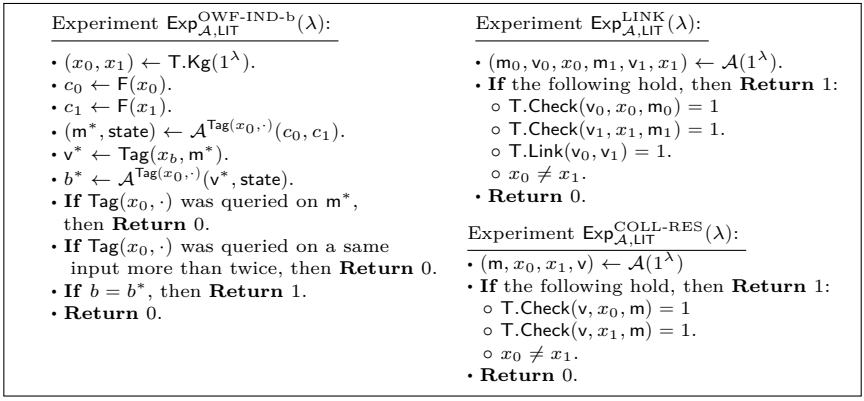

FIGURE 3.1: Security games of a collision resistant LIT.

$\mathsf{SPK.Sign}\{\mathsf{w} \mid x\}(\mathsf{m}) \to ß$: This signing algorithms takes as input a witness w to some statement x in a language L, and a message m. It outputs a signature of knowledge ß for instance $x \in L$ on m.

$\mathsf{SPK.Verify}(x, \mathsf{m}, ß) \to 1/0$: This verify algorithm takes as input a statement x, a message m and a purported signature ß. It outputs 1 or 0 to indicate accept or reject.

Remark 3.2.3. *For readability, we omit the polynomial-time Turing machine M_L from the syntax which accepts inputs (x, w) if and only if w is a witness showing that x is in a language L. We also omit* sp *as an input to* SPK *for the same reason.*

SECURITY. Informally we say that a signature of knowledge is SimExt-secure if it is correct, simulatable and extractable.

Correctness simply says that any signatures created using SPK should be accepted by SPK.Verify.

Simulatability says that there exists a simulator who, given some trapdoor information on the parameters, can create valid signatures without knowing a witness. The simulator is divided into two parts; SPK.SimSetup, which outputs a set of parameters with trapdoor information and SPK.SimSign, which then signs using these parameters. It is required that no adversary can distinguish between the output distributions of (SPK.SimSetup, SPK.SimSign) and (SPK.Setup, SPK.Sign).

Extractability says that there exists an extractor, with knowledge of the

trapdoor information, who, given a signature of knowledge on some statement $x \in L$, can produce a valid witness showing that $x \in L$. This property ensures that it is impossible to create a valid signature of knowledge without knowing a witness.

For the formal definition of SimExt-security we refer to Definition 2.2 of [CL06].

3.3 A PROPERTY-BASED DEFINITION OF DAA SECURITY

We first develop the syntax in Section 3.3.1 and then present the *property-based* security definition of a DAA scheme in Section 3.3.3. Due to the complex nature of the security notion for DAA, in Section 3.3.1, we also prepare some oracles and auxiliary algorithms needed to formally define the security requirements. However, before getting into details, we informally discuss how DAA works and the desired security properties below.

In a DAA scheme, we have four main entities: a number of trusted platform modules (TPM), a number of hosts, an issuer, and a number of verifiers. A TPM and a host together form a platform which performs the *join protocol* with the issuer who decides if the platform is allowed to become a member. Once a member, the TPM and host (i.e., platform) can together sign messages with respect to a basename bsn. If a platform signs with $\mathsf{bsn} = \perp$ or a fresh basename, the signature must be anonymous and unlinkable to previous signatures. That is, any verifier can check that the signature stems from a legitimate platform via a deterministic verify algorithm, but the signature does not leak any information about the identity of the platform. Only when the platform signs repeatedly with the same basename $\mathsf{bsn} \neq \perp$, it will be clear that the resulting signatures were created by the same platform, which can be publicly tested via a (deterministic) link algorithm.

The desirable security properties of a DAA scheme can be described informally as follows:

Correctness: When an honest platform generates a signature on message m and basename bsn, an honest verifier will accept this signature. Furthermore, two signatures σ_1 and σ_2 generated by the same honest platform using the same basename $\mathsf{bsn} \neq \perp$ should link according to the link algorithm. To an honest verifier, it should not matter in which order two signatures are supplied when testing their linkability.

Anonymity: An adversary that is given two signatures, w.r.t. two different basenames or $\mathsf{bsn} = \perp$, cannot distinguish whether both signatures were

created by one honest platform, or whether two different honest platforms created the signatures. We require this property to hold even when the issuer is corrupt.

Unforgeability: No adversary can produce more unlinkable signatures than he has corrupt TPMs and cannot produce a signature that identifies to an honest TPM who never took part in its signing. In particular, this captures the scenario where a platform with an honest TPM and corrupt host cannot produce a signature attesting to the fact that the platform is in the 'right' configuration. This property only holds when the issuer is honest.

Non-frameability: No adversary can create signatures on a message m w.r.t. basename bsn that links to a signature created by an honest platform, when this honest platform never signed m w.r.t. bsn. We require this property to hold even when the issuer is corrupt.

3.3.1 *Syntax*

We begin by providing the syntax of DAA. The following syntax represents the algorithms run by the parties involved in a DAA scheme: an *issuer* $\mathcal{I}$ which controls who can obtain signing credentials; a set of *platforms* each comprised of a *TPM* and *Host*, and a set of verifiers who may request attestations from a platform. Formally, a DAA scheme DAA consists of the following (interactive) PPT algorithms:

$\mathsf{Setup}(1^\lambda) \to \mathsf{sp}$: The set-up algorithm takes as input the security parameter 1^λ and outputs a description of the system parameters sp.

$\mathsf{TPM.Kg}(\mathsf{sp}) \to (\mathsf{esk}, \mathsf{epk})$: The TPM key generation algorithm takes as input the system parameters sp and outputs an endorsement secret/public key pair $(\mathsf{esk}, \mathsf{epk})$ for a TPM.

$\mathsf{I.Kg}(\mathsf{sp}) \to (\mathsf{isk}, \mathsf{ipk}, \mathsf{ML})$: The issuer key generation algorithm takes as input the system parameters sp and outputs an issuer secret/public key pair $(\mathsf{isk}, \mathsf{ipk})$ for the issuer $\mathcal{I}$ and initialises an empty members list ML.

$\mathsf{IpkVf}(\mathsf{ipk}, \mathsf{sp}) \to 1/0$: The issuer public key verification algorithm takes as input the issuer's public key and the system parameters and outputs 1 or 0 to indicate accept or reject.

$\langle \mathsf{JoinTPM}(\mathsf{esk}), \mathsf{JoinH}(\mathsf{epk}, \mathsf{ipk}), \mathsf{Issue}(\mathsf{isk}, \mathsf{epk}, \mathsf{ML})\rangle$: This is an interactive join protocol between a TPM, host, and issuer. At the end of the interactive protocol, the algorithms output a TPM signing key sk, a credential cre, and an updated members list $\mathsf{ML} \leftarrow \mathsf{ML} \cup \{\mathsf{epk}\}$, respectively.

$\langle \mathsf{SignTPM}(\mathsf{sk}, \mathsf{m}, \mathsf{bsn}), \mathsf{SignH}(\mathsf{cre}, \mathsf{m}, \mathsf{bsn})\rangle$ This is an interactive sign protocol between a TPM with signing key sk, message m, and a basename bsn and host with credential cre, message m, and basename bsn as inputs. At the end of the interactive protocol, the algorithms output $\perp$ and σ, respectively.

$\mathsf{Verify}(\mathsf{ipk}, \sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{RL}) \rightarrow 1/0$: The deterministic verification algorithm takes as input the issuer's public key ipk, a signature σ, a message m, a basename bsn, and a revocation list RL, and returns 1 or 0 to indicate accept or reject.

$\mathsf{Link}(\mathsf{ipk}, \sigma, \sigma', \mathsf{m}, \mathsf{m}', \mathsf{bsn}) \rightarrow 1/0$: The deterministic linking algorithm takes as input the issuer's public key ipk, two signatures σ, σ' w.r.t two messages m, m', and a common basename bsn. The algorithm returns 1 or 0 to indicate accept or reject.

The correctness and security requirements for DAA will be formalized via an experiment between an adversary and a challenger in the following Section 3.3.3. Below, we provide some details on the above syntax so as to help understand DAA and the following sections better. We present them in a sequence or remarks.

Remark 3.3.1 (Significance of endorsement keys)**.** *The endorsement keys* $(\mathsf{esk}, \mathsf{epk})$ *of the TPM are mainly used to create an authenticated channel between a TPM and the issuer during the join protocol. This is necessary since the TPM can only communicate with the issuer via the (possibly malicious) host. However, as in other advanced signature schemes such as group signatures [BMW03, BSZ05], this will be implicit in the scheme for simplicity.*

Remark 3.3.2 (Significance of members list ML)**.** *The issuer manages the members list* ML *created by the issuer key generation algorithm* $\mathsf{I.Kg}$*. The list* ML *keeps track of which TPMs have joined through the interactive join protocol. We will always assume membership management is done by including the endorsement public keys of the TPMs to* ML *and make it explicit in the syntax. This is how membership is managed in practice and makes use of the authentication that the endorsement keys allow.*

Remark 3.3.3 (Key-based revocation)**.** *Our syntax captures the notion of* key-based revocation*. As one of the popular methods in the literature, e.g., [BS04, CDL16] and preference of the TCG, we model* key-based revocation *where we include the signing secret keys* sk *of corrupted TPMs in* RL *to revoke signatures. More discussions will be provided in the following section.*

3.3.2 *Security Model*

Before we formally state our security definition, we define several oracles and auxiliary algorithms $\mathsf{Ext}, \mathsf{Identify}_{\mathsf{T}}$, and $\mathsf{Identify}$, which allow for a more succinct and intuitive description of our security definition.

ORACLE DESCRIPTIONS. Since DAA must capture various security notions such as anonymity, unforgeability, and non-frameability, it would be helpful to abstract what the adversary can do in each of these property-based security definitions in the form of oracle access. In this section, we define several oracles that help us simplify the property-based definition provided in Section 3.3.3.
As mentioned above, the oracles are an abstraction of the adversary's ability in the property-based security definition. For example, we may want an adversary to be able to obtain signatures from honest platforms. In the descriptions below, we omit the input protocol messages M_{in} for readability and adopt the same convention when referring to the oracles in the security games and proofs but note that they are included in the formal definitions of Figure A.3 in Appendix A.2.

The following global lists, initially set to empty, are maintained by the oracles: HT is a list of honest TPMs (whose host is corrupt); HP is a list of TPMs which are part of an entirely honest platform $\mathcal{P}$, i. e. the host is also honest; CP is a list of corrupt TPMs which are part of an entirely corrupt platform; ESK and EPK are lists of secret and public TPM endorsement keys respectively; HSK is a list of honest TPM signing keys; CRE is a list of signing credentials; SL is a list of signatures obtained from the SignO oracle; CL records the identity pair and list of basenames sent in queries to ChalO; TS is a list of communication transcripts of calls to the IssueO oracle; CSK is a list of corrupt TPM's secret keys extracted . We summarise these for reference in Table 3.1.

The following is an informal explanation of the seven oracles which will be used in our property-based definition. Note that some of the following oracles take an extra party parameter as input. The party parameter may take one of the values in $\{\mathcal{M}, \mathcal{P}\}$ (TPM or entire platform) and is used to

indicate the honest party which the oracle is to play the part of (where it is unclear).

List label	List description	Entry Format
HT	IDs of honest TPMs w/ corrupt host.	i
HP	IDs of honest TPMs w/ honest host i. e. honest platform.	i
CP	IDs of corrupt TPMs w/ corrupt host i. e. corrupt platform.	i
ESK	Secret TPM endorsement keys.	esk_i
EPK	Public TPM endorsement keys.	epk_i
HSK	Honest TPM (secret) signing keys.	sk_i
CSK	Corrupt TPM signing keys (extracted)	sk_i
CRE	Signing credentials.	cre_i
SL	Sig. queries made to SignO.	$(i, \mathsf{m}, \mathsf{bsn})$
CL	Queries made to ChalO.	(i_0, i_1, bsn).
TS	Transcript of calls to IssueO.	Scheme dependent

TABLE 3.1: Global lists maintained by oracles.

AddHonO$(\cdot; \mathsf{party})$: The *add honest party* oracle on input a TPM identity i, creates an honest TPM $\mathcal{M}_i$ (if $\mathsf{party} = \mathcal{M}$) or a platform containing TPM $\mathcal{M}_i$ (if $\mathsf{party} = \mathcal{P}$). It generates a pair of TPM endorsement keys $(\mathsf{esk}, \mathsf{epk})$ for the TPM. Finally, it returns the TPM's public endorsement key epk.

AddCrptO$(\cdot, \mathsf{epk}; \mathsf{party})$: The *add corrupt party* oracle on input a TPM identity i and an arbitrary string epk, creates a corrupt TPM $\mathcal{M}_i$ (if $\mathsf{party} = \mathcal{M}$) or platform containing TPM $\mathcal{M}_i$ (if $\mathsf{party} = \mathcal{P}$) and sets the TPM's public endorsement key as epk. It returns 1 to indicate completion of this task.

InitO$\check{}$P$(\cdot)$[1]: The *initialise platform* oracle on input a TPM identity i, creates a signing key sk and credential cre for the platform with TPM i. This is done by retrieving the TPM endorsement keys $(\mathsf{esk}, \mathsf{epk})$ and running the entire interactive join protocol on behalf of the TPM, host and issuer. The oracle returns 1 to indicate completion of this task.

JoinO$(\cdot; \mathsf{party})$: The *join party* oracle on input a TPM identity i, allows an adversary to perform the join protocol with an honest TPM $\mathcal{M}_i$ (if $\mathsf{party} = \mathcal{M}$) or a platform containing TPM $\mathcal{M}_i$ (if $\mathsf{party} = \mathcal{P}$). If

1 We note that this is a special oracle only used to define correctness of the DAA scheme.

$\mathsf{party} = \mathcal{M}$, then the oracle plays the role of the honest TPM $\mathcal{M}_i$ and executes the interactive join protocol with the adversary by running the $\mathsf{JoinTPM}$ algorithm. If $\mathsf{party} = \mathcal{P}$, then the oracle plays the role of the honest platform and executes the interactive join protocol with the adversary by running the $\mathsf{JoinTPM}$ and JoinH algorithms. The oracle returns 1 to indicate completion of this task.

$\mathsf{IssueO}(\cdot)$: The *issue* oracle on input a TPM identity i, allows an adversary to perform the join protocol with an issuer to obtain signing credentials. The oracle runs the Issue algorithm to create a credential cre for the platform with TPM i. If TPM i is corrupt, then the oracle stores the transcript $\mathsf{TS}[i]$ of the communication with the adversary. Finally, the oracle returns cre.

$\mathsf{SignO}(\cdot, \cdot, \cdot, \cdot; \mathsf{party})$: The *signing* oracle on input a TPM identity i, a message m, a basename bsn, generates a signature made by an honest TPM $\mathcal{M}_i$ (if $\mathsf{party} = \mathcal{M}$) or a platform containing TPM $\mathcal{M}_i$ (if $\mathsf{party} = \mathcal{P}$). If $\mathsf{party} = \mathcal{M}$, the oracle adds an entry $(i, \mathsf{m}, \mathsf{bsn}, *)$ to the signing queries list SL and returns 1 to indicate completion of this task. Here, '$*$' captures the fact that the TPM (played by the oracle) does *not* see the final signature. If $\mathsf{party} = \mathcal{P}$, the oracle adds an entry $(i, \mathsf{m}, \mathsf{bsn}, \sigma)$ to SL and returns the signature σ.

$\mathsf{ChalO}(\cdot, \cdot, \cdot, \cdot; b)$: The *challenge* oracle on input two TPM identities i_0 and i_1, a message m and basename bsn, generates a signature on message m and basename bsn under identity i_b. The oracle returns the signature σ.

$\mathsf{IdentifyO}(\cdot, \cdot, \cdot, \cdot)$: The *identify* oracle, on input a TPM identity i, a signature σ, message m, and basename bsn, runs $\mathsf{Identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{HSK}[i])$[2]. The oracle returns the resulting output bit.

NON-CONCURRENT ORACLE ACCESS. Building on the intuitive explanation of the oracles, we first present in Figure 3.2 the formal oracle definitions with *non-concurrent* access. We emphasize that this definition is not used for our UC implication, but rather provided for the purpose of conveying intuition of the oracle roles. This definition will be a stepping stone towards *concurrent* oracle definitions. In Figure 3.2, we assume that the adversary cannot interleave the interactive protocols run between the JoinO, IssueO,

2 This algorithm determines whether a given signature was signed using a given key. See Definition 3.3.4 for a formal definition of this algorithm.

and SignO oracles. We also assume that algorithms JoinTPM, JoinH, Issue, and SignTPM output some special symbol indicating failure of execution and say that the algorithm "terminates" when it outputs a valid value.

AddHonO$(i;$ party$)$
- **If** $i \in \mathsf{HT} \cup \mathsf{HP} \cup \mathsf{CT} \cup \mathsf{CP}$, then **Return** $\perp$.
- **If** party $= \mathcal{M}$, then $\mathsf{HT} \leftarrow \mathsf{HT} \cup \{i\}$.
- **If** party $= \mathcal{P}$, then $\mathsf{HP} \leftarrow \mathsf{HP} \cup \{i\}$.
- $(\mathsf{esk}, \mathsf{epk}) \leftarrow \mathsf{TPM.Kg}(\mathsf{sp})$.
- $(\mathsf{ESK}[i], \mathsf{EPK}[i]) := (\mathsf{esk}, \mathsf{epk})$.
- **Return** $\mathsf{EPK}[i]$.

AddCrptO$(i, \mathsf{epk};$ party$)$
- **If** $i \in \mathsf{HT} \cup \mathsf{HP} \cup \mathsf{CT} \cup \mathsf{CP}$, then **Return** $\perp$.
- **If** party $= \mathcal{M}$, then $\mathsf{CT} \leftarrow \mathsf{CT} \cup \{i\}$.
- **If** party $= \mathcal{P}$, then $\mathsf{CP} \leftarrow \mathsf{CP} \cup \{i\}$.
- $(\mathsf{ESK}[i], \mathsf{EPK}[i]) := (\perp, \mathsf{epk})$.
- **Return** accept.

JoinO$(i;$ party$)$
– **If** party $= \mathcal{P}$:
- **If** $i \notin \mathsf{HP}$ or $\mathsf{CRE}[i] \neq \perp$, then **Return** $\perp$.
- Run $\langle \mathsf{JoinTPM}(\mathsf{ESK}[i]), \mathsf{JoinH}(\mathsf{EPK}[i], \mathsf{ipk}), \star \rangle$, where $\star$ is an arbitrary algorithm run by the adversary.
- **If** JoinTPM terminates, then $\mathsf{HSK}[i] := \mathsf{sk}_i$.
- **If** JoinH terminates, then $\mathsf{CRE}[i] := \mathsf{cre}_i$.
- **Return** 1.

– **If** party $= \mathcal{M}$:
- **If** $i \notin \mathsf{HT}$, then **Return** $\perp$.
- Run $\langle \mathsf{JoinTPM}(\mathsf{ESK}[i]), \star, \star \rangle$, where $\star$ is an arbitrary algorithm run by the adversary.
- **If** JoinTPM terminates, $\mathsf{HSK}[i] := \mathsf{sk}_i$.
- **Return** 1.

IssueO(i)
– **If** $i \in \mathsf{HT}$:
- Run $\langle \mathsf{JoinTPM}(\mathsf{ESK}[i]), \star, \mathsf{Issue}(\mathsf{isk}, \mathsf{EPK}[i], \mathsf{ML}) \rangle$, where $\star$ is an arbitrary algorithm run by the adversary.
- **If** Issue terminates, then $\mathsf{CRE}[i] := \mathsf{cre}_i$.
- **Return** $\mathsf{CRE}[i]$.

– **If** $i \in \mathsf{CP}$:
- Run $\langle \star, \star, \mathsf{Issue}(\mathsf{isk}, \mathsf{EPK}[i], \mathsf{ML}) \rangle$, where $\star$ is an arbitrary algorithm run by the adversary.
- **If** Issue terminates, then $\mathsf{CRE}[i] := \mathsf{cre}_i$, and record the view of Issue in the transcript list $\mathsf{TS}[i]$.
- **Return** $\mathsf{CRE}[i]$.

InitO$^{\mathsf{ver}}$P(i)
- **If** $i \notin \mathsf{HP}$, then **Return** $\perp$.
- Run $\langle \mathsf{JoinTPM}(\mathsf{ESK}[i]), \mathsf{JoinH}(\mathsf{EPK}[i], \mathsf{ipk}), \mathsf{Issue}(\mathsf{isk}, \mathsf{EPK}[i], \mathsf{ML}) \rangle$ to obtain $\mathsf{HSK}[i]$, $\mathsf{CRE}[i]$, and an updated members list ML respectively.
- **Return** 1.

SignO$(i, \mathsf{m}, \mathsf{bsn};$ party$)$
– **If** party $= \mathcal{P}$:
- **If** $i \notin \mathsf{HP}$ or $\mathsf{CRE}[i] = \perp$ or $\mathsf{HSK}[i] = \perp$, then **Return** $\perp$.
- Run $\langle \mathsf{SignTPM}(\mathsf{HSK}[i], \mathsf{m}, \mathsf{bsn}), \mathsf{SignH}(\mathsf{CRE}[i], \mathsf{m}, \mathsf{bsn}) \rangle$ to obtain signature σ.
- $\mathsf{SL} \leftarrow \mathsf{SL} \cup \{(i, \mathsf{m}, \mathsf{bsn}, \sigma)\}$.
- **Return** σ.

– If party $= \mathcal{M}$:
- **If** $i \notin \mathsf{HT}$ or $\mathsf{HSK}[i] = \perp$, then **Return** $\perp$.
- Run $\langle \mathsf{SignTPM}(\mathsf{HSK}[i], \mathsf{m}, \mathsf{bsn}), \star \rangle$, where $\star$ is an arbitrary algorithm run by the adversary.
- **If** SignTPM terminates, then $\mathsf{SL} \leftarrow \mathsf{SL} \cup \{(i, \mathsf{m}, \mathsf{bsn}, \perp)\}$.
- **Return** 1.

ChalO$(i_0, i_1, \mathsf{bsn}, \mathsf{m}; b)$
- **If** $i_{b'} \notin \mathsf{HP}$ or $\mathsf{CRE}[i_{b'}] = \perp$ or $\mathsf{HSK}[i_{b'}] = \perp$ for $b' \in \{0, 1\}$, then **Return** $\perp$.
- **If** $|\mathsf{CL}| \geq 1$ and $(i_0, i_1, *) \notin \mathsf{CL}$, then **Return** $\perp$.
- Run $\langle \mathsf{SignTPM}(\mathsf{HSK}[i_b], \mathsf{m}, \mathsf{bsn}), \mathsf{SignH}(\mathsf{CRE}[i_b], \mathsf{m}, \mathsf{bsn}) \rangle$ to obtain challenge signature σ.
- $\mathsf{CL} \leftarrow \mathsf{CL} \cup \{(i_0, i_1, \mathsf{bsn})\}$.
- **Return** σ.

IdentifyO$(i, \sigma, \mathsf{m}, \mathsf{bsn})$
- If $i \notin \mathsf{HT} \cup \mathsf{HP}$, then **Return** $\perp$.
- $b \leftarrow \mathsf{Identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{HSK}[i])$.
- **Return** b.

FIGURE 3.2: Oracles with non-concurrent access.

CONCURRENT ORACLE ACCESS. While the property-based definition defined through the oracles with non-concurrent access is very handy and intuitive, unfortunately, it is not enough to imply UC-security. We thereofre generalize the oracles in Figure 3.2 to capture concurrent access by using the notion of stateful oracles as done by Bellare et al. [BSZ05] and Bootle et al. [BCC+16] for groups signatures. The formal description is provided in Appendix A.2, Figure A.3. As can be seen there, the adversary can interleave oracle calls and interact with the oracles in a concurrent fashion; this

is captured by slightly modifying the interactive oracles JoinO, IssueO, and SignO to take an extra input called the protocol message M_{in}. More details will be provided in Appendix A.2. Although these may look quite different from Figure 3.2, we emphasize that in essence they both capture the aforementioned intuitive oracle description, and the only difference lies in how we model concurrent access.

AUXILIARY ALGORITHMS. To argue some of the security notions of DAA, the challenger will need to check, given a signature σ, message m, and basename bsn from an adversary, whether this signature was produced by a TPM who has joined. Put differently, the challenger would need to be able to check if the signature was signed with a key that has been used earlier in a join session (and thus on which signing credentials have been issued). As we detailed in the introduction, we make this possible by defining three auxiliary algorithms $\mathsf{Ext}, \mathsf{Identify}_\mathsf{T}$. and $\mathsf{Identify}$. Algorithm Ext which, given a join transcript and a set of trap-doored parameters, can extract the TPM secret key used in that transcript. Then, given a TPM key and a join transcript, we also define an $\mathsf{Identify}_\mathsf{T}$ algorithm which determines whether the key was used in a given transcript or not. Finally, a third algorithm $\mathsf{Identify}$ is used to determine whether a given signature was generated using a given TPM key. The interdependencies of these three algorithms defined in Definition 3.3.4 provide us a framework for determining whether a signature was created using a TPM who has joined or not.

In the following definition we denote the space of all TPM secret keys by $\mathcal{K}$.

Definition 3.3.4. *For any DAA scheme* DAA*, we require the existence of auxiliary algorithms* $\mathsf{SimSetup}$*, and deterministic algorithms* Ext*,* $\mathsf{Identify}$*, and* $\mathsf{Identify}_\mathsf{T}$ *with the following properties where the oracles provided to the adversary* $\mathcal{A}$ *are defined in Appendix A.2, Figure A.3:*

1. *For all PPT adversaries* $\mathcal{A}$*, if we run* $(\mathsf{sp}, \mathsf{td}) \leftarrow \mathsf{SimSetup}(1^\lambda)$ *and* $\mathsf{sp}' \leftarrow \mathsf{Setup}(1^\lambda)$*, then we have*

$$\left|\Pr[\mathcal{A}(1^\lambda, \mathsf{sp}) \to 1] - \Pr[\mathcal{A}(1^\lambda, \mathsf{sp}') \to 1]\right| = \mathsf{negl}(\lambda),$$

   where the probability is taken over the randomness used by $\mathsf{SimSetup}$ *and* Setup*.*

2. *For all* ipk *such that* $\mathsf{IpkVf}(\mathsf{ipk}, \mathsf{sp}) = 1$*, where* $\mathsf{sp} \leftarrow \mathsf{Setup}(1^\lambda)$ *or* $\mathsf{sp} \leftarrow \mathsf{SimSetup}(1^\lambda)$*, given a signature* σ *on some message* m *and basename*

bsn *such that* $\mathsf{Verify}(\mathsf{ipk}, \sigma, \mathsf{m}, \mathsf{bsn}) = 1$*, there exists a unique* $\mathsf{sk} \in \mathcal{K}$ *such that* $\mathsf{Identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}) = 1$.

3. *For all* $(\mathsf{sp}, \mathsf{td}) \in \mathsf{SimSetup}(1^\lambda)$*,* $(\mathsf{ipk}, \mathsf{isk}, \mathsf{ML}) \in \mathsf{I.Kg}(\mathsf{sp})$*, and PPT adversary* $\mathcal{A}$ *with oracle access to* $(\mathsf{AddHonO}(\cdot; \mathcal{M})$*,* $\mathsf{AddCrptO}(\cdot, \cdot; \mathcal{P})$*,* $\mathsf{JoinO}(\cdot; \mathcal{M})$*,* $\mathsf{SignO}(\cdot, \cdot, \cdot; \mathcal{M})$*,* $\mathsf{IssueO}(\cdot)$*,* $\mathsf{IdentifyO}(\cdot, \cdot, \cdot, \cdot))$*, and any* i *such that* $\mathsf{CRE}[i] \neq \perp$ *we have:*
   - $\mathsf{Identify}_\mathsf{T}(\mathsf{TS}[i], \mathsf{sk}) = 1$*, where* $\mathsf{sk} \leftarrow \mathsf{Ext}(\mathsf{sp}, \mathsf{td}, \mathsf{ipk}, \mathsf{isk}, \mathsf{TS}[i])$*, and*
   - *there is a unique* $\mathsf{sk} \in \mathcal{K}$ *such that* $\mathsf{Identify}_\mathsf{T}(\mathsf{TS}[i], \mathsf{sk}) = 1$.

Since Ext will be implemented in the ROM we require online-extractability here. More discussion is provides in Appendix **??**.

DEFINING UNFORGEABILITY WITH AUXILIARY ALGORITHMS. Although the auxiliary algorithms are vital to DAA as a whole, we believe the use-case of these in the unforgeability game highlights their importance in the most intuitive manner. In the unforgeability game, we allow the adversary to make join requests to the honest issuer (played by the challenger) on behalf of a corrupt host or entirely corrupt platform. Informally, the adversary should win the unforgeability game if it is able to output a signature which wasn't signed using any TPM that has joined or that originates from an honest TPM who never signed it. The question is, how can the challenger check this winning condition in a well-defined manner.

Below, we explain how the challenger does this using the auxiliary algorithms. The challenger first extracts (using Ext) the TPM secret key from the join transcript of each corrupt platform that joins. By definition of $\mathsf{Identify}_\mathsf{T}$, we know that an extracted key is exactly the one used in the given join transcript. The challenger then runs $\mathsf{Identify}$ with the given signature on each of the honest TPM signing keys and also the extracted secret keys (for the corrupt TPMs) in turn. If the signature doesn't identify to any of these then we say that the adversary wins the game. This precisely captures the property that no adversary can make valid signatures without using a key for which signing credentials have been issued. Finally, the adversary also wins if the signature identifies to an honest TPM who never signed it.

Readers familiar with group signatures may notice the similarity of this notion to traceability. We note that the traceability game in group signatures does not require these complex algorithms since the winning condition can be efficiently checked by the challenger owing to the fact that the challenger knows the identity of all users in the group.

KEY-BASED REVOCATION. To instantiate secret key-based revocation we assume that the $\mathsf{Identify}$ algorithm is run as a subroutine of Verify to check that the given signature doesn't identify to any of the compromised TPM keys listed in the revocation list RL. Precisely, in order to verify a signature σ with respect to a message m, basename bsn, and revocation list RL one does the following.

1. Set $b \leftarrow \mathsf{Verify}(\sigma, \mathsf{m}, \mathsf{bsn}, \emptyset)$, where $\emptyset$ is the empty list.
2. For each $\mathsf{sk}_i \in \mathsf{RL}$, set $b_i \leftarrow \mathsf{Identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}_i)$.
3. If $b = 0$ or $\exists\, i$ such that $b_i = 1$, then verification fails.
4. Else, verification passes.

We note that this 'hard-coding' of the Verify algorithm might appear quite restrictive. However, this assumption allows for a smooth proof of our UC implication Theorem 3.4.1. Whilst this may be an artifact of the proof, we don't consider it to be a strong assumption since all existing secure DAA schemes satisfy this. We further note that this assumption naturally imposes the desired security properties on the revocation process. For example, a signature cannot pass verification if it identifies to some key on the revocation list. Furthermore, a previously invalid signature cannot be made valid by adding extra keys to the revocation list. Finally, if one only requires stand-alone security, i.e., security which does not necessarily imply UC security, it may be more intuitive and allow for a more flexible scheme design if one formulated the above revocation guarantees as an experiment instead of hard-coding Verify in this way.

3.3.3 *Security Definitions.*

We are finally ready to define the properties of a secure DAA scheme: correctness, anonymity, unforgeability, and non-frameability. The experiments referred to in the following definitions are presented in Figure 3.3. We try to provide intuition and discussions on the reasoning for our definition.

CORRECTNESS. We require that signatures produced by honest platforms are accepted by the Verify algorithm. In addition, we require that two signatures produced by the same platform with the same basename $\mathsf{bsn} \neq \perp$ are linked w.r.t. Link. Finally, Link is required to be symmetric.

Definition 3.3.5. *A DAA scheme* DAA *is said to be correct if for all* $\lambda \in \mathbb{N}$*, the advantage,*

$$\mathsf{Adv}^{Corr}_{\mathsf{DAA},\mathcal{A}}(\lambda) := Pr[\mathsf{Exp}^{\mathrm{Corr}}_{\mathsf{DAA},\mathcal{A}}(\lambda) = 1]$$

is negligible (in λ*) for all PPT adversaries* $\mathcal{A}$*.*

ANONYMITY. It is essential that attestations made by a platform do not reveal anything about its identity. Precisely, we require that no adversary should be able to determine which of two honest platforms has produced a given signature, even if he controls the issuer. Note that anonymity can only be preserved for fully honest platforms. Clearly, a corrupt host can append identifying information to any signature to reveal the platform's identity since it is the only one able to interact directly with the outside world.

Experiment: $\mathsf{Exp}^{\mathrm{Corr}}_{\mathsf{DAA},\mathcal{A}}(\lambda)$
- $\mathsf{sp} \leftarrow \mathsf{Setup}(1^\lambda)$; $\mathsf{HP}, \mathsf{EPK}, \mathsf{ESK}, \mathsf{CRE}, \mathsf{HSK} := \emptyset$.
- $(\mathsf{isk}, \mathsf{ipk}, \mathsf{ML}) \leftarrow \mathsf{I.Kg}(\mathsf{sp})$
- **If** $\mathsf{IpkVf}(\mathsf{ipk}, \mathsf{sp}) = 0$, then **Return** 0.
- $(i, \mathsf{m}_0, \mathsf{m}_1, \mathsf{bsn}) \leftarrow \mathcal{A}(\mathsf{ipk}; \mathsf{AddHonO}(\cdot;\mathcal{P}), \mathsf{InitO}^{\cdot}\mathsf{P}(\cdot))$
- $\sigma_b \leftarrow \mathsf{SignO}(i, \mathsf{m}_b, \mathsf{bsn}; \mathcal{P})$ for $b \in \{0, 1\}$
- **If** $\mathsf{Verify}(\mathsf{ipk}, \sigma_0, \mathsf{m}_0, \mathsf{bsn}, \emptyset) = 0$ or $\mathsf{Verify}(\mathsf{ipk}, \sigma_1, \mathsf{m}_1, \mathsf{bsn}, \emptyset) = 0$, then **Return** 0.
- **If** $\mathsf{bsn} \neq \perp$ and $\mathsf{Link}(\mathsf{ipk}, \sigma_0, \sigma_1, \mathsf{m}_0, \mathsf{m}_1, \mathsf{bsn}) = 0$ or $\mathsf{Link}(\mathsf{ipk}, \sigma_1, \sigma_0, \mathsf{m}_0, \mathsf{m}_1, \mathsf{bsn}) = 0$, then **Return** 1.
- **Return** 0.

$\mathsf{Exp}^{\mathrm{Anon-b}}_{\mathsf{DAA},\mathcal{A}}(\lambda)$
- $\mathsf{sp} \leftarrow \mathsf{Setup}(1^\lambda)$; $\mathsf{HP}, \mathsf{EPK}, \mathsf{ESK}, \mathsf{CRE}, \mathsf{HSK}, \mathsf{SL}, \mathsf{CL} := \emptyset$.
- $(\mathsf{ipk}, \mathsf{st}) \leftarrow \mathcal{A}(\mathsf{sp})$.
- **If** $\mathsf{IpkVf}(\mathsf{ipk}, \mathsf{sp}) = 0$, then **Return** 0.
- $b^* \leftarrow \mathcal{A}(\mathsf{sp}, \mathsf{st}; \mathsf{AddHonO}(\cdot;\mathcal{P}), \mathsf{JoinO}(\cdot;\mathcal{P}), \mathsf{SignO}(\cdot,\cdot,\cdot;\mathcal{P}), \mathsf{ChalO}(\cdot,\cdot,\cdot,\cdot;b))$.
- **If** $\exists(i, \mathsf{m}, \mathsf{bsn}, \sigma) \in \mathsf{SL}$ s.t. $\mathsf{bsn} \neq \perp$ and $(i, *, \mathsf{bsn}) \in \mathsf{CL}$ or $(*, i, \mathsf{bsn}) \in \mathsf{CL}$ then **Return** 0.
- **If** $b^* = b$, then **Return** 1.
- **Return** 0.

$\mathsf{Exp}^{\mathrm{UF}}_{\mathsf{DAA},\mathcal{A}}(\lambda)$
- $(\mathsf{sp}, \mathsf{td}) \leftarrow \mathsf{SimSetup}(1^\lambda)$; $\mathsf{HT}, \mathsf{CP}, \mathsf{EPK}, \mathsf{ESK}, \mathsf{CRE}, \mathsf{HSK}, \mathsf{SL} := \emptyset$.
- $(\mathsf{ipk}, \mathsf{isk}, \mathsf{ML}) \leftarrow \mathsf{I.Kg}(\mathsf{sp})$.
- $(\sigma, \mathsf{m}, \mathsf{bsn}) \leftarrow \mathcal{A}(\mathsf{sp}, \mathsf{ipk}; \mathsf{AddHonO}(\cdot;\mathcal{M}), \mathsf{AddCrptO}(\cdot,\cdot;\mathcal{P}), \mathsf{JoinO}(\cdot;\mathcal{M}), \mathsf{IssueO}(\cdot), \mathsf{SignO}(\cdot,\cdot,\cdot;\mathcal{M}), \mathsf{IdentifyO}(\cdot,\cdot,\cdot,\cdot))$.
- **If** $\mathsf{Verify}(\mathsf{ipk}, \sigma, \mathsf{m}, \mathsf{bsn}, \emptyset) = 0$, then **Return** 0.
- For all $i \in \mathsf{CP}$, run $\mathsf{sk}_i \leftarrow \mathsf{Ext}(\mathsf{sp}, \mathsf{td}, \mathsf{ipk}, \mathsf{isk}, \mathsf{TS}[i])$, and set $\mathsf{sk}_i \leftarrow \mathsf{CSK}[i]$.
- **If** $\exists i \in \mathsf{HT}$ s.t $\mathsf{Identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{HSK}[i]) = 1$ and $(i, \mathsf{m}, \mathsf{bsn}, \perp) \notin \mathsf{SL}$, then **Return** 1.
- **If** $\forall i \in \mathsf{HT}$, $\mathsf{Identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{HSK}[i]) = 0$ and $\forall j \in \mathsf{CP}$, $\mathsf{Identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{CSK}[j]) = 0$, then **Return** 1.
- **Return** 0.

$\mathsf{Exp}^{\mathrm{NF}}_{\mathsf{DAA},\mathcal{A}}(\lambda)$
- $\mathsf{sp} \leftarrow \mathsf{Setup}(1^\lambda)$; $\mathsf{HP}, \mathsf{EPK}, \mathsf{ESK}, \mathsf{CRE}, \mathsf{HSK}, \mathsf{SL} := \emptyset$.
- $(\mathsf{ipk}, \mathsf{state}) \leftarrow \mathcal{A}(\mathsf{sp})$.
- **If** $\mathsf{IpkVf}(\mathsf{ipk}, \mathsf{sp}) = 0$, then **Return** 0.
- $(i, \sigma, \mathsf{m}, \mathsf{bsn}) \leftarrow \mathcal{A}(\mathsf{sp}, \mathsf{state}; \mathsf{AddHonO}(\cdot;\mathcal{P}), \mathsf{JoinO}(\cdot;\mathcal{P}), \mathsf{SignO}(\cdot,\cdot,\cdot;\mathcal{P}), \mathsf{IdentifyO}(\cdot,\cdot,\cdot,\cdot))$.
- **If** $\mathsf{Verify}(\mathsf{ipk}, \sigma, \mathsf{m}, \mathsf{bsn}, \emptyset) = 0$, then **Return** 0.
- **If** $i \notin \mathsf{HP}$ or $(i, \mathsf{m}, \mathsf{bsn}, *) \in \mathsf{SL}$, then **Return** 0.
- **If** $\exists\,(i, \mathsf{m}^*, \mathsf{bsn}, \sigma^*) \in \mathsf{SL}$ s.t. $\mathsf{Link}(\sigma, \sigma^*, \mathsf{m}, \mathsf{m}^*, \mathsf{bsn}) = 1$, then **Return** 1.
- **If** $\mathsf{bsn} = \perp$, $\mathsf{Identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{HSK}[i]) = 1$, and $(i, \mathsf{m}, \mathsf{bsn}, *) \notin \mathsf{SL}$ then **Return** 1.
- **Return** 0.

FIGURE 3.3: Security experiments for property-based definition of DAA.

The adversary's aim is to guess which of two TPMs (of his choice) has signed a chosen message and basename. The powers afforded to $\mathcal{A}$ include control of the issuer and so his first action must be to output the issuer's public key. Secondly, he is given oracle access allowing him to create honest platforms which he can instruct to join and produce signatures. $\mathcal{A}$ may also call the challenge oracle ChalO on two honest platforms, and a sequence of message and a basename pairs on which he receives signatures signed under one of those identities (fixed for all queries). He must then output a bit b^* to indicate which of the two TPMs he thinks has signed the signatures returned by ChalO. $\mathcal{A}$ wins the game if he guesses correctly. We indicate the shared state of the two adversarial actions using st. Note that we do not explicitly give $\mathcal{A}$ the power to create corrupt TPMs and platforms since he controls the issuer and can therefore simulate corrupt TPMs and platforms internally.

Since we instantiate our model with the $\mathsf{Identify}$ algorithm which tells us whether a given signature was created under a given secret key, any user can trivially determine whether a signature was created under their own secret key. Furthermore, $\mathcal{A}$ cannot query both the signing oracle and the challenger with the same identity and basename. Otherwise, $\mathcal{A}$ could simply win the game by using the Link algorithm.

FLAW IN PREVIOUS ANONYMITY GAMES. We present a toy DAA scheme which is clearly not anonymous but that is provably secure according to existing property-based anonymity definitions. In particular, we exploit the fact that in [BFG+11], the adversary is allowed only a single call to the challenge oracle. Notably, due to the linkability feature of DAA, there cannot exist a hybrid argument showing that one challenge query provides the same guarantees as allowing for multiple calls as with symmetric encryption [BDJR97].

Consider a scheme in which the first signature of a platform is appended with $\mathsf{H}(i) \oplus \mathsf{r}$ where H is some secure hash function and $\mathsf{r} := \mathsf{H}(\mathsf{sk}, \mathsf{bsn})$ is the hash of the TPM secret key and chosen basename. All subsequent signatures are simply appended with r. Clearly the first signature is fully anonymous but anyone with a second signature can trivially win the anonymity experiment simply by computing $\mathsf{H}(i_0)$, $\mathsf{H}(i_1)$ and comparing the values to $(\mathsf{H}(i_b) \oplus \mathsf{r}) \oplus \mathsf{r}$.

This weakness of previous anonymity experiments emerges when aiming to prove Theorem 3.4.1 since the UC definition implicitly allows an adversary to obtain multiple challenge signatures relating to the embedded hard problem. Whilst this is a serious flaw in previous definitions, we can apply

the simple fix of allowing an adversary multiple challenge queries in the new model.

Definition 3.3.6 (Anonymity of DAA)**.** *A DAA scheme* DAA *is said to be anonymous if and only if for all* $\lambda \in \mathbb{N}$*, the advantage*

$$\mathsf{Adv}^{\text{Anon}}_{\mathsf{DAA},\mathcal{A}}(\lambda) := |\Pr[\mathsf{Exp}^{\text{Anon-0}}_{\mathsf{DAA},\mathcal{A}} = 1] - \Pr[\mathsf{Exp}^{\text{Anon-1}}_{\mathsf{DAA},\mathcal{A}} = 1]|$$

is negligible (in λ*) for all PPT adversaries* $\mathcal{A}$*.*

UNFORGEABILITY. Unforgeability ensures that no adversary can produce more unlinkable signatures than he has corrupt TPMs and cannot produce a signature that identifies to an honest TPM who never took part in its signing. This property is unique in that it can only hold when the issuer is honest, else a corrupt issuer simply creates dummy users which cannot be identified by the challenger. Unforgeability must be maintained for platforms containing an honest TPM, even if the host is corrupt and says that no adversary can create a signature that identifies to an honest TPM who never signed it *or* that does not identify to any TPM (honest or corrupt) that has joined.

The challenger $\mathcal{C}$ begins by generating the issuer's secret and public key. Next, the adversary $\mathcal{A}$ makes a round of oracle queries. These allow him to create honest TPMs, create corrupt platforms, join using an honest TPM or corrupt platform, use an honest TPM to create signatures, and determine whether a given signature was signed by a given TPM. The adversary must then output a signature on a message and basename of his choice. The Ext algorithm is used by the challenger to extract secret keys for each of the corrupt TPM's that $\mathcal{A}$ is using to join. The challenger then checks that the output signature is valid. Next, the challenger checks whether the signature identifies to any of the honest TPMs (for which $\mathcal{C}$ knows sk) but was never signed by them. If so, $\mathcal{A}$ wins the game. Finally, $\mathcal{A}$ also wins the game if the output signature does not identify to any of the TPMs that have joined (honest or corrupt).

After creating a corrupt TPM, an adversary can simulate the part of an honest host himself. Thus, where there is a corrupt TPM, we assume the host to be corrupt also. Similarly, when a TPM is honest we assume the host to be corrupt. This is also because we do not require unforgeability to hold for a corrupt TPM and honest host.

Definition 3.3.7 (Unforgeability of DAA)**.** *A DAA scheme* DAA *is said to be unforgeable if and only if for all* $\lambda \in \mathbb{N}$*, the advantage,*

$$\mathsf{Adv}^{UF}_{\mathsf{DAA},\mathcal{A}}(\lambda) := Pr[\mathsf{Exp}^{\mathrm{UF}}_{\mathsf{DAA},\mathcal{A}}(\lambda) = 1]$$

is negligible (in λ*) for all PPT adversaries* $\mathcal{A}$*.*

Remark 3.3.8 (Absence of revocation list in unforgeability experiment)**.** *We note that our unforgeability experiment makes no mention of revocation lists.*

This might seem strange to the reader, however the security properties relating to the revocation process are covered by the assumption that Verify *is hard-coded to internally run* Identify *on each secret key of* RL*. Indeed, this is proven formally via our UC implication Theorem 3.4.1. One can see this intuitively in the following way. Consider a modified unforgeability experiment where winning conditions depend on a non-empty revocation list as done in security experiments of group signature schemes such as [BS04, NF05]. It is easy to see that winning outputs of such an experiment would also constitute a winning output of our current experiment* assuming *that* Verify *is hard-coded in this way.*

NON-FRAMEABILITY. This ensures that one cannot create signatures on messages that an honest platform never signed but that link to signatures the platform did create. As in anonymity, we allow the issuer to be corrupt and can only guarantee security for an entirely honest platform. In particular, honest TPMs with a corrupt host cannot be protected from framing. This is because the host controls the output of signatures and could also run a corrupt TPM that had joined and and use the corrupt TPM's key to create signatures instead of using the honest TPM's contribution. The resulting signatures can't be protected from framing since they use the corrupt TPM's signing key.

The adversary $\mathcal{A}$ first chooses the issuer's secret key and announces the corresponding public key since all further actions will depend on this value. Next, he can make queries to oracles that allow him to create honest platforms, join using an honest platform and request signatures from them, and determine whether a given signature was signed by a given TPM. He must then output a TPM identity i and a valid signature σ on a message m and basename bsn that isn't in the signing queries list. $\mathcal{A}$ wins the non-frameability game if the output identity is part of an honest platform and there exists a signature signed by that platform which links to the output signature. $\mathcal{A}$ can also win if the output signature, with $\mathsf{bsn} = \perp$, identifies

to some honest platform who never signed it.

Note that we do not explicitly allow $\mathcal{A}$ to create corrupt TPMs and platforms since he controls the issuer and can simulate these internally as in the anonymity game.

Definition 3.3.9 (Non-frameability of DAA)**.** *A DAA scheme* DAA *is said to be non-frameable if and only if for all* $\lambda \in \mathbb{N}$*, the advantage,*

$$\mathsf{Adv}^{NF}_{\mathsf{DAA},\mathcal{A}}(\lambda) := Pr[\mathsf{Exp}^{\mathrm{NF}}_{\mathsf{DAA},\mathcal{A}}(\lambda) = 1],$$

is negligible (in λ*) for all PPT adversaries* $\mathcal{A}$*.*

Remark 3.3.10 (Absence of revocation list in non-frameability experiment)**.** *As with our unforgeability experiment, we demonstrate that although our non-frameability experiment makes no mention of revocation, the assumption that* Verify *is hard-coded to run* $\mathsf{Identify}$ *on each value in* RL *covers all desirable properties relating to revocation within non-frameability. Indeed, this is proven via our UC implication Theorem 3.4.1. Below we give the intuition as to why this is true.*

Consider a modified non-frameability experiment in which the adversary may additionally output a revocation list RL *and we check that his signature is valid by running* $\mathsf{Verify}(\mathsf{ipk}, \sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{RL})$*. Note that this is what is done in the security experiments of group signatures with verifier local revocation [BS04, NF05]. One can easily show that if* $\mathcal{A}$ *can give a winning output in this modified experiment, it gives us one for the current experiment* assuming *that* Verify *is hard-coded as described.*

Now that we have formalised all security requirements of DAA we can define the overall DAA security. The definition acknowledges three extra assumptions. Firstly, the existence of the auxiliary algorithms $\mathsf{SimSetup}$, $\mathsf{Identify}$, $\mathsf{Identify}_\mathsf{T}$, and Ext defined in Definition 3.3.4. Secondly, that KBR is instantiated using a Verify algorithm that internally runs $\mathsf{Identify}$ on each key in the revocation list. Finally, we assume signatures link if and only if they identify to a common signing key. Including this last condition allows for much more succinct property definitions.

Definition 3.3.11 (DAA security)**.** *A DAA scheme* DAA *is said to be secure if and only if it is correct, anonymous, unforgeable, non-frameable, and the following hold:*

1. *There exist auxiliary algorithms* $\mathsf{SimSetup}$*,* $\mathsf{Ext}, \mathsf{Identify}_\mathsf{T}$*, and* $\mathsf{Identify}$ *as defined in Definition 3.3.4.*

2. *We assume that key-based revocation is instantiated by a verification algorithm that implicitly runs* $\mathsf{Identify}$ *on each secret key of the provided revocation list.*

3. *Given two valid signature-message-basename tuples* $(\sigma, \mathsf{m}, \mathsf{bsn})$ *and* $(\sigma', \mathsf{m}', \mathsf{bsn})$ *for a common basename* $\mathsf{bsn} \neq \perp$*, it holds that* $\mathsf{Link}(\sigma, \sigma', \mathsf{m}, \mathsf{m}', \mathsf{bsn}) = 1$ *if and only if there exists an input* sk *such that* $\mathsf{Identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}) = \mathsf{Identify}(\sigma', \mathsf{m}', \mathsf{bsn}, \mathsf{sk}) = 1$.

3.4 PROPERTY-BASED TO UC MODEL IMPLICATION

In order to prove the security of a protocol in the UC model one must first define an 'ideal functionality' for that protocol. This is a description of how the protocol acts if it were perfectly secure. When presented with a particular instantiation of the protocol one proves its security by showing that no PTT adversary (or environment) is able to distinguish between the ideal functionality and the given instance of the protocol. It is common to present a series of modifications to the protocol until we end up with the ideal functionality itself. One then argues the indistinguishability of each iteration from its predecessor by using any such distinguisher to break some assumed-to-be-hard problem. For a complete overview of the UC model we refer the reader to [Can01].

The ideal functionality for DAA, $\mathcal{F}_{\mathsf{daa}}$ was first presented in [CDL16]. This has proved to be a sound model of security, having given rise to several DAA schemes since its inception. Nevertheless, it is unwieldy to work with and requires arduous proofs that are often difficult to verify. Since our primary aim is to promote the use of an alternative security framework in the property-based world, we must be sure that our new definition achieves at least the same security guarantees as the existing UC definition. We dedicate this section to providing such an assurance.

We begin with a discussion of the UC definition from [CDL16] and highlighting a few redundancies in it.

REDUNDANCIES IN $\mathcal{F}_{\mathsf{daa}}$. In order to design a property-based definition of DAA, our starting point was to extract the 'essence' of the security guarantees provided by $\mathcal{F}_{\mathsf{daa}}$; the ideal DAA functionality [CDL16]. During this process, it became clear that certain checks of $\mathcal{F}_{\mathsf{daa}}$ are unnecessarily strong in the sense that they could be weakened without losing any meaningful security. These checks are in the signature verification interface. When verifying a signature, $\mathcal{F}_{\mathsf{daa}}$ first collates all TPM secret keys to which the given

signature identifies. The signature is marked as invalid if one of a number of checks does not pass. We argue that two of these checks are unnecessarily restrictive. Said differently, they can be weakened without any loss of meaningful security. We list these checks.

1. The signature fails verification if it identifies to an honest TPM who never signed it, even if the issuer is corrupt.

2. The signature is rejected if it identifies to some key on the revocation list but does not identify to an honest TPM.

Let us now discuss why each of these checks can be relaxed whilst maintaining the same security guarantees. Our discussion closely follows the argumentation of Section 3.2 of [CDL16] in which the authors analyse the security guarantees of $\mathcal{F}_{\mathsf{daa}}$. The reader may find it useful to refer to Figures 3.4 and 3.5 in the following discussion.

Our first observation is that it is not necessary to reject signatures according to the first check if the issuer is corrupt and the matching TPM is paired with a corrupt host. Indeed, we split this check into two sub-cases (checks (**xi**) and **xii**). In check (**xi**) the signature is rejected if, when the issuer is honest, it identifies to an honest TPM (with possibly corrupt Host) who never signed it. Check (**xi**) rejects the signature if, even when the issuer is corrupt, it identifies to some honest platform who never signed it. We argue that the combination of these two checks, which excludes the case 'corrupt issuer, honest TPM, corrupt host', is still enough to achieve the security guarantees claimed in [CDL16].

Unforgeability is still guaranteed since check (**xi**) applies in the case of an honest issuer where the host may be corrupt as needed for unforgeabilitiy.

For non-frameability, it is important that one cannot create a signature that identifies to some honest platform who never signed it since this implies that it would link to ones it had signed. Check (**xii**) guarantees this, even when the issuer is corrupt. Note that in the original formulation of $\mathcal{F}_{\mathsf{daa}}$ [CDL16], this was guaranteed even if the host of that platform was corrupt. However, as argued in Section 3.3.2, honest TPMs with a corrupt host cannot be protected from framing. This is because the host controls the output of signatures and could also run a corrupt TPM that had joined and and use the corrupt TPM's key to create signatures instead of using the honest TPM's contribution. The resulting signatures can't be protected from framing since they use the corrupt TPM's signing key.

Checks needed for completeness/correctness properties only need to be enforced for honest platforms. Therefore, the arguments guaranteeing that the

extra verification checks won't trigger for platforms with an honest TPM apply to honest platforms. For these reasons, the previous formulation of $\mathcal{F}_{\mathsf{daa}}$ was overly restrictive in this case without providing extra security guarantees and we can make the modifications described above with no meaningful change to the security notion of $\mathcal{F}_{\mathsf{daa}}$.

Our second observation is that check (**xiii**) only needs to be made for honest platforms. Secret key-based revocation is enabled by the revocation list RL in the VERIFY interface and allows for the 'blocking' of signatures from exposed TPM keys.

This check is important in anonymity since it should not be exploitable for the tracing of honest platforms. Recall that anonymity can only hold if the whole platform is honest since a corrupt host could simply append its TPM's identity to each signature. Thus, in its original form [CDL16], check (**xiii**) gave no meaningful security improvement by preventing the exploitation of this check to trace honest TPMs (with corrupt host) rather than only protecting honest platforms. We therefore relax this check to reject signatures identifying to some key on the revocation list where no matching honest *platform* was found.

For the completeness of honest signatures, check (**xiii**) will not trigger since, by check (**vii**), we know that there is an honest platform with a key matching this signature. Finally, consistency of the verify interface is maintained w.r.t. check (**xiii**) since $\mathsf{identify}$ is deterministic by check (**i**) and executed without maintaining state so the result will be the same each time.

We conclude that these two modifications made to $\mathcal{F}_{\mathsf{daa}}$ in no way reduce its security guarantees. Furthermore, they allow for a more intuitive correspondence between the security properties of our new model and the security guarantees of $\mathcal{F}_{\mathsf{daa}}$. The reader should note that Theorem 3.4.1 gives necessary conditions such that Π_{DAA} implies this modified DAA functionality, not $\mathcal{F}_{\mathsf{daa}}$ in its original form as presented in [CDL16]. However, the above argumentation serves to demonstrate that these two variants provide exactly the same security guarantees.

3.4.1 *Wrapping A Property-Based DAA Scheme.*

We wish to prove that a scheme secure in our new model inherits essentially the same security guarantees as one proven secure in the UC framework. The DAA wrapper protocol Π_{DAA} shown in Figures 3.6 and 3.7 provides a generic transformation of a property-based DAA scheme DAA to one compatible with the universal composability framework. Those familiar with UC

Setup:

1. `Issuer Setup`. On input $(\mathsf{SETUP}, \mathsf{sid})$ from $\mathcal{I}$,
   - Verify that $\mathsf{sid} = (\mathcal{I}, \mathsf{sid}')$ and output $(\mathsf{SETUP}, \mathsf{sid})$ to Sim.
2. `Set Algorithms.` On input $(\mathsf{ALG}, \mathsf{sid}, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$ from Sim,
   - Check that ver, link, and $\mathsf{identify}$ are deterministic (**i**).
   - Store $(\mathsf{sid}, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify}, \mathsf{ukgen})$.

Join:

1. `Join Request`. On input $(\mathsf{JOIN}, \mathsf{sid}, \mathsf{jsid}, \mathcal{M}_i)$ from host $\mathcal{H}_j$,
   - Create a join session record $(\mathsf{jsid}, \mathcal{M}_i, \mathcal{H}_j, \mathit{status})$ with $\mathit{status} \leftarrow \mathit{request}$.
   - Output $(\mathsf{JOINSTART}, \mathsf{sid}, \mathsf{jsid}, \mathcal{M}_i, \mathcal{H}_j)$ to Sim.
2. `Join Request Delivery`. On input $(\mathsf{JOINSTART}, \mathsf{sid}, \mathsf{jsid},)$ from Sim,
   - Update the session record $(\mathsf{jsid}, \mathcal{M}_i, \mathcal{H}_j, \mathit{status})$ to $\mathit{status} \leftarrow \mathit{delivered}$.
   - Output $(\mathsf{JOINPROCEED}, \mathsf{sid}, \mathsf{jsid}, \mathcal{M}_i)$ to $\mathcal{I}$.
3. `Complete Join`. On input $(\mathsf{JOINPROCEED}, \mathsf{sid}, \mathsf{jsid})$ from $\mathcal{I}$,
   - Update session record $(\mathsf{jsid}, \mathcal{M}_i, \mathcal{H}_j, \mathit{status})$ to $\mathit{status} \leftarrow \mathit{complete}$.
   - Output $(\mathsf{JOINCOMPLETE}, \mathsf{sid}, \mathsf{jsid}, \mathsf{Sim})$.
4. `KeyGeneration`. On input $(\mathsf{JOINCOMPLETE}, \mathsf{sid}, \mathsf{jsid}, \mathsf{sk})$ from Sim.
   - Look up record $(\mathsf{jsid}, \mathcal{M}_i, \mathcal{H}_j, \mathit{status})$ with $\mathit{status} = \mathit{complete}$.
   - Abort if $\mathcal{I}$ or $\mathcal{M}_i$ is honest and a record $(\mathcal{M}_i, *, *) \in$ `Members` already exists (**ii**).
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, set $\mathsf{sk} \leftarrow \perp$.
   - Else, verify that the provided sk is eligible by checking
     - $\mathsf{CheckSkHonest}(\mathsf{sk}) = 1$ (**iii**) if $\mathcal{H}_j$ is corrupt and $\mathcal{M}_i$ is honest, or
     - $\mathsf{CheckSkCorrupt}(\mathsf{sk} = 1$ (**iv**) if $\mathcal{M}_i$ is corrupt.
   - Insert $\langle \mathcal{M}_i, \mathcal{H}_j, \mathsf{sk} \rangle$ into `Members` and output $(\mathsf{JOINED}, \mathsf{sid}, \mathsf{jsid})$ to $\mathcal{H}_j$.

FIGURE 3.4: The Setup and Join related interfaced of $\mathcal{F}_{\mathsf{daa}}$.

Sign:

1. `SignRequest`. On input $(\mathsf{SIGN}, \mathsf{sid}, \mathsf{ssid}, \mathcal{M}_i, \mathsf{m}, \mathsf{bsn})$ from $\mathcal{H}_j$,
   - If $\mathcal{I}$ is honest and no entry $\langle \mathcal{M}_i, \mathcal{H}_j, * \rangle$ exists in `Members`, abort.
   - Create a sign session record $(\mathsf{ssid}, \mathcal{M}_i, \mathcal{H}_j, \mathsf{m}, \mathsf{bsn}, \mathit{status})$ with $\mathit{status} \leftarrow \mathit{request}$.
   - Output $(\mathsf{SIGNSTART}, \mathsf{sid}, \mathsf{ssid}, \mathsf{m}, \mathsf{bsn}, \mathcal{M}_i, \mathcal{H}_j)$ to Sim.
2. `Sign Request Delivery`. On input $(\mathsf{SIGNSTART}, \mathsf{sid}, \mathsf{ssid})$ from Sim,
   - Update the session record $(\mathsf{ssid}, \mathcal{M}_i, \mathcal{H}_j, \mathsf{m}, \mathsf{bsn}, \mathit{status})$ to $\mathit{status} \leftarrow \mathit{delivered}$.
   - Output $(\mathsf{SIGNPROCEED}, \mathsf{sid}, \mathsf{ssid}, \mathsf{m}, \mathsf{bsn})$ to $\mathcal{M}_i$.
3. `SignProceed`. On input $(\mathsf{SIGNPROCEED}, \mathsf{sid}, \mathsf{ssid})$ from $\mathcal{M}_i$,
   - Look up record $(\mathsf{ssid}, \mathcal{M}_i, \mathcal{H}_j, \mathsf{m}, \mathsf{bsn}, \mathit{status})$ with $\mathit{status} = \mathit{delivered}$.
   - Output $(\mathsf{SIGNCOMPLETE}, \mathsf{sid}, \mathsf{ssid})$ to Sim.
4. `Signature Generation`. On input $(\mathsf{SIGNCOMPLETE}, \mathsf{sid}, \mathsf{ssid}, \sigma)$ from Sim,
   - If $\mathcal{M}_i$ and $\mathcal{H}_j$ are honest, ignore he adversary's signature and internally generate the signature for a fresh or established sk:
     - If $\mathsf{bsn} \neq \perp$, retrieve sk from $\langle \mathcal{M}_i, \mathcal{H}_j, \mathsf{bsn}, \mathsf{sk} \rangle \in$ `DomainKeys` for $(\mathcal{M}_i, \mathsf{bsn})$. If no such sk exists or $\mathsf{bsn} = \perp$, set $\mathsf{sk} \leftarrow \mathsf{ukgen}()$. Check $\mathsf{CheckSkHonest}(\mathsf{sk}) = 1$ (**v**) and store $\langle \mathcal{M}_i, \mathcal{H}_j, \mathsf{bsn}, \mathsf{sk} \rangle$ in `DomainKeys`.
     - Compute signature as $\sigma \leftarrow \mathsf{sig}(\mathsf{sk}, \mathsf{m}, \mathsf{bsn})$ and check $\mathsf{ver}(\sigma, \mathsf{m}, \mathsf{bsn}) = 1$ (**vi**).
     - Check $\mathsf{identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}) = 1$ (**vii**) and check that there is no $\mathcal{M}_i' \neq \mathcal{M}_i$ with signing key sk' registered in `Members` or `DomainKeys` with $\mathsf{identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}') = 1$ (**viii**).
   - If $\mathcal{M}_i$ is honest, store $(\sigma, \mathsf{m}, \mathsf{bsn}, \mathcal{M}_i)$ in `Signed`.
   - Output $(\mathsf{SIGNATURE}, \mathsf{sid}, \mathsf{ssid}, \sigma)$ to $\mathcal{H}_j$.

Verify:

1. On input $(\mathsf{VERIFY}, \mathsf{sid}, \mathsf{m}, \mathsf{bsn}, \sigma, \mathsf{RL})$ from some party $\mathcal{V}$,
   - Retrieve all pairs $\langle \mathcal{M}_i, *, \mathsf{sk} \rangle \in$ `Members` such that $\mathsf{identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}_i) = 1$ and all $\langle \mathcal{M}_i, \mathcal{H}_j, *, *, \mathsf{sk}_i \rangle \in$ `DomainKeys` where $\mathsf{identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}_i) = 1$. Set $f \leftarrow 0$ if at least one of the following conditions hold:
     - More than one such value sk_i was found (**ix**).
     - $\mathcal{I}$ is honest an no identifying value sk_i was found (**x**).
     - $\mathcal{I}$ is honest and there is an honest $\mathcal{M}_i$ but no entry $\langle *, \mathsf{m}, \mathsf{bsn}, \mathcal{M}_i \rangle \in$ `Signed` exists (**xi**).
     - There is an honest platform $(\mathcal{M}_i, \mathcal{H}_j)$, but no entry $\langle *, \mathsf{m}, \mathsf{bsn}, \mathcal{M}_i \rangle \in$ `Signed` exists (**xii**).
     - There is a $\mathsf{sk}' \in \mathsf{RL}$ where $\mathsf{identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}') = 1$ and no identifying value sk_i was found for an honest platform $(\mathcal{M}_i, \mathcal{H}_j)$ (**xiii**).
   - If $f \neq 0$, set $f \leftarrow \mathsf{ver}(\sigma, \mathsf{m}, \mathsf{bsn})$ (**xiv**).
   - Add $\langle \sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{RL}, f \rangle$ to `VerResults`, output $(\mathsf{VERIFIED}, \mathsf{sid}, f)$ to $\mathcal{V}$.

Link:

1. On input $(\mathsf{LINK}, \mathsf{sid}, \sigma, \mathsf{m}, \sigma', \mathsf{m}', \mathsf{bsn})$ from $\mathcal{V}$ with $\mathsf{bsn} \neq \perp$,
   - Output $\perp$ to $\mathcal{V}$ if at least one signature tuple $(\sigma, \mathsf{m}, \mathsf{bsn})$ or $(\sigma', \mathsf{m}', \mathsf{bsn})$ is not valid (verified via the Verify interface with $\mathsf{RL} = \emptyset$) (**xv**).
   - For each sk_i in `Members` and `DomainKeys` compute $b_i \leftarrow \mathsf{identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}_i)$ and $b_i' \leftarrow \mathsf{identify}(\sigma', \mathsf{m}', \mathsf{bsn}, \mathsf{sk}_i)$ and do the following:
     - Set $f \leftarrow 0$ if $b_i \neq b_i'$ for some i (**xvi**).
     - Set $f \leftarrow 1$ if $b_i = b_i' = 1$ for some i (**xvii**).
   - If f is not yet defined, set $f \leftarrow \mathsf{link}(\sigma, \mathsf{m}, \sigma', \mathsf{m}', \mathsf{bsn})$.
   - Output $(\mathsf{LINK}, \mathsf{sid}, f)$ to $\mathcal{V}$.

FIGURE 3.5: The Sign, Verify, and Link interfaces of $\mathcal{F}_{\mathsf{daa}}$.

proofs can consider protocol Π_{DAA} as *a* functionality that we wish to prove indistinguishable from *the ideal* functionality $\mathcal{F}_{\mathsf{daa}}$. This transformation is essentially a syntactic re-structuring of the algorithms comprising DAA. Herein we refer to the transformed protocol as the 'wrapper protocol', the functionality, or simply the 'wrapper'.

At a high level, the role of the wrapper is to ensure that input and output interfaces of our property-based syntax match up with those of the ideal functionality $\mathcal{F}_{\mathsf{daa}}$. Namely, an adversary (or environment) interacting with the property-based scheme should have the same number and type of input/output interfaces that he would when interacting with $\mathcal{F}_{\mathsf{daa}}$. We illustrate this with an example. Notice that the join phase of $\mathcal{F}_{\mathsf{daa}}$ is made up of two sub-phases; 'join request' and 'join proceed'. This is not the case in the property-based syntax where the join phase is simply described by the interactive join algorithm $\langle \mathsf{JoinTPM}, \mathsf{JoinH}, \mathsf{Issue} \rangle$. The join request phase of $\mathcal{F}_{\mathsf{daa}}$ involves only the host and issuer. Thus, the wrapper uses the two constituent algorithms JoinH and Issue to describe how these parties respectively respond to inputs sent from the adversary to the functionality in the UC setting.

Processes that do not require the interaction of two parties like verification and linking are more straightforward. Here, the wrappper simply takes the input (provided by an adversary) and transforms it into one compatible with the property-based algorithms Verify and Link. Again, we illustrate with an example. Consider the Verify interface of $\mathcal{F}_{\mathsf{daa}}$ which takes inputs of the form $(\mathsf{VERIFY}, \mathsf{sid}, \mathsf{m}, \mathsf{bsn}, \sigma, \mathsf{RL})$. The role of the first two entries will be discussed in the section below but are merely for book keeping in the UC framework. Now, the wrapper fetches $(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{RL})$ and ipk and feeds these into the $\mathsf{Verify}(\mathsf{ipk}, \sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{RL})$ algorithm which produces the output bit f. The wrapper then embellishes this to produce the final output $(\mathsf{VERIFIED}, \mathsf{sid}, f)$.

Clearly, this is only a syntactic restructuring of the DAA algorithms. Moreover, the internal algorithms run by each interface of the wrapper protocol are exactly those of the property-based protocol. Thus, the security properties and operation of the transformed scheme are unchanged by this transformation. We now describe two features of the wrapper. The first allows it to keep track of protocol messages whilst the second is the way in which it uses 'ideal hybrid-functionalities' to model parts of DAA which are not considered cryptographically in the security model.

Setup:

1. $\mathcal{I}$ upon input $(\mathsf{SETUP}, \mathsf{sid})$:
   - Checks that $\mathsf{sid} = (\mathcal{I}, \mathsf{sid}')$ for some sid'.
   - Retrieves $\mathsf{sp} \leftarrow \mathcal{F}_{\mathsf{crs}}^{D}$.
   - Runs $(\mathsf{ipk}, \mathsf{isk}, \mathsf{ML}) \leftarrow \mathsf{I.Kg}(\mathsf{sp})$ and store $(\mathsf{isk}, \mathsf{ML}, \mathsf{sid})$
   - Registers ipk with $\mathcal{F}_{\mathsf{ca}}$.
   - Outputs $(\mathsf{SETUPDONE}, \mathsf{sid})$.

Join Request:

1. $\mathcal{H}_j$ upon input $(\mathsf{JOIN}, \mathsf{sid}, \mathsf{jsid}, \mathcal{M}_i)$:
   - Parses $\mathsf{sid} = (\mathcal{I}, \mathsf{sid}')$, retrieves ipk and epk_i from $\mathcal{F}_{\mathsf{ca}}$.
   - Sets $\mathsf{st}^i_{\mathsf{JoinH}} := (\mathsf{epk}_i, \mathsf{ipk})$, $\mathsf{dec}^i_{\mathsf{JoinH}} := \mathsf{cont}$.
   - Runs $(\mathsf{st}^j_{\mathsf{JoinH}}, \mathsf{M}_{\mathsf{Issue}}, \mathsf{dec}^j_{\mathsf{JoinH}}) \leftarrow \mathsf{JoinH}(\mathsf{st}^i_{\mathsf{JoinH}}; \perp)$.
   - Sends $(\mathsf{JOIN}, \mathsf{sid}, \mathsf{jsid}, (\mathcal{M}_i, \mathsf{M}_{\mathsf{Issue}}))$ to $\mathcal{I}$.
2. $\mathcal{I}$ upon receiving $(\mathsf{JOIN}, \mathsf{sid}, \mathsf{jsid}, (\mathcal{M}_i, \mathsf{M}_{\mathsf{Issue}}))$ from $\mathcal{H}_j$:
   - Creates a join request record $\langle \mathsf{sid}, \mathsf{jsid}, \mathcal{M}_i, \mathcal{H}_j, \mathsf{M}_{\mathsf{Issue}}, \rangle$.
   - Outputs $(\mathsf{JOINPROCEED}, \mathsf{sid}, \mathsf{jsid}, \mathcal{M}_i)$.

Join Proceed:

1. $\mathcal{I}$ upon input $(\mathsf{JOINPROCEED}, \mathsf{sid}, \mathsf{jsid})$:
   - Retrieves both the join record $\langle \mathsf{sid}, \mathsf{jsid}, \mathcal{M}_i, \mathcal{H}_j, \mathsf{M}_{\mathsf{Issue}} \rangle$ and $(\mathsf{isk}, \mathsf{ML})$ from its records. It may also retrieve epk_i from $\mathcal{F}_{\mathsf{ca}}$ if not included in $\mathsf{M}_{\mathsf{Issue}}$.
   - Sets $\mathsf{st}^i_{\mathsf{Issue}} := (\mathsf{isk}, \mathsf{epk}_i, \mathsf{ML})$, $\mathsf{dec}^i_{\mathsf{Issue}} := \mathsf{cont}$.
   - Runs $(\mathsf{st}^i_{\mathsf{Issue}}, \mathsf{M}_{\mathsf{JoinH}}, \mathsf{dec}^i_{\mathsf{Issue}}) \leftarrow \mathsf{Issue}(\mathsf{st}^i_{\mathsf{Issue}}; \mathsf{M}_{\mathsf{Issue}})$.
   - Sends $(\mathsf{JOIN}, \mathsf{sid}, \mathsf{jsid}, \mathsf{M}_{\mathsf{JoinH}})$ to $\mathcal{H}_j$.
2.($\mathcal{H}$) If $\mathcal{H}_j$ receives a message of the form $(\mathsf{JOIN}, \mathsf{sid}, \mathsf{jsid}, \mathsf{M}_{\mathsf{JoinH}})$ from $\mathcal{I}$, $\mathcal{M}_i$, or $\mathcal{F}_{auth^*}$:
   - If $\mathsf{dec}^i_{\mathsf{JoinH}} \neq \mathsf{cont}$, does nothing. records.
   - Runs $(\mathsf{st}^i_{\mathsf{JoinH}}, \mathsf{M}_{\mathsf{JoinTPM/Issue}}, \mathsf{dec}^i_{\mathsf{JoinH}}) \leftarrow \mathsf{JoinH}(\mathsf{st}^i_{\mathsf{JoinH}}, \mathsf{M}_{\mathsf{JoinH}})$.
   - If $\mathsf{dec}^i_{\mathsf{JoinH}} = \mathsf{accept}$, sets $\mathsf{cre}_i := \mathsf{st}^i_{\mathsf{JoinH}}$ and outputs $(\mathsf{JOINED}, \mathsf{sid}, \mathsf{jsid})$.
   - If $\mathsf{dec}^i_{\mathsf{JoinH}} = \mathsf{cont}$, sends $(\mathsf{JOIN}, \mathsf{sid}, \mathsf{jsid}, \mathsf{M}_{\mathsf{JoinTPM/Issue}})$ to $\mathcal{M}_i/\mathcal{I}$.
2.($\mathcal{M}$) If $\mathcal{M}_i$ receives a message of the form $(\mathsf{JOIN}, \mathsf{sid}, \mathsf{jsid}, \mathsf{M}_{\mathsf{JoinTPM}})$ from $\mathcal{H}_j$:
   - If $\mathsf{st}^i_{\mathsf{JoinTPM}}$ is undefined, retrieves esk_i from its records and sets $\mathsf{st}^i_{\mathsf{JoinTPM}} := \mathsf{esk}_i$, $\mathsf{dec}^i_{\mathsf{JoinTPM}} := \mathsf{cont}$.
   - If $\mathsf{dec}^i_{\mathsf{JoinTPM}} \neq \mathsf{cont}$, does nothing.
   - Runs $(\mathsf{st}^i_{\mathsf{JoinTPM}}, \mathsf{M}_{\mathsf{JoinH/Issue}}, \mathsf{dec}^i_{\mathsf{JoinTPM}}) \leftarrow \mathsf{JoinTPM}(\mathsf{st}^i_{\mathsf{JoinTPM}}; \mathsf{M}_{\mathsf{JoinTPM}})$.
   - If $\mathsf{dec}^i_{\mathsf{JoinTPM}} = \mathsf{accept}$, sets $\mathsf{sk}_i := \mathsf{st}^i_{\mathsf{JoinTPM}}$.
   - Sends $(\mathsf{JOIN}, \mathsf{sid}, \mathsf{jsid}, \mathsf{M}_{\mathsf{JoinH/Issue}})$ to $\mathcal{H}_j$ directly or via $\mathcal{F}_{auth^*}$ respectively.
2.($\mathcal{I}$) If $\mathcal{I}$ receives a message of the form $(\mathsf{JOIN}, \mathsf{sid}, \mathsf{jsid}, \mathsf{M}_{\mathsf{Issue}})$ from $\mathcal{H}_j$ or $\mathcal{F}_{auth^*}$:
   - If $\mathsf{dec}^i_{\mathsf{Issue}} \neq \mathsf{cont}$, does nothing.
   - Runs $(\mathsf{st}^i_{\mathsf{Issue}}, \mathsf{M}_{\mathsf{JoinH}}, \mathsf{dec}^i_{\mathsf{Issue}}) \leftarrow \mathsf{Issue}(\mathsf{st}^i_{\mathsf{Issue}}; \mathsf{M}_{\mathsf{Issue}})$
   - If $\mathsf{dec}^i_{\mathsf{Issue}} = \mathsf{accept}$, sets $\mathsf{ML} = \mathsf{st}^i_{\mathsf{Issue}}$.
   - Sends $(\mathsf{JOIN}, \mathsf{sid}, \mathsf{jsid}, \mathsf{M}_{\mathsf{JoinH}})$ to $\mathcal{H}_j$

FIGURE 3.6: DAA wrapper protocol Π_{DAA}: setup and join phases.

Sign Request:

1. $\mathcal{H}_j$ upon input $(\mathsf{SIGN}, \mathsf{sid}, \mathsf{ssid}, \mathcal{M}_i, \mathsf{m}, \mathsf{bsn})$:
   - Creates a sign record $\langle \mathsf{sid}, \mathsf{ssid}, \mathcal{M}_i, \mathsf{m}, \mathsf{bsn} \rangle$.
   - Retrieves cre_j from its records.
   - Sets $\mathsf{st}^i_{\mathsf{SignH}} := (\mathsf{cre}_i, \mathsf{m}, \mathsf{bsn})$, $\mathsf{dec}^i_{\mathsf{SignH}} := \mathsf{cont}$.
   - Runs $(\mathsf{st}^i_{\mathsf{SignH}}, \mathsf{M}_{\mathsf{SignTPM}}, \mathsf{dec}^i_{\mathsf{SignH}}) \leftarrow \mathsf{SignH}(\mathsf{st}^i_{\mathsf{SignH}}; \mathsf{M}_{\mathsf{SignH}})$.
   - Sends $(\mathsf{SIGN}, \mathsf{sid}, \mathsf{ssid}, \mathsf{M}_{\mathsf{SignTPM}})$ to $\mathcal{M}_i$.
2. $\mathcal{M}_i$ upon receiving $(\mathsf{SIGN}, \mathsf{sid}, \mathsf{ssid}, \mathsf{M}_{\mathsf{SignTPM}})$ from $\mathcal{H}_j$:
   - Creates a sign record $\langle \mathsf{sid}, \mathsf{ssid}, \mathcal{H}_j, \mathsf{m}, \mathsf{bsn} \rangle$.
   - Outputs $(\mathsf{SIGNPROCEED}, \mathsf{sid}, \mathsf{ssid}, \mathsf{m}, \mathsf{bsn})$.

Sign Proceed:

1. $\mathcal{M}_i$ upon input $(\mathsf{SIGNPROCEED}, \mathsf{sid}, \mathsf{ssid})$:
   - Retrieves the sign record $\langle \mathsf{sid}, \mathsf{ssid}, \mathcal{H}_j, \mathsf{m}, \mathsf{bsn} \rangle$ and sk_i from its records.
   - Sets $\mathsf{st}^i_{\mathsf{SignTPM}} := (\mathsf{sk}_i, \mathsf{m}, \mathsf{bsn})$ and $\mathsf{dec}^i_{\mathsf{SignTPM}} := \mathsf{cont}$.
   - Runs $(\mathsf{st}^i_{\mathsf{SignTPM}}, \mathsf{M}_{\mathsf{SignH}}, \mathsf{dec}^i_{\mathsf{SignTPM}}) \leftarrow \mathsf{SignTPM}(\mathsf{st}^i_{\mathsf{SignTPM}}, \mathsf{M}_{\mathsf{SignTPM}})$.
   - Sends $(\mathsf{SIGN}, \mathsf{sid}, \mathsf{ssid}, \mathsf{M}_{\mathsf{SignH}})$ to $\mathcal{H}_j$.

2.($\mathcal{M}_i$) If $\mathcal{M}_i$ receives a message of the form $(\mathsf{SIGN}, \mathsf{sid}, \mathsf{ssid}, \mathsf{M}_{\mathsf{SignTPM}})$ from $\mathcal{H}_j$:
   - If $\mathsf{dec}^i_{\mathsf{SignTPM}} \neq \mathsf{cont}$, does nothing.
   - Runs $(\mathsf{st}^i_{\mathsf{SignTPM}}, \mathsf{M}_{\mathsf{SignH}}, \mathsf{dec}^i_{\mathsf{SignTPM}}) \leftarrow \mathsf{SignH}(\mathsf{st}^i_{\mathsf{SignTPM}}; \mathsf{M}_{\mathsf{SignTPM}})$.
   - Sends $(\mathsf{SIGN}, \mathsf{sid}, \mathsf{ssid}, \mathsf{M}_{\mathsf{SignH}})$ to $\mathcal{H}_j$.

2.($\mathcal{H}$) If $\mathcal{H}_j$ receives a message of the form $(\mathsf{SIGN}, \mathsf{sid}, \mathsf{ssid}, \mathsf{M}_{\mathsf{SignH}})$ from tpm_i:
   - If $\mathsf{dec}^i_{\mathsf{SignH}} \neq \mathsf{cont}$, does nothing.
   - Runs $(\mathsf{st}^i_{\mathsf{SignH}}, \mathsf{M}_{\mathsf{SignTPM}}, \mathsf{dec}^i_{\mathsf{SignH}}) \leftarrow \mathsf{SignH}(\mathsf{st}^i_{\mathsf{SignTPM}}; \mathsf{M}_{\mathsf{SignTPM}})$.
   - If $\mathsf{dec}^i_{\mathsf{SignH}} = \mathsf{accept}$, sets $\sigma := \mathsf{st}^i_{\mathsf{SignH}}$.
   - Outputs $(\mathsf{SIGNATURE}, \mathsf{sid}, \mathsf{ssid}, \sigma)$.

Verify:

1. $\mathcal{V}$ upon input $(\mathsf{VERIFY}, \mathsf{sid}, \mathsf{m}, \mathsf{bsn}, \sigma, \mathsf{RL})$:
   - Sets $f \leftarrow \mathsf{Verify}(\mathsf{ipk}, \sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{RL})$.
   - Outputs $(\mathsf{VERIFIED}, \mathsf{sid}, f)$.

Link:

1. $\mathcal{V}$ upon input $(\mathsf{LINK}, \mathsf{sid}, \sigma, \mathsf{m}, \sigma', \mathsf{m}', \mathsf{bsn})$ with $\mathsf{bsn} \neq \perp$:
   - Sets $f \leftarrow \mathsf{Link}(\sigma, \mathsf{m}, \sigma', \mathsf{m}', \mathsf{bsn})$
   - Outputs $(\mathsf{LINK}, \mathsf{sid}, f)$.

FIGURE 3.7: DAA wrapper protocol Π_{DAA}: sign, verify, and link phases.

KEEPING TRACK OF PROTOCOL MESSAGES. We now describe how messages sent to and from the functionality are labelled to ensure they are processed and delivered correctly.

In the property-based world, security experiments provide the adversary with a set of dedicated oracles allowing him to make queries to honest parties. These oracles allow an adversary granular access to the interfaces of the party he queries. In the UC framework, no such oracles exist and all queries are sent to the functionality. One can think of the functionality as one big oracle to which all queries from the adversary are sent. It is therefore important that messages to and from the functionality are labelled according to the type of query that is being made. In the example examined above, the verification query begins with the VERIFY identifier. This tells us which interface of the functionality the appended input should be sent to.

At the core of the UC framework is the adversary's ability to run multiple instances of the DAA protocol. This is central to the model's composability guarantees. In contrast, the property-based world only considers that the adversary interacts with a single instance of the protocol. To remedy this disparity, our wrapper appends protocol messages with session identifies sid to indicate which instance of the protocol the message is intended for. Also commonplace in UC security proofs is the use of sub-session identifiers. These can be thought of as way for the functionality to keep state about which future messages relate to previous ones and are analogous to the state and decision objects which the property-based algorithms maintain. We highlight the join and signing sub-session identifiers jsid and ssid used by the wrapper which allow the functionality to identify which join proceed and sign proceed sessions correspond to earlier join request and sign request sessions respectively.

IDEAL, HYBRID FUNCTIONALITIES. In the property-based model, there are a few aspects of DAA that we do not consider cryptographically but whose presence we depend on. In particular, it is assumed that some trusted TPM manufacturer creates a set of parameters and provides each TPM with a set of endorsement keys used to establish an authenticated channel with the issuer. Additionally, we assume that there exists a trusted certificate authority with whom the issuer registers its public key, and with whom the TPM manufacturer maintains a list of all public TPM endorsement keys. Once again, these features of DAA are outside the scope of this work and are abstracted. The same thing is done in the UC framework however these features are abstracted by the 'ideal, hybrid-functionalities' $\mathcal{F}_{\mathsf{crs}}^{D}, \mathcal{F}_{auth^*}$ and

$\mathcal{F}_{\mathsf{ca}}$ respectively. For formal definitions of $\mathcal{F}_{\mathsf{ca}}$, $\mathcal{F}_{\mathsf{crs}}^{D}$ and $\mathcal{F}_{auth^*}$ we refer the reader to [Can03], [Can01], and [CDL16] respectively. Protocol Π_{DAA} includes calls to these hybrid-functionalities where necessary.

3.4.2 *UC Implication*

We now present the main result of this chapter; showing that the wrapper protocol, if secure according to the new model, enjoys the same security guarantees as provided by the UC definition. There are two additional conditions that a (wrapped) DAA scheme, secure in the property-based model, must satisfy in order to achieve UC security. Firstly, the issuer must register its public key using an extractable proof of knowledge (of isk). Secondly, the host is assumed to begin and end the interactive join and sign protocols. All of these conditions are met by existing DAA schemes except the first which, where omitted, has lead to security flaws.

Theorem 3.4.1. *Let* DAA *be a DAA scheme that is secure according to Definition 3.3.11. Then* Π_{DAA} *presented in Figures 3.6 and 3.7 securely realises* $\mathcal{F}_{\mathsf{daa}}$ *in the* $(\mathcal{F}_{auth^*}, \mathcal{F}_{\mathsf{ca}}, the)$*-hybrid random oracle model if the following hold:*

1. *The issuer registers its public key using an online-extractable proof of knowledge* $\pi_{\mathcal{I}}$ *of the underlying secret key, with some certificate authority.*

2. *The host initializes and terminates the interactive join and signing protocols.*

We now give a high level proof overview describing the game hops and the techniques used to prove indistinguishably between each hop and refer the reader to the full publication for intermediate functionality figures and simulators.

In order to prove that the DAA scheme Π_{DAA} satisfies the UC DAA definition of [CDL16] we proceed as follows. We must show that no environment $\mathcal{E}$ can distinguish the real world, in which it is working with Π_{DAA} and adversary $\mathcal{A}$ from the ideal world in which it uses the DAA ideal functionality $\mathcal{F}_{\mathsf{daa}}$ as defined in [CDL16] (see Appendix 3.4) with simulator Sim.

We begin with the real world protocol Π_{DAA}. Next we define an entity $\mathcal{C}$, which we call the challenger, who runs this real world protocol for all parties. $\mathcal{C}$ is then split into two parts; a functionality $\mathcal{F}$ and a simulator Sim. We first

have a useless functionality which forwards all inputs from honest parties to the simulator and then relays the output back to those honest parties. Over a series of game hops we modify Sim and $\mathcal{F}$ so that $\mathcal{F}$ handles more of the inputs from honest parties on its own. At the end we are left with the ideal functionality $\mathcal{F}_{\mathsf{daa}}$ and a satisfying simulator.

Proof (Sketch).

GAME 1: This is the real world protocol.

GAME 2: The challenger $\mathcal{C}$ now receives all inputs and simulates the real world for all honest parties. It also simulates the hybrid functionalities honestly. By construction this is equivalent to the previous game.

GAME 3: We now split $\mathcal{C}$ into two parts: a 'dummy' functionality $\mathcal{F}$ and a simulator Sim. $\mathcal{F}$ simply forwards all inputs from honest parties to Sim. Sim simulates the real world protocol and relays the outputs back to $\mathcal{F}$ who in turn passes them onto $\mathcal{E}$. This is simply a restructuring of the previous game and is thus equivalent to the previous game.

GAME 4: In this game we let our intermediate $\mathcal{F}$ handle the Setup related interfaces using the procedure specifed in $\mathcal{F}_{\mathsf{daa}}$. Consequently, $\mathcal{F}$ expects to receive the algorithms (ukgen, sig, ver, link, $\mathsf{identify}$) from the simulator. For ukgen, ver, link, and $\mathsf{identify}$, Sim can simply provide the algorithms from the real-world protocol, where it omits the revocation check from ver. The sig algorithm, though, must contain the issuer's private key, so Sim has to be able to get that value.

When $\mathcal{I}$ is honest, Sim will receive a message from $\mathcal{F}$ asking for the algorithms, which informs Sim what is happening and allows him to start simulating the issuer. Since Sim is running the issuer, it knows its secret key and sets the sig algorithm accordingly.

When $\mathcal{I}$ is corrupt, Sim starts the simulation when the issuer registers his key with $\mathcal{F}_{\mathsf{ca}}$ that is controlled by the simulator. Since the public key includes an (on-line extractable) proof of knowledge of the issuer's secret key, Sim can extract the secret key from there and define the sig algorithm accordingly. By the simulation soundness of the SPK, this game hop is indistinguishable for the adversary.

GAME 5: $\mathcal{F}$ now handles the verify and link queries using the provided algorithms ver and link from the previous game, rather than forwarding the

queries to Sim. We do not let $\mathcal{F}$ perform the additional checks (Check (**ix**) - Check (**xvii**)) done by $\mathcal{F}_{\mathsf{daa}}$, though, but add these only later. For Check (**xiii**), $\mathcal{F}$ rejects a signature when a matching $\mathsf{sk}' \in \mathsf{RL}$ is found, but does not exclude honest TPMs from this check yet.

Because verify and link do not involve network traffic, the simulator does not have to simulate traffic either, we must only make sure the outputs do not change. The verification algorithm ver that $\mathcal{F}$ uses is almost equal to the real world protocol. The only difference is that ver used by $\mathcal{F}$ does not contain the revocation check. $\mathcal{F}$ now performs this check separately. Since, by an assumption of Definition 3.3.11, key-based revocation is instantiated precisely by running the Identify algorithm on each key in RL this separation will not change the verification outcome.

For linking, $\mathcal{F}$ executes the Link algorithm that Sim supplied, which is equivalent to the real world algorithm, so the outcome will clearly be equivalent.

GAME 6: In this step we change $\mathcal{F}$ to also handle the join-related interfaces, meaning it will receive the inputs and generate the outputs. We let $\mathcal{F}$ run the same procedure as $\mathcal{F}_{\mathsf{daa}}$, but again omit the additional checks (Check (**ii**)- Check (**iv**)).

We have to ensure that $\mathcal{F}$ outputs the same values as the real world does. As the join interfaces do not output crypto values, but only messages like 'start' and 'complete', we simply have to guarantee that whenever the real world protocol would reach a certain output, the functionality also allows that output, and vice versa.

For the direction from the real world to the functionality this is clearly given, since $\mathcal{F}$ does not perform additional checks and thus will always proceed for any input it receives from Sim. For all outputs triggered by $\mathcal{F}$, the simulator has to give explicit approval which allows Sim to block any output by $\mathcal{F}$ if the real world protocol would not proceed at a certain point.

If the host and/or TPM are honest, the simulator knows the identities $\mathcal{M}_i$, $\mathcal{H}_j$ and correctly uses them towards $\mathcal{F}$ as well as in the simulation. If both, the TPM and host are corrupt but the issuer is honest, then Sim cannot determine the identity of the host, as the host does not authenticate towards the issuer in the real world. This does not impact the real world simulation of the issuer, but the simulator has to choose an arbitrary corrupt host $\mathcal{H}_j$ now when invoking $\mathcal{F}$ with the JOIN call. This will only result in a different host being stored in the ML list in $\mathcal{F}$, but $\mathcal{F}$ never uses this identity when the corresponding TPM is corrupt.

In the final join interface JOINCOMPLETE, the simulator has to provide signing key sk of the TPM. When the TPM is honest, Sim knows the key anyway and if the TPM is corrupt, Sim extracts the signing key from the join transcript using Ext (see Definition 3.3.4). Note that $\mathcal{F}$ sets $\mathsf{sk} \leftarrow \perp$ when both the TPM and host are honest. However, this has no impact yet, as the signatures are still created by the simulator and the verify and link interfaces of $\mathcal{F}$ do not run the additional checks that make use of the internally stored records and keys. Overall, this game hop is indistinguishable by the simulation soundness of join transcripts.

GAMES 7, 8, 9, 10: Over the next four game hops, we gradually modify $\mathcal{F}$ so that it eventually handles all signing queries instead of merely forwarding them to Sim. As before, $\mathcal{F}$ will use the (sign) interfaces from $\mathcal{F}_{\mathsf{daa}}$, with the difference that it does not perform the Checks (**v**)-(**viii**) which we add in a later game.

When the TPM or the host is corrupt, Sim has to provide the signature value, which it takes from the real world simulation, and thus perfectly mimics the real world output. When both the TPM and the host are honest, $\mathcal{F}$ creates the signatures internally in an unlinkable way; it chooses a new sk per basename and TPM, or per signature when $\mathsf{bsn} = \perp$ and then runs the sig algorithm for that fresh key. $\mathcal{F}$ keeps the internally chosen keys $\langle \mathcal{M}_i, \mathcal{H}_j, \mathsf{bsn}, \mathsf{sk}_i \rangle$ in a list `DomaninKeys` to ensure consistency if a TPM wishes to reuse the basename

This change is indistinguishable under the assumed anonymity of protocol Π_{DAA}. Suppose an environment can distinguish a signature by an honest party with the sk it joined with from a signature by the same party but with a different sk. Then one can use such an environment to win the 'real-or-random' anonymity game described in Definition A.1.2, Appendix A.1. To see this, we consider a sequence of sub-games where, in one game, a signature is generated using the true sk used in the joining of that TPM and in the next game that same signature is created by sampling a fresh secret key. To show that these two sub-games are indistinguishable, we embed the challenge oracle $\mathsf{ChalO}^{ror}(\cdot,\cdot,\cdot)$ into the simulator so that the signature Sim returns is the one obtained on querying the $\mathsf{ChalO}^{ror}(\cdot,\cdot,\cdot)$ oracle. If the environment thinks it is observing the second sub-game, it must be that we are in the 'random' case, else we are in the 'real case'. Thus we can use the distinguishing power of the environment to win the real-or-random anonymity game. By Lemma A.1.3 (see Appendix A.1), we can then win the 'left-or-right' anonymity game. Since protocol Π_{DAA} is assumed to satisfy the left-or-right

notion of anonymity, this change is indistinguishable.

Another important modification made to $\mathcal{F}$ is a check that all honestly generated signatures verify and identify correctly. Once again, we reduce to a property of real-world protocol Π_{DAA}; its correctness.

The last important modification to $\mathcal{F}$ is that for honest platforms, $\mathcal{F}$ now checks that a newly generated signature doesn't identify to the signing key of some other TPM. Since there is a unique value sk which identifies to each valid signature, this matching key corresponds to some honest platform. We then argue that due to the exponential size of the key space, the probability than one samples the same key as some other TPM is negligible.

GAME 11: When verifying signatures, $\mathcal{F}$ now rejects any signature identifying to two different signing keys. By assumption that there is a unique TPM signing key to which a valid signature identifies, this new check does not alter the verification outcome.

GAME 12: $\mathcal{F}$ now rejects signatures that do not identify to a TPM secret key used in an earlier join session. Importantly this check is only triggered if the issuer is also honest. In order to show that this check will not trigger, we demonstrate how such a triggering signature constitutes a game winning forgery in the unforgeability game defined in Figure 3.3. In particular, our reductionist simulates Game 12 to the environment by making use of $\mathsf{IssueO}(\cdot)$ and the other oracles available to it in the unforgeability game. Since Π_{DAA} is assumed unforgeable, Game 12 is indistinguishable from Game 11.

GAME 13: $\mathcal{F}$'s verify interface now rejects signatures that identify to some honest TPM who never signed them. We show that if an environment is able to distinguish Games 12 and 13, it must be able to create a signature triggering this check which constitutes a game-winning output in the unforgeability game. Since Π_{DAA} is assumed unforgeable, Game 13 is indistinguishable from Game 12.

GAME 14: $\mathcal{F}$ now introduces a new check that disallows signatures identifying to some honest platform who never signed them, even when the issuer is corrupt. We show that if there is a distinguisher who can distinguish between Games 13 and 14, then one can construct a reductionist who wins the non-framability game described in Figure 2. We consider the cases when this check triggers for $\mathsf{bsn} \neq \perp$ and $\mathsf{bsn} = \perp$ separately, showing that neither situ-

ations occur with more than negligible probability if Π_{DAA} is non-frameable and unforgeable as assumed.

GAME 15: Previously, $\mathcal{F}$ rejected any signature that identified to some signing key on the revocation list RL. Now such signatures are only rejected if they additionally identify to no honest platforms. To show that this is indistinguishable from the previous game, we must show that signatures rejected by the revocation list check that also identify to some honest platform occur with negligible probability. In other words, the additional constraint of this check affects the outcome of verification with negligible probability. We assume that there is a distinguisher, able to tell the difference between Games 14 and 15. We construct a reductionist $\mathcal{R}$ who uses this distinguisher to win the anonymity experiment defined in Figure 3.3. Once this reductionist observes a signature rejected by this new check, he learns the TPM key of an honest platform. He can then call the challenge oracle with this platform and some other randomly chosen honest platform. If the challenge signature identifies to this newly discovered key then it must have come from that platform. If not, it came from the second platform. Since protocol Π_{DAA} is assumed to be anonymous, Game 13 is indistinguishable from Game 14.

GAME 16: F now puts requirements on the link algorithm. We show that these requirements do not change the output of the Link interface. The introduction of this new check would only change the output distribution in the following two cases. Firstly, two signatures that previously linked now do not. This would happen if $\mathsf{Link}(\sigma, \sigma', \mathsf{m}, \mathsf{m}', \mathsf{bsn}) = 1$ but either $\mathsf{Identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}) = 0$ or $\mathsf{Identify}(\sigma', \mathsf{m}', \mathsf{bsn}, \mathsf{sk}) = 0$. Alternatively two signatures that previously did not link but now do, i. e. $\mathsf{Link}(\sigma, \sigma', \mathsf{m}, \mathsf{m}', \mathsf{bsn}) = 0$ but $\mathsf{Identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk}) = \mathsf{Identify}(\sigma', \mathsf{m}', \mathsf{bsn}, \mathsf{sk}) = 1$. By assumption, (see Definition 3.3.11) neither of these situations can occur since it is assumed that signatures have the property that they link if and only if there is some value sk such that both signatures identify to sk.

The functionality $\mathcal{F}$ described in Game 15 is exactly $\mathcal{F}_{\mathsf{daa}}$ as defined in [CDL16] (See Figures 3.4 and 3.5 in Section 3.4). Since we have shown the indistinguishability of each game hop, we have that protocol Π_{DAA} realizes $\mathcal{F}_{\mathsf{daa}}$. □

3.5 A GENERIC CONSTRUCTION

In this section, we provide a generic construction of DAA, $\mathsf{DAA}_{\mathsf{Gen}}$, satisfying our property-based security definition. The generic construction allows us to view prior constructions of DAA in a more modular manner, and in particular, it provides better intuition on why the constructions are secure/insecure. Compared with some DAA schemes [BE17, KCB+19] which require the use of homomorphic commitments or non-standard signatures allowing for a two-party signing protocol, our generic construction relies on simpler primitives. In particular these previous schemes require the composition of signature schemes, where one party completes a signature begun by another, yielding hard to verify proofs.

3.5.1 *Description of* $\mathsf{DAA}_{\mathsf{Gen}}$

Our construction employs the following cryptographic building blocks. We believe all components other than the *tag scheme* are well-known. A *tag scheme*, first introduced by [BFG+11] in the context of DAA, allows us to construct a tag v for a message m using a secret key x. The tag itself leaks no information of the message nor secret key. However, once two tags on the same secret key and message are produced, they will be publicly linkable. Due to this useful notion of anonymity *and* linkability, it has been used explicitly or implicitly in many primitives such as linkable ring signatures [LWW04], group signatures with verifier local revocation [BS04], E-cash [CHL05], and anonymous authentication [DDP06, CHK+06]. We refer to Section 3.2 for a more formal definition of the building blocks below.

1. An existentially unforgeable signature scheme $\mathcal{S} = (\mathsf{S.Kg}, \mathsf{S.Sign}, \mathsf{S.Ver})$;
2. An injective one-way function F;
3. A tag scheme $\mathcal{T} = (\mathsf{T.Kg}, \mathsf{Tag}, \mathsf{T.Link}, \mathsf{T.Check})$ compatible with F, that is linkable, indistinguishable and collision-resistant;
4. A hiding and binding commitment scheme $\mathcal{C} = \mathsf{Com}$;
5. Signatures of knowledge $\mathsf{SPK}_1, \mathsf{SPK}_2, \mathsf{SPK}_3$, which are online-extractable.

Since all building blocks above can be instantiated assuming the hardness of the discrete logarithm (DL) problem (in the random oracle model), our generic construction provides a DAA scheme based on DL. Our construction of DAA works as follows.

$\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{sp}$: On input security parameter 1^λ, the setup algorithm generates sp as a collection of public parameters specified by $\mathcal{S}$, $\mathcal{T}$, F, $\mathcal{C}$ and the signatures of knowledge.

$\mathsf{TPM.Kg}(\mathsf{sp}) \rightarrow (\mathsf{esk}, \mathsf{epk})$: The TPM key generation algorithm takes as input the system parameters sp and outputs an endorsement secret/public key pair $(\mathsf{esk}, \mathsf{epk})$ for TPM.

$\mathsf{I.Kg}(\mathsf{sp}) \rightarrow (\mathsf{isk}, \mathsf{ipk}, \mathsf{ML})$: The issuer key generation algorithm takes as input the system parameters sp, runs the key generation algorithm of $\mathcal{S}$ to obtain a signing/verifying key pair and sets it as the secret/public key pair $(\mathsf{isk}, \mathsf{ipk})$ for the issuer $\mathcal{I}$. The algorithm also initialises an empty members list ML.

$\langle \mathsf{JoinTPM}(\mathsf{esk}), \mathsf{JoinH}(\mathsf{epk}, \mathsf{ipk}), \mathsf{Issue}(\mathsf{isk}, \mathsf{epk}, \mathsf{ML}) \rangle$: The interactive join protocol between a TPM, host and issuer is as follows.

1. The host sends the TPM's public endorsement key epk to the issuer. The latter checks whether $\mathsf{epk} \in \mathsf{ML}$. If not, it sends back a nonce ρ.
2. Receiving ρ, the host passes it to the TPM who then proceeds as follows.
   a) Run the key generation algorithm of $\mathcal{T}$ to obtain a key x.
   b) Compute $\mathsf{tpk} = \mathsf{F}(x)$.
   c) Generate a signature of knowledge π_{join} to prove knowledge of the preimage x of tpk:
   $$\pi_{join} \leftarrow \mathsf{SPK}_1.\mathsf{Sign}\{(\mathsf{sp}, \mathsf{tpk}), x \mid \mathsf{F}(x) = \mathsf{tpk}\}(\rho).$$
   d) Send the pair $(\mathsf{tpk}, \pi_{join})$ to the host who then forwards it onto the issuer.
3. If π_{join} is valid, then the issuer generates a signature on the TPM's public key, $\mathsf{sig} \leftarrow \mathsf{S.Sign}(\mathsf{isk}, \mathsf{tpk})$, updates the member list $\mathsf{ML} \leftarrow \mathsf{ML} \cup \{\mathsf{epk}\}$, and sends sig to the host.
4. The host verifies sig and stores the credential $\mathsf{cre} = (\mathsf{sig}, \mathsf{tpk})$, while the TPM sets its signing key as $\mathsf{sk} = x$.

$\langle \mathsf{SignTPM}(\mathsf{sk}, \mathsf{m}, \mathsf{bsn}), \mathsf{SignH}(\mathsf{cre}, \mathsf{m}, \mathsf{bsn}) \rangle$: This interactive sign protocol between TPM and host works as follows.

1. The host samples a commitment randomness r for $\mathcal{C}$, commits to tpk as $c = \mathsf{Com}(\mathsf{tpk}, r)$ and sends (tpk, r, c) to the TPM.
2. The TPM verifies that $\mathsf{Com}(\mathsf{tpk}, r) = c$. Then, depending on whether $\mathsf{bsn} = \perp$, it uses $\mathsf{sk} = x$ and r to generate a signature of knowledge $ß_{\mathsf{T}}$ as follows.
   - If $\mathsf{bsn} \neq \perp$, redefine $\mathsf{bsn} := (0, \mathsf{bsn})$, compute $v = \mathsf{Tag}(x, \mathsf{bsn})$ and let

     $$ß_{\mathsf{T}} \leftarrow \mathsf{SPK}_2.\mathsf{Sign}\{(\mathsf{sp}, c, v, \mathsf{bsn}), (\mathsf{tpk}, x, r) \mid \mathsf{F}(x) = \mathsf{tpk} \wedge \mathsf{Com}(\mathsf{tpk}, r) = c \wedge \mathsf{T.Check}(v, x, \mathsf{bsn}) = 1\}(\mathsf{m}, \mathsf{bsn}, c, v).$$

   - If $\mathsf{bsn} = \perp$, sample uniformly random $\mathsf{bsn}^\star \leftarrow \{0,1\}^\lambda$, redefine $\mathsf{bsn} := (1, \mathsf{bsn}^\star)$, compute $v = \mathsf{Tag}(x, \mathsf{bsn})$, and generate $ß_{\mathsf{T}}$ as above.
3. Send $(\mathsf{bsn}, v, ß_{\mathsf{T}})$ to the host.
4. If $ß_{\mathsf{T}}$ is invalid, the host outputs $\perp$ and aborts. Otherwise, it uses $\mathsf{cre} = (\mathsf{sig}, \mathsf{tpk})$ and r to generate a signature of knowledge

   $$ß_{\mathsf{H}} \leftarrow \mathsf{SPK}_3.\mathsf{Sign}\{(\mathsf{sp}, \mathsf{ipk}, c), (\mathsf{cre}, r) \mid \mathsf{S.Ver}(\mathsf{ipk}, \mathsf{tpk}, \mathsf{sig}) = 1 \wedge \mathsf{Com}(\mathsf{tpk}, r) = c\}(\mathsf{m}, c, v, ß_{\mathsf{T}})$$

   and outputs signature $\sigma = (c, v, ß_{\mathsf{T}}, ß_{\mathsf{H}})$.

$\mathsf{Verify}(\mathsf{ipk}, \sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{RL})$: On input the issuer's public key ipk, a signature σ, a message m, a basename bsn, and a revocation list RL consisting of secret signing keys of corrupted TPMs, this algorithm parses σ as $(c, v, ß_{\mathsf{T}}, ß_{\mathsf{H}})$ proceeds as follows.

1. If $ß_{\mathsf{T}}$ is invalid, return 0.
2. If $ß_{\mathsf{H}}$ is invalid, return 0.
3. If there exists an entry $\mathsf{sk}' = x'$ in RL such that $\mathsf{T.Check}(v, x', \mathsf{bsn}) = 1$, return 0.
4. Return 1.

$\mathsf{Link}(\mathsf{ipk}, \sigma, \sigma', \mathsf{m}, \mathsf{m}', \mathsf{bsn})$: On input the issuer's public key ipk, two valid signatures σ, σ' w.r.t to two messages m, m' and a common basename bsn, the linking algorithm proceeds as follows.

1. Parse σ and σ' as $(c, v, ß_{\mathsf{T}}, ß_{\mathsf{H}})$ and $(c', v', ß_{\mathsf{T}}', ß_{\mathsf{H}}')$, respectively.

2. Return 1 (i.e., linked) if $\mathsf{bsn} \neq \perp$ and $\mathsf{T.Link}(v, v') = 1$. Otherwise, return 0.

Remark 3.5.1. *In the construction above, although we do not explicitly specify, we assume that the issuer generates an online-extractable signature of knowledge* π_{I} *of his secret key* isk *in the registration of* ipk *with the CA. Furthermore, the algorithm* IpkVf *should take* π_{I} *as input and output* accept *if* π_{I} *is valid and* reject *otherwise. These are necessary to achieve composable security. Otherwise* IpkVf *may simply check the well-formedness of* ipk.

3.5.2 *Security of* $\mathsf{DAA_{Gen}}$

Before proving the security of generic construction $\mathsf{DAA_{Gen}}$ in full detail, we begin by showing how the buildings blocks of $\mathsf{DAA_{Gen}}$ allow one to construct the auxiliary algorithms SimSetup, Ext, $\mathsf{Identify_T}$ and Identify defined in Definition 3.3.4.

The role of these algorithms is central to DAA security. Indeed, one could argue that the existence of an Ext algorithm satisfying Definition 3.3.4, along with the other auxiliary algorithms, is almost equivalent to the property of unforgeability; the most subtle and historically troublesome property of DAA.

For this reason we present a proof of their existence from our generic building blocks in this section.

Lemma 3.5.2 (Auxiliary algorithms)**.** *Assume that the signatures of knowledge* $\mathsf{SPK}_1, \mathsf{SPK}_2, \mathsf{SPK}_3$ *are simulatable and online extractable,* $\mathcal{T}$ *is a collision resistance tag scheme that is compatible with an injective one-way function* F*. Then the DAA system* $\mathsf{DAA_{Gen}}$ *satisfies the requirements of Definition 3.3.4.*

Proof. We construct algorithms SimSetup, Ext, IdentifyT, and Identify as follows.

SimSetup. The only difference between SimSetup and the real Setup algorithm is that the public parameters associated with the signatures of knowledge are generated using SPK_1.SimSetup, SPK_2.SimSetup and SPK_3.SimSetup, with trapdoors τ_1, τ_2, and τ_3, respectively. The algorithm outputs the simulated parameter sp together with trapdoor $\mathsf{td} = (\tau_1, \tau_2, \tau_3)$.

Ext. On input sp, trapdoor $\mathsf{td} = (\tau_1, \tau_2, \tau_3)$, ipk, isk, and a join transcript $\mathsf{TS}[i]$, this algorithm fetches $(\mathsf{tpk}_i, \pi_{join,i})$ from $\mathsf{TS}[i]$, and runs the

extractor of SPK_1 with trapdoor τ_1 to obtain x_i such that $\mathsf{F}(x_i) = \mathsf{tpk}_i$. Finally, output x_i.

Identify. On input a valid signature σ, a message m, a basename bsn, and a secret key sk, the identifying algorithm parses σ as $(c, v, \mathsf{ß}_\mathsf{T}, \mathsf{ß}_\mathsf{H})$ and returns 1 if $\mathsf{T.Check}(v, \mathsf{sk}, \mathsf{bsn}) = 1$. Otherwise, return 0. If the input signature is invalid, return 0.

$\mathsf{Identify}_\mathsf{T}$. On input a join transcript $\mathsf{TS}[i]$ and a TPM key sk, this algorithm fetches tpk_i from $\mathsf{TS}[i]$ and returns 1 if $\mathsf{F}(\mathsf{sk}) = \mathsf{tpk}_i$ and 0 otherwise.

We now prove that the four auxiliary algorithms we constructed above satisfy the conditions of Definition 3.3.4. By the simulatability of the signatures of knowledge, the sp output by $\mathsf{SimSetup}$ is indistinguishable from the one output by the real setup algorithm. Thus, item 1 of the definition is satisfied.

We now show item 2 is also satisfied. Given a valid signature $(\sigma, \mathsf{m}, \mathsf{bsn})$, parse σ as $\sigma = (c, v, \pi_\mathsf{T}, \pi_\mathsf{H})$. Due to the extractability of SPK_2, we can extract x such that $\mathsf{T.Check}(v, x, \mathsf{bsn}) = 1$. Now suppose, by contradiction, there exists $x' \neq x$ such that $\mathsf{T.Check}(v, x', \mathsf{bsn}) = 1$. Then, (m, x, x', v) represents a winning output in $\mathsf{Exp}^{\text{COLL-RES}}_{\mathcal{A},\mathsf{LIT}}(\lambda)$ as presented in Figure 3.1. Since, by the collision resistance of LIT, this cannot happen with more than negligible probability, we have that $x' = x$. Thus we arrive at a contradiction. Since $\mathsf{Identify}$ is instantiated with $\mathsf{T.Check}$, we conclude that $\mathsf{sk} := x$ is the unique key to which σ identifies.

Finally, we show that item 3 of Definition 3.3.4 is satisfied. Observe that if $\mathsf{CRE}[i] \neq \perp$ for some TPM i, then by definition of the IssueO oracle we have a complete transcript $\mathsf{TS}[i]$ of the join session for TPM i. We can then run $\mathsf{Ext}(\mathsf{sp}, \mathsf{td}, \mathsf{ipk}, \mathsf{isk}, \mathsf{TS}[i])$ as defined above to extract x_i such that $\mathsf{F}(x_i) = \mathsf{tpk}_i$, where tpk_i is from $\mathsf{TS}[i]$. By definition of $\mathsf{Identify}_\mathsf{T}$, we have that $\mathsf{Identify}_\mathsf{T}(\mathsf{TS}[i], x_i) = 1$. This satisfies the first point of item 3. Finally we must show that there exists a unique sk such that $\mathsf{Identify}_\mathsf{T}(\mathsf{TS}[i], \mathsf{sk}) = 1$. This follows from the injectivity of F since, given tpk_i, there is a unique x such that $\mathsf{F}(x) = \mathsf{tpk}_i$.

□

We now present results claiming the security of $\mathsf{DAA}_\mathsf{Gen}$ with respect to our new model and subsequently that $\Pi_{\mathsf{DAA}_\mathsf{Gen}}$ realises $\mathcal{F}_\mathsf{daa}$.

Theorem 3.5.3 (Security Of Generic Construction)**.** *The generic construction* $\mathsf{DAA}_\mathsf{Gen}$ *presented in Section 3.5.1 is a secure DAA scheme with respect to Definition 3.3.11.*

Proof Overview. It remains to show that $\mathsf{DAA}_{\mathsf{Gen}}$ satisfies the properties of correctness, anonymity, unforgeability, and non-frameability as well as the final condition of Definition 3.3.11 that two signatures under $\mathsf{bsn} \neq \perp$ link if and only if they identify to a common key.

The five aforementioned properties will be formally proven as Lemmata 3.5.5 to 3.5.7, 3.5.9 and 3.5.10. Here, we identify a few of the more subtle points of those proofs which we wish to draw the reader's attention to.

The proof of unfogeability is slightly unusual in the following sense. In this experiment, the 'winning' outputs are tested using the $\mathsf{Identify}$ algorithm. We modify this algorithm in a way that is undetectable to the adversary but which means that no such winning outputs can exist. The subtlety of the proof is that this same $\mathsf{Identify}$ algorithm must be used by the challenger to respond to queries to the $\mathsf{IdentifyO}$ oracle. This oracle, queried on input $(i, \mathsf{m}, \mathsf{bsn}, \sigma)$, should tell the adversary whether this signature was signed using TPM i or not. We show that if there exists an adversary who can distinguish between responses from the oracle based on the modified $\mathsf{Identify}$ algorithm and one based on the original $\mathsf{Identify}$ algorithm, then we can use such an adversary to break either the existential unforgeability of the signature scheme $\mathcal{S}$ or the one-way property of F.

The proof of non-frameability follows a similar structure since there the challenger must also provide an $\mathsf{IdentifyO}$ oracle for the adversary. Here, however, an adversary able to detect these modifications to the $\mathsf{Identify}$ algorithm can be used to break the one-way property of F or the linkability of the tag scheme $\mathcal{T}$.

Since, in $\mathsf{DAA}_{\mathsf{Gen}}$, the host initialises and terminates both the join and sign protocols, and we assume that the issuer registers his public key with an online extractable proof of correctness, Theorem 3.4.1 gives us the following Corollary.

Corollary 3.5.4 (UC Security of $\mathsf{DAA}_{\mathsf{Gen}}$)**.** *When defined with respect to the DAA system* $\mathsf{DAA}_{\mathsf{Gen}}$*, the DAA protocol* Π_{DAA} *defined in Figures 3.6 and 3.7 securely realises* $\mathcal{F}_{\mathsf{daa}}$ *in the* $(\mathcal{F}_{auth^*}, \mathcal{F}_{\mathsf{ca}}, \mathcal{F}_{\mathsf{crs}}^{D})$*-hybrid random oracle model.*

Corollary 3.5.4 demonstrates that, under a well defined syntactic transformation, our generic construction $\mathsf{DAA}_{\mathsf{Gen}}$ can, not only be proven in our new property-based model, but also achieves composable security as defined in [CDL16].

We prove, in detail the remaining lemmata required for the proof of Theorem 3.5.3.

CORRECTNESS OF DAA_{Gen}.

Lemma 3.5.5 (Correctness)**.** *The DAA system* $\mathsf{DAA}_{\mathsf{Gen}}$ *satisfies correctness, as defined in Section 3.3.2, if all the building blocks are correct.*

Proof. Assuming the correctness of $\mathcal{S}$, F, $\mathcal{C}$, $\mathcal{T}$, and SPK_1 in the signing protocol, an honest platform should possess $(x, \mathsf{tpk}, \mathsf{sig}, r)$ satisfying $\mathsf{S.Ver}(\mathsf{ipk}, \mathsf{tpk}, \mathsf{sig}) = 1$, $\mathsf{Open}(c, \mathsf{tpk}, r) = 1$, $\mathsf{F}(x) = \mathsf{tpk}$, $\mathsf{T.Check}(v, x, \mathsf{bsn}) = 1$. Then, by the correctness of SPK_2 and SPK_3, the signature $\sigma = (c, v, \mathfrak{B}_{\mathsf{T}}, \mathfrak{B}_{\mathsf{H}})$ should be accepted by the verifying algorithm. Indeed, if σ was not accepted by the Verify algorithm then either $\mathfrak{B}_{\mathsf{T}}$ or $\mathfrak{B}_{\mathsf{H}}$ is invalid, breaking the correctness of SPK_2 or SPK_3 resp.

Moreover, two signatures produced by the same platform with the same $\mathsf{bsn} \neq \perp$ are linked, and the Link algorithm is symmetric, thanks to the correctness of the tag system $\mathcal{T}$.

□

ANONITMIY OF DAA_{Gen}.

Lemma 3.5.6 (Anonymity)**.** *If the commitment scheme* $\mathcal{C}$ *is hiding, and if the tag system* $\mathcal{T}$ *is indistinguishable w.r.t.* F*, and if the signatures of knowledge* $\mathsf{SPK}_1, \mathsf{SPK}_2, \mathsf{SPK}_3$ *are simulatable, then the DAA system* $\mathsf{DAA}_{\mathsf{Gen}}$ *satisfies the anonymity property, as defined in Definition 3.3.6.*

Proof. Consider experiment $\mathsf{Exp}^{\text{Anon–b}}_{\mathcal{DAA},\mathcal{A}}(\lambda)$ of **Fig.** 3.3. We define a sequence of games, where the first game corresponds to $\mathsf{Exp}^{\text{Anon–0}}_{\mathcal{DAA},\mathcal{A}}(\lambda)$, and the last game to $\mathsf{Exp}^{\text{Anon–1}}_{\mathcal{DAA},\mathcal{A}}(\lambda)$. For each case, we argue that the adversary's behaviour changes only negligibly from one game to the next one. As a result, the advantage of the adversary in distinguishing the first game and the last game is at most negligible in λ. We consider the following sequence of games.

GAME 1. This is the anonymity experiment $\mathsf{Exp}^{\text{Anon–0}}_{\mathcal{DAA},\mathcal{A}}(\lambda)$ of **Fig.** 3.3, where the challenger chooses bit $b = 0$. In this game, given $\mathsf{sp} \leftarrow \mathsf{Setup}(1^\lambda)$, the adversary first sends back $(\mathsf{ipk}, \mathsf{ML}, \mathsf{st})$. Then, it is given access to oracles specified on Line 3 of the experiment. When the adversary queries ChalO with $(i_0, i_1, \mathsf{bsn}, \mathsf{m})$, the challenger returns with a signature $\sigma^\star$ generated honestly using key $(\mathsf{sk}_0, \mathsf{cre}_0)$ of platform i_0, i.e.,

$$\sigma^\star = (c_0, v_0, \mathfrak{B}_{\mathsf{T}0}, \mathfrak{B}_{\mathsf{H}0}) \leftarrow \langle \mathsf{SignTPM}(\mathsf{sk}_0, \mathsf{m}, \mathsf{bsn}), \mathsf{SignH}(\mathsf{cre}_0, \mathsf{m}, \mathsf{bsn}) \rangle.$$

GAME 2. In this game, we make the following change in the generation of sp. Instead of faithfully running the setup algorithms of SPK_1, SPK_2 and SPK_3, the challenger runs $\mathsf{SPK}_1.\mathsf{SimSetup}$, $\mathsf{SPK}_2.\mathsf{SimSetup}$ and $\mathsf{SPK}_3.\mathsf{SimSetup}$ to generate their public parameters together with simulation trapdoors τ_1, τ_2 and τ_3, respectively. Assuming the simulatability of SPK_1, SPK_2 and SPK_3, this game is indistinguishable from Game 1.

GAME 3. In this game, we modify how to react when the adversary queries the join and sign oracles $\mathsf{JoinO}(\cdot;\mathcal{P})$, $\mathsf{SignO}(\cdot;\mathcal{P})$ oracles. For a join query w.r.t honest platform i, the challenger still honestly generates $\mathsf{tpk}_i = F(x_i)$, but simulates $\pi_{join,i}$ as $\pi^{\star}_{join,i}$ using trapdoor τ_1. For a sign query, the challenger simulates π_{T} and π_{H} using trapdoors τ_2 and τ_3 respectively. By the simulatability of SPK_1, SPK_2 and SPK_3, this game is indistinguishable from Game 2.

GAME 4. In this game, we change how the challenge signature $\sigma^{\star}$ is generated. The commitment c_0 and the tag v_0 are still the same as in Game 3, i.e., $c_0 = \mathsf{Com}(\mathsf{tpk}_0, r_0)$ and $v_0 = \mathsf{Tag}(x_0, \mathsf{bsn})$, but $\mathfrak{s}_{\mathsf{T}0}$ and $\mathfrak{s}_{\mathsf{H}0}$ are simulated as $\mathfrak{s}_{\mathsf{T}}{}^{\star}$ and $\mathfrak{s}_{\mathsf{H}}{}^{\star}$, using τ_2 and τ_3, respectively. Let $\sigma^{\star} = (c_0, v_0, \mathfrak{s}_{\mathsf{T}}{}^{\star}, \mathfrak{s}_{\mathsf{H}}{}^{\star})$. By the simulatability of SPK_2 and SPK_3, this game is indistinguishable from Game 3.

GAME 5. In this game, we make the following modification w.r.t. Game 4. The commitment c_0 is replaced by $c_1 = \mathsf{Com}(\mathsf{tpk}_1, r_1)$, where tpk_1 is the TPM public key of platform i_1, and r_1 is a fresh commitment randomness. Let $\sigma^{\star} = (c_1, v_0, \mathfrak{s}_{\mathsf{T}}{}^{\star}, \mathfrak{s}_{\mathsf{H}}{}^{\star})$. Assuming the hiding property of $\mathcal{C}$, the advantage of the adversary in distinguishing this game from Game 4 is negligible.

GAME 6. In this game, we modify the generation of the tag, by replacing v_0 by $v_1 = \mathsf{Tag}(x_1, \mathsf{bsn})$. The challenge signature is set as $\sigma^{\star} = (c_1, v_1, \mathfrak{s}_{\mathsf{T}}{}^{\star}, \mathfrak{s}_{\mathsf{H}}{}^{\star})$. Note that, in Game 5, the adversary sees $(\mathsf{tpk}_0 = \mathsf{F.Eval}(x_0), \mathsf{tpk}_1 = \mathsf{F.Eval}(x_1), v_0 = \mathsf{Tag}(x_0, \mathsf{bsn}))$, and it sees $(\mathsf{tpk}_0 = \mathsf{F.Eval}(x_0), \mathsf{tpk}_1 = \mathsf{F.Eval}(x_1), v_1 = \mathsf{Tag}(x_1, \mathsf{bsn}))$ in Game 6. If it can distinguish Game 6 from Game 5 with non-negligible advantage, then we can use it to break the indistinguishability of the tag system $\mathcal{T}$ w.r.t. $\mathcal{F}$. Thus, by contraposition, we deduce that Game 6 and Game 5 are indistinguishable from the adversary's view.

GAME 7. This game is symmetric to Game 4, where we replace the simulated $\mathsf{B}_{\mathsf{T}}{}^{\star}$, $\mathsf{B}_{\mathsf{H}}{}^{\star}$ by $\mathsf{B}_{\mathsf{T}1}$, $\mathsf{B}_{\mathsf{H}1}$ honestly generated using key $(\mathsf{sk}_1, \mathsf{cre}_1)$ of platform i_1. The challenge signature is set as $\sigma^{\star} = (c_1, v_1, \mathsf{B}_{\mathsf{T}1}, \mathsf{B}_{\mathsf{H}1})$. By the simulatability of SPK_2 and SPK_3, this game is indistinguishable from Game 6.

GAME 8. This game is symmetric to Game 3, where we replace the simulated $\pi^{\star}_{join,i}$ used to answer to the adversary's joining queries by honestly generated $\pi_{join,i}$. By the simulatability of SPK_1, this game is indistinguishable from Game 7.

GAME 8. This game is symmetric to Game 2, where we replace the simulated public parameters sp in Game 6 by sp faithfully generated by $\mathsf{Setup}(1^{\lambda})$. Thanks to the simulatability of SPK_2 and SPK_3, this game is indistinguishable from Game 7.

GAME 9. This game is original $\mathsf{Exp}^{\text{Anon-1}}_{\mathcal{DAA},\mathcal{A}}(\lambda)$, with the challenger's bit $b = 1$. Note that it is identical to Game 8.

Since the adversary's advantage in distinguishing Game 9 from Game 1 is at most negligible, the construction satisfies the anonymity property defined in Definition 3.3.6.

□

UNFORGEABILITY OF $\mathsf{DAA}_{\mathsf{Gen}}$.

Lemma 3.5.7 (Unforgeability)**.** *Assume that* $\mathsf{SPK}_1, \mathsf{SPK}_2$*, and* SPK_3 *are simulatable and online extractable, the commitment scheme* $\mathcal{C}$ *is binding, the signature scheme* $\mathcal{S}$ *is existentially unforgeable under chosen message attacks, and* F *is an injective one-way function with respect to the collision resistant tag scheme. Then the DAA system* $\mathsf{DAA}_{\mathsf{Gen}}$ *satisfies unforgeability, as defined in Section 3.3.3.*

Proof. Let $\mathcal{A}$ be an adversary that breaks the unforgeability game with advantage ϵ. We consider the following game sequence and denote the event on which $\mathcal{A}$ wins in **Game** i as E_i. We first present the game sequence and then show that the advantage of $\mathcal{A}$ changes only negligibly in adjacent games.

GAME 0. This is the original unforgeability game. By assumption, we have $\Pr[\mathsf{E}_0] = \epsilon$.

GAME 1. In this game we modify the challenger so that it uses $\mathsf{SPK}_1.\mathsf{SimSign}$, $\mathsf{SPK}_2.\mathsf{SimSign}$, and $\mathsf{SPK}_3.\mathsf{SimSign}$ instead of $\mathsf{SPK}_1.\mathsf{Sign}$, $\mathsf{SPK}_2.\mathsf{Sign}$, and $\mathsf{SPK}_3.\mathsf{Sign}$. In particular, when the adversary queries oracle JoinO and SignO, the challenger simply uses $\mathsf{SPK}_1.\mathsf{SimSign}$, $\mathsf{SPK}_2.\mathsf{SimSign}$, and $\mathsf{SPK}_2.\mathsf{SimSign}$ respectively, to respond. This game is indistinguishable from Game 0 due to the simulatability of the SPKs. Namely, $|\Pr[\mathsf{E}_0] - \Pr[\mathsf{E}_1]| = \mathsf{negl}(\lambda)$.

GAME 2. In this game, we modify the challenger so that it stores all tpks of honest TPMs and corrupt TPMs that it has provided credentials to. In particular, let L_{HT} be the set of tpks that are generated by the challenger when queried the oracle JoinO. Moreover, let L_{CP} be the set of tpks that are queried to the oracle IssueO by the adversary. Since this change has no effect on the adversary, we have $\Pr[\mathsf{E}_1] = \Pr[\mathsf{E}_2]$.

GAME 3. In this game we modify how the challenger answers when queried the oracle $\mathsf{IdentifyO}$. In particular, we modify the algorithm $\mathsf{Identify}$ run internally in $\mathsf{IdentifyO}$. In the previous game, algorithm $\mathsf{Identify}$ was defined as follows.

$\mathsf{Identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk})$: Parse σ as $(c, v, \mathsf{ß}_\mathsf{T}, \mathsf{ß}_\mathsf{H})$ and return 1 if $\mathsf{T.Check}(v, \mathsf{sk}, \mathsf{bsn}) = 1$, and $\mathsf{ß}_\mathsf{T}$ and $\mathsf{ß}_\mathsf{H}$ are valid. Otherwise, return 0.

Here note that the algorithm $\mathsf{identify}$ run internally in $\mathsf{IdentifyO}$ only ever takes $\mathsf{sk} \in \mathsf{HSK}$ as input. However, since we later use the modified $\mathsf{Identify}$ algorithm for a different purpose, we provide the description in its full generality. In this game, the challenger replaces $\mathsf{identify}$ by the following algorithm $\mathsf{Identify}_1$.

$\mathsf{Identify}_1(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk})$: Parse σ as $(c, v, \mathsf{ß}_\mathsf{T}, \mathsf{ß}_\mathsf{H})$ and check $\mathsf{ß}_\mathsf{T}$ and $\mathsf{ß}_\mathsf{H}$ are valid and $\mathsf{T.Check}(v, \mathsf{sk}, \mathsf{bsn}) = 1$. If not output 0. Otherwise, extract (tpk, x, r) from $\mathsf{ß}_\mathsf{T}$ and $(\mathsf{cre} = (\mathsf{tpk}', \mathsf{sig}), r')$ from $\mathsf{ß}_\mathsf{H}$. If $\mathsf{tpk} \neq \mathsf{tpk}'$ output 0. Otherwise, output 1.

Due to the extractability of SPK_2 and SPK_3, and the binding property of the commitment scheme, we have $\mathsf{tpk} = \mathsf{tpk}'$ with overwhelming probability. Hence, $|\Pr[\mathsf{E}_2] - \Pr[\mathsf{E}_3]| = \mathsf{negl}(\lambda)$.

GAME 4. In this game we further modify how the challenger answers when queried the oracle $\mathsf{IdentifyO}$. Namely, we internally run the following algorithm $\mathsf{Identify}_2$.

$\mathsf{Identify}_2(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk})$: Same as $\mathsf{Identify}_1$ except that it further checks whether $\mathsf{tpk} \in L_{\mathsf{HT}} \cup L_{\mathsf{CP}}$ at the end. If not output 0. Otherwise output 1.

To detect the difference between Game 3 and Game 4, an adversary $\mathcal{A}$ must make a query such that $\mathsf{Identify}_1$ outputs 1 and $\mathsf{Identify}_2$ output 0. That is, query a signature such that when the challenger extracts tpk from the signature it results in $\mathsf{tpk} \notin L_{\mathsf{HT}} \cup L_{\mathsf{CP}}$. However, due to extractability and soundness of SPK_3, this means $\mathcal{A}$ outputs a valid signature sig for the (message) $\mathsf{tpk} \notin L_{\mathsf{HT}} \cup L_{\mathsf{CP}}$ which the challenger has not singed on behalf of the issuer during JoinO or IssueO. Therefore, if such an $\mathcal{A}$ existed, it can be used to break the underlying signature scheme $\mathcal{S}$. Hence, $|\Pr[\mathsf{E}_3] - \Pr[\mathsf{E}_4]| = \mathsf{negl}(\lambda)$

GAME 5. In this game we make a final modification to how the challenger answers when queried the oracle $\mathsf{IdentifyO}$. Namely, we internally run the following algorithm $\mathsf{Identify}_3$.

$\mathsf{Identify}_3(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk})$: If $\mathsf{sk} \neq \mathsf{HSK}[i]$ for any $i \in \mathsf{HT}$, then output $\mathsf{Identify}_2(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{sk})$. Otherwise, if $\mathsf{sk} = \mathsf{HSK}[i]$ for some $i \in \mathsf{HT}$, we slightly modify $\mathsf{Identify}_2$. Instead of checking whether $\mathsf{T.Check}(v, \mathsf{HSK}[i], \mathsf{bsn}) = 1$, $\mathsf{Identify}_3$ checks, right after it parses σ, whether the challenger has ever generated $\mathfrak{ß}_{\mathsf{T}}$ with respect to TPM_i during SignO. If outputs 0 if not and proceeds with the remaining checks if it has.

To detect the difference between Game 4 and Game 5, an adversary $\mathcal{A}$ must make a query such that

$$\big(\mathsf{T.Check}(v, \mathsf{HSK}[i], \mathsf{bsn}) = 0 \wedge \ \mathfrak{ß}_{\mathsf{T}} \text{ was generated w.r.t. } \mathrm{TPM}_i\ \big) \quad \text{or} \quad (3.1)$$

$$\big(\mathsf{T.Check}(v, \mathsf{HSK}[i], \mathsf{bsn}) = 1 \wedge \ \mathfrak{ß}_{\mathsf{T}} \text{ was never generated w.r.t. } \mathrm{TPM}_i\big). \quad (3.2)$$

We show in Lemma 3.5.8 that the above query can only happen with negligible probability assuming that F is injective and one-way with respect to the collision resistant tag scheme. Hence, $|\Pr[\mathsf{E}_4] - \Pr[\mathsf{E}_5]| = \mathsf{negl}(\lambda)$. We postpone the proof of Lemma 3.5.8 so as not to interrupt the main proof.

From the above argument, we obtain $\epsilon = \Pr[\mathsf{E}_0] = \mathsf{negl}(\lambda)$ if $\Pr[\mathsf{E}_5] = \mathsf{negl}(\lambda)$. Therefore, for the rest of the proof we prove that no $\mathcal{A}$ can win the unforgeability game in Game 5 with more than negligible probability. Recall that the adversary wins the unforgeability game if it outputs a valid signature σ on message m and basename bsn, such that either one of the condition holds:

- there exists an honest platform i s.t. $\mathsf{Identify}_3(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{HSK}[i]) = 1$ and $(i, \mathsf{m}, \mathsf{bsn}, \perp) \notin \mathsf{SL}$;
- $\forall i \in \mathsf{HT}, \mathsf{Identify}_3(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{HSK}[i]) = 0$ and $\forall j \in \mathsf{CP}, \mathsf{Identify}_3(\sigma, \mathsf{m}, \mathsf{bsn}, x_j) = 0$, where $x_j \leftarrow \mathsf{Ext}(\mathsf{sp}, \mathsf{td}, \mathsf{ipk}, \mathsf{isk}, \mathsf{TS}[j])$.

Here, we replace the original $\mathsf{Identify}$ algorithm in the winning condition by our modified $\mathsf{Identify}_3$ algorithm. Due to the above argument, this change in the winning condition has negligible difference in $\mathcal{A}$'s advantage since otherwise $\mathcal{A}$ can be used to distinguish between Game 2 and Game 5.

First, we show that an adversary making the former type of forgery does not exist by definition. Notice that due to $(i, \mathsf{m}, \mathsf{bsn}, \perp) \notin \mathsf{SL}$, the challenger has never generated $\mathfrak{B}_\mathsf{T}$ (which is part of the forgery σ) with respect to TPM_i. Then by definition we have $\mathsf{Identify}_3 = 0$. However, this contradicts the winning condition $\mathsf{Identify}_3(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{HSK}[i]) = 1$. Hence, in Game 5, no adversary can win by making the former type of forgery.

Next, let us assume $\mathcal{A}$ makes the latter type of forgery. We consider the scenarios in which $\mathsf{Identify}_3$ outputs 0 for all $\mathsf{sk}_i \in \mathsf{HSK}$ and $x_j \in \mathsf{CSK}$ and show that this cannot happen with non-negligible probability. First of all, since σ is a valid signature, $\mathfrak{B}_\mathsf{T}$ and $\mathfrak{B}_\mathsf{H}$ are valid. Let us extract (tpk, x, r) from $\mathfrak{B}_\mathsf{T}$ and $(\mathsf{cre} = (\mathsf{tpk}', \mathsf{sig}), r')$ from $\mathfrak{B}_\mathsf{H}$. Then following the same argument used to move from Game 2 to Game 5, we have $\mathsf{tpk} = \mathsf{tpk}'$, $r = r'$, and $\mathsf{tpk} \in L_\mathsf{HT} \cup L_\mathsf{CP}$ with overwhelming probability. Since F is injective, the unique secret key associated to tpk is x and hence $x = \mathsf{HSK}[i]$ for some $i \in \mathsf{HT}$ or $x = x_j$ for some $j \in \mathsf{CP}$. In either case, we have $\mathsf{T.Check}(v, x, \mathsf{bsn}) = 1$ due to correctness of extraction. Therefore, at this point, the only situation which triggers $\mathsf{Identify}_3$ to output 0 is if the forgery made by $\mathcal{A}$ satisfies Eq. 3.4 above. However, as we show in Lemma 3.5.8, no PPT adversary has non-negligible advantage. In summary, any such $\mathcal{A}$ must have negligible probability.

Finally, to conclude the proof, it remains to prove Lemma 3.5.8.

Lemma 3.5.8. *For all PPT adversary, we have* $|\Pr[\mathsf{E}_4] - \Pr[\mathsf{E}_5]| = \mathsf{negl}(\lambda)$.

Proof. We prove that no PPT adversary $\mathcal{A}$ can make queries which satisfy Eq (3.3) or Eq (3.4). First of all, the former case can never happen. This is because if the challenger generated $\mathfrak{B}_\mathsf{T}$ with respect to TPM_i, then we must have $\mathsf{T.Check}(v, \mathsf{HSK}[i], \mathsf{bsn}) = 1$ by construction. Therefore, below we focus on the latter case. Specifically, we construct an adversary $\mathcal{B}$ which wins the injective one-wayness of F with respect to the tag scheme by using $\mathcal{A}$ which makes queries satisfying Eq (3.4) as a subroutine. The description of $\mathcal{B}$ is as follows.

$\mathcal{B}^{\mathsf{F}(\cdot),\mathsf{Tag}(x^*,\cdot)}(y^*)$: Let Q be the upper bound on the number of honest TPMs constructed by $\mathcal{A}$. $\mathcal{B}$ samples $i^* \leftarrow [Q]$ and implicitly sets x and y as the secret and public TPM key of TPM_{i^*}, respectively. Namely, $\mathsf{HSK}[i^*] = x^*$ and $\mathsf{tpk}_{i^*} = y^*$. It honestly prepares the TPM keys for all other honest TPMs. Notice that $\mathcal{B}$ can respond identically to the Game 4 and Game 5 challenger to all queries made to oracles other than $\mathsf{IdentifyO}$. This is because it either knows all the secrets or because it can simulate them using $\mathsf{SPK.SimSetup}$ and the oracles $\mathsf{F}(\cdot)$ and $\mathsf{Tag}(x^*,\cdot)$ it is provided with.

When $\mathcal{A}$ queries oracle $\mathsf{IdentifyO}$ on input $(i,\sigma,\mathsf{m},\mathsf{bsn})$, $\mathcal{B}$ first parses σ as $(c,v,\mathsf{ß}_\mathsf{T},\mathsf{ß}_\mathsf{H})$. If $i \neq i^*$, then it simply runs $\mathsf{Identify}_2(\sigma,\mathsf{m},\mathsf{bsn},\mathsf{HSK}[i])$. If $i = i^*$, it checks whether it has ever generated $\mathsf{ß}_\mathsf{T}$ with respect to TPM_{i^*}. If it has, output 1. Otherwise, it extracts (tpk,x,r) from $\mathsf{ß}_\mathsf{T}$ and $(\mathsf{cre} = (\mathsf{tpk}',\sigma),r')$ from $\mathsf{ß}_\mathsf{H}$ and checks $\mathsf{tpk} = \mathsf{tpk}'$ and $\mathsf{tpk} \in L_{\mathsf{HT}} \cup L_{\mathsf{CP}}$. If any of the checks fail, output 0. Otherwise, it checks whether $\mathsf{F}(x) = \mathsf{tpk}_{i^*}$. If so, then abort the game and output x as the preimage of F. Otherwise, output 0.

Due to the injectivity of F and soundness of SPK_2, if $\mathsf{F}(x) \neq \mathsf{tpk}_{i^*}$, then we have $\mathsf{HSK}[i^*] \neq x$. Moreover, due to the collision resistance of the tag scheme, we must have $\mathsf{T.Check}(v,\mathsf{HSK}[i^*],\mathsf{bsn}) = 0$ since the extracted witness satisfies $\mathsf{T.Check}(v,x,\mathsf{bsn}) = 1$. Therefore, unless $\mathcal{B}$ aborts in the above simulation, it simulates the view of Game 4 and Game 5 perfectly to $\mathcal{A}$. On the other hand, $\mathcal{B}$ aborts (i.e., win the injective one-wayness game) whenever $\mathcal{A}$ succeeds in making a query satisfying Eq (3.4) with respect to TPM_{i^*}. However, by assumption, $\mathcal{B}$ can only have negligible advantage. Therefore, $\mathcal{A}$ cannot make such queries with more than negligible probability. □

□

NON-FRAMEABILITY OF $\mathsf{DAA}_{\mathsf{Gen}}$.

Lemma 3.5.9 (Non-Frameability)**.** *Assume that the signatures of knowledge* SPK_1, SPK_2, *and* SPK_3 *are simulatable and online extractable, the tag system* $\mathcal{T}$ *is linkable and collision resistant, the commitment scheme* $\mathcal{C}$ *is binding, and* F *is one-way. Then the DAA system* $\mathsf{DAA}_{\mathsf{Gen}}$ *satisfies non-frameability, as defined in Section 3.3.3.*

Proof. Let $\mathcal{A}$ be an adversary that breaks the non-frameability game with advantage ϵ. We consider the following game sequence and denote the event on which $\mathcal{A}$ wins in Game i as E_i. We first present the game sequence

and then show that the advantage of $\mathcal{A}$ changes only negligibly in adjacent games.

GAME 0. This is the original non-frameability game. By assumption, we have $\Pr[\mathsf{E}_0] = \epsilon$.

GAME 1. In this game we modify the challenger so that it uses $\mathsf{SPK}_1.\mathsf{SimSign}$, $\mathsf{SPK}_2.\mathsf{SimSign}$, and $\mathsf{SPK}_3.\mathsf{SimSign}$ instead of $\mathsf{SPK}_1.\mathsf{Sign}$, $\mathsf{SPK}_2.\mathsf{Sign}$ and $\mathsf{SPK}_3.\mathsf{Sign}$. In particular, when the adversary queries oracle JoinO and SignO, the challenger simply uses $\mathsf{SPK}_1.\mathsf{SimSign}$, $\mathsf{SPK}_2.\mathsf{SimSign}$, and $\mathsf{SPK}_3.\mathsf{SimSign}$, to generate π_{join}, π_{T}, and π_{H} respectively. This game is indistinguishable from Game 0 due to the simulatability of the SPKs. Namely, $|\Pr[\mathsf{E}_0] - \Pr[\mathsf{E}_1]| = \mathsf{negl}(\lambda)$.

GAME 2. In this game we modify how the challenger answers when queried the oracle $\mathsf{IdentifyO}$. In particular, we modify the algorithm $\mathsf{Identify}$ run internally in $\mathsf{IdentifyO}$. In the previous game, algorithm $\mathsf{Identify}$ was defined as follows.

$\mathsf{Identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{HSK}[i])$: Parse σ as $(c, v, \mathsf{ß}_{\mathsf{T}}, \mathsf{ß}_{\mathsf{H}})$ and return 1 if $\mathsf{T}.\mathsf{Check}(v, \mathsf{HSK}[i], \mathsf{bsn}) = 1$, and $\mathsf{ß}_{\mathsf{T}}$ and $\mathsf{ß}_{\mathsf{H}}$ are valid. Otherwise, return 0.

Here, since $\mathsf{Identify}$ is only ever invoked with respect to an honest TPM throughout the game, we assume without loss of generality that it is provided $\mathsf{HSK}[i]$ for some $i \in \mathsf{HP}$ as input. In this game, the challenger replaces $\mathsf{identify}$ by the following algorithm $\mathsf{Identify}_1$.

$\mathsf{Identify}_1(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{HSK}[i])$: Parse σ as $(c, v, \mathsf{ß}_{\mathsf{T}}, \mathsf{ß}_{\mathsf{H}})$ and check $\mathsf{ß}_{\mathsf{T}}$ and $\mathsf{ß}_{\mathsf{H}}$ are valid and $\mathsf{T}.\mathsf{Check}(v, \mathsf{HSK}[i], \mathsf{bsn}) = 1$. If not output 0. Otherwise, extract (tpk, x, r) from $\mathsf{ß}_{\mathsf{T}}$ and $(\mathsf{cre} = (\mathsf{tpk}', \mathsf{sig}), r')$ from $\mathsf{ß}_{\mathsf{H}}$. If $\mathsf{tpk} \neq \mathsf{tpk}'$ output 0. Otherwise, output 1.

Due to the extractability of SPK_2 and SPK_3, and the binding property of the commitment scheme, we have $\mathsf{tpk} = \mathsf{tpk}'$ with overwhelming probability. Hence, $|\Pr[\mathsf{E}_1] - \Pr[\mathsf{E}_2]| = \mathsf{negl}(\lambda)$.

GAME 3. In this game, we further modify how the challenger answers when queried the oracle $\mathsf{IdentifyO}$. Namely, we run the following algorithm $\mathsf{Identify}_2$.

$\mathsf{Identify}_2(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{HSK}[i])$: Defined exactly as $\mathsf{Identify}_1$, except that in case $\mathsf{Identify}_1$ outputs 1, it further checks if the extracted tpk satisfies $\mathsf{tpk} = \mathsf{tpk}_i$. If so, output 1 and otherwise output 0.

This game is indistinguishable from the previous game due to the collision resistance of the tag scheme. In particular, assume an adversary made a query to oracle $\mathsf{IdentifyO}$ such that $\mathsf{Identify}_1$ outputs 1 but $\mathsf{Identify}_2$ outputs 0. Let tpk and x be the witness extracted from such a query. By assumption and the infectivity of F, we have $x \neq \mathsf{HSK}[i]$. Also, by soundness of SPK_2, we have $\mathsf{T.Check}(v, x, \mathsf{bsn}) = 1$. Hence, since we have $\mathsf{T.Check}(v, \mathsf{HSK}[i], \mathsf{bsn}) = 1$ conditioned on $\mathsf{Identify}_1$ outputting 1, $(\mathsf{bsn}, x, \mathsf{HSK}[i], v)$ constitutes as a valid attack for the collision resistance of the tag scheme. Hence, we have $|\Pr[\mathsf{E}_2] - \Pr[\mathsf{E}_3]| = \mathsf{negl}(\lambda)$.

GAME 4. In this game, we further modify how the challenger answers when queried the oracle $\mathsf{IdentifyO}$. Namely, we internally run the following algorithm $\mathsf{Identify}_2$.

$\mathsf{Identify}_3(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{HSK}[i])$: Same as $\mathsf{Identify}_2$ except that instead of checking whether $\mathsf{T.Check}(v, \mathsf{HSK}[i], \mathsf{bsn}) = 1$, the challenger now checks that it previously signed π_{T} at the beginning, outputting 0 if not and proceeding with the remaining checks if so.

To detect the difference between Game 3 and Game 4, an adversary $\mathcal{A}$ must make a query such that

$$\big(\mathsf{T.Check}(v, \mathsf{HSK}[i], \mathsf{bsn}) = 0 \wedge \ \text{ß}_{\mathsf{T}} \text{ was generated w.r.t. TPM}_i\ \big) \quad \text{or} \tag{3.3}$$

$$\big(\mathsf{T.Check}(v, \mathsf{HSK}[i], \mathsf{bsn}) = 1 \wedge \ \text{ß}_{\mathsf{T}} \text{ was never generated w.r.t. TPM}_i\big). \tag{3.4}$$

The proof that neither of these cases can arise with more than negligible probability is almost identical to that for Lemma 3.5.8. The only difference is that the check for tpk we had for Lemma 3.5.8 (i.e., $\mathsf{tpk} \in L_{\mathsf{HT}} \cup L_{\mathsf{CP}}$) is replaced by $\mathsf{tpk} = \mathsf{tpk}_i$ in this proof. The proof again assumes that F is injective and one-way with respect to the collision resistant tag scheme. Hence, $|\Pr[\mathsf{E}_3] - \Pr[\mathsf{E}_4]| = \mathsf{negl}(\lambda)$.
From the above argument, we obtain $\epsilon = \Pr[\mathsf{E}_0] = \mathsf{negl}(\lambda)$ if $\Pr[\mathsf{E}_4] = \mathsf{negl}(\lambda)$. Therefore, for the rest of the proof we prove that no $\mathcal{A}$ can win the non-frameability game in Game 4 with more than negligible probability. Recall that the adversary wins the non-frameability game if it outputs a TPM identity i, a valid signature σ on message m, and basename bsn, such that $(i, \mathsf{m}, \mathsf{bsn}, *) \notin \mathsf{SL}$ and either one of the following conditions holds:

- There exists $(i, \mathsf{m}^*, \mathsf{bsn}, \sigma^*) \in \mathsf{SL}$ s.t. $\mathsf{Link}(\sigma, \sigma^*, \mathsf{m}, \mathsf{m}^*, \mathsf{bsn}) = 1$;
- Or, if $\mathsf{bsn} = \perp$ and $\mathsf{Identify}_2(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{HSK}[i]) = 1$.

Here, we replace the original $\mathsf{Identify}$ algorithm in the winning condition by our modified $\mathsf{Identify}_2$ algorithm. Due to the above argument, this change in the winning condition has negligible difference in $\mathcal{A}$'s advantage since otherwise $\mathcal{A}$ can be used to distinguish between Game 2 to Game 4.

First, we show the easy case that an adversary making the second type of forgery does not exist. Notice that due to $(i, \mathsf{m}, \mathsf{bsn}, *) \notin \mathsf{SL}$, the challenger has never generated $\mathsf{ß}_\mathsf{T}$ (which is part of the output σ) with respect to TPM_i. Then by definition $\mathsf{Identify}_2$ outputs 0. However, this contradicts the winning condition $\mathsf{Identify}_2(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{HSK}[i]) = 1$. Hence, in Game 4, no adversary can win by making the latter type of output.

Now consider an adversary $\mathcal{A}$ winning by an output of the first type. Parse σ as $(c, v, \mathsf{ß}_\mathsf{T}, \mathsf{ß}_\mathsf{H})$. By definition of algorithm Link, we have $\mathsf{T.Link}(v, v^*) = 1$ for some $(i, \mathsf{m}^*, \mathsf{bsn}, \sigma^* = (c^*, v^*, \mathsf{ß}_\mathsf{T}{}^*, \mathsf{ß}_\mathsf{H}{}^*)) \in \mathsf{SL}$. Now, due to the extractability of SPK_2, we are able to extract tpk and x from $\mathsf{ß}_\mathsf{T}$ such that $\mathsf{F}(x) = \mathsf{tpk}$ and $\mathsf{T.Check}(v, x, \mathsf{bsn}) = 1$. Moreover, since σ^* was honestly generated by $\mathcal{B}$, there is a corresponding tpk_i and $x_i = \mathsf{HSK}[i]$ such that $\mathsf{F}(x_i) = \mathsf{tpk}_i$ and $\mathsf{T.Check}(v^*, x_i, \mathsf{bsn}) = 1$. Therefore, in case $\mathsf{tpk} \neq \mathsf{tpk}_i$, we can simply reduce it to the linkability security of the tag scheme. Namely, $(\mathsf{bsn}, v, x, \mathsf{bsn}, v^*, x_i)$ forms a valid output for an adversary against the linkability game. On the other hand, in case $\mathsf{tpk} = \mathsf{tpk}_i$, we can make similar arguments made to show indistinguishability of Game 3 and Game 4 and reduce it to the injective and one-wayness of F with respect to the collision resistant tag scheme. In particular, the reduction algorithm guesses the i for which $\mathsf{tpk} = \mathsf{tpk}_i$, and simulate the Game 4 challenger without knowledge of the corresponding x_i (which it can do due to the changes we made to algorithm $\mathsf{Identify}_3$). □

LINK-IDENTIFY DEPENDENCE OF DAA_{Gen}.

Lemma 3.5.10 (Link-Identify Dependence)**.** *Assume that the hypotheses of Lemma 3.5.2 hold and $\mathcal{T}$ is a correct and linkable tag system. Then, the DAA system $\mathsf{DAA}_\mathsf{Gen}$ satisfies the requirement in Item 2 of Definition 3.3.11.*

Proof. Let $(\sigma, \mathsf{m}, \mathsf{bsn})$ and $(\sigma', \mathsf{m}', \mathsf{bsn}')$ are two valid signature-message-basename tuple with $\mathsf{bsn} \neq \perp$.

Suppose that $\mathsf{Link}(\sigma, \sigma', \mathsf{m}, \mathsf{m}', \mathsf{bsn}) = 1$. Let v and v' be the tags associated with σ and σ', respectively. Then, since the scheme satisfies Definition 3.3.4, there exist secret keys x, x' such that $\mathsf{Identify}(\sigma, \mathsf{m}, \mathsf{bsn}, x) = \mathsf{Identify}(\sigma', \mathsf{m}', \mathsf{bsn}, x') = 1$. This implies that $\mathsf{T.Check}(v, x, \mathsf{bsn}) = \mathsf{T.Check}(v', x', \mathsf{bsn}') = 1$. Moreover, we have $\mathsf{T.Link}(v, v') = 1$. It follows from the linkability of the tag system $\mathcal{T}$ that $x = x'$.

In the other direction, suppose that there exists a signing key x such that $\mathsf{Identify}(\sigma, \mathsf{m}, \mathsf{bsn}, x) = \mathsf{Identify}(\sigma', \mathsf{m}', \mathsf{bsn}, x) = 1$. Then $\mathsf{T.Check}(v, x, \mathsf{bsn}) = \mathsf{T.Check}(v', x, \mathsf{bsn}') = 1$. By the correctness of the tag system $\mathcal{T}$, we deduce that $\mathsf{Link}(\sigma, \sigma', \mathsf{m}, \mathsf{m}', \mathsf{bsn}) = 1$. □

4

AN EFFICIENT LATTICE-BASED DAA SCHEME

4.1 INTRODUCTION

In this chapter, we move our focus from the security definition of DAA to designing an *efficient* lattice-based construction. As mentioned in Section 1.1.1, this has been a long-standing open problem. Prior to this, proposed schemes have either provide only asymptotic security analysis or, where concrete parameters are provided, they are computed in a very heuristic manner. To date, there have been no post-quantum DAA constructions instantiated with truly practical parameters.

Our scheme takes inspiration from [BLNS23] in which the authors give a framework for anonymous credentials. We will show one can modify their framework to use module-lattice throughout and furthermore, demonstrate that the recent zero-knowledge proof operations in [LNP22] can be split between two proving parties whilst preserving the key confidentiality of TPM key material.

We begin by covering some of the preliminary building blocks used in our construction.

4.2 PRELIMINARIES

4.2.1 *Notation*

We introduce some notation used in this chapter. When using hash functions, we will denote their image space using subscripts. For example, H_{R_q} maps an input to an element of R_q. Give a basis B, we denote its Gram-Schmidt orthogonalisation by $\tilde{B}$.

4.2.2 *The Interactive NTRU-ISIS-f Assumption*

We recall the falsifiable (interactive) $\mathsf{Int\text{-}NTRU\text{-}ISIS}_f$ assumption as introduced by Bootle et. al in [BLNS23]. Note the slight modification we make to the relation a winning output must satisfy. In particular, the vector $\mathbf{1}^\top$ is used to compress the right hand side of the relation into a single ring

element. This is needed in order to apply the trapdoor sampling techniques of [CPS+20] in the *module* NTRU setting.

Definition 4.2.1 (The Int-MNTRU-ISIS$_f$ Problem (Interactive))**.** *Define the system parameters* $\mathsf{sp} = (q, d, n, m, N, \sigma, B_s, B_m)$ *as a tuple of functions of the security parameter* λ*. Let* $f : [N] \to R_q^n$ *be an efficiently computable function. We define the advantage of an adversary* $\mathcal{A}$ *in solving the* Int-MNTRU-ISIS$_f$ *problem as*

$$\mathcal{A}_{\mathsf{sp}}^{\mathsf{Int\text{-}MNTRU\text{-}ISIS}_f}(\mathcal{A}) = \Pr\left[\mathsf{Exp}_{\mathsf{sp}}^{\mathsf{Int\text{-}MNTRU\text{-}ISIS}_f}(\mathcal{A}) \to 1\right],$$

where the experiment $\mathsf{Exp}_{\mathsf{sp}}^{\mathsf{Int\text{-}MNTRU\text{-}ISIS}_f}(\mathcal{A})$ *is defined in Section 4.2.2. The* Int-MNTRU-ISIS$_f$ *assumption states that for every PPT adversary* $\mathcal{A}$*,* $\mathcal{A}_{\mathsf{sp}}^{\mathsf{Int\text{-}MNTRU\text{-}ISIS}_f}(\mathcal{A})$ *is negligible in* λ*.*

$\underline{\mathsf{Exp}_{\mathsf{sp}}^{\mathsf{Int\text{-}MNTRU\text{-}ISIS}_f}(\mathcal{A})}$

1: $(\mathbf{h}, \mathbf{B}_{\mathbf{F},\mathbf{g}}) \leftarrow \mathsf{NTRU.TrapGen}(q, n, d)$

2: $\mathbf{A} := \begin{bmatrix} 1 & \mathbf{h}^T \end{bmatrix}$

3: $\mathbf{C} \leftarrow R_q^{n \times (n+m)}$

4: $\mathcal{M} = \emptyset$

5: $(x^*, \mathbf{s}^*, \mathbf{m}^*, \mathbf{r}^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{pre}}}(\mathbf{A}, \mathbf{C})$

6: **if** $(\mathbf{m}^* \notin \mathcal{M})$
$\wedge \left(\mathbf{A}\mathbf{s}^* = \mathbf{1}^\top \left(f(x^*) + \mathbf{C}\begin{bmatrix}\mathbf{m}^* \\ \mathbf{r}^*\end{bmatrix}\right)\right)$
$\wedge\, (0 < \|\mathbf{s}^*\| \leq B_s) \wedge (\|\mathbf{m}^*\|, \|\mathbf{r}^*)\| \leq B_m)$

7: **then Return** 1

8: **ElseReturn** 0

$\underline{\mathcal{O}_{\mathsf{pre}}(\mathbf{m}, \mathbf{r})}$

1: **if** $\|\mathbf{m}, \mathbf{r}\| < B_m$

2: **then Return** $\perp$

3: $x \leftarrow [N]$

4: $\mathbf{s} \leftarrow \mathbf{A}_s^{-1}\left(\mathbf{1}^\top \left(f(x) + \mathbf{C}\begin{bmatrix}\mathbf{m} \\ \mathbf{r}\end{bmatrix}\right)\right)$

5: $\mathcal{M} \leftarrow \mathcal{M} \cup \{\mathbf{m}\}$

6: **Return** $(x, \mathbf{s})$

FIGURE 4.1: The interactive ISIS_f experiment

Concrete instantiation of f. We instantiate f, as in [BLNS23], using a binary encoding function parameterized by an integer $t \in \mathbb{N}$ and a matrix $B \in \mathbb{Z}_q^{nd \times t}$. The function f is then defined as

$$f(x) := \Phi^{-1}(B \cdot \mathsf{bin}(x)) \in R_q^n,$$

where $\mathsf{bin} \in \{0,1\}^t$ is a binary decomposition of $x - 1$. Hence, $N = 2^t$. It is easy to see that proving knowledge of a preimage x of f, i.e. $f(x) = \mathbf{y}$, is

equivalent to proving knowledge of binary $\vec{u} \in \{0,1\}^t$ such that $B\vec{u} = \Phi(\mathbf{y})$. This is the formalisation we will use in order to apply the proof techniques of [LNP22].

4.2.3 *NTRU Lattices*

In this section, and wherever NTRU trapdoor sampling is referred to in this work, we consider a ring $\hat{R} := \mathbb{Z}[X]/(X^{\hat{d}} + 1)$ of dimension $\hat{d}$. This is to distinguish between the ring over which M-LWE and MSIS are defined. This distinction will be important when eventually setting parameters.

Let $\hat{d}$ be a power of two, q be a positive integer, $\mathbf{g} \in \hat{R}_q^{\hat{n}}$, and $\mathbf{F} \in \hat{R}_q^{\hat{n}\times\hat{n}}$ be invertible. Let $\mathbf{h} = \mathbf{F}^{-1}\mathbf{g} \in \hat{R}_q^{\hat{n}}$ and define the $NTRU$ module as

$$\mathcal{L}_{\mathsf{NTRU}} := \{(u, \mathbf{v}) \in \hat{R}^{\hat{n}+1} \,:\, u + \mathbf{v}^T\mathbf{h} = \quad \bmod q\}.$$

$\mathcal{L}_{\mathsf{NTRU}}$ admits bases in the form of

$$\mathbf{B}_{\mathsf{NTRU}} := \begin{pmatrix} -\mathbf{h} & \mathbf{I}_{\hat{n}} \\ q & \mathbf{0}_{\hat{n}} \end{pmatrix} \text{ and } \mathbf{B}_{\mathbf{F},\mathbf{g}} := \begin{pmatrix} \mathbf{g} & -\mathbf{F} \\ g_0 & -\mathbf{f}_0^T \end{pmatrix}.$$

As with any module, $\mathcal{L}_{\mathsf{NTRU}}$ can be seen as a $\mathbb{Z}$-lattice over $\mathbb{Q}^{(\hat{n}+1)\hat{d}}$.

Lemma 4.2.2 ([CPS+20])**.** *There is an efficient algorithm* $\mathsf{NTRU.TrapGen}(q, \hat{d}, \hat{n})$ *which outputs* $\mathbf{h}$ *and a basis* $\mathbf{B}$ *of* $\mathcal{L}_{\mathsf{NTRU}}$ *such that* $\|\tilde{\mathbf{B}}\|_2 \leq \gamma \cdot q^{1/(\hat{n}+1)}$*, where* $\gamma = 1.17$ *for* $\hat{n} = 1, 2$ *and* $\gamma = 1.24$ *for* $\hat{n} = 3$*.*

Lemma 4.2.3 ([CPS+20])**.** *Let* $\lambda \in \mathbb{N}$ *and* $\epsilon = 1/(4\sqrt{\lambda})$*. There is a PPT algorithm* $\mathsf{GSampler}$*, which takes* $(\mathbf{h}, \mathbf{B}_{\mathbf{F},\mathbf{g}}) \leftarrow \mathsf{NTRU.TrapGen}(\hat{d}, \hat{n}, q)$*, standard deviation* $\sigma > 0$ *and a target vector* $c \in \hat{R}_q$ *as input and outputs* $\mathbf{e} \in \hat{R}_q^{\hat{n}+1}$ *such that*

$$\Delta\left(\begin{bmatrix}1 & \mathbf{h}^T\end{bmatrix}_\sigma^{-1}(c), \mathsf{GSampler}(\mathbf{h}, \mathbf{B}_{\mathbf{F},\mathbf{g}}, \sigma, c)\right) \leq 2^{-\lambda},$$

as long as

$$\sigma \geq \gamma q^{1/(\hat{n}+1)} \cdot \eta_\epsilon(\mathbb{Z}^{(\hat{n}+1)\hat{d}}) \text{ where } \eta_\epsilon(\mathbb{Z}^{(\hat{n}+1)\hat{d}}) \approx \frac{1}{\pi} \cdot \sqrt{\frac{1}{2}\log\left(2\hat{d}(\hat{n}+1)(1+\frac{1}{\epsilon})\right)}.$$

As noted in [CPS+20, Section 3.1], in order to generate a trapdoor basis as in Lemma 4.2.2 with $\|\tilde{\mathbf{B}}\|_2 \leq \gamma \cdot q^{1/(\hat{n}+1)}$, it is important to sample each row of $[\mathbf{f}_{i1}, \ldots, \mathbf{f}_{i\hat{n}}, \mathbf{g}_i]$ according to a discreet Gaussian with standard deviation

$\gamma \cdot \frac{1}{\sqrt{\hat{d}(2+\hat{n}-i)}} \cdot q^{1/(\hat{n}+1)}$. We emphasise that this limits how large we can take $\hat{n}$ since a higher value would lead to a smaller NTRU key and may take us into so-called 'overstretched' parameter regimes as highlighted by the cryptanalysis of [ABD16, Dv21]. We will discuss this in far more detail in Section 5.3 but for now it suffices to bare this in mind when setting parameters for out DAA scheme.

4.2.4 *The LNP Lattice-Based Proof System*

Here, we recall the non-interactive zero-knowledge proof techniques first introduced by Lyubashevsky, Nguygen, and Plançon [LNP22]. We will use these techniques to prove knowledge of short $\mathbf{s}_1$ satisfying equations of the form

$$\mathbf{P}\mathbf{s}_1 = \mathbf{v} \text{ over } R_q, \tag{4.1}$$

where $\mathbf{P} \in R_q^{n \times m}$ and $\mathbf{v} \in R^m$ are public matrices and vectors (resp.) of elements in R_q. We begin by some preliminary setup and notation.

SETUP. In this work we will work with a modulus $q = p_1 p_2$, a product of two primes of the form $p_i = 5 \mod 8$ and $p_1 < p_2$. We denote by $\tilde{x}$, the constant coefficient of the polynomial x.

Let $\sigma() : R_q \to R_q$ be the ring automorphism defined by $\sigma(X) = X^{-1}$. We extend this notation naturally to vectors $\mathbf{x} \in R_q^n$ by defining $\sigma(\mathbf{x}) = (\sigma(x_1), \dots \sigma(x_n))$ and similarly to matrices. Finally, let $\tau \in \mathbb{N}$ be such that $1/p_1^\tau = \mathsf{negl}(\lambda)$. This will be the parameter used for soundness amplification.

ABDLOP COMMITMENT. We recall the commitment scheme introduced in [LNP22], which combines both the Ajtai [Ajt96] and BDLOP [BDL+18] constructions. The advantage of this hybrid scheme is that it allows one to commit to elements of both small norm (in the Ajtai part) and ones of large norm (in the BDLOP) part and reducing the overall communication cost by their combination. The scheme is parameterized by the natural numbers $k_{\mathsf{MSIS}}, m_1, m_2, \ell \in \mathbb{N}$, with $m_2 \geq k_{\mathsf{MSIS}} + l$. The commitment key (which will be form the common reference string in the proofs) consists of the uniformly random matrices

$$\mathsf{crs} := (\mathbf{A}_1, \mathbf{A}_2, \mathbf{B}) \in R_q^{k_{\mathsf{MSIS}} \times m_1} \times R_q^{k_{\mathsf{MSIS}} \times m_2} \times R_q^{\ell \times m_2}.$$

To commit to a short element $\mathbf{s}_1 \in R_q^{m_1}$ (with small coefficients) as well as a "fully-fledged' element $\mathbf{m} \in R_q^{\ell}$, one sampled $\mathbf{s}_2 \leftarrow \chi$ from some distribution χ^{m_2} over R_q and computes

$$\mathsf{ABDLOP.Com}(\mathsf{crs}, \mathbf{s}_1, \mathbf{m};\, \mathbf{m}_2) := \begin{bmatrix} \mathbf{t}_A \\ \mathbf{t}_B \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{0} \end{bmatrix} \mathbf{s}_1 + \begin{bmatrix} \mathbf{A}_2 \\ \mathbf{B} \end{bmatrix} \mathbf{s}_2 + \begin{bmatrix} \mathbf{0} \\ \mathbf{m} \end{bmatrix}.$$

Looking ahead, when employing the proof techniques of [LNP22] we will commit to the element $\mathbf{s}$ of Equation (4.1) in the Ajtai part (i.e. in position $\mathbf{s}_1$) and commit to so-called 'garbage' terms, used in the proof, in the position $\mathbf{m}$.

CHALLENGE SPACE. We define the challenge space that will be used in the zero-knowledge proofs. Let $\xi, \mu > 0$ and k be a power-of-two. Define the challenge space $\mathcal{C}$ as

$$\mathcal{C} := \{c \in R_q \,:\, \|c\|_\infty \leq \xi \wedge \sigma(c) = c \wedge \sqrt[2k]{\|c^{2k}\|_1} \leq \nu\}.$$

We use the experimental values from [BLNS23], which ensure that for c sampled uniformly from the space of elements bounded by ξ in infinity norm, it holds that $\sqrt[2k]{\|c^{2k}\|_1} \leq \nu$ with probability at least 99%.

d	ξ	ν	k	$\lvert\mathcal{C}\rvert$
64	8	140	32	2^{129}
128	2	59	32	2^{147}

TABLE 4.1: Example parameters for defining $\mathcal{C}$ for a modulus q whose smallest prime divisor is larger than 16.

We are now ready to state the hiding and binding properties of the ABDLOP commitment scheme [LNP22, Lemma 3.1].

Definition 4.2.4 (Relaxed opening)**.** *A relaxed opening of the ABDLOP commitment* $(\mathbf{t}_A, \mathbf{t}_B)$ *with respect to a commitment key* crs *is a tuple* $(\mathbf{s}_1, \mathbf{m}, \mathbf{s}_2, c) \in R_q^{m_1} \times R_q^{\ell} \times R_q^{m_2} \times \bar{\mathcal{C}}$ *satisfying*

$$\mathsf{ABDLOP.Com}(\mathsf{crs}, \mathbf{s}_1, \mathbf{m};\, \mathbf{s}_2) = (\mathbf{t}_A, \mathbf{t}_B) \text{ with}$$
$$\|c\mathbf{s}_1\|_2 \leq B_1 \text{ and } \|c\mathbf{s}_2\|_2 \leq B_2.$$

Lemma 4.2.5 (Binding)**.** *Let $\xi < p_1/2$ where p_1 is the smallest prime dividing q. Then the ADBLOP commitment is computationally binding with respect to relaxed openings under the $\mathsf{MSIS}_{k_{\mathsf{MSIS}},m_1+m_2,B}$ assumption, where $B := 4\nu\sqrt{B_1^2 + B_2^2}$.*

Lemma 4.2.6 (Hiding)**.** *The* ABDLOP *commitment scheme is computaitonally hiding under the $\mathsf{M\text{-}LWE}_{k_{\mathsf{MSIS}}+\ell,m_2,\chi,q}$ assumption.*

APPROXIMATE RANGE PROOFS. As part of the overarching zero-knowledge argument, we will want to prove that $\mathbf{s}$ of Equation (4.1) has *quite* small coefficients. That is, will give an approximate range proof for the norm of $\mathbf{s}$.

We recall the results in [GHL22, LNP22] which allow us to bound the operator norm of a matrix $\mathfrak{R} \leftarrow \mathsf{Bin}_{\xi}^{256\times m}$, where Bin is the binomial distribution with positive integer parameter ξ defined as $\sum_{i=1}^{\xi}(a_i - b_i)$, where $a_i, b_i \xleftarrow{\$} \{0,1\}$. Let us define the functions $\omega_{\min}(\lambda)$ and $\omega_{\max}(\lambda)$ such that for any $\vec{w} \in \mathbb{Z}^m$ and any $\xi \in \mathbb{N}$ we have

$$\Pr_{\mathfrak{R}\leftarrow\mathsf{Bin}_{\xi}^{256\times m}}[\|\mathfrak{R}\vec{w}\|_2^2 > \xi \cdot \|\vec{w}\|_2^2 \cdot \omega_{\min}(\lambda)^2] \leq 2^{-2\lambda},$$

$$\Pr_{\mathfrak{R}\leftarrow\mathsf{Bin}_{\xi}^{256\times m}}[\|\mathfrak{R}\vec{w}\|_2^2 < \xi \cdot \|\vec{w}\|_2^2 \cdot \omega_{\max}(\lambda)^2] \leq 2^{-\lambda}.$$

As shown in [GHL22], adopting a few natural heuristic assumptions, that $\omega_{\min}(128) = \sqrt{13}$ and $\omega_{\max}(128) = \sqrt{337}$.

Finally, we use the following result from [LNP22, Lemma2.9], which allows will allow one to verify the approximate norm of $\mathbf{s}$, given the response in the approximate range proof.

Lemma 4.2.7. *Fix $m, P \in \mathbb{N}$, and a bound $b \leq P/(8\sqrt{2}\cdot\omega_{min}(\lambda)\cdot m)$, and let $\vec{w} \in [\pm P/2]^m$ with $\|\vec{w}\|_2 \geq b$, and let $\vec{y}$ be an arbitrary vector in $[\pm P/2]^m$. Then*

$$\Pr_{\mathfrak{R}\leftarrow\mathsf{Bin}_{1}^{256\times m}}\left[\|\mathfrak{R}\vec{w} + \vec{y} \mod P\|_2 < \frac{b}{\sqrt{2}}\cdot\omega_{min}(\lambda)\right] < 2^{-\lambda}.$$

noindent We are now ready to describe the non-interactive zero-knowledge proof (NIZK) as introduced in [LNP22] for proving knowledge of a short $\mathbf{s}_1 \in R^{m_1}$ satisfying Equation (4.1). We begin by re-defining the relation Equation (4.1) as a system of linear equations over $\mathbb{Z}_q$. Note, this is done by considering the rotation matrix of $\mathbf{P}$, which we will denote by $P = \mathsf{Rot}(\mathbf{P}) \in$

$\mathbb{Z}_q^{nd \times m_1 d}$ and the coefficient vectors of $\mathbf{s}_1 \in R_q^{m_1}$ and $\mathbf{v} \in R_q^n$ which we denote by $\vec{\mathbf{s}}_1 \in \mathbb{Z}_q^{m_1 d}$ and $\vec{\mathbf{v}} \in \mathbb{Z}_q^{nd}$ respectively so that we aim to prove the equation

$$P\vec{\mathbf{s}_1} = \vec{\mathbf{v}}, \tag{4.2}$$

over $\mathbb{Z}_q$. Thus, using the language of ZKPs, we can define the corresponding general relation

$$\mathcal{R}_{\mathsf{Gen}} := \{x := (P, \vec{\mathbf{v}}), \mathsf{w} := \vec{\mathbf{s}}_1 \mid P \in \mathbb{Z}_q^{nd \times m_1 d}, \vec{\mathbf{v}} \in \mathbb{Z}_q^{nd}, \vec{\mathbf{s}}_1 \in \mathbb{Z}_q^{m_1 d} : P\vec{\mathbf{s}_1} = \vec{\mathbf{v}} \wedge \|\vec{\mathbf{s}}_1\|_2 \leq B_s\}. \tag{4.3}$$

We will denote the following proof system by $\Pi_{\mathsf{NIZK}}^{\mathsf{Gen}}$, where Gen refers to the relation 4.3 to be proven.

We begin by restructuring Equation (4.2) so that it suffices to prove this modified equation whilst also allowing us to prove an exact norm value for $\mathbf{s}_1$ (rather than a range). Observe that, by Lagrange's sum-of-four-squares theorem, $\|\vec{\mathbf{s}_1}\|_2 \leq B_{s_1}$ is equivalent to the existence of four integers $a_1, a_2, a_3, a_4 \in \mathbb{Z}$ such that $B_{s_1}^2 - \|\vec{\mathbf{s}_1}\|_2^2 = a_1^2 + a_2^2 + a_3^2 + a_4^2$. Thus, we can define a vector $a := (a_1, a_2, a_3, a_4, 0, \ldots, 0)^\top \in \mathbb{Z}_q^d$ and observe that

$$\left\| \begin{bmatrix} \vec{\mathbf{s}}_1 \\ \vec{a} \end{bmatrix} \right\|_2 = B_{s_1} \text{ and } \begin{bmatrix} P & 0 \end{bmatrix} \begin{bmatrix} \vec{\mathbf{s}}_1 \\ \vec{a} \end{bmatrix} = P\vec{\mathbf{s}_1}.$$

Let us redefine, via abuse of notation, $\vec{\mathbf{s}_1} := \begin{bmatrix} \vec{\mathbf{s}_1} & \vec{a} \end{bmatrix}^\top$. Furthermore, we redefine $m_1 = m_1 + 1$. We also define the common *random* string consisting of uniformly random matrices which are used of the ABDLOP commitment:

$$\mathsf{crs} := (\mathbf{A}_1, \mathbf{A}_2, \mathbf{B}_y, \mathbf{B}_g, \mathbf{b}) \in R_q^{k_{\mathsf{MSIS}} \times m_1} \times R^{k_{\mathsf{MSIS}} \times m_2} \times R^{256/d \times m_2} \times R^{\tau \times k_{\mathsf{M\text{-}LWE}}} \times R^{m_2},$$

where k_{MSIS} and m_2 are parameters set so that MSIS and $\mathsf{M\text{-}LWE}$ are hard.

We now give a round-by-round description of the procedure prove relation $\mathcal{R}_{\text{GEN}}$.

ROUND 1. In this round, we create an ABDLOP commitment to $\mathbf{s}_1$ as well as commitments to masking vectors $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3$ and to a vector $\mathbf{g}$ with constant coefficients equal to zero. We sample $\mathbf{s}_2 \leftarrow \chi^{m_2}$ where χ is a probability distribution on ternary polynomials in R_q and also $\mathbf{y}_i \leftarrow D_{\sigma_i}^{m_i}$ for $i = 1, 2$ and $\mathbf{y}_3 \leftarrow D_{\sigma_3}^{256}$. Finally we sample a polynomial vector $\mathbf{g} \leftarrow \{x \in R_q : \tilde{x} = 0\}$ and compute the commitments:

$$\mathbf{t}_A := \mathbf{A}_1\mathbf{s}_1 + \mathbf{A}_2\mathbf{s}_2,$$

$$\mathbf{w} := \mathbf{A}_1\mathbf{y}_2 + \mathbf{A}_2\mathbf{y}_2$$
$$\mathbf{t}_y := \mathbf{B}_y\mathbf{s}_2 + \mathbf{y}_3$$
$$\mathbf{t}_g := \mathbf{B}_g\mathbf{s}_2 + \mathbf{g}.$$

The first message and corresponding challenge are then

$$a_1 := (\mathbf{t}_A, \mathbf{t}_y, \mathbf{t}_g, \mathbf{w}) \text{ and}$$
$$(\mathfrak{R}_0, \mathfrak{R}_1) := \mathsf{H}(1, \mathsf{crs}, x, a_1) \in \{0,1\}^{256\times m_1 d} \times \{0,1\}^{256\times m_1 d}.$$

ROUND 2. We now create the response for the approximate proof of shortness for $\mathbf{s}_1$. Define the matrix $\mathfrak{R} := \mathfrak{R}_0 - \mathfrak{R}_1$. Compute the response element $\vec{z}_3 := \vec{\mathbf{y}}_3 = \mathfrak{R}\vec{\mathbf{s}}_1$ and apply rejection sampling on $\vec{z}_3$. The second message and corresponding challenge are then:

$$a_2 := \vec{z}_3 \text{ and}$$
$$(\gamma_{i,j})_{i\in[\tau],j\in[256+nd+1]} := \mathsf{H}(2, \mathsf{crs}, x, a_1, a_2) \in \mathbb{Z}_q^{\tau\times(256+nd+1)}.$$

ROUND 3. The is now to prove the following relations over $\mathbb{Z}_q$:

$$\begin{cases} \vec{z}_3 = \vec{\mathbf{y}}_3 + \mathfrak{R}\vec{\mathbf{s}}_1 & (4.4)\\ P\vec{\mathbf{s}}_1 = \vec{\mathbf{v}} & (4.5)\\ \langle \vec{\mathbf{s}}_1, \vec{\mathbf{s}}_1 \rangle = B_{s_1}^2 & (4.6) \end{cases}$$

By way of example, let us consider Equation 4.6. Proving this equation over $\mathbb{Z}_q$ is equivalent to proving that the constant coefficient of $\sigma(\mathbf{e})_1^\top \mathbf{s}_1 - B_{s_1}^2$ is equal to zero. Using this observation, we now aim to create a polynomial h_i for which, if its constant coefficient is zero, the relations above hold with high probability over $\mathbb{Z}_q$. This is done as follows. For each $i \in [\tau]$ we construct the polynomial

$$\begin{aligned} h_i^{(t)} := g_i &+ \sum_{j=1}^{256} \gamma_{i,j} \cdot (\sigma(\mathbf{r}_j)^\top \mathbf{s}_1 + \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{y}_3 - z_j) \\ &+ \sum_{j=1}^{nd} \gamma_{i,256+j} \cdot (\sigma(\mathbf{p}_j)^\top \mathbf{s}_1 - \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{v}) \\ &+ \gamma_{i,256+nd+1} \cdot (\sigma(\mathbf{e})_1^\top \mathbf{s}_1 - B_{s_1}^2) \end{aligned}$$

where $\mathbf{r}_j$ is the polynomial vector such that its coefficient vector is the j-th row of $\mathfrak{R}$. Similarly, $\mathbf{p}$ corresponds to the first row of P and $\hat{\mathbf{e}}_j$ is the

polynomial vector so that $\Phi(\hat{\mathbf{e}}_j)$ is a unit vector with its j-th element being 1. z_j is simply the j-th component of $\vec{z}_3$. By definition of $\mathbf{g}$, the constant coefficients of $h_1, ..., h_\tau$ are all zero and the other coefficients are masked by g_i. The third message and the corresponding challenge are then:

$$\begin{aligned} a_3 &:= (h_1, \ldots, h_\tau) \text{ and} \\ \boldsymbol{\mu} &:= (\mu_i)_{i\in[\tau]} = \mathsf{H}(3, \mathsf{crs}, x, a_1, a_2, a_3) \in R_q^\tau. \end{aligned}$$

ROUND 4. The goal is not to prove the τ quadratic equations defined by the h_i above. With high probability, it suffices to prove the following equation, constructed by linearly combining the τ equations

$$\begin{aligned} 0 = \sum_{i=1}^{\tau} \Big(\sum_{j=1}^{256} \gamma_{i,j} \cdot (\sigma(\mathbf{r}_j)^\top \mathbf{s}_1 + \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{y}_3 - z_j) \\ + \sum_{j=1}^{nd} \gamma_{i,256+j} \cdot (\sigma(\mathbf{p}_j)^\top \mathbf{s}_1 - \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{v}) \\ + \gamma_{i,256,nd+1} \cdot (\sigma(\mathbf{s}_1)^\top \mathbf{s}_1 - B_{s_1}^2) \\ + g_i - h_i \Big) \end{aligned} \tag{4.7}$$

Let us define

$$\mathbf{B} := \begin{bmatrix} \mathbf{B}_y \\ \mathbf{B}_g \end{bmatrix}, \ \mathbf{t}_B := \begin{bmatrix} \mathbf{t}_y \\ \mathbf{t}_g \end{bmatrix}, \ \mathbf{m} := \begin{bmatrix} \mathbf{y}_3 \\ \mathbf{g} \end{bmatrix}, \hat{\mathbf{s}} := \begin{bmatrix} \mathbf{s}_1 \\ \sigma(\mathbf{s}_1) \\ \mathbf{m} \\ \sigma(\mathbf{m}) \end{bmatrix}.$$

We note that Equation (4.7) may be written as

$$\hat{\mathbf{s}}^\top \mathbf{D}_2 \hat{\mathbf{s}} + \mathbf{d}_1^\top \hat{\mathbf{s}} + d_0 = 0,$$

where $\mathbf{D}_2$, $\mathbf{d}_1$, and d_0 are public matrices (see [BLNS23, Section 6] for how they are constructed). Now, we run the sub-protocol for proving a single quadratic equation with automorphisms as described in [LNP22]. We begin by computing the garbage term

$$f_1 := (\hat{\mathbf{s}})^\top \mathbf{D}_2 \mathbf{y} + (\hat{\mathbf{s}})^\top \mathbf{D}_2 \mathbf{y} + \mathbf{y}^\top \mathbf{D}_2 \hat{\mathbf{s}} + \mathbf{y}^\top \mathbf{D}_2 \hat{\mathbf{s}} + \mathbf{d}_1 \mathbf{y},$$

where

$$\mathbf{y} := \begin{bmatrix} \mathbf{y}_1 \\ \sigma(\mathbf{y}_1) \\ -\mathbf{B}\mathbf{y}_2 \\ -\sigma(\mathbf{B}\mathbf{y}_2) \end{bmatrix}.$$

We further create the commitment $t := \mathbf{b}^\top \mathbf{s}_2 + f_1$ to f_1 and set $f_0 := \mathbf{y}^\top \mathbf{D}_2 \mathbf{y} + \mathbf{b}^\top \mathbf{y}_2$. Thus the fourth message and corresponding challenge are:

$$a_4 := (t, f_0) \text{ and } c := \mathsf{H}(4, \mathsf{crs}, x, a_1, a_2, a_3, a_4) \in \mathcal{C}.$$

FINAL ROUND. Given the challenge c, we compute responses $\mathbf{z}_i := c\mathbf{s}_i + \mathbf{y}_i$ for $i = 1, 2$, applying rejection sampling to them to mask the $\mathbf{s}_i$. The final message is then $a_5 := (\mathbf{z}_1, \mathbf{z}_2)$ and the final proof is

$$\pi := (a_1, a_2, a_3, a_4, a_5).$$

VERIFICATION. Given a proof π the verifier recomputes the corresponding challenges as well as $\mathbf{D}_2$, $\mathbf{d}_1$, and d_0. Denoting

$$\mathbf{z} := \begin{bmatrix} \mathbf{z}_1 \\ \sigma(\mathbf{z}_1) \\ c\mathbf{t}_B - \mathbf{B}\mathbf{z}_2 \\ \sigma(c\mathbf{t}_B - \mathbf{B}\mathbf{z}_2) \end{bmatrix},$$

the verifier outputs 1 if *all* the following relations hold:

$$\begin{cases} \|\mathbf{z}_1\|_2 \overset{?}{\leq} B_1, \|\mathbf{z}_2\|_2 \overset{?}{\leq} B_2, \|\vec{z_3}\|_2 \overset{?}{\leq} B_3, \\ \tilde{h}_i \overset{?}{=} 0 \text{ for } i \in [\tau], \\ \mathbf{A}_1\mathbf{z}_1 + \mathbf{A}_2\mathbf{z}_2 \overset{?}{=} \mathbf{w} + c\mathbf{t}_A, \\ \mathbf{z}^\top \mathbf{D}_2 \mathbf{z} + c\mathbf{d}_1^\top \mathbf{z} + c^2 d_0 - (ct - \mathbf{b}^\top \mathbf{z}_2) \overset{?}{=} f_0, \end{cases}$$

and 0 otherwise.

SECURITY. We now give lemmas for the security of $\Pi^{\mathsf{Gen}}_{\mathsf{NIZK}}$. These results are due to [LNP22] and so we only state them here. For their proof we refer the reader to the aforementioned work.

Lemma 4.2.8 (Correctness)**.** *Let $\alpha_1, \alpha_2, \alpha_3 \in O(\sqrt{\lambda})$, $d \in O(\lambda)$. Recall ν, ω_{max} from Section 4.2.4 and define the following standard deviations:*

$$\sigma_1 := \alpha_1 \nu \|\mathbf{s}_1\|_2,\ \sigma_2 := \alpha_2 \nu \sqrt{m_2 d},\ \sigma_3 := \alpha_3 \|\mathbf{s}_1\|_2 \omega_{max},$$

and the corresponding rejection rates for $i \in \{1,2,3\}$:

$$M_i := \exp\left(\sqrt{\frac{2(\lambda+1)}{\log e}} \cdot \frac{1}{\alpha_i} + \frac{1}{2\alpha_i^2}\right).$$

Then the proof system $\Pi^{\mathsf{Gen}}_{\mathsf{NIZK}}$ *is correct.*

Lemma 4.2.9 (Zero-Knowledge)**.** $\alpha_1, \alpha_2, \alpha_3 \in O(\sqrt{\lambda})$, $d \in O(\lambda)$. *Recall* ν, ω_{max} *from Section 4.2.4 and define the following standard deviations:*

$$\sigma_1 := \alpha_1 \nu \|\mathbf{s}_1\|_2,\ \sigma_2 := \alpha_2 \nu \sqrt{m_2 d},\ \sigma_3 := \alpha_3 \|\mathbf{s}_1\|_2 \omega_{max},$$

and the corresponding rejection rates for $i \in \{1,2,3\}$:

$$M_i := \exp\left(\sqrt{\frac{2(\lambda+1)}{\log e}} \cdot \frac{1}{\alpha_i} + \frac{1}{2\alpha_i^2}\right).$$

Then $\Pi^{\mathsf{Gen}}_{\mathsf{NIZK}}$ *is zero-knowledge under the* $\mathsf{M\text{-}LWE}_{(k_{\mathsf{MSIS}}+256/d+\tau+1),m_2,\chi,q}$ *assumption.*

Finally, we come to soundness. The proof of this lemma can be found in [LNP22, Theorem B.7].

Lemma 4.2.10 (Knowledge Soundness)**.** *Let* $B := 8\nu\sqrt{B_1^2 + B_2^2}$ *and*

$$q > \max\left(B, 16 m_1 d B_3, \frac{2}{\omega_{max}(\lambda)} B_3^2\right).$$

Then there exists an extractor Ext *with the following properties. Given oracle access to any (potentially dishonest) prover* Prover^**, which outputs a valid poof with probability* ϵ*, the extractor runs in expected polynomial (in statement) number of steps and with probability at least*

$$\epsilon - \left(\frac{2}{|\mathcal{C}|} + p_1^{-d/2} + p_1^{-\lambda} + 2^{-128}\right),$$

it either outputs $\mathbf{s}_1 \in R_q^{m_1}$ *such that* $P\vec{\mathbf{s}_1} = \vec{\mathbf{v}}$ *and* $\|\mathbf{s}\|_2 = B$*, or an* $\mathsf{MSIS}_{k_{\mathsf{MSIS}},m_1+m_2,B}$ *solution for the matrix* $\left[\mathbf{A}_1 \| \mathbf{A}_2\right]$.

4.3 MAIN CONSTRUCTION

We begin by giving a high-level overview of our new DAA scheme suppressing, for now, the artefacts needed for proving its security in the UC model such as session identifiers. In the same vein, we present the join and signing protocols in their complete, interactive form without separating them into sub-stages. This is done to aid the reader's intuition and again we leave the syntactic rearrangements needed for UC compatibility to Section 4.4. Sections 4.3.1 and 4.3.1 display the join and sign phases of the protocol and the reader may find it useful to refer to these.

4.3.1 *Protocol Description*

SETUP. Let $q, n, \hat{n}, m$, and N be positive integers such that $q = p_1 p_2$, a product of two primes of the form $p_i = 5 \mod 8$, $p_1 < p_2$. Let d be a power of two and [1] and let σ, σ_{NTRU} B_s, and B_m be positive reals. Let $\hat{d}$ be a positive multiple of d. Define the system parameters $\mathsf{sp} = (q, d, n, \hat{n}, m, N, B, U, \sigma, \sigma_{\mathsf{NTRU}}, B_s, B_{\mathsf{tsk}})$. $B \in \mathbb{Z}^{nd \times t}$ is sampled uniformly and used to define the function f as described in Section 4.2.2 and N is the maximum number of platforms that can join. U is sampled uniformly from $\mathbb{Z}_q^{n^2 d \times 256}$. The issuer's public key is $\mathsf{ipk} = (\mathbf{C}_1, \mathbf{C}_2, \mathbf{h})$ where $\mathbf{C}_1, \mathbf{C}_2 \in R^{n \times n}$, $\mathbf{h} \in R_q^n$, and its secret key is $\mathsf{isk} = \mathbf{B}_{\mathbf{F},\mathbf{g}}$ which is a short basis for the NTRU lattice $\mathcal{L}_{\mathsf{NTRU}}$ defined by $\mathbf{h}$. Note, $\mathbf{h}$, and $\mathbf{B}_{\mathbf{F},\mathbf{g}}$ are generated using $\mathsf{NTRU.TrapGen}$ as described in Lemma 4.2.2.

JOIN. The TPM samples $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2) \leftarrow R_3^n \times R_3^n$ and $e_3 \leftarrow \{0,1\}^{256}$ and sets its secret key $\mathsf{tsk} = (\mathbf{e}_1, \mathbf{e}_2, e_3)$. Next, the TPM computes

$$\mathbf{u}_1 := \mathbf{C}_1 \mathbf{e}_1 + \mathbf{C}_2 \mathbf{e}_2 \in R_q^n, \tag{4.8}$$

and sends $\mathbf{u}_1$ along with a proof of knowledge $\pi_{\mathsf{join}} \leftarrow \mathsf{Prove}_{\mathsf{join}}^{\mathsf{H}}(\mathbf{u}_1; (\mathbf{e}_1, \mathbf{e}_2))$ of $(\mathbf{e}_1, \mathbf{e}_2)$ satisfying Equation (4.8) such that $\|\mathbf{e}_1\|_2, \|\mathbf{e}_2\|_2 \leq B_{\mathsf{tsk}}$.

The issuer, upon receiving $(\mathbf{u}_1, \pi_{\mathsf{join}})$ will check the validity of the proof, aborting if it does not pass. Next the issuer samples $x \xleftarrow{\$} [N]$, computes $f(x) := \mathsf{Coeffs}^{-1}(B \cdot \mathsf{bin}(x)) \in R_q^n$. Next, the issuer constructs

$$c := \mathbf{1}^\top (f(x) + \mathbf{u}_1) \in R_q,$$

1 for the purposes of this work, the reader can think of d as being 64 or 128.

where $\mathbf{1}$ is the column vector containing n entries of the constant polynomial 1. The issuer then samples a credential as $\mathbf{s} \leftarrow \mathsf{GSampler}(\mathbf{h}, \mathbf{B}_{\mathbf{F},\mathbf{g}}, \sigma_{\mathsf{cred}}, c)$, satisfying

$$\begin{bmatrix} 1 & \mathbf{h}^T \end{bmatrix} \mathbf{s} = c \mod R_q, \tag{4.9}$$

where $\|\mathbf{s}\|_2 \leq B_s$.

It then passes $(\mathbf{s}, x)$ to the host who checks that $\mathbf{s}$ is shot and satisfies Equation (4.9). It then stores the credential $\mathsf{cred} = (\mathbf{s}, x)$.

SIGN. To sign a message m with respect to a basename bsn (if $\mathsf{bsn} = \perp$, the TPM chooses one uniformly at random from the space of basenames, $\mathfrak{B}$), the TPM creates the value $\mathbf{D} := \mathsf{H}_{R_q^{n \times n}}(\mathsf{bsn})$. It then computes the error term $\mathbf{e}' := \mathsf{H}_{R_3^n}(e_3, \mathsf{bsn})$ and outputs the pseudonym $\mathsf{nym} = \mathbf{D}\mathbf{e}_1 + \mathbf{e}' \in R_q^n$.

At this point, the platform (TPM and Host) collectively know short $\mathbf{e}_1, \mathbf{e}_2$, $\mathbf{e}', \mathbf{s}$, and x, satisfying

$$\mathsf{nym} = \mathbf{D}\mathbf{e}_1 + \mathbf{e}', \tag{4.10}$$

$$\begin{bmatrix} 1 & \mathbf{h}^T \end{bmatrix} \mathbf{s} = \mathbf{1}^\top \left(f(x) + \mathbf{C}_1\mathbf{e}_1 + \mathbf{C}_2\mathbf{e}_2 \right). \tag{4.11}$$

In order to create a signature on m with respect to bsn, the platform creates a proof of knowledge $\pi_{\mathsf{sign}} \leftarrow \mathsf{Prove}^{\mathsf{H}}_{\mathsf{sign}}((\mathsf{nym}, \mathbf{D}, \mathbf{h}, \mathbf{C}_1, \mathbf{C}_2), (\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}', \mathbf{s}))$ of these short elements satisfying the equation above (where the message is included in the hash input of the non-interactive proof). In particular, one proves that $\|\mathbf{e}_1\|_2, \|\mathbf{e}_2\|_2, \|\mathbf{e}'\|_2 \leq B_{\mathsf{tsk}}$, $\|\mathbf{s}\|_2 \leq B_s$. The proof system used is as described in Section 4.2.4. The host then outputs the signature $\mathsf{sig} = (\mathsf{nym}, \mathsf{bsn}, \pi_{\mathsf{sign}})$.

VERIFY. Given a message m, signature $\mathsf{sig} = (\mathsf{nym}, \mathsf{bsn}, \pi_{\mathsf{sign}})$, and issuer public key ipk, the verifier checks the proof π_{sign} (recovering $\mathbf{D}$ from bsn). Further, the verifier checks the signature against the list of revoked TPMs; for each $\mathbf{e}'_1 \in \mathsf{RL}$, the verifier checks that $\|\mathsf{nym} - \mathbf{D}\mathbf{e}'_1\|_2$ is not less than B_{tsk}. If all checks pass, it outputs 1 and 0 otherwise.

LINK. On input two verifying signatures $(\mathsf{M}_1, \mathsf{bsn}, \mathsf{sig}_1)$ and $(\mathsf{M}_2, \mathsf{bsn}, \mathsf{sig}_2)$ on messages M_1 and M_2 for the same basename, the linking algorithm proceeds as follows:

1. If $\mathsf{nym}_1 = \mathsf{nym}_2$, it outputs 1
2. Otherwise it checks that $\|\mathsf{nym}_1 - \mathsf{nym}_2\|_2 \leq 2B_{\mathsf{tsk}}$, outputting 1 if so. Otherwise it returns 0.

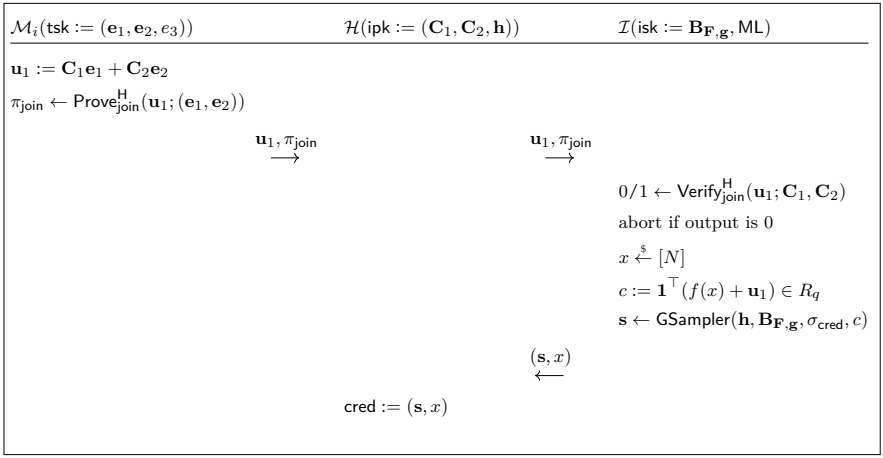

FIGURE 4.2: The Join protocol for Π_{LDAA}.

4.3.2 *Constructing* π_{join}

We now detail the non-interactive zero=knowledge proof framework used by the platform in the join protocol.

It suffices for the TPM to prove knowledge of short $\mathbf{e}_1, \mathbf{e}_2 \in R_q^n$ satisfying the module-LWE instance

$$\begin{bmatrix}\mathbf{C}_2^{-1}\mathbf{C}_1 & \mathbf{I}_n\end{bmatrix}\begin{bmatrix}\mathbf{e}_1\\ \mathbf{e}_2\end{bmatrix} = \mathbf{C}_2^{-1}\mathbf{u}_1. \tag{4.12}$$

More precisely, the TPM proves that $\|\mathbf{e}_1\|_2, \|\mathbf{e}_2\|_2 \leq B_{\mathsf{tsk}}$ for $B_{\mathsf{tsk}} = \sqrt{nd}$. Prove this relation is exactly dealt with in [LNP22, Section 6.2] where the authors show how one only needs to commit to $\mathbf{s}_1 := \mathbf{e}_1$ and prove that

$$\left\|\begin{bmatrix}\mathbf{e}_1\\ \mathbf{C}_2^{-1}\mathbf{C}_1\mathbf{e}_1 - \mathbf{C}_2^{-1}\mathbf{u}\end{bmatrix}\right\|_2 = \left\|\begin{bmatrix}\mathbf{I}_n\\ \mathbf{C}_2^{-1}\mathbf{C}_1\end{bmatrix}\mathbf{e}_1 - \begin{bmatrix}\mathbf{0}\\ \mathbf{C}_2^{-1}\mathbf{u}_1\end{bmatrix}\right\|_2 \leq B_{\mathsf{tsk}}.$$

We adopt the same parameters as given in [LNP22, Section 6.2] which yield a proof size for π_{join} of 14.4KB. Finally, the proof inherits the security properties of correctness, zero-knowledge, and knowledge soundness described in Lemmata 4.2.8 to 4.2.10.

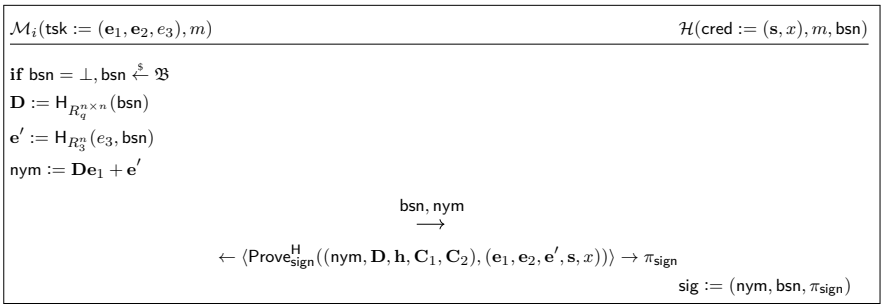

FIGURE 4.3: The Sign protocol for Π_{LDAA}. The penultimate line denotes interaction between the TPM and Host to produce π_{sign} (see Section 4.3.3 for details).

4.3.3 *Constructing* π_{sign}

Here, we give a details overview of how the LNP NIZK protocol described in Section 4.2.4 can be used to generate π_{sign}. We stress that splitting the prover work between TPM and host is non-trivial and requires careful attention. Thus, we present the proof rounds in full detail rather than simply referring to the work in [LNP22] as a plugin as in the generation of π_{join}.

We first combine Equations (4.10) and (4.11) into an equation of the form defined by the relation in Equation (4.3). It is easily derived that, after some rearrangement, Equations (4.10) and (4.11) can be written as one equation over $\mathbb{Z}_q$ of the form

$$\begin{bmatrix} I_d & \Phi(\mathbf{h}^\top) & -J_1 & -J_2 & -J_3 & 0 \\ 0 & 0 & 0 & \mathsf{Rot}(\mathbf{D}) & 0 & I_{nd} \end{bmatrix} \begin{bmatrix} \vec{\mathbf{s}} \\ \vec{u} \\ \vec{\mathbf{e}_1} \\ \vec{\mathbf{e}_2} \\ \vec{\mathbf{e}'} \end{bmatrix} = \begin{bmatrix} \vec{0} \\ \vec{\mathsf{nym}} \end{bmatrix} \in \mathbb{Z}_q^{(n+1)d}, \tag{4.13}$$

where $J_1 \in \mathbb{Z}_q^{d\times t}$ and $J_2, J_3 \in \mathbb{Z}_q^{d\times nd}$ matrix resulting from the product of $\mathbf{1}^\top$ with $(f(x) + \mathbf{C}_1\mathbf{e}_1 + \mathbf{C}_2\mathbf{e}_2)$ on the righ-hand side of Equation (4.11). As well as the linear relation Equation (4.13), we must further prove the following quadratic relations: $\|\vec{\mathbf{s}}\|_2 \leq B_s$, $\langle \vec{u}, \vec{u} - 1\rangle = 0$, and $\|\vec{\mathbf{e}}_1\|_2, \|\vec{\mathbf{e}}_2\|_2, \|\vec{\mathbf{e}'}\|_2 \leq B_{\mathsf{tsk}}$. Note, the two-norm of a ring element can be equivalently expressed as the inner product of its coefficient vector with itself.

We now observe that, by Lagrange's sum-of-four-squares theorem, $\|\vec{\mathbf{s}}\|_2 \leq$

B_s is equivalent to the existence of four integers $a_1, a_2, a_3, a_4 \in \mathbb{Z}$ such that $B_s^2 - \|\vec{s}\|_2^2 = a_1^2 + a_2^2 + a_3^2 + a_4^2$. Thus, we can define a vector $a := (a_1, a_2, a_3, a_4, 0, \ldots, 0)^\top \in \mathbb{Z}_q^d$ and observe that

$$\| \begin{bmatrix} \vec{s} \\ \vec{a} \end{bmatrix} \|_2 = B_s.$$

Let us redefine, via abuse of notation, $\vec{s} := \begin{bmatrix} \vec{s} & \vec{a} \end{bmatrix}^\top$. A similar elongation and reassignment can be performed on $\vec{e}_1, \vec{e}_2$, and $\vec{e}'$. Further, note that by inserting columns of zeros in the correct places to first matrix on the left-hand side of Equation (4.13), we can replace the secret vectors in the second matrix with their elongated versions whilst preserving the linear relation. The advantage now is that all norm bounds (quadratic relations) to be proven are *exact* rather than inequalities. This makes the proof structure much cleaner. Finally, we extend the vector $\vec{u}$ with zeros to have length d so that the whole secret vector has length a multiple of d. Again we insert $(d - t)$ columns of zeros into the public matrix to preserve the original equation. At this point, let us note the length of the secret vector in Equation (4.13): $(3n + 1)d + (\hat{n} + 1)\hat{d}$. In the following, we define $m_1 := (3n + 1)d + (\hat{n} + 1)\hat{d}/d$, where we assume $\hat{d}$ is a multiple of d. Thus, m_1 is the number of ring elements in the secret vector. We are now ready to describe the detailed round of the zero-knowledge protocol. In order to refer to objects in Equation (4.13), let us label the matrices and vectors (left to right) as P, $\mathbf{s}_1$, and $\mathbf{v}$ respectively. We also define the common *random* string consisting of uniformly random matrices which are used of the ABDLOP commitment:

$$\mathsf{crs} := (\mathbf{A}_1, \mathbf{A}_2, \mathbf{B}_y, \mathbf{B}_g, \mathbf{b}) \in R_q^{k_{\mathsf{MSIS}} \times m_1} \times R^{k_{\mathsf{MSIS}} \times m_2} \times R^{256/d \times m_2} \times R^{\tau \times k_{\mathsf{M\text{-}LWE}}} \times R^{m_2},$$

where k_{MSIS} and m_2 are parameters set so that MSIS and M-LWE are hard. We denote the following NIZK protocol defined by the proof and verification phases by Π_{Sign}.

ROUND 1. In this round, the platform needs to create a commitment to the vector $\mathbf{s}_1$ as well as commitments to masking vectors $\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3$ and to a vector $\mathbf{g}$ with constant coefficients equal to zero. This work is split between TPM and Host as follows.

Let us define $\mathbf{s}_1^{(t)}$ and $\mathbf{s}_1^{(h)}$ to be the TPM and Host's additive shares of $\mathbf{s}_1$ respectively so that

$$\mathbf{s}_1^{(t)} := \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{e}_1 \\ \mathbf{e}_2 \\ \mathbf{e}' \end{bmatrix} \text{ and } \mathbf{s}_1^{(h)} := \begin{bmatrix} \mathbf{s} \\ \mathbf{u} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix},$$

where $\mathbf{u} := \Phi^{-1}(\vec{u})$. The TPM then samples $\mathbf{s}_2 \leftarrow \chi^{m_2}$ where χ is a probability distribution on ternary polynomials in R_q. It also samples $\mathbf{y}_i \leftarrow D_{\sigma_i}^{m_i}$ for $i = 1, 2$ and $\mathbf{y}_3 \leftarrow D_{\sigma_3}^{256}$. Finally it samples a polynomial vector $\mathbf{g} \leftarrow \{x \in R_q : \tilde{x} = 0\}$ and computes the commitments

$$\begin{aligned} \mathbf{t}_A^{(t)} &:= \mathbf{A}_1\mathbf{s}_1^{(t)} + \mathbf{A}_2\mathbf{s}_2, \\ \mathbf{w} &:= \mathbf{A}_1\mathbf{y}_2 + \mathbf{A}_2\mathbf{y}_2 \\ \mathbf{t}_y &:= \mathbf{B}_y\mathbf{s}_2 + \mathbf{y}_3 \\ \mathbf{t}_g &:= \mathbf{B}_g\mathbf{s}_2 + \mathbf{g}. \end{aligned}$$

The TPM then sends these to the host who computes $\mathbf{t}_A := \mathbf{t}_A^{(t)} + \mathbf{A}_1\mathbf{s}_1^{(h)}$. The first message and corresponding challenge are then

$$\begin{aligned} a_1 &:= (\mathbf{t}_A, \mathbf{t}_y, \mathbf{t}_g, \mathbf{w}) \text{ and} \\ (\mathfrak{R}_0, \mathfrak{R}_1) &:= \mathsf{H}(1, \mathsf{crs}, x, a_1) \in \{0,1\}^{256\times m_1 d} \times \{0,1\}^{256\times m_1 d}. \end{aligned}$$

ROUND 2. The TPM and host now jointly create the response for the approximate proof of shortness for $\mathbf{s}_1$. The host begins by passing thee TPM the challenge $\mathfrak{R} := \mathfrak{R}_0 - \mathfrak{R}_1$. The TPM then computes $\vec{z}_3^{(t)} := \vec{\mathbf{y}}_3 = \mathfrak{R}\vec{\mathbf{s}}_1^{(t)}$ and applies rejection sampling on $\vec{z}_3$ before passing $\vec{z}_3^{(t)}$ to the host. The host then computes $\vec{z}_3 := \vec{z}_3^{(t)} + \mathfrak{R}\vec{\mathbf{s}}_1^{(h)}$, performing a second round of rejection sampling, returning to the beginning of the round if upon rejection. The second message and corresponding challenge are then

$$\begin{aligned} a_2 &:= \vec{z}_3 \text{ and} \\ (\gamma_{i,j})_{i\in[\tau],j\in[256+(n+1)d+5]} &:= \mathsf{H}(2, \mathsf{crs}, x, a_1, a_2) \in \mathbb{Z}_q^{\tau\times(256+(n+1)d+5)}. \end{aligned}$$

ROUND 3. The goal of the platform is now to jointly prove the following relations over $\mathbb{Z}_q$:

$$\begin{cases} \vec{z}_3 = \vec{\mathbf{y}}_3 + \mathfrak{R}\vec{\mathbf{s}}_1 & (4.14) \\ P\vec{\mathbf{s}}_1 = \vec{\mathbf{v}} & (4.15) \\ \langle \vec{\mathbf{s}}, \vec{\mathbf{s}} \rangle = B_s^2 & (4.16) \\ \langle \vec{u}, \vec{u} \rangle = 0 & (4.17) \\ \langle \vec{e}_1, \vec{e}_1 \rangle = B_{\mathsf{tsk}}^2 & (4.18) \\ \langle \vec{e}_2, \vec{e}_2 \rangle = B_{\mathsf{tsk}}^2 & (4.19) \\ \langle \vec{e}', \vec{e}' \rangle = B_{\mathsf{tsk}}^2 & (4.20) \end{cases}$$

By way of example, let us consider Equation 4.18. Proving this equation over $\mathbb{Z}_q$ is equivalent to proving that the constant coefficient of $\sigma(\mathbf{e}_1)^\top \mathbf{e}_1 - B_{\mathsf{tsk}}^2$ is equal to zero. In a similar way, the platform now aims to create a polynomial h_i for which, if its constant coefficient is zero, the relations above hold with high probability over $\mathbb{Z}_q$. This is done as follows. For each $i \in [\tau]$ the TPM creates the polynomial

$$\begin{aligned} h_i^{(t)} := g_i &+ \sum_{j=1}^{256} \gamma_{i,j} \cdot (\sigma(\mathbf{r}_j)^\top \mathbf{s}_1^{(t)} + \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{y}_3 - z_j^{(t)}) \\ &+ \sum_{j=1}^{(n+1)d} \gamma_{i,256+j} \cdot (\sigma(\mathbf{p}_j)^\top \mathbf{s}_1^{(t)} - \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{v}) \\ &+ \gamma_{i,256,(n+1)d+3} \cdot (\sigma(\mathbf{e}_1)^\top \mathbf{e}_1 - B_{\mathsf{tsk}}^2) \\ &+ \gamma_{i,256,(n+1)d+4} \cdot (\sigma(\mathbf{e}_2)^\top \mathbf{e}_2 - B_{\mathsf{tsk}}^2) \\ &+ \gamma_{i,256,(n+)d+5} \cdot (\sigma(\mathbf{e}')^\top \mathbf{e}' - B_{\mathsf{tsk}}^2) \end{aligned}$$

where $\mathbf{r}_j$ is the polynomial vector such that its coefficient vector is the j-th row of $\mathfrak{R}$. Similarly, $\mathbf{p}$ corresponds to the first row of P and $\hat{\mathbf{e}}_j$ is the polynomial vector so that $\Phi(\hat{\mathbf{e}}_j)$ is a unit vector with its j-th element being 1. z_j is simply the j-th component of $\vec{z}_3$. It then sends $h_i^{(t)}$ to the host. The host then computes

$$\begin{aligned} h_i^{(h)} := &\sum_{j=1}^{256} \gamma_{i,j} \cdot (\sigma(\mathbf{r}_j)^\top \mathbf{s}_1^{(h)} + \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{y}_3 - z_j^{(h)}) \\ &+ \sum_{j=1}^{(n+1)d} \gamma_{i,256+j} \cdot (\sigma(\mathbf{p}_j)^\top \mathbf{s}_1^{(h)} - \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{v}) \end{aligned}$$

$$+ \gamma_{i,256,(n+1)d+1} \cdot (\sigma(\mathbf{s})^\top \mathbf{s} - B_s^2)$$
$$+ \gamma_{i,256,(n+1)d+2} \cdot (\sigma(\mathbf{u})^\top (\mathbf{u} - \mathbf{1})).$$

The host then computes $h_i := h_i^{(t)} + h_i^{(h)}$ for each $i \in [\tau]$. By definition of $\mathbf{g}$, the constant coefficients of $h_1, ..., h_\tau$ are all zero and the other coefficients are masked by g_i. The third message and the corresponding challenge are then:

$$a_3 := (h_1, \ldots, h_\tau) \text{ and}$$
$$\boldsymbol{\mu} := (\mu_i)_{i \in [\tau]} = \mathsf{H}(3, \mathsf{crs}, x, a_1, a_2, a_3) \in R_q^\tau.$$

ROUND 4. The goal of the platform is to now jointly prove the τ quadratic equations defined by the h_i above. With high probability, it suffices to prove the following equation, constructed by linearly combining the τ equations

$$\begin{aligned} 0 = \sum_{i=1}^{\tau} \Big(& \sum_{j=1}^{256} \gamma_{i,j} \cdot (\sigma(\mathbf{r}_j)^\top \mathbf{s}_1 + \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{y}_3 - z_j) \\ & + \sum_{j=1}^{(n+1)d} \gamma_{i,256+j} \cdot (\sigma(\mathbf{p}_j)^\top \mathbf{s}_1 - \sigma(\hat{\mathbf{e}}_j)^\top \mathbf{v}) \\ & + \gamma_{i,256,(n+1)d+1} \cdot (\sigma(\mathbf{s})^\top \mathbf{s} - B_s^2) \\ & + \gamma_{i,256,(n+1)d+2} \cdot (\sigma(\mathbf{u})^\top (\mathbf{u} - \mathbf{1})) \\ & + \gamma_{i,256,(n+1)d+3} \cdot (\sigma(\mathbf{e}_1)^\top \mathbf{e}_1 - B_{\mathsf{tsk}}^2) \\ & + \gamma_{i,256,(n+1)d+4} \cdot (\sigma(\mathbf{e}_2)^\top \mathbf{e}_2 - B_{\mathsf{tsk}}^2) \\ & + \gamma_{i,256,(n+)d+5} \cdot (\sigma(\mathbf{e}')^\top \mathbf{e}' - B_{\mathsf{tsk}}^2) \\ & + g_i - h_i \Big) \end{aligned} \tag{4.21}$$

Let us define

$$\mathbf{B} := \begin{bmatrix} \mathbf{B}_y \\ \mathbf{B}_g \end{bmatrix}, \mathbf{t}_B := \begin{bmatrix} \mathbf{t}_y \\ \mathbf{t}_g \end{bmatrix}, \mathbf{m} := \begin{bmatrix} \mathbf{y}_3 \\ \mathbf{g} \end{bmatrix}, \hat{\mathbf{s}}^{(t)} := \begin{bmatrix} \mathbf{s}_1^{(t)} \\ \sigma(\mathbf{s}_1^{(t)}) \\ \mathbf{m} \\ \sigma(\mathbf{m}) \end{bmatrix}, \text{ and } \hat{\mathbf{s}}^{(h)} := \begin{bmatrix} \mathbf{s}_1^{(h)} \\ \sigma(\mathbf{s}_1^{(h)}) \\ \mathbf{m} \\ \sigma(\mathbf{m}) \end{bmatrix}$$

Further, defining $\hat{\mathbf{s}} := \hat{\mathbf{s}}^{(t)} + \hat{\mathbf{s}}^{(h)}$, we note that Equation (4.21) may be written as

$$\hat{\mathbf{s}}^\top \mathbf{D}_2 \hat{\mathbf{s}} + \mathbf{d}_1^\top \hat{\mathbf{s}} + d_0 = 0,$$

where $\mathbf{D}_2$, $\mathbf{d}_1$, and d_0 are public matrices (see [BLNS23, Section 6] for how they are constructed). Now, we run the sub-protocol for proving a single quadratic equation with automorphisms as described in [LNP22] but we split the work between TPM and host.

The host begins by sending $(\hat{\mathbf{s}}^{(h)})^\top \mathbf{D}_2$ and $\mathbf{D}_2 \hat{\mathbf{s}}^{(h)}$ to the TPM. The TPM can now compute the garbage term

$$f_1 := (\hat{\mathbf{s}}^{(t)})^\top \mathbf{D}_2 \mathbf{y} + (\hat{\mathbf{s}}^{(h)})^\top \mathbf{D}_2 \mathbf{y} + \mathbf{y}^\top \mathbf{D}_2 \hat{\mathbf{s}}^{(t)} + \mathbf{y}^\top \mathbf{D}_2 \hat{\mathbf{s}}^{(h)} + \mathbf{d}_1 \mathbf{y},$$

where

$$\mathbf{y} := \begin{bmatrix} \mathbf{y}_1 \\ \sigma(\mathbf{y}_1) \\ -\mathbf{B}\mathbf{y}_2 \\ -\sigma(\mathbf{B}\mathbf{y}_2) \end{bmatrix}.$$

TPM can then further create the commitment $t := \mathbf{b}^\top \mathbf{s}_2 + f_1$ to f_1. The it sets $f_0 := \mathbf{y}^\top \mathbf{D}_2 \mathbf{y} + \mathbf{b}^\top \mathbf{y}_2$. These can then be passed to the host who computes the challenge. Thus the fourth message and corresponding challenge are:

$$a_4 := (t, f_0) \text{ and } c := \mathsf{H}(4, \mathsf{crs}, x, a_1, a_2, a_3, a_4) \in \mathcal{C}.$$

FINAL ROUND. Given the challenge c, the TPM begins by computing responses $\mathbf{z}_i^{(t)} := c\mathbf{s}_i^{(t)} + \mathbf{y}_i$ for $i = 1, 2$, applying rejection sampling to them and passing them to the host. The host then computes $\mathbf{z}_i := \mathbf{z}_i^{(t)} + c\mathbf{s}_i^{(h)}$. The final message is then $a_5 := (\mathbf{z}_1, \mathbf{z}_2)$ and the proof output by the platform (via the host) is

$$\pi_{\mathsf{sign}} := (a_1, a_2, a_3, a_4, a_5).$$

When made non-interactive via the Fiat-Shamir transform, we note that $\mathbf{w}$ need not be sent, not any of the challenges since except c since these are all computable from the other proof elements.

VERIFICATION. Given a proof π the verifier recomputes the corresponding challenges as well as $\mathbf{D}_2$, $\mathbf{d}_1$, and d_0. Denoting

$$\mathbf{z} := \begin{bmatrix} \mathbf{z}_1 \\ \sigma(\mathbf{z}_1) \\ c\mathbf{t}_B - \mathbf{B}\mathbf{z}_2 \\ \sigma(c\mathbf{t}_B - \mathbf{B}\mathbf{z}_2) \end{bmatrix},$$

the verifier outputs 1 if *all* the following relations hold:

$$\begin{cases} \|\mathbf{z}_1\|_2 \overset{?}{\leq} B_1, \|\mathbf{z}_2\|_2 \overset{?}{\leq} B_2, \|\vec{z_3}\|_2 \overset{?}{\leq} B_3, \\ \tilde{h}_i \overset{?}{=} 0 \text{ for } i \in [\tau], \\ \mathbf{A}_1\mathbf{z}_1 + \mathbf{A}_2\mathbf{z}_2 \overset{?}{=} \mathbf{w} + c\mathbf{t}_A, \\ \mathbf{z}^\top \mathbf{D}_2 \mathbf{z} + c\mathbf{d}_1^\top \mathbf{z} + c^2 d_0 - (ct - \mathbf{b}^\top \mathbf{z}_2) \overset{?}{=} f_0, \end{cases}$$

and 0 otherwise.

SECURITY. We now give lemmas for the security of π_{sign}. These results are due to [LNP22] and so we only state them here. For their proof we refer the reader to the aforementioned work. Since we assume an honest TPM in this work, any added security concerns that might arise from the splitting of proof work between TPM and host could only come from elements sent from the TPM to the host. It is easy to see that, if parameters are chosen according to the following lemmas, no information about the TPM secret key is leaked to the host. Furthermore, any objects passed from the host to the TPM need not hide the credential from the TPM.

Lemma 4.3.1 (Correctness of π_{sign})**.** *Let $\alpha_1, \alpha_2, \alpha_3 \in O(\sqrt{\lambda})$, $d \in O(\lambda)$. Recall ν, ω_{max} from Section 4.2.4 and define the following standard deviations:*

$$\sigma_1 := \alpha_1 \nu \|\mathbf{s}_1\|_2,\ \sigma_2 := \alpha_2 \nu \sqrt{m_2 d},\ \sigma_3 := \alpha_3 \|\mathbf{s}_1\|_2 \omega_{max},$$

and the corresponding rejection rates for $i \in \{1, 2, 3\}$:

$$M_i := \exp\left(\sqrt{\frac{2(\lambda+1)}{\log e}} \cdot \frac{1}{\alpha_i} + \frac{1}{2\alpha_i^2}\right).$$

Then the proof system Π_{Sign} for generation and verification of π_{sign} is correct.

Lemma 4.3.2 (Zero-Knowledge)**.** *$\alpha_1, \alpha_2, \alpha_3 \in O(\sqrt{\lambda})$, $d \in O(\lambda)$. Recall ν, ω_{max} from Section 4.2.4 and define the following standard deviations:*

$$\sigma_1 := \alpha_1 \nu \|\mathbf{s}_1\|_2,\ \sigma_2 := \alpha_2 \nu \sqrt{m_2 d},\ \sigma_3 := \alpha_3 \|\mathbf{s}_1\|_2 \omega_{max},$$

and the corresponding rejection rates for $i \in \{1, 2, 3\}$:

$$M_i := \exp\left(\sqrt{\frac{2(\lambda+1)}{\log e}} \cdot \frac{1}{\alpha_i} + \frac{1}{2\alpha_i^2}\right).$$

Then Π_{Sign} is zero-knowledge under the $\mathsf{M\text{-}LWE}_{(k_{\mathsf{MSIS}}+256/d+\tau+1), m_2, \chi, q}$ assumption.

Finally, we come to soundness. The proof of this lemma can be found in [LNP22, Theorem B.7].

Lemma 4.3.3 (Knowledge Soundness)**.** *Let* $B := 8\nu\sqrt{B_1^2 + B_2^2}$ *and*

$$q > \max\left(b, 16m_1 dB_3, \frac{2}{\omega_{max}(\lambda)}B_3^2\right).$$

Then there exists an extractor Ext *with the following properties. Given oracle access to any (potentially dishonest) prover* Prover^**, which outputs a valid poof with probability* ϵ*, the extractor runs in expected polynomial (in statement) number of steps and with probability at least*

$$\epsilon - \left(\frac{2}{|\mathcal{C}|} + p_1^{-d/2} + p_1^{-\lambda} + 2^{-128}\right),$$

it either outputs $\mathbf{s}_1 \in R_q^{m_1}$ *such that* $P\vec{\mathbf{s}_1} = \vec{\mathbf{v}}$ *and* $\|\mathbf{s}\|_2 = B$*, or an* $\mathsf{MSIS}_{k_{\mathsf{MSIS}},m_1+m_2,B}$ *solution for the matrix* $\left[\mathbf{A}_1|\mathbf{A}_2\right]$*.*

4.4 SECURITY PROOF

We now show that our DAA protocol satisfies the desired DAA security guarantees captured through the ideal functionality $\mathcal{F}_{\mathsf{daa}}$ [CDL16]. Before presenting our proof sketch, we first discuss how the protocols and algorithms presented in Section 4.3 have to be "UC-fied".

4.4.1 *UC Wrapper for our Protocol*

To date, the only sound security notion for DAA is an ideal functionality in the Universal Composability framework. Describing protocols in the UC framework requires some extra care to include session identifiers, explicit party inputs in interactive protocols, as well as to reflect the abstract modeling of keys and secure channels. We start by describing the necessary subfunctionalities our protocol relies on, and then discuss how to map our protocols to the interfaces required by the DAA ideal functionality $\mathcal{F}_{\mathsf{daa}}$.

SUB-FUNCTIONALITIES We assume that a common reference string functionality $\mathcal{F}_{\mathsf{crs}}^D$ and a certificate authority functionality $\mathcal{F}_{\mathsf{ca}}$ are available to all parties. The later allows the issuer to register his public key, and $\mathcal{F}_{\mathsf{crs}}^D$ is used to provide all entities with the system parameters comprising of the random seed to generate the commitments, and the issuer's public key.

For the communication between the TPM and issuer (via the host) in the join protocol, we use the semi-authenticated channel $\mathcal{F}_{auth^*}$ introduced in [CDL16]. For all communication between a host and TPM we assume the secure message transmission functionality $\mathcal{F}_{smt}$ (enabling authenticated and encrypted communication). In practice, $\mathcal{F}_{smt}$ is naturally guaranteed by the physical proximity of the host and TPM forming the platform, i.e., if both are honest an adversary can neither alter nor read their internal communication.

In the description of the protocol, we assume that parties call $\mathcal{F}_{\mathsf{crs}}^{D}$ and $\mathcal{F}_{\mathsf{ca}}$ to retrieve the necessary key material whenever they use a public key of another party. Further, if any of the checks in the protocol fails, the protocol ends with a failure message $\perp$.

The protocol also outputs $\perp$ whenever a party receives an input or message it does not expect (e.g., protocol messages arriving in the wrong order.)

$\mathcal{F}_{\mathsf{daa}}$ INTERFACES. The $\mathcal{F}_{\mathsf{daa}}$ functionality considers an issuer $\mathcal{I}$ and the platform consisting of a TPM $\mathcal{M}_i$ and a host $\mathcal{H}_i$. In UC, different instances of a protocol are separated through unique session identifiers $\mathsf{sid} = (\mathcal{I}, \mathsf{sid}')$. In the real-world these are mapped to the issuer public key, and all parties use the sid to link their stored key material to the particular issuer.

SETUP. $\mathcal{I}$ upon input $(\mathsf{SETUP}, \mathsf{sid})$ generates his key pair $(\mathsf{ipk}, \mathsf{isk})$. It registers the public key $(\mathsf{sid}, \mathsf{ipk})$ at $\mathcal{F}_{\mathsf{ca}}$, stores the secret key as $(\mathsf{sid}, \mathsf{isk})$ and ends with output $(\mathsf{SETUPDONE}, \mathsf{sid})$.

JOIN. To distinguish several join sessions that might run in parallel, we use a unique sub-session identifier jsid that is given as additional input to all parties. The join protocol starts when $\mathcal{H}_j$ receives the input $(\mathsf{JOIN}, \mathsf{sid}, \mathsf{jsid}, \mathcal{M}_i)$ upon which it triggers $\mathcal{M}_i$ to generate $\mathbf{u}_1$ along with the proof π_{join}, and send it via $\mathcal{F}_{auth^*}$ to $\mathcal{I}$. When $\mathcal{I}$ receives the message it outputs $(\mathsf{JOINPROCEED}, \mathsf{sid}, \mathsf{jsid}, \mathcal{M}_i)$. The join session is completed when the issuer receives an input telling him to proceed with join session jsid, upon which it returns $\mathsf{cred} = (\mathbf{s}, x)$. This explicit interaction with the issuer allows the issuer to perform some additional check to decide whether $\mathcal{M}_i$ is allowed to join. The host stores the credential as $(\mathsf{sid}, \mathcal{M}_i, \mathsf{cred})$ and the TPM stores its secret key as $(\mathsf{sid}, \mathcal{H}_j, \mathsf{tsk})$ i.e., both "remember" with whom they joined. The join ends with $\mathcal{M}_i$ outputting $(\mathsf{JOINCOMPLETE}, \mathsf{sid}, \mathsf{jsid})$.

SIGN. Signing is a protocol run between a TPM $\mathcal{M}_i$ and a host $\mathcal{H}_j$. Again, we use a unique sub-session identifier ssid to allow for multiple sign sessions and unique identification of the particular session in the UC interfaces. The host $\mathcal{H}_j$ upon input $(\mathsf{SIGN}, \mathsf{sid}, \mathsf{ssid}, \mathcal{M}_i, m, \mathsf{bsn})$, retrieves its join record $(\mathsf{sid}, \mathcal{M}_i, \mathsf{cred})$ and aborts if no such record is found. It sends $(\mathsf{ssid}, m, \mathsf{bsn})$ to the TPM which then checks that a key record $(\mathsf{sid}, \mathcal{H}_j, \mathsf{tsk})$ exists and outputs $(\mathsf{SIGNPROCEED}, \mathsf{sid}, \mathsf{ssid}, m, \mathsf{bsn})$. The signature is completed when $\mathcal{M}_i$ receives the input $(\mathsf{SIGNPROCEED}, \mathsf{sid}, \mathsf{ssid})$, upon which it computes nym. The explicit input from the TPM is necessary to ensure that the TPM in fact "approved" the attestation of m and bsn. Finally, the host outputs the jointly computed signature as $(\mathsf{SIGNATURE}, \mathsf{sid}, \mathsf{ssid}, \sigma)$.

VERIFY AND LINK. Here both algorithms are simply re-labeled as UC interfaces with the ipk being replaced with sid, i.e. both algorithms are made available through interfaces $(\mathsf{VERIFY}, \mathsf{sid}, m, \mathsf{bsn}, \sigma, \mathsf{RL})$ and $(\mathsf{LINK}, \mathsf{sid}, \sigma, m, \sigma', m', \mathsf{bsn})$ respectively.

We are now ready to state the security of Π_{LDAA} in the following theorem.

Theorem 4.4.1. *The protocol Π_{LDAA} presented in Section 4.3 securely realizes $\mathcal{F}_{\mathsf{daa}}$ [CDL16] in the $(\mathcal{F}_{auth^*}, \mathcal{F}_{\mathsf{ca}}, \mathcal{F}_{smt}, \mathcal{F}^{D}_{\mathsf{crs}})$-hybrid and random oracle model under static corruptions, if the* M-LWE*,* MSIS*,* M-NTRU*, and* Int-MNTRU-ISIS$_f$ *assumptions hold.*

4.4.2 *Proof Sketch*

To show that no environment, $\mathcal{E}$, can distinguish the real world, in which it is working with Π_{LDAA} and adversary $\mathcal{A}$, from the ideal world, in which it uses $\mathcal{F}_{\mathsf{daa}}$ with simulator $\mathcal{S}$, we use a sequence of games. We start with the real world protocol execution. In the next game we construct one entity C that runs the real world protocol for all honest parties. Then we split C into two pieces, a functionality $\mathcal{F}$ and a simulator $\mathcal{S}$, where $\mathcal{F}$ receives all inputs from honest parties and sends the outputs to honest parties. We start with a useless functionality, and gradually change $\mathcal{F}$ and update $\mathcal{S}$ accordingly, to end up with the full $\mathcal{F}_{\mathsf{daa}}$ and a satisfying simulator.

The proof closely follows the structure of the UC proof by Camenisch et al. in [CDL16], with the crucial steps occurring in Game 7, where the signatures of honest platforms are replaced by signatures on "dummy" keys (guaranteeing *anonymity*), and Games 12–15 where we let the functionality enforce the expected *unforgeability* and *non-frameability* properties. For unforgeability we rely on the security of credentials created by the issuer. This is exactly

the Int-MNTRU-ISIS$_f$ assumption which guarantees that only the issuer can produce the preimage $\mathbf{s}$ which forms part of the credential.

GAME 1. This is the real world protocol.

GAME 2. The challenger $\mathcal{C}$ now receives all inputs and simulates the real world protocol for honest parties. Since $\mathcal{C}$ gets all inputs, it can simply run the real world protocol. It also simulates all hybrid functionalities, but does so honestly, so $\mathcal{E}$ does not see any difference. By construction, this is equivalent to the previous game.

GAME 3. We now split $\mathcal{C}$ into a "dummy functionality" $\mathcal{F}$ and simulator $\mathcal{S}$. $\mathcal{F}$ receives all inputs, and simply forwards them to $\mathcal{S}$. $\mathcal{S}$ simulates the real world protocol and sends the outputs it generates to $\mathcal{F}$, who then outputs them to $\mathcal{E}$. This game only restructures the previous game.

GAME 4. In this game we let our intermediate $\mathcal{F}$ handle the setup related interfaces using the procedure specified in $\mathcal{F}$. Consequently, $\mathcal{F}$ expects to receive the algorithms $(\mathsf{ukgen}, \mathsf{sig}, \mathsf{ver}, \mathsf{link}, \mathsf{identify})$ from the simulator. For $\mathsf{ukgen}, \mathsf{ver}$, and link, $\mathcal{S}$ can simply provide the algorithms from the real-world protocol, where it omits the revocation check from ver. The sig algorithm will be a combination of the join and sign procedure though, as it will be used to create anonymous signatures for honest platforms for which it uses a fresh TPM key whenever the platform signs w.r.t. a new basename. Thus, to internally create signatures via sig, the algorithm must first create a valid membership credential for the freshly chosen tsk and then sign with this new credential. So sig must contain the issuer's private key, which the simulator $\mathcal{S}$ has to be able to get.

When $\mathcal{I}$ is honest, $\mathcal{S}$ is running the issuer, i.e., it knows its secret key and sets the sig algorithm accordingly.

When $\mathcal{I}$ is corrupt, $\mathcal{S}$ starts the simulation when the issuer registers his key with $\mathcal{F}_{\mathsf{ca}}$ that is controlled by the simulator. Since the public key comes with a proof of knowledge of the issuer's secret key, $\mathcal{S}$ can extract the secret key from there and define the sig algorithm accordingly. By the zero-knowledge property of the proof system, this game hop is indistinguishable for the adversary.

Finally, for $\mathsf{identify}$ which is used to check whether a signature $(\sigma, m, \mathsf{bsn})$ belongs to a certain tsk, we use roughly the same procedure as for revocation checks. That is, the algorithm parses $\sigma = (\mathsf{nym}, d, \mathsf{bsn})$, $\mathsf{tsk} = (\mathbf{e}_1, *)$, and

checks that $\|(\mathsf{nym} - \mathbf{De}_1)\|$ is small. If so it outputs 1, and 0 otherwise. Recall that we compute pseudonyms for random d when $\mathsf{bsn} = \perp$, so this check works for all cases.

GAME 5. $\mathcal{F}$ now handles the verify and link queries using the provided algorithms ver and link from the previous game, rather than forwarding the queries to $\mathcal{S}$. We do not let $\mathcal{F}$ perform the additional checks (Checks (ix) – (xvi)) done by $\mathcal{F}$, though, but add these only later. For Check (xii), $\mathcal{F}$ rejects a signature when a matching $\mathsf{tsk}' \in \mathsf{RL}$ is found, but does not exclude honest TPMs from this check yet.

Because verify and link do not involve network traffic, the simulator does not have to simulate traffic either, we must only make sure the outputs do not change. $\mathcal{F}$ executes the algorithms that $\mathcal{S}$ supplied, and $\mathcal{S}$ supplied them in such a way that they are equivalent to the real world algorithms, so the outcome will clearly be equivalent.

GAME 6. In this step we change $\mathcal{F}$ to also handle the join-related interfaces, meaning it will receive the inputs and generate the outputs. We let $\mathcal{F}$ run the same procedure as $\mathcal{F}$, but again omit the additional checks (Checks (ii)–(iv)).

In the final join interface $\mathsf{JOINCOMPLETE}$, the simulator has to provide the secret key tsk of the TPM. When the TPM is honest, $\mathcal{S}$ knows the key anyway and uses it towards $\mathcal{F}$. If the TPM is corrupt and either the issuer or host is honest, $\mathcal{S}$ extracts the vector $\mathbf{e}_1$ from the proof π_{join} that it receives in the role of the honest $\mathcal{I}$ or $\mathcal{H}_j$ using the knowledge soundness of the proof system and sets $\mathsf{tsk} \leftarrow (\mathbf{e}_1, \perp)$. Note that we do not extract nor set the part e_3 of the TPM's secret key. This has no impact though, as tsk will only be used for internal checks by $\mathsf{identify}$ for which only $\mathbf{e}_1$ is used.

Finally, note that $\mathcal{F}$ sets $\mathsf{tsk} := \perp$ when both the TPM and host are honest. However, this has no impact yet, as the signatures are still created by the simulator and the verify and link interfaces of $\mathcal{F}$ do not run the additional checks that make use of the internally stored records and keys.

Overall, this game hop is indistinguishable by the knowledge soundness of π_{join}.

GAME 7. We now transform $\mathcal{F}$ such that it internally handles the signing queries of *honest platforms* instead of merely forwarding them to $\mathcal{S}$. Thus, this game hop proves the **anonymity** of our DAA scheme.

Again, $\mathcal{F}$ uses the sign interfaces from $\mathcal{F}$, with the difference that it does not perform the Check (v) – (vii) which we only add in a later game.

When both the TPM and the host are honest, $\mathcal{F}$ creates the signatures internally in an unlinkable way: It chooses a new tsk per basename and TPM, or per signature when $\mathsf{bsn} = \perp$ and then runs the sig algorithm for that fresh key. As described earlier, sig starts by internally "issuing" a membership credential on tsk using the issuer's secret key that is included in sig. $\mathcal{F}$ keeps the internally chosen keys $\langle \mathcal{M}_i, \mathsf{bsn}, \mathsf{tsk} \rangle$ in a list `DomainKeys` to ensure consistency if a TPM wishes to reuse the basename.

This change is indistinguishable under the M-LWE assumption, the reduction can be done in a straightforward manner, using a hybrid argument to to replace the signatures one-by-one.

GAME 8. We change $\mathcal{F}$ such that it no longer informs $\mathcal{S}$ which message and basename are being signed. Thus, when the TPM and host are both honest, $\mathcal{S}$ does not learn m or bsn but only its leakage $l(m, \mathsf{bsn})$. Recall, that signatures for honest platforms are generated by the functionality now, so $\mathcal{S}$ merely as to mimic communication between the honest TPM and host.

GAME 9. We now add the constraint that when $\mathcal{I}$ is honest, $\mathcal{F}$ only allows platforms that joined to sign, which is checked via the list `Members`. Note that for our simulation we only care about platforms that are at least "partially" honest, i.e., the host and/or TPM are honest, as otherwise there is nothing to simulate. For such platforms, this check will not change the view of $\mathcal{E}$ using the simulator $\mathcal{S}$ from the previous game: in the real world, an honest host and TPM both check that they have a joined before signing. In the ideal world, $\mathcal{S}$ makes join queries towards $\mathcal{F}$ ensuring that the joined platforms (with honest entities) are in `Members`, and thus $\mathcal{F}$ still allows any signing that could take place in the real world.

GAME 10. In this game we let $\mathcal{F}$ additionally check the validity of every new tsk that is generated or received in the join and sign interface.

If the TPM is corrupt, $\mathcal{F}$ checks that $\mathsf{CheckSkCorrupt}(\mathsf{tsk}) = 1$ for the tsk that the simulator extracted from π_{join} (Check (iv)). This check prevents the adversary from choosing different keys $\mathsf{tsk} \neq \mathsf{tsk}'$ that both fit to the same signature. My uniqueness of solutions to M-LWE only one secret $\mathbf{e}_1$ satisfying the nym-part of the signature (also for the "random" pseudonyms when $\mathsf{bsn} = \perp$). As there is only a single tsk for every valid signature where $\mathsf{identify}(\sigma, m, \mathsf{bsn}, \mathsf{tsk}) = 1$, this check will never fail.

For keys of honest TPMs, $\mathcal{F}$ verifies that $\mathsf{CheckSkHonest}(\mathsf{tsk}) = 1$ whenever it receives or generates a new tsk (Check (iii) and (v)). With these checks we avoid the registration of keys for which matching signatures already exist. Since keys for honest TPMs are chosen uniformly at random from an exponentially large group and every signature has exactly one matching key, the chance that a signature under that key already exists is negligible.

GAME 11. We now add the checks to $\mathcal{F}$ that $\mathcal{F}$ runs in the sign interfaces when internally generating signatures for honest platforms. After creating a signature, $\mathcal{F}$ checks whether the signature verifies and matches the right key (Check (vi) and (vii)). As $\mathcal{S}$ supplied proper algorithms and the signature scheme is complete, these checks will obviously always succeed.

$\mathcal{F}$ also checks with the help of its internal key records `Members` and `DomainKeys` that no one else already has a key which would match this newly generated signature (Check viii). As signatures match only a single TPM key and we choose keys of honest platforms at random from a large domain, this can happen only with negligible probability.

In the next four game hops, we let $\mathcal{F}$ perform the four additional checks that are done by $\mathcal{F}$ in the verification interface, i.e., we rely on $\mathcal{F}$ to enforce the desired **unforgeability** and **non-frameability** guarantees. We now show that this check does not change the verification outcome, as any signature that would previously pass will still pass.

GAME 12. $\mathcal{F}$ now performs an additional check during verification, it checks whether it finds multiple tsk values matching this signature, and if so, it rejects the signature. It is easy to see that there is only one tsk per signature for which $\mathsf{identify}$ will output 1, as there is only one $\mathbf{e}_1$ that can lead to the pseudonym nym (by the hardness of $\mathsf{M\text{-}LWE}$).

GAME 13. When $\mathcal{I}$ is honest, $\mathcal{F}$ now only accepts signatures on tsk values that $\mathcal{I}$ has issued a membership credential on. Under the existential unforgeability of the membership credential, this check changes the verification outcome only with negligible probability. The unforgeability of our signature scheme used by the issuer to (blindly) sign the TPM's key is based on the $\mathsf{Int\text{-}MNTRU\text{-}ISIS}_f$ assumption.

GAME 14. $\mathcal{F}$ now prevents forging signatures using an honest TPM's tsk. If the environment can distinguish this game hop, i.e., it can create a valid

signature σ for message m and basename bsn that traces via $\mathsf{identify}$ to an honest TPM, but that TPM has never signed m, bsn. We can use this to break the M-LWE problem. Again, the reduction can be done in a straightforward manner: We use the M-LWE challenge as $\mathbf{u}_1$ for a randomly chosen honest TPM during the join protocol (with the possibly corrupt issuer). DAA signatures for this honest TPM are merely simulated. If we see a valid signature that we never created, we extract $\mathbf{e}_1$ from the accompanying π_{sign} and output that as the M-LWE solution.

GAME 15. Check (xii) is added to $\mathcal{F}$, this ensures that honest TPMs are not being revoked. If an honest TPM is simulated by means of the Ring-LWE problem instance, if a proper key RL is found, it must be the secret key of the target instance. This is again equivalent to solving the M-LWE problem.

GAME 16. We now let $\mathcal{F}$ perform all the additional checks $\mathcal{F}$ makes for link queries. The output of $\mathcal{F}$ based on these checks is still consistent with the output which the link algorithm would give: If there is a tsk that matches one signatures but not the other, by the knowledge soundness of π_{sign} we have that the pseudonyms are not based on the same tsk, and thus must differ which results in link outputting 0. If there is a tsk that matches both signatures, again by knowledge soundness of π_{sign} we have that the pseudonyms are based on the same tsk and must be equal, resulting in link outputting 1.

Now $\mathcal{F}$ is equal to $\mathcal{F}_{\mathsf{daa}}$, concluding our proof sketch.

4.5 PERFORMANCE

In this section we analyze the performance of our new lattice-based DAA scheme.

4.5.1 *Computing parameters.*

Let us begin by setting the target bit-security parameter $\lambda = 128$. Throughout this section we aim for an implicit lattice dimension of at least 1024 for all lattice assumptions. As is common for achieving the above security level for this dimension, we use a prime 32-bit modulus $q = 2^{32} - 99$. We then ensure a negligible soundness error as prescribed Lemma 4.2.10 by choosing the soundness boosting parameter $\tau = 4$, which ensures that $q^{-\tau}$ and q^{-d} are negligible for all sensible values of d. Let us denote by $m_1^{\mathsf{join}}, m_2^{\mathsf{join}}$ the

lengths of the vectors $\mathbf{s}_1$ and $\mathbf{s}_2$ in the join protocol. Similarly, we define $m_1^{\mathsf{join}}, m_2^{\mathsf{join}}$ to be the corresponding quantities in the sign protocol.

We now choose values of $k_{\mathsf{MSIS}}^{\mathsf{Join}}$ and m_2^{join} so that

$$\mathsf{M\text{-}LWE}_{(k_{\mathsf{MSIS}}^{\mathsf{sign}}+256/d+\tau+1),m_2^{\mathsf{join}}-(k_{\mathsf{MSIS}}^{\mathsf{sign}}+256/d+\tau+1),\chi,q}$$
$$\text{and } \mathsf{MSIS}_{k_{\mathsf{MSIS}}^{\mathsf{sign}},m_1^{\mathsf{join}}+m_2^{\mathsf{join}},B_{\mathsf{join}}},$$

are hard. Similarly, we choose $k_{\mathsf{MSIS}}^{\mathsf{sign}}$ and m_2^{sign}, so that both

$$\mathsf{M\text{-}LWE}_{(k_{\mathsf{MSIS}}^{\mathsf{sign}}+256/d+\tau+1),m_2^{\mathsf{sign}}-(k_{\mathsf{MSIS}}^{\mathsf{sign}}+256/d+\tau+1),\chi,q}$$
$$\text{and } \mathsf{MSIS}_{k_{\mathsf{MSIS}}^{\mathsf{sign}},m_1^{\mathsf{sign}}+m_2^{\mathsf{sign}},B_{\mathsf{sign}}}$$

are hard where χ is the uniform distribution of ternary elements of R and B_{join} and B_{sign} are the two-norms of the vector $[\mathbf{s}_1\ \mathbf{s}_2]^\top$ in each phase.

In order to estimate the hardness of MSIS we use the relation due to Micciancio and Regev [MR09], which states that LLL will recover a short vector a vector of 2-norm $2^{(2\sqrt{d\log_2 q\log_2\delta})}$. δ is the root Hermite factor and $\delta < 1.0045$ gives rise to at least 128 bits of security.

For $\mathsf{M\text{-}LWE}$ hardness estimation we follow standard convention by using the estimator [APS15]. This estimates the cost of BKZ conservatively by focusing only on the cost of a single uSVP oracle call, a core operation in BKZ. The number of such calls required has been estimated to be $8d$ for a lattice dimension d, and we follow this estimate.

We must also estimate the security of the issuer's NTRU public key. Here, we use an implicit lattice dimension of $\hat{n}\cdot\hat{d}$ and modulus q. We use standard deviation

$$\sigma_{\mathsf{NTRU}} = \gamma\cdot\frac{1}{\sqrt{\hat{d}(\hat{n}+2-i)}}\cdot q^{1/(\hat{d}+1)}, \tag{4.22}$$

for constructing the i-th row of $\mathbf{h} := \mathbf{F}^{-1}\cdot\mathbf{g}$ as prescribed by the pre-image sampling techniques of [CPS+20], which requires $\gamma = 1.24$ for $\hat{d} = 3$. We then run the estimator of [Dv21] for the hardness of NTRU with these parameters. Note that these calculations reveal that taking $\hat{n} > 3$ would lead us into so-called 'overstretched' parameters since larger $\hat{n}$ forces smaller NTRU keys by Equation (4.22).

Finally, we set the standard deviations σ_1, σ_2 and σ_3 needed for the correctness and zero-knowledge properties of the LNP proof system. These are set as follows:

Scheme	TPM key size	signature size
[CKLL19]	3KB	> 2MB
Ours	0.77KB	37.7KB

TABLE 4.2: Key and signatures sizes for the previous state-of-the-art lattice DAA scheme [CKLL19] as compared to this work.

$$\sigma_1 := \nu\|\mathbf{s}_1\|_2,\ \sigma_2 := \nu\sqrt{m_2 d},\ \sigma_3 := \|\mathbf{s}_1\|_2\omega_{\max}.$$

Finally, we set the bounds B_1, B_2, and B_3 for the verification checks using the standard tail bound of $\sigma\sqrt{2d}$ for the two-norm of an elements of length d, sampled from a discreet gaussian of parameter σ. The resulting parameter set is shown in Table 4.3.

TOTAL SIZES. We now compute the total communication cost for Π_{LDAA}. Beginning with the proofs, we inherit the proof size for π_{join} from [LNP22, Section 6.2] which yield a proof size for π_{join} of 14.4KB, where one can use their Huffman encoding technique on $\mathbf{z}_1$ since the committed vector $\mathbf{e}_1$ can be seen as coming from a discreet gaussian on parameter $\sqrt{2/3}$ (for ternary $\mathbf{e}_1$).

Next, using the parameters of Table 4.3, we have that the size of π_{sign} is 33.6KB. A signature also comprises a pseudonym nym and basename bsn size $8d\log q$ and 128 bits respectively. Thus, the final signature size is 37.7 KB.

COMPARISON WITH EXISTING WORKS. Table 4.2 displays the TPM key sizes and DAA signature sizes for our work and we include the estimates provided for the same quantities in [CKLL19] which represents the current state-of-the-art. Furthermore, we note that the signature size provided in [CKLL19] is a very rough lower bound since the authors calculate this figure by arguing that 'most polynomials in R_q have coefficients smaller than q'. Our numbers are based on the actual storage needed for fully-fledged ring elements. All previous works give only asymptotic parameters. The scheme in [ECE+18] uses TPM keys with 24 polynomials and the one in [EE17] requires 49 polynomials. Even if they were able to use the same ring dimension as we do, their key and signature sizes would be vastly larger and impractical.

Thus, our work represents a four-fold reduction in TPM key size and

parameters	description	value
λ	security parameter	128
q	modulus	$2^{32} - 99$
d	ring dimension for R	128
$\hat{d}$	ring dimension for NTRU ring $\hat{R}$	384
ξ	max coeff. of chal. space element	2
ν	k-bound on chal. space element	59
τ	# gargage terms g_j for boosting soundness	4
m_2	len. of $\mathbf{s}_2$ in ABDlOP commitment	25
$k^{\mathsf{join}}_{\mathsf{MSIS}}$	height of $\mathbf{A}_1, \mathbf{A}_2$ in ABDLOP for join	9
$k^{\mathsf{sign}}_{\mathsf{MSIS}}$	height of $\mathbf{A}_1, \mathbf{A}_2$ in ABDLOP for sign	9
χ	distribution from which $\mathbf{s}_2$ is sampled	S_1
N	max # of platforms that can join	2^{40}
n	dim. of TPM keys $\mathbf{e}_1, \mathbf{e}_2 \in R_q$	8
γ	Gram-Schmidt slack for falcon sampling	1.5
$\hat{n}$	dimension of NTRU public key $\mathbf{h}$	3
σ_{NTRU}	s.d. for sampling NTRU trapdoor	8.1
m^{join}_1	len. of witness $\mathbf{s}_1$ in join	8
m^{sign}_1	len. of witness $\mathbf{s}_1$ in sign	34
$\omega_{\max}(128)$	approx. rang proof param.	$\sqrt{337}$
σ^{join}_1	s.d. for sampling y_1 in join	4480
σ^{join}_2	s.d. for sampling y_2 in join	7920
σ^{join}_3	s.d. for sampling y_3 in join	587
σ^{sign}_1	s.d. for sampling y_1 in sign	1929375
σ^{sign}_2	s.d. for sampling y_2 in sign	7920
σ^{sign}_3	s.d. for sampling y_3 in sign	252990
B^{join}_1	verification bound for $\mathbf{z}_1$ in join	202742
B^{join}_2	verification bound for $\mathbf{z}_2$ in join	633568
B^{join}_3	verification bound for $\mathbf{z}_3$ in join	$1.7\sqrt{256}$
B^{sign}_1	verification bound for $\mathbf{z}_1$ in sign	87313590
B^{sign}_2	verification bound for $\mathbf{z}_2$ in sign	633568
B^{sign}_3	verification bound for $\mathbf{z}_3$ in sign	$1.7\sqrt{256}$

TABLE 4.3: Example parameters for Π_{LDAA}.

importantly a reduction by 1.5 orders of magnitude (53 times) in signature size over the state-of-the-art.

5

PRACTICAL LATTICE-BASED ELECTRONIC VOTING

5.1 INTRODUCTION

In this chapter, we turn our attention to electronic voting. This work follows naturally from our previous focus on NTRU lattices and seeks to make contributions to one of the most fundamental tools in a modern democratic society; voting. Moreover, the electronic voting protocol we present seeks to further embed privacy as a core design principle of our digital infrastructure.

5.2 PRELIMINARY

Here we details the preliminary material, local to this chapter, necessary for the presentation of our NTRU analysis and electronic voting scheme.

5.2.1 *Computational Assumptions*

THE NTRU PROBLEM. We give the historical presentation of the NTRU problem as it is more convenient for our analysis in Section 5.3. We note that some works refer to this problem as the 'search/decisional short polynomial ratio' problem. Furthermore, one can consider the so-called 'module' NTRU problem [CPS+20], which considers the ratio of matrices of polynomials $\mathbf{F}$ and $\mathbf{G}$, (S/D)SMR as defined in the preliminary of this thesis. Our analysis and applications can naturally be extended to the module setting, so for ease of presentation, we use the basic (polynomial) NTRU formulation [HPS98b]. We will also only work with the ring variants of other lattice assumptions (as opposed to their module variants) and so we give their explicit formulations below for completeness.

Definition 5.2.1 (Search/Decision NTRU)**.** *Let $q > 2$ be a prime, d be the ring dimension, and $D_{\sigma_{\mathsf{NTRU}}}$ be a distribution over R_q. Sampling $(f, g) \leftarrow D^2_{\sigma_{\mathsf{NTRU}}}$ with rejection if f is not invertible in R_q, define $h = g/f \in R_q$. The search-$\mathsf{NTRU}_{q,d,\sigma_{\mathsf{NTRU}}}$ problem is, given h, to recover any rotation $(X^i f, X^i g)$ of the pair (f, g). The decision-$\mathsf{NTRU}_{q,d,\sigma_{\mathsf{NTRU}}}$ problem is, given h, to decide if h is computed as $h = g/f$ for $(f, g) \leftarrow D^2_{\sigma_{\mathsf{NTRU}}}$ or if h is sampled uniformly from R_q.*

THE RLWE AND RSIS PROBLEMS. We define the standard lattice-hardness problems over rings [Ajt96, Reg05, LPR10]. Whilst we already define the following two problems in their more general, module, form in Section 2.3.2, we will be using their ring variant (i.e. module rank 1) in this chapter and thus provide their full definition in this setting for completeness.

Definition 5.2.2 (Ring Learning with Errors)**.** *Let $q > 2$ be a prime, d be the ring dimension, $D_{\sigma_{\mathsf{R\text{-}LWE}}}$ be a distribution over R_q, and $\mathcal{A}$ a PPT algorithm that makes at most Q oracle queries. Then the advantage of $\mathcal{A}$ in solving the ring learning with errors $\mathsf{R\text{-}LWE}_{d,q,Q,\sigma_{\mathsf{R\text{-}LWE}}}$ problem is defined as*

$$\mathcal{A}^{\mathsf{R\text{-}LWE}}_{d,q,Q,\sigma_{\mathsf{R\text{-}LWE}}}(\mathcal{A}) = \left|\Pr[\mathcal{A}^{\mathcal{O}_{\mathsf{R\text{-}LWE}}}(1^\lambda) \to 1] - \Pr[\mathcal{A}^{\mathcal{O}_{\$}}(1^\lambda) \to 1]\right|,$$

where oracles $\mathcal{O}_{\mathsf{R\text{-}LWE}}$ and $\mathcal{O}_{\$}$ are defined as

- *$\mathcal{O}_{\mathsf{R\text{-}LWE}}$: Samples $a \leftarrow R_q$, $(s_1, s_2) \leftarrow D^2_{\sigma_{\mathsf{R\text{-}LWE}}}$, and then output $(a, as_1 + s_2)$;*
- *$\mathcal{O}_{\$}$: Samples $(a, b) \leftarrow R_q \times R_q$, and then output (a, b).*

Definition 5.2.3 (Ring Short Integer Solutions)**.** *Let $q > 2$ be a prime, d be the ring dimension, $\|\cdot\|$ a norm, and $\beta \in \mathbb{R}^+$ a positive integer. The $\mathsf{R\text{-}SIS}_{d,q,\beta}$ problem is, given a uniformly random $a \in R_q$, find $s_1, s_2 \in R_q$ such that $as_1 + s_2 = 0 \in R_q$ and $\|s_1, s_2\| \leq \beta$.*

5.2.2 *NTRU Cryptosystem*

In this chapter, we will use the provably secure variant of the NTRU cryptosystem first presented by Steinfeld and Stehlé in [SS13]. This scheme relies on the hardness of both the RLWE and NTRU assumptions. Note we make two minor modifications to ensure perfectly correct decryption: (1) encryption randomness is sampled from a bounded distribution, and (2) the secret keys f and g are rejected unless their ℓ_2 norm is below a given bound. When sampled accordingly, this limitation has only a negligible effect on the completion probability of the key generation algorithm and the entropy of resulting keys.

Setup. Let $p \ll q$ be primes and d a power of two which define the rings R_p and R_q. Messages lie in R_p. Let $\sigma_{\mathsf{NTRU}} \in \mathbb{R}$ and $D_{\sigma_{\mathsf{NTRU}}}$ a discrete Gaussian distribution over R with standard deviation σ_{NTRU}, $t \in (1, 2]$ and $\sigma_{\mathsf{LWE}} \in \mathbb{N}$. Let the setup parameters be $\mathsf{sp} = (d, p, q, \sigma_{\mathsf{NTRU}}, t, \nu)$. The encryption scheme is described in Figure 5.1.

Key Generation $\mathsf{KeyGen}_{\mathsf{NTRU}}(\mathsf{sp})$. Given input $\mathsf{sp} = (d, p, q, \sigma_{\mathsf{NTRU}}, t, \nu)$:

1. $f, g \leftarrow D_{\sigma_{\mathsf{NTRU}}}$; if $f \notin R_q^\times$ or $f \not\equiv 1 \in R_p$, resample.
2. If $\|f\|_2, \|g\|_2 > t \cdot \sqrt{d} \cdot \sigma_{\mathsf{NTRU}}$, restart.
3. Return $\mathsf{sk} = f$, $\mathsf{pk} = h := g/f \in R_q$.

Encryption $\mathsf{Enc}_{\mathsf{NTRU}}(m, \mathsf{pk})$. Given message $m \in R_p$ and public key $\mathsf{pk} = h$:

1. Sample encryption randomness $s, e \leftarrow S_\nu$.
2. Return ciphertext $c = p \cdot (hs + e) + m \in R_q$.

Decryption $\mathsf{Dec}_{\mathsf{NTRU}}(c, \mathsf{sk})$. Given ciphertext c and key $\mathsf{sk} = f$:

1. Return message $m = (f \cdot c \mod q) \mod p$.

FIGURE 5.1: Adapted `NTRU-Encrypt` [SS13].

Lemma 5.2.4 (`NTRU-Encrypt` Security)**.** *Let* $p \cdot d \cdot t \cdot \sigma_{\mathsf{NTRU}}(2\nu + 1/2) < \lfloor q/2 \rfloor$. *Then the encryption scheme in Figure 5.1 is (perfectly) correct. Moreover, assuming the hardness of the* $\mathsf{NTRU}_{q,d,\sigma_{\mathsf{NTRU}}}$ *and* $\mathsf{R\text{-}LWE}_{d,q,\chi}$ *problems, the scheme is* IND-CPA *secure.*

5.2.3 *The BDLOP Commitment Scheme*

Here we recall the BDLOP commitment scheme from [BDL+18]. For simplicity, we present the scheme instantiated over rings instead of modules, committing to only one ring element at a time:

$\mathsf{Setup}(1^\lambda)$: On input a security parameter λ, samples uniformly random a_1, a_2, a_3 from R_q and outputs the public commitment key pk_C defined as:

$$\mathsf{pk}_\mathsf{C} = \begin{bmatrix} \vec{a}_1 & 0 \\ \vec{a}_2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & a_1 & a_2 & 0 \\ 0 & 1 & a_3 & 1 \end{bmatrix}.$$

$\mathsf{Com}(\mathsf{pk}_C, x)$: On input a public commitment key pk_C and an element x in R_q, samples a vector $\vec{r} \in R_q^3$ such that $\|\vec{r}\|_\infty \leq B_{\mathsf{Com}}$, and computes the commitment as:

$$\mathsf{com} = \begin{bmatrix}\vec{c}_1\\ \vec{c}_2\end{bmatrix} = \begin{bmatrix}1 & a_1 & a_2 & 0\\ 0 & 1 & a_3 & 1\end{bmatrix} \begin{bmatrix}\mathsf{r}_1\\ \mathsf{r}_2\\ \mathsf{r}_3\\ x\end{bmatrix} = [\![x]\!].$$

It outputs the commitment com and the opening $d = (x, \vec{r}, 1)$.

$\mathsf{Open}(\mathsf{pk}_C, \mathsf{com}, d)$: On input a public commitment key pk_C, the commitment com and the opening $d = (x, \vec{r}, f)$ where $f \in \bar{\mathcal{C}}$. It verifies:

$$f \cdot \mathsf{com} \stackrel{?}{=} \begin{bmatrix}1 & a_1 & a_2 & 0\\ 0 & 1 & a_3 & 1\end{bmatrix} \begin{bmatrix}\mathsf{r}_1\\ \mathsf{r}_2\\ \mathsf{r}_3\\ f \cdot x\end{bmatrix},$$

and $\forall i \in [3]: \|\mathsf{r}_i\|_2 \stackrel{?}{\leq} 4 \cdot \sigma_{\mathsf{Com}}\sqrt{d}$. It outputs 1 if the relations hold and 0 otherwise.

The BDLOP commitment scheme is hiding if the RLWE problem is hard for vectors of ℓ_∞ norm B_{Com} over a lattice of dimension $2 \cdot d$. Furthermore, the scheme is binding if the RSIS problem is hard for vectors of ℓ_2 norm $16\sigma_{\mathsf{Com}}\sqrt{\kappa d}$ over a lattice of dimension $2 \cdot d$ [BDL+18].

5.2.4 *Proof Systems*

We now detail the various proof systems needed in the construction of our voting scheme. In particular, these building blocks are used in Section 5.4.3 to transform the passively-secure scheme into one with active security.

PROOF OF LINEARITY. In the distributed decryption phase of our scheme (Step 2 in particular), we prove well-formedness of the linear decryption share. The protocol Π_{LIN} produces a proof that a committed value v is a multiple of another committed value u with respect to a public scalar g. In our setting, we will prove $[\![E_{ij} - p^{-1}\mathsf{ds}_{ij}]\!] = -p^{-1}\mathsf{ctx}_i[\![\mathsf{dk}_j]\!]$.

The exact relation for the proof system is:

$$\mathcal{R}_{\text{LIN}} := \left\{ (x, w) \;\middle|\; \begin{array}{c} x := (\mathsf{pk}_C, \mathsf{com}_u, \mathsf{com}_v, g) \;\wedge \\ w := (d_u = (u, \vec{r}_u, f_u), d_v = (v, \vec{r}_v, f_v)) : \\ u = g \cdot v \;\wedge\; \mathsf{Open}(\mathsf{pk}_C, \mathsf{com}_u, d_u) \;\wedge \\ \mathsf{Open}(\mathsf{pk}_C, \mathsf{com}_v, d_v) \end{array} \right\}.$$

The proof of linearity π_{LIN} is computed as follows [BDL+18]:

1. Sample vectors $\vec{y}_u$ and $\vec{y}_v$ of length k over R_q according to $D_{\sigma_{\text{LIN}}}$ and compute $\vec{w}_u = \vec{a}_1 \cdot \vec{y}_u$ and $\vec{w}_v = \vec{a}_1 \cdot \vec{y}_v$ and $t = g \cdot \vec{a}_2 \cdot \vec{y}_u - \vec{a}_2 \cdot \vec{y}_v$.

2. Hash $(\vec{w}_u, \vec{w}_v, t)$ to c in $\mathcal{C}_\kappa$, and compute $\vec{z}_u = \vec{y}_u + c \cdot \vec{r}_u, \vec{z}_v = \vec{y}_v + c \cdot \vec{r}_v$.

3. Rejection sample with respect to $(\vec{y}_u, \vec{z}_u)$, and $(\vec{y}_v, \vec{z}_v)$. If it outputs 1 then output $\pi_{\text{LIN}} = (c, \vec{z}_u, \vec{z}_v)$ and otherwise restart by sampling new $(\vec{y}_u, \vec{y}_v)$.

The verifier checks if $\|\vec{z}_u, \vec{z}_v\|_2 \leq 2\sigma_{\text{LIN}}\sqrt{k \cdot d}$ and if the hash of $(\vec{a}_1 \cdot \vec{z}_u - c \cdot c_{u,1}, \vec{a}_1 \cdot \vec{z}_v - c \cdot c_{v,1}, g \cdot \vec{a}_2 \cdot \vec{z}_u - \vec{a}_2 \cdot \vec{z}_v + c_{v,2} + g \cdot c_{u,2})$ equals c. It outputs 1 if all checks verify, and otherwise it outputs 0.

Using the improved rejection sampling techniques from [LNS21], we set $\sigma_{\text{LIN}} = B_{\mathsf{Com}} \cdot \kappa\sqrt{d}$. The size of π_{LIN} is $2kd \log_2(4\sigma_{\text{LIN}})$ bits.

PROOF OF SHUFFLE. The exact relation for the proof system is:

$$\mathcal{R}_{\text{SHUF}} := \left\{ (x, w) \;\middle|\; \begin{array}{c} x := (\{(\mathsf{com}_i, \bar{u}_i)\}_{i \in [\tau]}) \;\wedge \\ w := (\{d_i = (u_i, \vec{r}_i, f_i)\}_{i \in [\tau]}, \rho) : \\ \rho \in \mathsf{Perm}[\tau] \;\wedge\; \forall i \in [\tau] : \; u_i = \bar{u}_{\rho(i)} \\ \wedge\; \mathsf{Open}(\mathsf{pk}_C, \mathsf{com}_i, d_i) \end{array} \right\}.$$

The proof of shuffle π_{SHUF} is computed as follows [ABG+21, Section 4]:

1. Hash the statement to get a uniform value and then shift all commitments and messages to u'_i and $\bar{u}'_i$ (the commitments are additionally homomorphic).

2. For all $i \in [\tau - 1]$, sample random values θ_i and commit to random linear combinations of the form $[\![D_i]\!] = [\![\theta_{i-1} \cdot u'_i + \theta_i \cdot \bar{u}'_i]\!]$ (where $\theta_0 = \theta_\tau = 0$).

3. Hash the commitments to get a uniform challenge β. Then for all $i \in [\tau]$ compute s_i so that it solves the linear system with respect to β.

4. For all $i \in [\tau]$, compute proofs of linearity for the commitment equations of the form $[\![D_i]\!] = s_{i-1}[\![u'_i]\!] + s_i \cdot \bar{u}'_i$ (where $s_0 = \beta$ and $s_\tau = (-1)^\tau \beta$).

The verifier accepts if all proofs of linearity are valid. This proof π_{SHUF} consists of one ring element, one commitment and one proof of linearity per shuffled element. Using the proof of linearity π_{LIN} from above, the size of π_{SHUF} is $\tau d(2k \log_2(4\sigma_{\text{LIN}}) + 3\log_2 q)$ bits.

AMORTIZED PROOFS OF SHORTNESS. The protocol Π_{SMALL}, produces a proof that a batch of equations $\vec{A}\vec{s}_i = \vec{t}_i$ for $i \in [\ell]$ is satisfied for a set of secret vectors $\vec{s}_i$ with ℓ_∞ norm bounded by ν. The exact relation for the proof system is:

$$\mathcal{R}_{\text{SMALL}} := \left\{ (x, w) \;\middle|\; \begin{array}{c} x := (\mathsf{pk}_C, \{\mathsf{com}_i\}_{i\in[\ell]}) \;\wedge \\ w := (\{d_i = (u_i, \vec{r}_i, f_i)\}_{i\in[\ell]}) : \\ \forall i \in [\ell] :\; \|u_i\|_\infty \le \nu \;\wedge\; \mathsf{Open}(\mathsf{pk}_C, \mathsf{com}_i, d_i) \end{array} \right\}.$$

The proof of shortness π_{SMALL} is quite involved, combining error-correcting codes, Merkle trees, Lagrange interpolation and proximity testing, and we refer to [ABGS23] for details. The proof size, for batch size ℓ of ternary secret vectors, is given in [ABGS23, Equation (1)] as

$$(3vd + (3\ell + 2)\eta)\log_2 q + 2\lambda\eta(1 + \log_2 \gamma) \text{ bits},$$

using an $[\gamma, \mu, \iota]$ Reed-Solomon Code with code-length γ, message length μ and minimal distance ι where $\mu = d(k+2) + \eta \le \gamma < q$ for encoding randomness of length η. λ is the security parameter. The soundness of the proof is given as

$$2 \cdot \max\left\{ 2\left(\frac{\mu'}{\gamma - \eta}\right)^\eta, \frac{1}{q-\ell} + \left(1 - \frac{\mu' - \mu}{6\gamma}\right)^\eta,\; 2 \cdot \left(1 - \frac{2(\mu' - \mu)}{3\gamma}\right)^\eta, \frac{18\ell}{q-\ell} \right\},$$

for a choice of message length μ' such that $\mu \le \mu' \le \gamma < q$.

AMORTIZED PROOFS OF BOUNDEDNESS. To prove boundedness on the large noise drowning terms E_i in our voting scheme, we define Π_{BND} to be a slightly adapted version of the Π_{SMALL}, protocol used to prove boundedness of very small values, see Section 5.2.4 for full details. The main idea of Π_{BND} is that we use bit-decomposition of the integers E_{ij} to produce a long vector with small entries. This allows for the application Π_{SMALL}, to prove an exact bound on E_i.

The previous work by Aranha et al. [ABGS23] used the amortised relaxed proofs by Baum et al. [BBC+18] to get smaller proof sizes of the cost of slightly increasing the overall parameters of the voting scheme because of the slack inherent in the proof system. In practice this leads to a slightly larger modulus q but have no impact on the ring dimension d. However, in our setting, we get better parameters in practice for the whole scheme when giving exact proofs of boundedness, even though the proofs themselves are larger. The exact relation for the proof system, with batch size ℓ' and secret vectors bounded in the ℓ_∞ norm by B_{Drown}, is:

$$\mathcal{R}_{\text{BND}} := \left\{ (x, w) \;\middle|\; \begin{array}{c} x := (\mathsf{pk}_C, \{\mathsf{com}_i\}_{i\in[\ell']}) \wedge \\ w := (\{d_i = (u_i, \vec{r}_i, f_i)\}_{i\in[\ell']}) : \\ \forall i \in [\ell'] : \ \|u_i\|_\infty \leq B_{\mathsf{Drown}} \wedge \\ \mathsf{Open}(\mathsf{pk}_C, \mathsf{com}_i, d_i) \end{array} \right\}.$$

Since Π_{SMALL}, is a proof system that scales with the number of possible values of the secret vectors, we use bit decomposition techniques to limit a blow-up in terms of running time, memory usage and proof size, to the cost of proving knowledge of longer secret vectors.

Any integer E between 0 and q can be represented in base b as $E = [b_0 \ b_1 \ ... \ b_\zeta] \circ [1 \ b \ ... \ b^\zeta]$ for unique coefficients b_i between 0 and $b-1$ and $\zeta = \lceil \log_b q \rceil - 1$ where $\circ$ is the dot product. This can be naturally extended to vectors, matrices and modules, particularly for our commitment matrix A. Since the commitment randomness is already short, we only need to decompose the last element in the secret vector, and we can do so in the following way (note that we abuse notation, where after $(*)$ the elements before $|$ are in R_q and the elements after are in $\mathbb{Z}_q$, but any element in R_q can be represented in $\mathbb{Z}_q^d$):

$$\begin{aligned} \mathbf{A}_{ij}\mathbf{s}_{ij} &= \begin{bmatrix} 1 & a_{1,1} & \mathbf{a}_{1,2} & |0 \\ 0 & 1 & \mathbf{a}_{2,2} & |1 \end{bmatrix} \begin{bmatrix} \vec{r}_{E_{ij}} \\ E_{ij} \end{bmatrix} \\ &\overset{(*)}{=} \begin{bmatrix} 1 & a_{1,1} & \mathbf{a}_{1,2} & |0 & \dots & 0 \\ 0 & 1 & \mathbf{a}_{2,2} & |1 & \dots & b^\zeta \end{bmatrix} \begin{bmatrix} \vec{r}_{E_{ij}} \\ E_{0ij} \\ \vdots \\ E_{\zeta ij} \end{bmatrix} = \bar{\mathbf{A}}_{ij}\bar{\mathbf{s}}_{ij}. \end{aligned}$$

Here, the ring element E_{ij} is decomposed, and elements $E_{0ij}, \dots, E_{\zeta ij}$ have integer values between 0 and $b-1$. We note that these statements are equivalent to prove, and that the length of $\bar{\mathbf{A}}_{ij}$ is $d(k+\zeta+1)$ over $\mathbb{Z}_q$ instead of $d(k+2)$.

Finally, we use the $\Pi_{\text{SMALL}_\infty}$ protocol to prove ternary secret values as above but with a tweak: the public matrix input to the protocol is $\bar{\mathbf{A}}_{ij}$ instead of $\mathbf{A}_{ij}$, and we change the coefficient values that we are checking for in the proof. For the first $d \cdot k$ values we are checking for $(0, 1, -1)$ coefficients but for the next $d(\zeta + 1)$ values we are checking for $(0, 1, 2)$ coefficients instead (this is a small tweak of line 3 in [ABGS23, Figure 5] that does not impact the performance of the protocol in any way, these values are initially arbitrary to the proof system). Since the other secret parts are ternary, we have that $\zeta = \lceil \log_3 B_{\mathsf{Drown}} \rceil - 1$.

5.2.5 *Verifiable Mixing and Distributed Decryption*

The modern framework for designing cryptographic voting systems can be viewed as the sequential composition of a verifiable mix-net with a distributed decryption protocol. We thus provide formal definitions of these components against which the security of our voting scheme will be judged.

VERIFIABLE MIXING. First, we define *completeness*, *soundness* and *simulatability* for a mixing protocol Π_{MIX} executed by a prover Prover, with respect to a generic encryption scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ [ABGS23].

We say that the mixing protocol Π_{MIX} is *complete* if for honest PPT parties Prover and $\mathsf{Verifier}$ that follows the protocol then Prover on input a set of honestly generated ciphertexts will output a new set of ciphertexts together with a proof such that $\mathsf{Verifier}$ accepts the proof and the output ciphertexts decrypt to the same set of messages as the input ciphertexts.

Definition 5.2.5 (Mixing Completeness)**.** *The mixing protocol* Π_{MIX} *is* complete *if for all* $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$, $\{c_i\}_{i\in[\tau]} \leftarrow \mathsf{Enc}(\mathsf{pk}, \{m_i\}_{i\in[\tau]})$, *and for PPT adversaries* $\mathcal{A}$ *such that* $(\{\hat{c}_i\}_{i\in[\tau]}, \pi) \leftarrow \mathsf{Prover}(\mathsf{pp}, \mathsf{pk}, \{c_i\}_{i\in[\tau]})$, *we have*

$$\Pr\left[\begin{matrix}\{m_i\}_{i\in[\tau]} = \mathsf{Dec}(\mathsf{sk}, \{\hat{c}_i\}_{i\in[\tau]} \\ 1 \leftarrow \mathsf{Verifier}(\mathsf{pp}, \mathsf{pk}, \{c_i\}_{i\in[\tau]}, \{\hat{c}_i\}_{i\in[\tau]}, \pi)\end{matrix}\right] \leq 1 - \epsilon(\lambda),$$

where the probability is taken over KeyGen, Enc *and* Prover.

We say that the mixing protocol Π_{MIX} is *sound* if a dishonest PPT adversary $\mathcal{A}$ that can behave arbitrarily on input a set of honestly generated ciphertexts will not be able to output a new set of ciphertexts together with a proof such that an honest $\mathsf{Verifier}$ accepts the proof but the output ciphertexts decrypt to a different set of messages than the input ciphertexts.

Definition 5.2.6 (Mixing Soundness)**.** *The mixing protocol* Π_{MIX} *is sound if for all* $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$*,* $\{c_i\}_{i\in[\tau]} \leftarrow \mathsf{Enc}(\mathsf{pk}, \{m_i\}_{i\in[\tau]})$*, and PPT adversaries* $\mathcal{A}$ *such that* $(\{\hat{c}_i\}_{i\in[\tau]}, \pi) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}, \{c_i\}_{i\in[\tau]})$*, we have*

$$\Pr\begin{bmatrix} \{m_i\}_{i\in[\tau]} \neq \mathsf{Dec}(\mathsf{sk}, \{\hat{c}_i\}_{i\in[\tau]} \\ 1 \leftarrow \mathsf{Verifier}(\mathsf{pp}, \mathsf{pk}, \{c_i\}_{i\in[\tau]}, \{\hat{c}_i\}_{i\in[\tau]}, \pi) \end{bmatrix} \leq \epsilon(\lambda),$$

where the probability is taken over KeyGen*,* Enc *and* $\mathcal{A}$*.*

We say that the mixing protocol Π_{MIX} is *simulatable* if a PPT adversary $\mathcal{A}$ that on input a set of honestly generated ciphertexts can not distinguish between a real execution of the mixing protocol with accepting output and a protocol execution from a PPT simulator $\mathcal{S}$ (given a set honestly mixed output ciphertexts) producing a simulated mixing proof

Definition 5.2.7 (Mixing Simulatability)**.** Π_{MIX} *is simulatable if there exists a PPT simulator* $\mathcal{S}$ *such that for all PPT adversaries* $\mathcal{A}$ *we have the following:*

$$\left|\Pr\left[b = b' \;:\; \begin{array}{c} (\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa); b \leftarrow \{0,1\} \\ \{c_i\}_{i\in[\tau]} \leftarrow \mathsf{Enc}(\mathsf{pk}, \{m_i\}_{i\in[\tau]}) \\ (\{\hat{c}_i\}_{i\in[\tau]}, \pi^{(0)}) \leftarrow \mathsf{Prover}(\mathsf{pp}, \mathsf{pk}, \{c_i\}_{i\in[\tau]}) \\ (\pi^{(1)}) \leftarrow \mathcal{S}(\mathsf{pp}, \mathsf{pk}, \{c_i\}_{i\in[\tau]}, \{\hat{c}_i\}_{i\in[\tau]}) \\ b' \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}, \{c_i\}_{i\in[\tau]}, \{\hat{c}_i\}_{i\in[\tau]}, \pi^{(b)}) \end{array}\right] - \frac{1}{2}\right| \leq \epsilon(\lambda),$$

where the probability is taken over KeyGen*,* Enc*,* Prover*,* $\mathcal{S}$ *and* $\mathcal{A}$*.*

DISTRIBUTED DECRYPTION. Here we define the syntax and security properties for a PKE with distributed decryption [ABGS23].

Definition 5.2.8 (PKE with Distributed Decryption)**.** *A* PKE *scheme with distributed decryption consists of five algorithms* (KeyGen, Enc, Dec, DDec, Comb)*, where*

$\mathsf{KeyGen}(\mathsf{sp}) \to (\mathsf{pk}, \mathsf{sk}, \{\mathsf{dk}_j\}_{j\in[\xi_2]})$: *On input security parameters* sp*, it returns public encryption key* pk*, as secret key* sk*, and a set of* ξ_2 *secret decryption shares* $\{\mathsf{dk}_j\}_{j\in[\xi_2]}$

$\mathsf{Enc}(\mathsf{pk}, \{m_i\}_{i\in[\tau]}) \to \{c_i\}_{i\in[\tau]}$: *On input public key* pk *and a set of messages* $\{m_i\}_{i\in[\tau]}$*, outputs a set of ciphertexts* $\{c_i\}_{i\in[\tau]}$

$\mathsf{Dec}(\mathsf{sk}, \{c_i\}_{i\in[\tau]}) \to \{m_i\}_{i\in[\tau]}$: *On input secret key* sk *and set of ciphertexts* $\{c_i\}_{i\in[\tau]}$*, outputs a set ofmessages* $\{m_i\}_{i\in[\tau]}$

$\mathsf{DDec}(\mathsf{dk}_j, \{c_i\}_{i\in[\tau]}) \to \{\mathsf{ds}_{i,j}\}_{i\in[\tau]}$: *On input decryption key share* dk_j, *set of ciphertexts* $\{c_i\}_{i\in[\tau]}$, *outputs set of decryption shares* $\{\mathsf{ds}_{i,j}\}_{i\in[\tau]}$.

$\mathsf{Comb}(\{c_i\}_{i\in[\tau]}, \{\mathsf{ds}_{i,j}\}_{i\in[\tau]}) \to \{m_i\}_{i\in[\tau]}/\perp$ *On input ciphertexts* $\{c_i\}_{i\in[\tau]}$ *and decryption shares* $\{\mathsf{ds}_{i,j}\}_{i\in[\tau]}$, *outputs either set of messages* $\{m_i\}_{i\in[\tau]}$ *or* $\perp$,

We first define standalone IND-CPA security of a PKE, concerning only the Enc and Dec algorithms.

Definition 5.2.9 (Chosen Plaintext Security)**.** *We say that the public key encryption scheme is secure against* chosen plaintext attacks *if an adversary* $\mathcal{A}$, *after choosing two messages* m_0 *and* m_1 *and receiving an encryption* c *of either* m_0 *or* m_1 *(chosen at random), cannot distinguish which message* c *is an encryption of. Hence, we want that*

$$\left|\Pr\left[b = b' \ : \ \begin{array}{c}(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{sp}) \\ (m_0, m_1, \mathsf{state}) \leftarrow \mathcal{A}(\mathsf{sp}, \mathsf{pk}) \\ b \xleftarrow{\$} \{0,1\}, c \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b) \\ b' \leftarrow \mathcal{A}(c, \mathsf{state})\end{array}\right] - \frac{1}{2}\right| \leq \epsilon(\lambda),$$

where the probability is taken over KeyGen *and* Enc.

Definition 5.2.10 (Threshold Correctness)**.** *We say that the public key distributed encryption scheme is* threshold correct *with respect to* $P_{\mathsf{sk}}(\cdot)$ *if the following probability equals* 1*:*

$$\Pr\left[\begin{array}{c}\mathsf{Comb}(\{c_i\}_{i\in[\tau]}, \{\mathsf{ds}_{i,j}\}_{i\in[\tau]}^{j\in[\xi_2]}) \\ = \\ \mathsf{Dec}(\mathsf{sk}, \{c_i\}_{i\in[\tau]})\end{array} : \begin{array}{c}(\mathsf{pk},\mathsf{sk},\{\mathsf{dk}_j\}_{j\in[\xi_2]}) \leftarrow \mathsf{KeyGen}(\mathsf{sp}) \\ \{c_i\}_{i\in[\tau]} \leftarrow \mathcal{A}(\mathsf{pp},\mathsf{pk}) \\ \forall i \in [\tau] : \ P_{\mathsf{sk}}(c_i) = 1, \forall j \in [\xi_2] : \\ \{\mathsf{ds}_{i,j}\}_{i\in[\tau]} \leftarrow \mathsf{DDec}(\mathsf{sk}_j, \{c_i\}_{i\in[\tau]})\end{array}\right],$$

where the probability is taken over KeyGen *and* DDec.

For the remaining properties we use the notion of a predicate, denoted $P_{\mathsf{sk}}(\cdot)$, defined with relation to secret key sk, which takes a single input and outputs either 1 indicating accept or 0 indicating reject. The reader should note that this is an abstraction of a verification check. In our schemes, this will involve the checking of zero-knowledge proofs attesting to the fact that each party has performed its part honestly.

Definition 5.2.11 (Threshold Verifiability)**.** *A PKE scheme with distributed decryption is* threshold verifiable *with respect to* $P_{\mathsf{sk}}(\cdot)$ *if an adversary* $\mathcal{A}$ *corrupting* $J \subseteq [\xi_2]$ *secret key shares* $\{\mathsf{sk}_j\}_{j\in J}$ *cannot convince* Comb *to accept*

maliciously created decryption shares $\{\mathsf{ds}_{i,j}\}_{i\in[\tau],j\in J}$. *More concretely, the following probability is bounded by a negligible* $\epsilon(\lambda)$*:*

$$\Pr\left[\begin{array}{c}\mathsf{Dec}(\mathsf{sk},\{c_i\}_{i\in[\tau]})\\ \neq \\ \mathsf{Comb}(\{c_i\}_{i\in[\tau]},\{\mathsf{ds}_{i,j}\}_{i\in[\tau]}^{j\in[\xi_2]}):\\ \neq \\ \perp\end{array}\begin{array}{c}(\mathsf{pp},\mathsf{pk},\mathsf{sk},\{\mathsf{sk}_j\}_{j\in[\xi_2]})\leftarrow\mathsf{KeyGen}(1^\lambda,\xi_2)\\ (\{c_1,\ldots,c_\tau\})\leftarrow\mathcal{A}(\mathsf{pp},\mathsf{pk},\{\mathsf{sk}_j\}_{j\in J})\\ \forall i\in[\tau]:\ P_{\mathsf{sk}}(c_i)=1,\forall j\notin J:\\ \{\mathsf{ds}_{i,j}\}_{i\in[\tau]}\leftarrow\mathsf{DDec}(\mathsf{sk}_j,\{c_i\}_{i\in[\tau]})\\ \{\mathsf{ds}_{i,j}\}_{i\in[\tau],j\in J}\leftarrow\mathcal{A}(\{\mathsf{ds}_{i,j}\}_{i\in[\tau],j\notin J})\end{array}\right],$$

where the probability is taken over KeyGen *and* DDec*.*

Definition 5.2.12 (Distributed Decryption Simulatability)**.** *A* PKE *scheme with distributed decryption is* simulatable *with respect to* $P_{\mathsf{sk}}(\cdot)$ *if an adversary* $\mathcal{A}$ *corrupting* $J\subsetneq[\xi_2]$ *secret decryptionkey shares* $\{\mathsf{dk}_j\}_{j\in J}$ *cannot distinguish the transcript of the decryption protocol from a simulation by a simulator* Sim *which only gets* $\{\mathsf{dk}_j\}_{j\in J}$ *as well as correct decryptions as input. More concretely, the following probability is bounded by a negligible* $\epsilon(\lambda)$*:*

$$\left|\Pr\left[b=b'\ :\ \begin{array}{c}(\mathsf{pk},\mathsf{sk},\{\mathsf{dk}_j\}_{j\in[\xi_2]})\leftarrow\mathsf{KeyGen}(\mathsf{sp})\\ \{c_i\}_{i\in[\tau]}\leftarrow\mathcal{A}(\mathsf{pp},\mathsf{pk},\{\mathsf{dk}_j\}_{j\in J})\\ \forall i\in[\tau]:\ P_{\mathsf{sk}}(c_i)=1\\ \{\mathsf{ds}_{i,j}^{(0)}\}\leftarrow\mathsf{DDec}(\{\mathsf{dk}_j\}_{j\in[\xi_2]},\{c_i\}_{i\in[\tau]})\\ \{\mathsf{ds}_{i,j}^{(1)}\}\leftarrow\mathsf{Sim}(\mathsf{pp},\{\mathsf{dk}_j\}_{j\in J},\{c_i,\mathsf{Dec}(\mathsf{dk}_j,c_i)\}_{i\in[\tau]})\\ b\leftarrow\{0,1\},b'\leftarrow\mathcal{A}(\{\mathsf{ds}_{i,j}^{b}\}_{i\in[\tau],j\in[\xi_2]})\end{array}\right]-\frac{1}{2}\right|,$$

where the probability is taken over KeyGen, DDec, Sim*.*

5.3 NTRU HARDNESS

Research on the security of the NTRU problem has revealed a significant improvement in the performance of lattice reduction algorithms when applied to NTRU lattices for so-called *overstretched* parameters. More precisely, analysis carried out over a series of works [ABD16, KF17, LW20] has shown this weakening of the NTRU problem to occur when the modulus q is very large compared to the ring dimension d and when secrets are small. Naturally, these works seek to determine the turning point at which q becomes large enough for such attacks to apply, so security concerns arise. We refer to this as the *fatigue point*.

5.3.1 *Extending NTRU Cryptanalysis*

Until recently, only an asymptotic result was known about the position of the fatigue point, determined by Kirchner and Fouque as $q = d^{2.783+o(1)}$ [KF17].

DUCAS-VAN WOERDEN ANALYSIS. In their recent paper [Dv21], Ducas and van-Woerden improve on the asymptotic result of Kirchner and Fouque, narrowing down the fatigue point for ternary secrets to $q = d^{2.484+o(1)}$. Building on this result, the authors perform an average-case analysis (rather than a worst-case bound) based on the volume of the relevant lattices and sublattices to arrive at a concrete prediction of the fatigue point. They identify two lattice reduction events that distinguish standard regimes from their overstretched counterparts to facilitate their analysis.

- Secret Key Recovery (SKR): The event in which a vector as short as the secret key is inserted into the lattice basis.
- Dense Sublattice Discover (DSD): The event in which a vector of the dense sublattice is inserted into the lattice basis.

A DSD event has been shown to shortly precede an SKR event by a cascading of further DSD events or to enable the decryption of fresh ciphertexts.

Through careful observation of the occurrence of these events, Ducas and van Woerden use their predictive model to determine the concrete fatigue point of NTRU with ternary secrets to be $q = 0.004 \cdot d^{2.484}$ for $d > 100$. One can use the scripts provided[1] in their work to estimate the concrete hardness of NTRU. We also affirm their predictive model by running real experiments on low-dimensional instances to confirm this relation.

BEYOND TERNARY SECRETS. The reader may have noticed that the discussion of the fatigue point, thus far, only focuses on the modulus and dimension of the ring. Recalling Definition 5.2.1 reminds us that f and g need not always be ternary. Indeed, many NTRU-based constructions use secrets with non-ternary coefficients. Let us consider f and g generated according to a Gaussian distribution D_σ of standard deviation σ. For convenience, the analysis of [Dv21] models the ternary secret case by sampling $f, g \leftarrow D_\sigma$ with $\sigma^2 = 2/3$. Varying σ can model any secret key size, and thus we will herein consider that f and g are always sampled according to some Gaussian D_σ. The natural question arises:

1 See github.com/WvanWoerden/NTRUFatigue for their code.

Does the choice of (secret size) σ influence the fatigue point, and if so, what is its impact?

To get some intuition on this, we recall the work of Steinfeld and Stehlé [SS13] in which the authors show how selecting σ sufficiently large gives rise to a public key $h = g/f$ that is statistically indistinguishable from uniform when f and g are sampled from D_σ. Moreover, they show that using such parameters allows one to remove the NTRU assumption from a proof of the NTRU cryptosystem. The σ needed for statistical security depends on the size of q and d. This suggests that fixing q and d and increasing σ makes the NTRU problem harder.

This observation goes some way to answering the first part of our question since it is clear that, for a sufficiently large σ, both SKR and DSD become ineffective.

Whilst using statistically uniform public keys provides peace of mind, this practice comes with significant efficiency losses. In addition to much larger key sizes, conditions for a cryptosystem's correctness can become much more constraining. Note that for correct decryption of the $\texttt{NTRU} - \texttt{Encrypt}$ cryptosystem defined in Figure 5.1, one needs the relation $\|p(gs + fe) + fm\|_\infty < q/2$ to hold. Clearly, using larger secrets f and g thus leads to less favorable parameters by pushing up the modulus q.

We, therefore, have a balancing act that needs to be performed when setting NTRU parameters; to keep parameters small whilst avoiding the attacks affecting overstretched regimes. Fortunately, the script provided in [**?**] also allows for NTRU hardness estimations using any choice of σ though no analysis is performed in their work outside the ternary case. Nevertheless, their estimator provides a tool, much like the LWE estimator of Albrecht et [APS15], to analyze the concrete hardness of any given NTRU parameter set.

A MORE GENERAL FATIGUE RELATION. In our analysis, we are interested in answering the second part of the above question. In particular, we would like to know by *how much* an increase in NTRU secret size affects the position of the fatigue point for a given ring dimension.

A simple calculation, following the analysis of [Dv21], Section 3.2, confirms that the asymptotic relation $q = d^{2.484+o(1)}$ holds regardless of the value of σ. This suggests that if the value of σ plays a role in the concrete, average case relation, it manifests in the leading constant. We can thus infer that, for some function ψ and constant c, the fatigue point is given by

$$q = c \cdot \psi(\sigma) \cdot d^{2.484}.$$

In order to determine the nature of ψ, we consider a range of $\sigma \in [2, 2^2, \ldots, 2^{20}]$. For each σ, we perform a loglog-linear regression on the estimated fatigue points overall prime ring dimensions $199, \ldots, 499$. This mimics the calculations of [Dv21] used in the ternary case. For a full explanation of why this is a sensible range to examine, we refer the reader to Section 5.5 of that work.

Next, we consider the predicted fatigue points as a geometric series. This reveals the predicted average-case fatigue point to be

$$q = 0.0058 \cdot \sigma^2 \cdot d^{\,2.484}. \tag{5.1}$$

The precision of this relation across all σ considered is very high. Whilst we could extend this part of our analysis to larger σ, it is highly unlikely that, for cryptographic applications, one would need to take σ higher than 2^{20}. We note also that, setting $\sigma^2 = 2/3$, we recover the fatigue point determined for the ternary case [Dv21].

This gives a definitive answer to our question about the impact of σ on the fatigue point. To give more gravity to this prediction, we also run a series of experiments for computable ring dimensions to validate this estimate. The results are displayed in Figure 5.2.

We also give a second figure (Figure 5.3) in which we plot q/σ^2 along the vertical axis. This reveals the accuracy of the preceding constant by revealing how closely bunched the estimations and experiments are when normalised across varying σ. As was observed by Ducas and Woerden in the ternary case, the estimator is slightly pessimistic, predicting a fatigue point roughly 15% lower than the one suggested by real experiments. They give potential explanations for this discrepancy, pointing to the slope parameter used in the estimator, which is not well calibrated for such small block sizes. In practice, though, this small error only translates to a difference of 2-3 in the block size needed to run BKZ and thus hardly affects the predicted security. Importantly, our experiments show that this error remains constant even at larger moduli.

THE SIGNIFICANCE OF σ^2. Having determined the impact of σ on the concrete fatigue point for NTRU, we reflect on the structure of Equation (5.1). As an illustrating example, let us return to the decryption correctness constraint for the NTRU cryptosystem. This can be written as $\sigma \cdot \mathcal{F}(p, d, \nu) < q$ for some function $\mathcal{F}$. Suppose for a given parameter set (d, q, σ), this constraint is satisfied, but the corresponding NTRU instance does not provide adequate security. Let us then increase q by a constant factor δ, say. According to the constraint, this gives room for an increase in

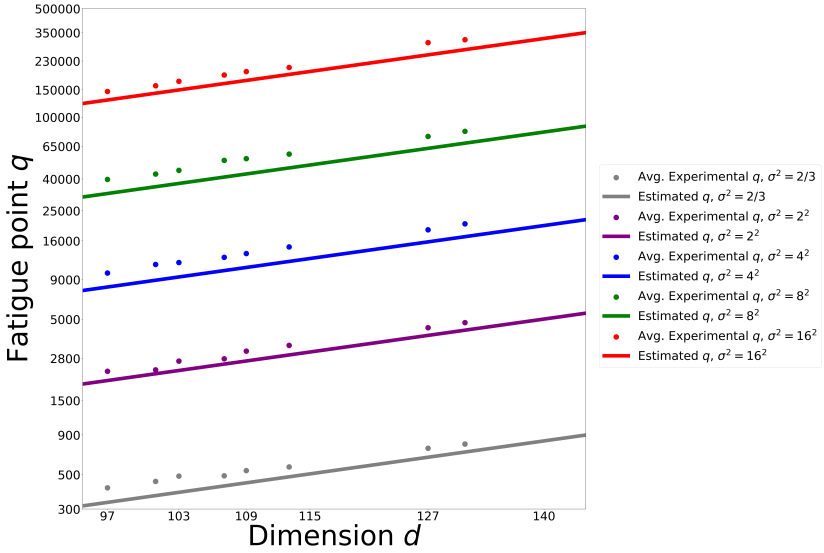

FIGURE 5.2: Experimental fatigue point values for a range of σ, calculated using BKZ with 8 tours on matrix NTRU instances. The straight-colored lines show the estimated values using the (modified) estimator from [Dv21].

σ to $\delta \cdot \sigma$. Then Equation (5.1) tells us that the new fatigue point for the set $(d, \delta \cdot q, \delta \cdot \sigma)$ increases by a factor of δ^2. The important observation here is that, while increasing q in the first move might weaken the NTRU instance, the same increase permitted for σ actually gives rise to a *net increase in the hardness of the instance*. In summary, Equation (5.1) tells us that it is possible to 'win' this cat-and-mouse game for NTRU that so often arises when setting lattice parameters.

We now consider how our analysis might be applied to existing works to refine parameter choices.

5.3.2 *Revisiting Existing NTRU-Based Cryptosystems*

While the authors of [Dv21] note that parameters used in the NTRU-based NIST finalists are still secure to the degrees claimed, many works in the literature use different sets, some of which may fall foul of the dense sublattice

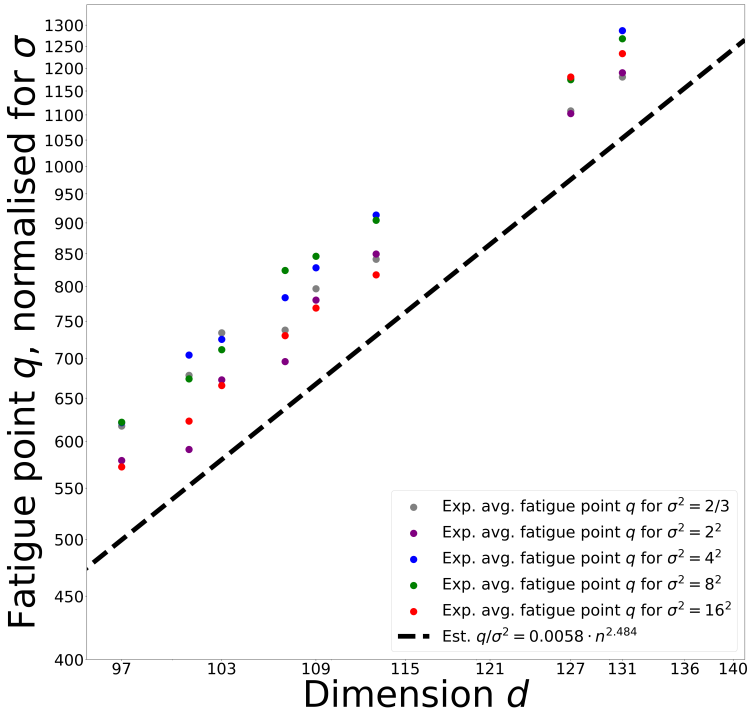

FIGURE 5.3: Experimental values for q/σ^2 illustrate that the fatigue point, when adjusted for σ, is modeled by $q/\sigma^2 = 0.0058 \cdot d^{2.484}$.

attack, and thus, one needs to use the techniques described in the previous section to set parameters.

We examine an existing primitive in which the parameters chosen fall short of providing the security levels claimed. However, as suggested by Equation (5.1), we can carefully re-select parameters so that a small increase in the size of secrets yields the security bump-up needed.

BLIND SIGNATURES [DK22]. del Pino and Katsumata present a lattice-based (partially) blind signature using trapdoor sampling. In the (round optimal) construction given, a user passes the message to be signed in a *blind* way so that the signer does not learn the message they sign. This is done by committing to the message and then proving the well-formedness of this commitment. We will call this the *first flow*. The signer then creates

an output passed back to the user (*second flow*). Finally, the user computes a signature for its original message using this response message from the signer.

In the first flow, Pino and Katsumata employ the NTRU-based linear homomorphic commitment scheme (LinHC) of [Kat21] to ensure the soundness and overall QROM security of the well-formedness proof. One must, therefore, choose parameters so that the relevant NTRU instance is hard. The choice of $d = 2048$, $q = 2^{66}$, and ternary NTRU secrets is informed by the constraint requiring straight-line extractability of the proof system. However, as we have observed, such large moduli run the risk of taking a parameter set into overstretched territory. Moreover, these values give rise to only 63 bits of security when run through the estimator of [Dv21] rather than the claimed 128.

To rectify this situation, there are two common strategies; either one can increase the ring dimension used throughout the scheme or use sufficiently large NTRU secrets that the corresponding public key is *statistically* indistinguishable from uniform.

Doubling the ring dimension in [dK22] from 2048 to 4096 (to retain the implementation benefits of a power-of-two dimension) and computing the other parameters accordingly, 128 bits of security is reached at the cost of doubling the sign-request flow (69.2MB), doubling the returned 'pre-signature', and doubling the user's final signature size to 200 KB.

The alternative method, using a statistically uniform public commitment key turns out to be impossible whilst satisfying all parameter constraints simultaneously.

We now exhibit the benefits of the relation Equation (5.1), as revealed by our analysis, when applied to the problem of setting NTRU parameters. with the same ring dimension $d = 2048$, we increase σ_{NTRU} (secret size). This has the effect of pushing up the modulus needed to facilitate the straight-line extraction condition. The reader might observe that increasing q reduces the hardness of the problem again. However, Equation (5.1) reveals that it is possible to 'win' this cat-and-mouse game since the fatigue point increases *quadratically* with the size of the secrets. We thus propose the following parameters to ensure the 128 bit security threshold is reached:

$$q \approx 2^{74},\ p \approx 2^{41},\ \sigma_{\mathsf{NTRU}} = 13,$$

where p is the prime used to commit to the witness in the LinHC protocol. Fortunately, this change only has a small effect on the total communication cost. In the first flow, the user signing query increases from 34 MB to 35.4

MB, and the sizes of the user's pre-signature and final signature output are unchanged. This significantly improves the sizes that arise from changing the ring dimension and avoids doubling the final signature.

SUMMARY. Simply increasing the size of the NTRU secrets may be all that is needed to ensure the correct security threshold is reached. In other settings, this also pushes up the modulus over which a scheme is defined, as in the examples above. However, the scheme may also rely on other hardness assumptions, such as RLWE, as in our voting scheme, defined over the same ring. Now, the RLWE problem may no longer be hard for the adjusted parameters and one may need to increase the ring *dimension* to find parameters for which both problems are hard. This can make what was an efficient scheme into one that cannot be deployed in practice.

Clearly, such balancing acts must be approached with a good understanding of the hardness of NTRU instances. We aim to further demonstrate the advantages of this approach when presenting our voting scheme in Section 5.4 where our fine-grained analysis allows us to bring down the overall communication cost dramatically.

5.4 LATTICE-BASED VOTING SCHEME

A *cryptographic voting scheme* is usually defined in terms of the algorithms for the tasks of election setup, casting ballots, and counting cast ballots. To accurately model the counting process, we need algorithms for shuffling and distributed decryption. To make such a scheme verifiable (actively secure), we additionally need a mechanism by which to verify that the encryption, shuffling, and decryption algorithms are computed honestly to ensure that the election output is correct. In this section, we follow the framework of Aranha et al. [ABGS23], replacing BGV encryption with one from NTRU, modifying decryption verification, and setting parameters based on our prior NTRU hardness analysis to achieve the efficiency gains claimed in **??**.

5.4.1 *Voting Overview*

SETUP PHASE. A trusted party runs the key generation algorithm for the PKE scheme with distributed decryption. In this work, we will assume a trusted key generation and leave the design of a distributed key generation algorithm for NTRU to future work, as this is common for voting schemes.

The generated public parameters sp are given to every participant, while the decryption key shares dk_j are distributed amongst the decryption servers.

CASTING PHASE. Each voter instructs their voting device to cast their chosen ballot. The device encrypts the ballot under the public key pk to create a ciphertext c, and it computes a *ballot proof*. The standard way to do this is to use a verifiable encryption scheme such as the one presented in [LNP22], proving that the submitted ciphertext contains a genuine ballot in zero-knowledge.

COUNTING PHASE. This is divided into three sequential processes. First, encrypted ballots are passed through a series of shuffle servers.

The ξ_1 shuffle servers $\mathcal{S}_1, ..., \mathcal{S}_{\xi_1}$ consecutively run the shuffle algorithm of the set of encrypted ballots $\{c_i^{(k-1)}\}$, passing the shuffled and re-encrypted ballots $\{c_i^{(k)}\}$ to the next shuffle server. They also generate a shuffle proof which anyone can verify. We may refer to this whole shuffle process as the *mix-net*.

Each of the ξ_2 decryption servers $\mathcal{D}_j$ receives the output of each shuffle server and verifies the corresponding shuffle proofs. Only after verifying each proof does a decryption server begin decryption. $\mathcal{D}_j$ then computes a set of partial decryption shares $\{\mathsf{ds}_{ij}\}$, one for each of the ciphertexts. Finally, it creates a proof of decryption to guarantee that it computed its shares correctly. Each decryption server passes its shares to the combining algorithm Comb.

The Comb algorithm performs the task of recovering the ballots. Having received all decryption shares from decryption servers, the Comb algorithm verifies the decryption proofs. If all decryption proofs are verified, it recovers the ballots by combining the decryption shares.

A schematic of these processes, in the malicious setting, is shown in Figure 5.4. This figure is adapted from [ABGS23] and shows the voting protocol beginning with input of a set of encrypted ballots and finishing with a set of ballots in plaintext. We note that some works consider an auditor whose role is to verify the processes at each step by checking the proofs provided. This is somewhat of a stylistic design choice. For the purposes of this chapter, it is useful to think of the proofs as providing verifiability of each phase by any third party and by the component servers before carrying out their roles.

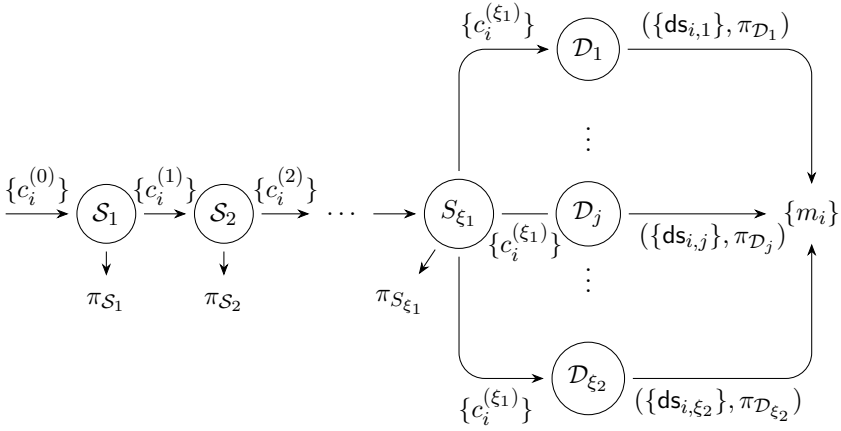

FIGURE 5.4: The voting protocol with verifiable mix-net and distributed decryption, adapted from [ABGS23, Figure 1].

5.4.2 *Passively-Secure Construction*

Here we present our passively secure voting scheme. Whilst our ultimate goal is to give a verifiable (actively secure) voting scheme, we first isolate the core, passively secure skeleton for clarity of presentation. We begin by defining the algorithms and syntax of this construction.

Definition 5.4.1 (Passively Secure Voting Scheme)**.** *Let τ be the number of voters, ξ_1 the number of shuffle servers, and ξ_2 the number of decryption servers. A passively secure cryptographic voting scheme Π_{PVote} consists of five algorithms* $(\mathsf{KeyGen}, \mathsf{Cast}, \mathsf{Shuffle}, \mathsf{DDec}, \mathsf{Comb})$.

$\mathsf{KeyGen}(\mathsf{sp}) \to (\mathsf{pk}, \mathsf{sk}, \{\mathsf{dk}_j\}_{j\in[\xi_2]})$: *On input setup parameters* sp*, it returns a public encryption key* pk*, a secret key* sk *and a set of ξ_2 secret decryption key shares $\{\mathsf{dk}_j\}_{j\in[\xi_2]}$.*

$\mathsf{Cast}(\mathsf{pk}, v) \to c$: *On input a public key* pk *and vote v it returns an encrypted ballot c.*

$\mathsf{Shuffle}(\{c_i\}_{i\in[\tau]}) \to \{\hat{c}_i\}_{i\in[\tau]}$: *On input a set of encrypted ballots $\{c_i\}_{i\in[\tau]}$ it returns a set of encrypted ballots $\{\hat{c}_i\}_{i\in[\tau]}$.*

$\mathsf{DDec}(\{c_i\}_{i\in\tau}, \mathsf{dk}_j) \to \{\mathsf{ds}_{i,j}\}$: *On input a set of encrypted ballots $\{c_i\}_{i\in[\tau]}$ and a decryption key dk_j, it returns a set of decryption shares $\mathsf{ds}_j = \{\mathsf{ds}_{i,j}\}_{i\in[\tau]}$.*

$\mathsf{Comb}(\{c_i\}_{i\in[\tau]}, \{\mathsf{ds}_j\}_{j\in[\xi_2]}) \to \{v\}_{i\in[\tau]}$: *On input a set of encrypted ballots* $\{c_i\}_{i\in[\tau]}$ *and a set of decryption shares* $\{\mathsf{ds}_j\}_{j\in[\xi_2]}$*, it outputs a set of votes* $\{v\}_{i\in[\tau]}$.

We instantiate the algorithms, present our passively secure voting scheme and give an overview in Figure 5.5.

Setup. Let $p \ll q$ be primes and d a power of two which define the rings R_p and R_q. Votes lie in R_p. Let $\sigma_{\mathsf{NTRU}}, B_{\mathsf{Dec}}, B_{\mathsf{Drown}} \in \mathbb{R}^+$, $t \in (1, 2]$, and $\nu, \tau, \xi_1, \xi_2 \in \mathbb{N}$. Let $\mathsf{sp} = (d, p, q, \sigma_{\mathsf{NTRU}}, t, \nu, \tau, \xi_1, \xi_2)$.

5.4.3 *Actively-Secure Construction*

We present our actively secure (verifiable) voting scheme. This can be seen as a natural extension of the passive protocol Π_{PVote} by adding verifiability to the shuffle and distributed decryption processes. This is done by applying the zero-knowledge proofs of Section 5.2.4 so that the outputs of $\mathsf{Shuffle}_{\mathsf{A}}$ and $\mathsf{DDec}_{\mathsf{A},}$ now include a proof of shuffle $\pi_{\mathcal{S}}$ and a proof of decryption $\pi_{\mathcal{D}}$ respectively.

Now, any third party can verify that the processes of shuffling and distributed decryption were carried out correctly without compromising the privacy or integrity of the voting system. As usual, we assume a trusted dealer for key generation and leave the construction of an NTRU-based distributed key generation for other applications to future work.

This construction implicitly defines a verifiable mix-net and PKE with distributed decryption from NTRU. We consider these of independent interest and give an overview of their workings as stand-alone protocols.

VERIFIABLE SHUFFLE. Our aim here is, given a set of input ciphertexts, to generate a new set of ciphertexts that decrypts to the same set of plaintexts. Crucially, input-output ciphertext correspondence must be obscured. Additionally, we would like any third party to verify that this process has been performed correctly without compromising the privacy of the mix.

To add this verifiability, we apply the scheme of [ABG+21], which allows one to prove a shuffle of openings of the lattice commitments in Section 5.2.3. We denote this proof system Π_{SHUF}. Since NTRU ciphertexts only contain a single element, we can import their scheme without modification where the committed messages are ciphertexts. We also employ the Π_{SMALL} proof systems described in Section 5.2.4 to prove that the new ciphertext noise is

$\mathsf{KeyGen}(\mathsf{sp})$. On input system parameters sp:

1. $(\mathsf{sk} = f, \mathsf{pk} = h) \leftarrow \mathsf{KeyGen}_{\mathsf{NTRU}}(d, p, q, \sigma_{\mathsf{NTRU}}, t)$.
2. For $j \in [\xi_2 - 1]$, $\mathsf{dk}_j \leftarrow \mathcal{U}(R_q)$ and set $\mathsf{dk}_{\xi_2} = \mathsf{sk} - \sum_{j=1}^{\xi_2 - 1} \mathsf{dk}_j \mod q$.
3. Return $(\mathsf{pk}, \mathsf{sk})$ and key shares $\{\mathsf{dk}_j\}_{j \in [\xi_2]}$.

$\mathsf{Cast}(\mathsf{pk}, v)$. On input the public key pk and a vote $v \in R_p$:

1. Compute $c = \mathsf{Enc}_{\mathsf{NTRU}}(\mathsf{pk}, v)$.
2. Return encrypted ballot c.

$\mathsf{Shuffle}(\{c_i\}_{i \in [\tau]})$. On input encrypted ballots $\{c_i\}_{i \in [\tau]}$:

1. For each $i \in [\tau]$, compute $c'_i = \mathsf{Enc}_{\mathsf{NTRU}}(\mathsf{pk}, 0)$.
2. For each $i \in [\tau]$, compute $\hat{c}_i = c_i + c'_i \mod q$.
3. Sample a random permutation $\pi \leftarrow \mathsf{Perm}[\tau]$.
4. Return re-encrypted ballots $\{\hat{c}_{\pi(i)}\}_{i \in [\tau]}$.

$\mathsf{DDec}(\{c_i\}_{i \in \tau}, \mathsf{dk}_j)$. On input a set of encrypted ballots $\{c_i\}_{i \in \tau}$ and a decryption key share dk_j:

1. For each $i \in [\tau]$, sample $E_{ij} \leftarrow S_{B_{\mathsf{Drown}}}$ and compute share $\mathsf{ds}_{ij} = \mathsf{dk}_j \cdot c_i + p \cdot E_{ij} \mod q$.
2. Return the set of decryption shares $\mathsf{ds}_j = \{\mathsf{ds}_{ij}\}_{i \in [\tau]}$.

$\mathsf{Comb}(\{c_i\}_{i \in [\tau]}, \{\mathsf{ds}_j\}_{j \in [\xi_2]})$. On input encrypted ballots $\{c_i\}_{i \in [\tau]}$ and decryption shares $\{\mathsf{ds}_j = \{\mathsf{ds}_{ij}\}_{i \in [\tau]}\}_{j \in [\xi_2]}$:

1. For each $i \in [\tau]$, $v_i = \left(\sum_{j \in [\xi_2]} \mathsf{ds}_{ij} \mod q\right) \mod p$.
2. Return the set of votes $\{v_i\}_{i \in [\tau]}$.

FIGURE 5.5: The passively-secure voting scheme Π_{PVote}.

sufficiently bounded. At a high level, our verifiable shuffle of NTRU ciphertexts $c_1, ..., c_\tau$ follows the framework of [ABGS23]:

1. The shuffle server creates encryptions $c'_1, ..., c'_\tau$ of 0 and commits to these as $[\![c'_i]\!]$ for each $i \in [\tau]$. Run the Π_{SMALL} protocol to prove that each committed ciphertext is honestly computed.

2. Adding the original ciphertexts c_i to these commitments homomorphically yields commitments $[\![\hat{c}_i]\!]$ to ciphertexts with the same plaintext as in $c_1, ..., c_\tau$, now with fresh randomness.

3. The server now reveals the openings $\hat{c}_i$ in a randomly permuted order and runs the Π_{SHUF} protocol to prove that these are indeed a permutation of the correct openings of the commitments.

We note that verification of the shuffle proof should be done before any ballot decryption begins. This can be seen as a first step of the DDec algorithm or as part of a global verification process carried out by an auditor. For simplicity of presentation and since this is covered in [ABGS23], we omit this from the full protocol.

VERIFIABLE DISTRIBUTED DECRYPTION. Our aim here is, given a set of input ciphertexts, to generate a set of decryption shares so that we can extract the encrypted plaintexts when all the shares are combined. Furthermore, each decryptor must prove that they decrypted their decryption share correctly using their secret key share. Therefore, in the active setting, the public key additionally contains a commitment $[\![\mathsf{dk}_j]\!]$ to each secret key share dk_j, and each decryptor holds an opening to exactly one of the commitments. The verifiable distributed decryption protocol works as follows:

1. For each $i \in [\tau]$, the decryptor samples a noise value $E_{ij} \leftarrow S_{B_{\mathsf{Drown}}}$, computes a decryption share $\mathsf{ds}_{ij} = \mathsf{dk}_j \cdot c_i + p \cdot E_{ij}$ and commits to the noise as $[\![E_{ij}]\!]$.

2. For each $i \in [\tau]$, it uses the Π_{Lin} protocol to prove that the linear decryption equation above is computed honestly with respect to $[\![\mathsf{dk}_j]\!]$ and $[\![E_{ij}]\!]$.

3. For each $i \in [\tau]$, it uses the Π_{Bnd} protocol to prove that $[\![E_{ij}]\!]$ is an honestly created commitment and the committed value is bounded by B_{Drown}.

We note that this framework closely follows the decryption protocol in [ABGS23]. Still, for the decryption protocol, we use a proof system that yields an exact upper bound on the noise values instead of a relaxed bound to keep the system parameters smaller when ensuring correct decryption. We now detail the complete verifiable voting scheme in Figures 5.6 and 5.7.

Setup. Let $p \ll q$ be primes and d a power of two which define the rings R_p and R_q. Votes lie in R_p. Let $\sigma_{\mathsf{NTRU}}, B_{\mathsf{Dec}}, B_{\mathsf{Drown}}, B_{\mathsf{Com}}$, $B_{\mathsf{Small}} \in \mathbb{R}^+$, $t \in (1, 2]$, and $\nu, \tau, l, \xi_1, \xi_2 \in \mathbb{N}$. Let $\mathsf{sp} = (d, p, q, \sigma_{\mathsf{NTRU}}, t, \nu, \tau, l, \xi_1, \xi_2, B_{\mathsf{Dec}}, B_{\mathsf{Drown}}, B_{\mathsf{Com}})$.

5.4.4 *ZKPs for Active Security*

In order to bootstrap the passively-secure scheme Π_{PVote} to one with active security we employ a number of zero-knowledge proof systems. Parties must present these proofs in order to prove that they have conducted their tasks correctly. Whilst presented in detail in Section 5.2.4, we briefly explain how each proof type is used in Π_{AVote}.

We wish to highlight, in particular, the way in which we adapt the amortized proof of boundedness Π_{BND} as compared to previous works [ABGS23]. The crucial observation here is that whilst we get slightly larger proofs there, the *exact* guarantees provided by the proof allow for better parameters to be chosen for the overall voting scheme. This leads to a net reduction in communication cost.

CHALLENGE SET. Let κ be an integer such that $\binom{d}{\kappa} \cdot 2^\kappa > 2^\lambda$ and define the two sets $\mathcal{C}_\kappa = \{c \in R_q \mid \|c\|_\infty = 1 \wedge \|c\|_1 = \kappa\}$ and $\bar{\mathcal{C}}_\kappa = \{c - c' \mid c, c' \in \mathcal{C}_\kappa \wedge c \neq c'\}$.

REJECTION SAMPLING. To ensure that the proofs do not leak any information about the secret values, we use rejection sampling to force the output to be independent of the secrets [Lyu09, Lyu12]. For an overview of rejection sampling techniques, see Section 2.3.3.

PROOF OF LINEARITY. In Step 2 of DDec, we prove well-formedness of the linear decryption share. The protocol Π_{LIN} produces a proof that a committed value v is a multiple of another committed value u with respect to a public scalar g. In our setting, we will prove $[\![E_{ij} - p^{-1}\mathsf{ds}_{ij}]\!] = -p^{-1}\mathsf{ctx}_i[\![\mathsf{dk}_j]\!]$. For the formal relation and a description of the proof procedure, see **??**.

$\mathsf{KeyGen}_{\mathsf{A}}(\mathsf{sp})$. On input system parameters sp:

1. Run $\Big((\mathsf{pk},\mathsf{sk}),\{\mathsf{dk}_j\}_{j\in[\xi_2]}\Big) \leftarrow \mathsf{KeyGen}(\mathsf{sp})$.
2. For $j \in [\xi_2]$, compute the commitments and openings $(\llbracket \mathsf{dk}_j \rrbracket, \vec{r}_{\mathsf{dk}_j}) \leftarrow \mathsf{Com}(\mathsf{dk}_j)$.
3. Return $\mathsf{pk}_{\mathsf{A}} = \big(\mathsf{pk}, \llbracket \mathsf{dk}_1 \rrbracket, ..., \llbracket \mathsf{dk}_{\xi_2} \rrbracket\big)$, $\mathsf{sk}_{\mathsf{A}} = \mathsf{sk}$, and key shares $\{\mathsf{dk}_{A,j} = \Big(\mathsf{dk}_j, \vec{r}_{\mathsf{dk}_j}\Big)\}_{j\in[\xi_2]}$.

$\mathsf{Cast}_{\mathsf{A}}(\mathsf{pk}_{\mathsf{A}}, v)$. On input a public key pk_A and a vote v in R_p, retrieving pk from pk_A:

1. Return $c \leftarrow \mathsf{Cast}(\mathsf{pk}, v)$.

$\mathsf{Shuffle}_{\mathsf{A}}(\{c_i\}_{i\in[\tau]})$. On input a set of encrypted ballots $\{c_i\}_{i\in[\tau]}$:

1. For $i \in [\tau]$, compute $c'_i \leftarrow \mathsf{Enc}_{\mathsf{NTRU}}(\mathsf{pk}, 0)$ using encryption randomness (s'_i, e'_i).
2. For $i \in [\tau]$, commit to c'_i as $\mathsf{com}'_i := \llbracket c'_i \rrbracket \leftarrow \mathsf{Com}(\mathsf{pk}_{\mathsf{C}}, c'_i)$ where $\vec{r}_{c'_i}$ is the commitment randomness used. Then denoting
$$\mathbf{A}_{\mathsf{M}} = \begin{bmatrix} 1 & a_{1,1} & a_{1,2} & 0 & 0 \\ 0 & 1 & a_{2,2} & ph & p \end{bmatrix},$$
and $\mathbf{s}_{c'_i} = [\vec{r}_{c'_i}, s'_i, e'_i]^T$ compute $\pi_{\mathsf{Small,i}} \leftarrow \Pi_{\mathrm{SMALL}}$, for matrix $\mathbf{A}_{\mathsf{M}}$, input vector $\mathbf{s}_{c'_i}$, targets com'_i, and bound B_{Small}. Set $\pi_{\mathsf{Small},} := \{\pi_{\mathsf{Small,i}}\}_{i\in[\tau]}$.
3. For $i \in [\tau]$, compute $\hat{c}_i = c_i + c'_i$. Sample $\pi \leftarrow \mathsf{Perm}([\tau])$, and compute $\pi_{\mathsf{Shuf}} \leftarrow \Pi_{\mathrm{SHUF}}$ with input commitments $\{\llbracket \hat{c}_i \rrbracket\}_{i\in[\tau]}$, randomness $\{\vec{r}_{c'_i}\}_{i\in[\tau]}$, ciphertexts $\{\hat{c}_i\}_{i\in\tau]}$, and permuted ciphertexts $\{\hat{c}_{\pi(i)}\}_{i\in[\tau]}$.
4. Return $\Big(\{\hat{c}_{\pi(i)}\}_{i\in[\tau]}, \pi_{\mathcal{S}}\Big)$, where $\pi_{\mathcal{S}} = \Big(\{\mathsf{com}'_i\}_{i\in[\tau]}, \pi_{\mathsf{Small},}, \pi_{\mathsf{Shuf}}\Big)$.

FIGURE 5.6: Π_{AVote} key generation, casting, and shuffle.

$\mathsf{DDec}_A(\{c_i\}_{i\in[\tau]}, \mathsf{dk}_{A,j})$. On input a set of ciphertexts $\{c_i\}_{i\in[\tau]}$ and decryption key share $\mathsf{dk}_{A,j} = \left(\mathsf{dk}_j, \vec{r}_{\mathsf{dk}_j}\right)$:

1. For $i \in [\tau]$, sample $E_{ij} \leftarrow S_{B_{\mathsf{Drown}}}$, and compute $\mathsf{ds}_{ij} = \mathsf{dk}_j \cdot c_i + p \cdot E_{ij}$.
2. For $i \in [\tau]$, compute $\left(\llbracket E_{ij} \rrbracket, \vec{r}_{E_{ij}}\right) \leftarrow \mathsf{Com}(E_{ij}, \mathsf{pk}_{\mathsf{C}})$ and use the Π_{LIN} protocol to compute a proof $\pi_{\mathsf{Lin}_{\mathsf{ij}}}$ for the linear relation $\mathsf{ds}_{ij} = \mathsf{dk}_j \cdot c_i + p \cdot E_{ij}$.
3. Apply the amortized proof Π_{BND} from Section 4.2.4, to create a proof π_{Bnd} that, for all $i \in [\tau]$, $\|E_{ij}\|_\infty \leq B_{\mathsf{Drown}}$ and $\|\vec{r}_{E_{ij}}\|_\infty \leq B_{\mathsf{Com}}$.
4. Return $\mathsf{ds}_j := \left(\{\mathsf{ds}_{ij}\}_{i\in[\tau]}, \pi_{\mathcal{D}}\right)$, where $\pi_{\mathcal{D}} = \left(\{\llbracket E_{ij} \rrbracket\}_{i\in[\tau]}, \{\pi_{\mathsf{Lin}_{\mathsf{ij}}}\}_{i\in[\tau]}, \pi_{\mathsf{Bnd}}\right)$.

$\mathsf{Comb}_A(\{c_i\}_{i\in[\tau]}, \{\mathsf{ds}_j\}_{j\in[\xi_2]})$. On input encrypted ballots $\{c_i\}_{i\in[\tau]}$ and decryption shares $\{\mathsf{ds}_j\}_{j\in[\xi_2]}$:

1. Parse ds_j as $\left(\{\mathsf{ds}_{ij}\}_{i\in[\tau]}, \pi_{\mathcal{D}_j}\right)$, and verify the proofs $\pi_{\mathsf{Lin}_{\mathsf{ij}}}$ and $\pi_{\mathsf{Bnd},ij}$, returning $\perp$ if either fails to verify.
2. Compute
$$v_i = \Big(\sum_{j\in[\xi_2]} \mathsf{ds}_{ij} \mod q\Big) \mod p.$$
3. Return the set of votes $\{v_i\}_{i\in[\tau]}$.

FIGURE 5.7: Π_{AVote} distributed decryption and combining.

PROOF OF SHUFFLE. In Step 3 of the shuffle, we use Π_{SHUF} to produce a proof that a set of committed values is a permutation of a set of public values.

In our context, the committed values will be the $\hat{\mathsf{ctx}}_i$ values. These can be constructed by the verifier from the ctx_i and the $[\![\mathsf{ctx}'_i]\!]$. The public values are the $\hat{\mathsf{ctx}}_i$. The proof then convinces the verifier that output ciphertexts are a genuine permutation of the set of re-randomized input ciphertexts. We present a full description of this proof system in Section 5.2.4.

AMORTIZED PROOF OF SHORTNESS. In Step 2 of the shuffle, we use Π_{SMALL}, to prove that we have committed to well-formed encryptions of zero. This is done in a batched way using a proof system that proves a set of equations of the form $\vec{A}\vec{s_i} = \vec{t_i}$. $i \in [\ell]$ is satisfied for a set of secret vectors $\vec{s_i}$ with ℓ_∞ norm bounded by ν.

The proof techniques are quite involved, combining error-correcting codes, Merkle trees, Lagrange interpolation and proximity testing. In Section 5.2.4 we formalise the relation to be proven and make explicit the relevant parameter constraints. We refer to [ABGS23] for full details.

AMORTIZED PROOFS OF BOUNDEDNESS. In Step 3 of DDec, we use Π_{BND} to prove boundedness on noise drowning terms E_{ij}. We define Π_{BND} to be an adapted version of the Π_{SMALL}, protocol, see Section 5.2.3 for full details.

5.4.5 *Security Analysis*

Here we analyze the security of our (verifiable) voting scheme presented in Figures 5.6 and 5.7. We begin by examining the implicit shuffle and distributed decryption protocols and prove their security. Then, we will consider the security of the overall voting scheme, showing how the security of the two aforementioned sub-protocols gives rise to the notions of integrity and privacy as required by an electronic voting scheme.

SHUFFLE PROTOCOL. We analyse the security of the verifiable shuffle protocol implicitly defined by the tuple of algorithms $\Pi_{\mathsf{AShuf}} := (\mathsf{KeyGen}_{\mathsf{A}}, \mathsf{Cast}_{\mathsf{A}}, \mathsf{Shuffle}_{\mathsf{A}}, \mathsf{Dec}_{\mathsf{NTRU}})$. We say that Π_{AShuf} is *secure* if it satisfies the properties of shuffle completeness, shuffle soundness, and shuffle simulatability. We refer the reader to Section 5.2.5 for formal definitions of these notions.

Lemma 5.4.2 (Π_{AShuf} Security)**.** *Suppose* Π_{SMALL}, *and* Π_{SHUF} *are complete, knowledge sound, and HVZK, and that* Com *is hiding. Furthermore, suppose that* $\mathtt{NTRU}-\mathtt{Encrypt}$ *is IND-CPA secure and let the total noise,* B_{Mix}*, added to ciphertexts in the shuffle be such that* $B_{\mathsf{Dec}} + B_{\mathsf{Mix}} \leq \lfloor q/2 \rfloor$*. Then the verifiable shuffle* Π_{AShuf} *is secure.*

We sketch the proof. Since Π_{SMALL}, and Π_{SHUF} are complete, the protocol will finish and the shuffle will verify correctly. Moreover, since $B_{\mathsf{Mix}} + B_{\mathsf{Dec}} \leq \lfloor q/2 \rfloor$, decryption will be correct. Thus, Π_{AShuf} is complete.

Now assume we have an adversary $\mathcal{A}$ breaking the knowledge soundness of the shuffle. That is, given a set of input ciphertexts, $\mathcal{A}$ can produce a new set of ciphertexts and a shuffle proof which verifies but the new ciphertexts do not decrypt to the original plaintexts. Without loss of generality, assume that ciphertext $\hat{c}_i$ decrypts incorrectly. By the knowledge soundness of $\pi_{\mathsf{Small},}$, one can extract commitment randomness $\mathbf{s}_{c'_i} = [\mathbf{r}c'_i, s'_i, e'_i]^T$ such that $\mathbf{A}_{\mathsf{M}}\mathbf{s}_{c'_i} = \mathbf{t}'_i$. By knowledge soundness of π_{Shuf}, it is easy to check that one can extract a second vector $\tilde{\mathbf{s}}_{c'_i} = [\tilde{\mathbf{r}}c'_i, \tilde{s}'_i, \tilde{e}'_i]^T$ satisfying $\mathbf{A}_{\mathsf{M}}\tilde{\mathbf{s}}_{c'_i} = \mathbf{t}'_i$. Then incorrect decryption of $\hat{c}_i$ corresponds to either $\tilde{s}'_i \neq s'_i$ or $\tilde{e}'_i \neq e'_i$. In either case, we break the binding property of the BDLOP commitment scheme.

Finally, we can argue the shuffle simulatability in the standard way. We construct a simulator that, given a set of input ciphertexts and output ciphertexts from an honest shuffle, simulates the proofs $\pi_{\mathsf{Small},}$ and π_{Shuf} using the HVZK property of those systems and replaces the commitments to ciphertexts with commitments to zero. Simulatability then follows from the hiding property of the commitments and IND-CPA security of $\mathtt{NTRU}-\mathtt{Encrypt}$.

DISTRIBUTED DECRYPTION SECURITY. We now analyse the security of the PKE with distributed decryption implicitly defined by the tuple of algorithms $\Pi_{\mathsf{ADDec}} := (\mathsf{KeyGen}_{\mathsf{A}}, \mathsf{Cast}_{\mathsf{A}}, \mathsf{Dec}_{\mathsf{NTRU}}, \mathsf{DDec}_{\mathsf{A},},$ $\mathsf{Comb}_{\mathsf{A}})$. We say that Π_{ADDec} is *secure* if it satisfies the properties of IND-CPA security, threshold correctness, threshold verifiability, and distributed decryption simulatability. For completeness, we give the full definitions of these notions in Section 5.2.5. Since many of these properties rely on building blocks used in previous works, we provide a proof sketch here and refer the reader to [ABGS23] for the full arguments. We will however make parameter constraints explicit to aid in the performance analysis of Section 5.5.

Lemma 5.4.3 (Π_{ADDec} Security)**.** *Let* $B_{\mathsf{Drown}} = 2^{\text{sec}}(B_{\mathsf{Dec}}/p\xi_2) < \lfloor q/2 \rfloor$. *Suppose that* Π_{LIN} *and* Π_{BND} *are complete, knowledge sound, and HVZK, and* $\mathtt{NTRU}-\mathtt{Encrypt}$ *is IND-CPA secure. Then the verifiable, distributed decryption protocol* Π_{ADDec} *is secure.*

We sketch the proof. IND-CPA security of Π_{ADDec} follows trivially from the IND-CPA security of the underlying NTRU encryption scheme and the HVZK property of the proofs Π_{LIN} and Π_{BND}.

Examining the threshold correctness, define the predicate $P_{\mathsf{sk_A}}(\cdot)$ so that $P_{\mathsf{sk_A}}(c) = 1$ if and only if $\|\mathsf{sk_A} \cdot c\|_\infty < B_{\mathsf{Dec}}$. Then given a set of adversarially generated ciphertexts $\{c\}_{i\in[\tau]}$ satisfying $P_{\mathsf{sk_A}}(c_i) = 1$ for all $i \in [\tau]$, we have that $\left\|\sum_{j\in[\xi_2]} \mathsf{ds}_{ij}\right\|_\infty < q/2$ and so the Comb algorithm will return the correct decryption of c_i. The completeness of Π_{LIN} and Π_{BND} ensure that the arguments will be accepted, thus, Π_{ADDec} is threshold correct.

For threshold verifiability we also consider only ciphertexts such that $P_{\mathsf{sk_A}}(c) = 1$. Note that if Comb accepts a ciphertext for which decryption is incorrect then, for some j, no relation $\mathsf{ds}_{ij} = \mathsf{dk}_j \cdot c_i + pE_{ij}$ holds for an E_{ij} of norm at most B_{Drown}. This implies either an adversary against the soundness of Π_{LIN} or of Π_{BND}.

Finally, we describe a simulator for our distributed decryption. Firstly, let us replace commitments to E_{ij} with commitments to zero. This change is indistinguishable from the hiding property of the BDLOP commitment scheme. Next, one can simulate the proofs $\pi_{\mathsf{Lin_{ij}}}$ and π_{Bnd}, otherwise a distinguisher breaks the HVZK of the respective proof systems. Lastly, the simulatability of the ds_{ij} follows from the choice of sec which is chosen so that ds_{ij} is statistically indistinguishable from uniform.

We now argue our voting protocol's overall security by discussing its integrity and privacy properties. We give a very high-level overview of the security properties a voting system requires and how our design achieves these properties. For a detailed overview and the formal proofs, we refer the reader to [ABGS23]. Whilst some building blocks are different in our work, they are combined in much the same way as the aforementioned work.

INTEGRITY. This property ensures that no adversary is able to cause inconsistent decryption or non-unique decryption, even if the adversary obtains all of the key material. It can be shown that an adversary succeeding in causing one of these events can be used to construct an adversary against either the threshold verifiability of the distributed decryption protocol Π_{ADDec} or the soundness of the shuffle protocol Π_{AShuf}.

PRIVACY. Privacy dictates that an adversary seeing the contents of the ballot box, the intermediate shuffles, and the decrypted ballot shares should not be able to determine who cast which ballot. This should hold even if the adversary learns some decryption key shares, inserts adversarially generated

Parameter	Explanation	Constraints
λ	Comp. sec. parameter	≥ 128
sec	Stat. sec. parameter	≥ 40
d	Ring dimension of R_p and R_q	d a power of two
p	Plaintext modulus	p a small prime
q	Ciphertext & com. modulus	prime $q = 1 \mod 2d$ s.t. $\|\sum_{j\in\xi_2} \mathsf{ds}_j\|_\infty \leq \lfloor q/2 \rfloor$
t	$\mathsf{KeyGen}_{\mathsf{NTRU}}$ rej, param	s.t. rej. prob. $< 1/1000$ (??)
k	BDLOP binding param.	> 2
$\mathcal{C}$	Challenge space	$\{c \in R_3 \mid \|c\|_1 = \kappa\}$
κ	Max ℓ_1-norm of elements in $\mathcal{C}$	$2^\kappa \cdot \binom{d}{\kappa} > 2^\lambda$
ξ_1, ξ_2	# shuffle/dec. servers	≥ 2
B_{Com}	Bound on the com. noise	so that SIS is hard
B_{Dec}	Noise in ciphertext	$pdt\sigma_{\mathsf{NTRU}}(2\xi_1\nu + \frac{1}{2})$
B_{Drown}	Inf. norm of drowning E_{ij}	$2^{\mathrm{sec}}(B_{\mathsf{Dec}}/p\xi_2)$
σ_{NTRU}	S.d. for NTRU key	$\mathsf{NTRU}_{d,q,\sigma_{\mathsf{NTRU}}}$ hard
ν	Bound on enc. randomness	$\mathsf{R\text{-}LWE}_{d,q,\sigma_{\mathsf{R\text{-}LWE}}}$ hard
σ_{Com}	S.d in ZKPs of lin. relations	$\sigma_{\mathsf{Com}} = \kappa B_{\mathsf{Com}}\sqrt{kd}$
τ	Total # votes cast	$(\tau^\delta + 1)/\lvert R_q \rvert < 2^{-\lambda}$
η	R.S. encoding rand. len.	s.t. soundness $\geq 1 - 2^{-\lambda}$
ℓ_{Small}	Batch size in Π_{SMALL},	same secret len as [ABGS23]
ℓ_{Bnd}	Batch size in Π_{BND}	same secret len. as [ABGS23]
μ_{Small}	R.S. message len. in Π_{SMALL},	$\mu_{\mathsf{Small}} = (k+2)d + \eta$
μ_{Bnd}	R.S. message len in Π_{BND}	$\mu_{\mathsf{Bnd}} = (k+1)\cdot d + \eta$
μ'_{Small}	R.S. message dim. in Π_{SMALL},	$\mu_{\mathsf{Small}} \leq \mu'_{\mathsf{Small}} \leq \gamma < q$
μ'_{Bnd}	R.S. message dim. in Π_{SMALL},	$\mu_{\mathsf{Bnd}} \leq \mu'_{\mathsf{Bnd}} \leq \gamma < q$
γ_{Small}	R.S. code len. in Π_{BND}	$\mu_{\mathsf{Small}} \leq \mu'_{\mathsf{Small}} \leq \gamma < q$
γ_{Bnd}	R.S. code len. in Π_{BND}	$\mu_{\mathsf{Bnd}} \leq \mu'_{\mathsf{Bnd}} \leq \gamma < q$

TABLE 5.1: System parameters and constraints.

ciphertexts into the ballot box, introduce adversarially generated intermediate shuffles, and publish adversarially chosen decrypted ballot shares. This reduces to the NTRU cryptosystem's CPA-security, commitments hiding, and zero-knowledge properties of the underlying proofs.

5.5 PERFORMANCE

5.5.1 *Parameter Constraints*

We begin by collecting all parameters of the scheme and noting any constraints applying to them in Table 5.1.

Next, we closely examine the constraint needed for the correct (perfect) decryption of votes as performed by the Comb algorithm. This turns out to be the most influential constraint on the overall efficiency of the scheme. In particular, this constraint informs our choice of the global ring dimension d and modulus q, which most directly affect the communication sizes.

DECRYPTION CORRECTNESS. After passing through the mix-net of ξ_1 shuffle servers, a ciphertext is of the form

$$c = p\left(h\sum_{k\in[\xi_1]} s_k + \sum_{k\in[\xi_1]} e_k\right) + m,$$

where the encryption randomness terms s_k and e_k are sampled from S_ν. Next, this ciphertext is passed to a decryption server, which computes a decryption share of the form $\mathsf{ds}_j = f_j \cdot c + pE_j$. Then the Comb algorithm, on collecting $\{\mathsf{ds}_j\}_{j\in[\xi_2]}$, outputs

$$v' = \left(\sum_{j\in[\xi_2]} \mathsf{ds}_j \mod q\right) \mod p.$$

In order for the result of this process to yield the original ballot cast, we require the infinity norm of the sum here to be bounded by $\lfloor q/2 \rfloor$. It follows that a sufficient constraint for this correct decryption is the following:

$$p \cdot d \cdot t \cdot \sigma_{\mathsf{NTRU}} \cdot (2\xi_1 \cdot \nu + 1/2)(1 + 2^{\mathrm{sec}}) < \lfloor q/2 \rfloor, \tag{5.2}$$

where t is the rejection parameter in $\mathsf{KeyGen}_{\mathsf{NTRU}}$.

COMPUTATIONAL SECURITY. Having chosen parameters satisfying the constraints of Table 5.1, we must ensure that the underpinning lattice problems are sufficiently hard for these parameters.

For RLWE we follow standard convention by using the estimator [APS15]. This estimates the cost of BKZ conservatively by focusing only on the cost of a single uSVP oracle call, a core operation in BKZ. The number of such calls required has been estimated to be $8d$ for a lattice dimension d, and we follow this estimate.

To determine the NTRU problem's hardness, we use the analysis of Section 5.3. Having settled on a ring dimension d and modulus q giving sufficient hardness of the RLWE problem, we use (5.2) to determine the maximum

standard deviation permissible for generating the NTRU secrets (f, g). Finally, following the procedure described in Section 5.3, one can calculate the estimated computational complexity of the given NTRU instance. Again, we employ the conservative formula $0.292\beta + 16.4 + \log_2(8d)$ used in works such as [DTGW17, SPL+17, BIP+22] to compute bit-security from an estimated blocksize β.

In order to ensure the binding property of the BDLOP commitment schemes we use, the RSIS problem must be hard. We use the relation due to Micciancio and Regev [MR09], which states that LLL will recover a short vector a vector of 2-norm $2^{(2\sqrt{d \log_2 q \log_2 \delta})}$. δ is the root Hermite factor and $\delta < 1.0045$ gives rise to at least 128 bits of security. Owing to the horizontally long nature of the commitment matrix used, the hardness of the corresponding RSIS instance easily meets this threshold.

5.5.2 *Sample Parameters and Total Size*

Table 5.2 gives a sample set of parameters generated by following the process described in the previous section.

In Table 5.3, we present the total sizes of objects in our voting scheme and compare them with those of [ABGS23]. We denote the output of each shuffle node by $\pi_{\mathcal{S}_i}$, including ciphertexts, commitments, proofs of shortness, and shuffle proofs. Similarly, we denote the total output of each decryption node as $\pi_{\mathcal{D}_j}$, consisting of decryption shares, commitments, proofs of linearity and boundedness.

Our scheme achieves a reduction in ciphertext size by over a factor of five. Moreover, the reduction in commitment sizes and constituent proofs leads to shuffle server outputs of 130 KB per vote, which are three times smaller, and decryption server outputs of 85 KB per vote, which are half of those in [ABGS23]. Overall this represents a 2.5× efficiency gain over [ABGS23] as summarized in **??**.

5.5.3 *Benchmarks*

We adapted the proof-of-concept implementation by [ABGS23] to fit our scheme since the framework is the same. Our benchmarks were collected on an Intel Kaby Lake Core i7-7700 CPU machine with 64 GB of RAM running single-threaded at 3.6 GHz, with Turbo Boost disabled to reduce measurement variability. This is a similar machine as in [ABGS23]. Our code is available at `https://anonymous.4open.science/r/pets-2024-submission-79`.

Parameter	Explanation	Value
λ	Computational security parameter	128
d	Ring dimension	2048
q	Ciphertext and commitment modulus	$\approx 2^{59}$
sec	Statistical security parameter	40
p	Plaintext modulus	2
t	$\mathsf{KeyGen}_{\mathsf{NTRU}}$ rejection parameter	1.058
ν	Infinity norm of encryption randomness	1
ξ_1, ξ_2	Number of shuffle and decryption servers	4
B_{Com}	Infinity norm of commitment randomness	1
B_{Dec}	Noise in ciphertext	262144
B_{Drown}	Infinity norm of noise drowning term E_{ij}	$\approx 2^{55}$
σ_{NTRU}	Standard deviation for encryption secret key	7.12
η	Reed-Solomon encoding randomness length	325
ℓ_{Small}	Proof batch size in Π_{SMALL},	9830
ℓ_{Bnd}	Proof batch size in Π_{BND}	12288
μ_{Small}	Reed-Solomon message length in Π_{SMALL},	10565
μ_{Bnd}	Reed-Solomon message length in Π_{BND}	8517
μ'_{Small}	Reed-Solomon message dimension in Π_{SMALL},	23988
μ'_{Bnd}	Reed-Solomon message dimension in Π_{BND}	181550
γ_{Small}	Reed-Solomon code length in Π_{SMALL},	26616
γ_{Bnd}	Reed-Solomon code length in Π_{BND}	198668

TABLE 5.2: Sample parameter set.

Scheme	c_i	$[\![R_q]\!]$	π_{Shuf}	π_{Lin}	π_{Small},	π_{Bnd}
[ABGS23] [KB]	80	80/120	150	35	20	2
Ours [KB]	15	30	63	18	22	22

TABLE 5.3: Ciphertext, commitment, and proof sizes per voter. Note that the two sizes in [ABGS23] reflect commitments to noise-drowning terms and ciphertexts, respectively.

Scheme	Com	Open	Enc	Dec	DDec
[ABGS23] [ms]	0.45	2.76	0.74	0.64	1.56
Ours [ms]	0.17	0.80	0.20	0.21	0.45

TABLE 5.4: Ciphertext and commitment timings. Numbers were obtained averaging over 10^4 executions measured using the cycle counter available on the platform.

Scheme	π_{Lin}	π_{Shuf}
[ABGS23] [ms]	$(43.4 + 6.4)$	$(44.9 + 7.9)$
Ours [ms]	$(16.9 + 2.0)$	$(17.5 + 2.1)$
Scheme	$\pi_{\mathsf{Small},}$	π_{Bnd}
[ABGS23] [ms]	$(214.4 + 10.0)$	$(92.7 + 23.9)$
Our [ms]	$(44.2 + 4.0)$	$(310.5 + 4.3)$

TABLE 5.5: Proving and verification times, obtained by computing the average of 100 executions with $\tau = 1000$.

We compare the timings in Table 5.4 and Table 5.5. Analysing our experiments, each shuffle server takes $(0.20 + 0.17 + 17.5 + 44.2) = 62$ ms and each decryption server takes $(0.17 + 0.45 + 16.9 + 310.5) = 328$ ms. Given four servers, where shuffles are performed sequentially and decryption is performed in parallel, the total time is 576 ms, making our scheme twice as fast as [ABGS23] as summarized in Table 5.5.

We finally note that the proofs of linearity in the shuffle and the batched proofs of shortness and boundedness during shuffles and decryption can be computed in parallel, and that powerful servers dedicated to an election with Turbo Boost enabled would most likely outperform our numbers by at least an order of magnitude.

6

CONCLUSIONS

In this thesis we studied cryptographic protocol lying in the intersection of post-quantum security, privacy preservation, and distributed trust.

Our work on DAA security should pave the way for more designs from post-quantum assumptions (in addition to ones over lattices). It shines a light on redundancies in the UC definition of [CDL16] and demonstrates the relative ease of providing security proofs in the game-based setting. Our generic construction acts as modular framework for building future DAA schemes. In particular, this means that as the component building blocks individually improve through the literature, the state-of-the-art DAA constructions can be easily updated with 'drop-in' replacement of these improved building blocks.

Our lattice-DAA construction finally puts to an end a long line of research towards a truly practical DAA with post-quantum security. We achieve this by weaving together recent progress in the space of lattice-based ZKPs and anonymous credentials, efficient instantiations of which have been long sought-after in their own right.

Prior to this work, there was little attention paid to the NTRU assumption in constructing privacy-preserving protocols. Through both our LDAA an voting protocols, we have demonstrated the potential for great efficiency saving through the adoption of NTRU; our LDAA scheme improving by an order of magnitude over previous works and our voting protocol improving the state-of-the-art by 2.5 times.

As a more general observation, we think it important to note that our works suggests the existing lattice-based zero-knowledge research has arrived at a place to provide not only sufficient efficiency but also flexibility for the design of a whole range of privacy-preserving protocols.

6.1 FUTURE RESEARCH DIRECTIONS

IMPLEMENTATION. With the widespread deployment of TPMs and urgency in transitioning to post-quantum cryptography, a crucial next step would be to implement our new LDAA. This would involve analysis both

of the running times and side channel attacks but also inform the memory requirements of the FutureTPM.

VOTING ADD-ONS. Our voting protocol is based on the most common framework involving both a mixnet and distributed decryption phase. Whilst it satisfactorily improves on the state-of-the-art here, there are additional features one might require for a voting implementation.

Firstly, one could incorporate return codes. Consider a voter, casting their ballot on their preferred digital device. The voter casts their ballot and is notified, on screen, that their vote has been submitted. This does note give any guarantees that their chosen ballot has been received. In particular, if their devices has been compromised, this system could easily be spoofed. Return codes give the voter a method by which to verify, mathematically, that the vote they selected has indeed been counted in the ballot counting process. We suggest that this could be one by extending the work of [HS22] from BGV to NTRU.

One of the main bottlenecks in achieving efficient parameters for out voting scheme is the need for a large noise drowning term in the creation of decryption shares. Our results can possibly be improved using techniques in [AKSY22, BS23, CSS+22]. We use 40 bits of statistical noise drowning to protect the secret key in the distributed decryption protocol. This can possibly be improved if we choose parameters based on how many ciphertexts we will decrypt or change noise drowning techniques to Gaussian and compute the Rényi divergence to estimate the leakage.

Having observed the flexibility of *module*-NTRU when setting parameters for our LDAA scheme, it would be interesting to see whether moving from the ring setting to the module setting would improve the efficiency of our voting scheme. Since our parameters are chosen very tightly to satisfy the decryption bound Equation (5.2), it is unlikely that much would be gained in the exact scheme design we give. However, if changing the noise drowning techniques as mentioned above, one might find reason to use a lattice dimension that could be chosen in a more granular way, as opposed to being constrained to power-of-two as in the ring setting.

DISTRIBUTED NTRU KEY GENERATION. The one area in which our voting scheme does not match the functionality of the previous works is that it assumes a trusted setup. LWE-based encryption schemes enjoy secret keys which can be combined homomorphically (due to their linear structure). This allows for fairly straight-forward distributed key generation using the sum

of individually chosen keys to implicitly define a global secret key. However, an NTRU key (being the quotient of two polynomials or matrices) has no such nice structure. In particular, linearly combining NTRU keys does not lead to additive shares in a new key. We thus suggest that distributed key generation for NTRU poses and interesting, albeit non-trivial, challenge.

A

APPENDIX

A.1 EQUIVALENT NOTIONS OF ANONYMITY

In this section we prove Lemma A.1.3 which shows that anyone able to win the 'real-or-random' anonymity game, can win the left-or-right variant used in the security model of Section 3.3.2.

This is the final piece in proving Theorem 3.4.1 since when proving the indistinguishability of games 6 and 7, we use a distinguishing environment to win the real-or-random anonymity game. By Lemma A.1.3 one can then break the anonymity (LOR version) of protocol Π_{DAA}. Since Π_{DAA} is assumed secure according to Definition 3.3.11, we have that no such adversary exists and so games 6 and 7 are indistinguishable. Moreover, we prove the equivalence of these two notions of anonymity, defining the security experiment and challenge oracle for the real-or-random variant.

We note that the equivalence of these two notions might be of independent interest when designing a concrete DAA instantiation since it might be easier to work with one or the other anonymity notion. We further note that these definitions are no longer equivalent if the issuer does not have to register its secret key with an extractable proof of knowledge. However, Theorem 3.4.1 requires that the issuer does register isk with an online-extractable proof of knowledge. Thus, as long as one aims to satisfy the conditions of Theorem 3.4.1 (as is desirable to imply UC), these anonymity variants can be used interchangeably. In the following discussion, both experiment variants will feature oracles allowing the adversary to create honest platforms, create credentials for these platforms, and request signatures from them. Where the two experiments vary is in how the challenge oracle responds to the *challenge* signing queries. For this reason, our discussion will focus on the behaviour of the challenge oracles of each experiment rather than the oracles common to both.

We first recall the notion of *left-or-right* anonymity. This is the definition we use in our security model (see Figure 3.3). Here, the adversary is allowed to make challenge queries of the form $(i_0, i_1, \mathsf{m}, \mathsf{bsn})$ to the challenge oracle ChalO^{lor} for TPMs $\mathcal{M}_{i_0}$and $\mathcal{M}_{i_1}$, a message m, and basename bsn. Two games are considered. In the first, each challenge query is responded to by signing under the credentials of $\mathcal{M}_{i_0}$; in the second, those of $\mathcal{M}_{i_1}$ are used.

Formally, we define the *left-or-right* challenge oracle $\mathsf{ChalO}^{lor}(\cdot,\cdot,\cdot,\cdot;b)$ in Figure A.3, however we include a 'compact' version in Figure A.1 for readability. We now recall the *left-or-right* anonymity experiment which is used in our security model in Figure 3.3.

Definition A.1.1 (**LOR-ANON**). *Let* DAA *be a DAA scheme. Let* $b \in \{0,1\}$ *and* $\lambda \in \mathbb{N}$. *Consider the adversary* $\mathcal{A}_{lor}$ *partaking in experiment* $\mathsf{Exp}^{\text{lor-Anon-b}}_{\mathsf{DAA},\mathcal{A}_{lor}}(\lambda)$ *defined in Figure A.1. We define the advantage function as follows.*

$$\mathsf{Adv}^{\text{lor-Anon}}_{\mathsf{DAA},\mathcal{A}_{lor}}(\lambda) = \left|\Pr\left[\mathsf{Exp}^{\text{lor-Anon-0}}_{\mathsf{DAA},\mathcal{A}_{lor}}(\lambda)=1\right] - \Pr\left[\mathsf{Exp}^{\text{lor-Anon-1}}_{\mathsf{DAA},\mathcal{A}_{lor}}(\lambda)=1\right]\right|$$

The scheme DAA *is said to be LOR-ANON secure if the function* $\mathsf{Adv}^{\text{lor-Anon}}_{\mathsf{DAA},\mathcal{A}_{lor}}(\lambda)$ *is negligible for any adversary* $\mathcal{A}_{lor}$ *whose time complexity is polynomial in* λ.

$\underline{\mathsf{ChalO}^{lor}(i_0, i_1, \mathsf{m}, \mathsf{bsn}; b)}$

- **If** $i_{b'} \notin \mathsf{HP}$ or $\mathsf{CRE}[i_{b'}] = \perp$ or $\mathsf{HSK}[i_{b'}] = \perp$ for $b' \in \{0,1\}$ then **Return** $\perp$.
- **If** $|\mathsf{CL}| \geq 1$ and $(i_0, i_1, *) \notin \mathsf{CL}$, then **Return** $\perp$.
- Run the interaction $\langle \mathsf{SignTPM}(\mathsf{HSK}[i_b], \mathsf{m}, \mathsf{bsn}), \mathsf{SignH}(\mathsf{CRE}[i_b], \mathsf{m}, \mathsf{bsn})\rangle$ to obtain σ.
- $\mathsf{CL} \leftarrow \mathsf{CL} \cup \{(i_0, i_1, \mathsf{bsn})\}$.
- **Return** σ.

$\underline{\mathsf{Exp}^{\text{lor-Anon-b}}_{\mathsf{DAA},\mathcal{A}_{lor}}(\lambda):}$

- $\mathsf{sp} \leftarrow \mathsf{Setup}(1^\lambda)$; $\mathsf{HP}, \mathsf{EPK}, \mathsf{ESK}, \mathsf{CRE}, \mathsf{HSK}, \mathsf{SL}, \mathsf{CL} := \emptyset$.
- $(\mathsf{ipk}, \mathsf{ML}, \mathsf{state}) \leftarrow \mathcal{A}(\mathsf{sp})$.
- **If** $\mathcal{A}$'s output is not well formed, then **Return** $\perp$.
- $d \leftarrow \mathcal{A}(\mathsf{ipk}, \mathsf{isk}, \mathsf{state}; \mathsf{ChalO}^{lor}(\cdot,\cdot,\cdot,\cdot;b), \mathsf{AddHonO}(\cdot,\mathcal{P}), \mathsf{JoinO}(\cdot;\mathcal{P}), \mathsf{SignO}(\cdot,\cdot,\cdot,\cdot;\mathcal{P})))$.
- **If** $\exists(i, \mathsf{m}, \mathsf{bsn}, \sigma) \in \mathsf{SL}$ s.t. $\mathsf{bsn} \neq \perp$ and $(i, *, \mathsf{bsn}) \in \mathsf{CL}$ or $(*, i, \mathsf{bsn}) \in \mathsf{CL}$, then **Return** 0.
- If $d = b$, then **Return** 1.
- **Return** 0.

FIGURE A.1: Challenge oracle and security experiment for 'left-or-right' anonymity.

We now define the *real-or-random* (ROR) notion of anonymity. We describe the ROR challenge oracle $\mathsf{ChalO}^{ror}(\cdot,\cdot,\cdot;b)$ parametrised by the bit b which takes as input a TPM identity i, a message m, and a basename bsn. This oracle will take as input a TPM identity i, a message m, and a basename bsn. Parametrised by a bit b, the oracle will either generate a signature on $(\mathsf{m}, \mathsf{bsn})$ under TPM i (if $b = 0$) or create a new TPM identity

(which is fixed once and for all) and sign using the freshly generated keys and credential (if $b = 1$). In the latter case, the oracle needs to be able to extract the issuer's secret key so that it can create a valid credential for this freshly generated TPM. Thus, we introduce the extraction algorithm that does this. As stated in Theorem 3.4.1, we consider only 'on-line' extraction (sometimes called straight-line extraction in the literature) since rewinding is not possible in the UC framework. One cannot rewind the environment who would notice that the issuer's behaviour is not as expected. We refer the reader to [Fis05] for a formal definition of on-line extraction and herein use the algorithm Ext to denote the process by which the simulator obtains the secret key of a corrupt issuer as needed in $\mathsf{ChalO}^{ror}(\cdot,\cdot,\cdot;\cdot)$. We describe $\mathsf{ChalO}^{ror}(\cdot,\cdot,\cdot;\cdot)$ in Figure A.2. Again, this is a 'compact' definition designed to provide intuition and the reader should note that to capture concurrency one needs to flesh this out using the 'state-message-decision' framework as demonstrated in Figure A.3.

Definition A.1.2 (**ROR-ANON**)**.** *Let* DAA *be a DAA scheme. Let* $b \in \{0,1\}$ *and* $\lambda \in \mathbb{N}$*. Consider the adversary* $\mathcal{A}_{ror}$ *partaking in experiment* $\mathsf{Exp}^{\text{ror-Anon-b}}_{\mathsf{DAA},\mathcal{A}_{ror}}(\lambda)$ *defined in Figure A.2. We define the advantage function as follows.*

$$\mathsf{Adv}^{\text{ror-Anon}}_{\mathsf{DAA},\mathcal{A}_{ror}}(\lambda) = \left|\Pr\left[\mathsf{Exp}^{\text{ror-Anon-0}}_{\mathsf{DAA},\mathcal{A}_{ror}}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\text{ror-Anon-1}}_{\mathsf{DAA},\mathcal{A}_{ror}}(\lambda) = 1\right]\right|$$

The scheme DAA *is said to be ROR-ANON secure if the function* $\mathsf{Adv}^{\text{ror-Anon}}_{\mathsf{DAA},\mathcal{A}_{ror}}(\lambda)$ *is negligible for any adversary* $\mathcal{A}_{ror}$ *whose time complexity is polynomial in* λ*.*

We now state and prove reductions between these two variant of anonymity. These lemmata suffice to show that the two notions are equivalent.

Lemma A.1.3 (LOR-ANON ⇒ ROR-ANON)**.** *For any DAA scheme* DAA*,*

$$\mathsf{Adv}^{\text{ror-Anon}}_{\mathsf{DAA},\mathcal{A}_{ror}}(\lambda) \leq \mathsf{Adv}^{\text{lor-Anon}}_{\mathsf{DAA},\mathcal{A}_{lor}}(\lambda).$$

Lemma A.1.4 (ROR-ANON ⇒ LOR-ANON)**.** *For any DAA scheme* DAA*,*

$$\mathsf{Adv}^{\text{lor-Anon}}_{\mathsf{DAA},\mathcal{A}_{lor}}(\lambda) \leq 2 \cdot \mathsf{Adv}^{\text{ror-Anon}}_{\mathsf{DAA},\mathcal{A}_{ror}}(\lambda) + \mathsf{negl}(\lambda).$$

Of Lemma A.1.3. Let $\mathcal{A}_2$ be an adversary that has non-negligible advantage ϵ in the ROR-ANON experiment. Then, we construct an adversary $\mathcal{A}_1$ with advantage ϵ in the LOR-ANON experiment which runs $\mathcal{A}_2$ as a subroutine.

$\underline{\mathsf{ChalO}^{ror}(i, \mathsf{m}, \mathsf{bsn}; b)}$

- **If** $i' = \perp$, sample i' at random.
- **If** $i \notin \mathsf{HP}$, $\mathsf{CRE}[i] = \perp$ or $\mathsf{HSK}[i] = \perp$ then **Return** $\perp$.
- **If** $|\mathsf{CL}| \geq 1$ and $(i, i', *) \notin \mathsf{CL}$, then **Return** $\perp$.
- **If** $b = 0$:
  - Run the interaction $\langle \mathsf{SignTPM}(\mathsf{sk}[i], \mathsf{m}, \mathsf{bsn}), \mathsf{SignH}(\mathsf{CRE}[i], \mathsf{m}, \mathsf{bsn}) \rangle$ to obtain σ.
- **If** $b = 1$:
  - **If** $\mathsf{HSK}[i'] \neq \perp$ and $\mathsf{CRE}[i'] \neq \perp$ then **Return** $\sigma := \langle \mathsf{SignTPM}(\mathsf{HSK}[i'], \mathsf{m}, \mathsf{bsn}), \mathsf{SignH}(\mathsf{CRE}[i'], \mathsf{m}, \mathsf{bsn}) \rangle$.
  - Run $\mathsf{isk} \leftarrow \mathsf{Ext}(\mathsf{sp}, \mathsf{td}, \mathsf{ipk})$.
  - Run $(\mathsf{ESK}[i'], \mathsf{EPK}[i']) \leftarrow \mathsf{TPM.Kg}(\mathsf{sp})$
  - Run the interactions $\langle \mathsf{JoinTPM}(\mathsf{ESK}[i']), \mathsf{JoinH}(\mathsf{EPK}[i'], \mathsf{ipk}), \mathsf{Issue}(\mathsf{isk}, \mathsf{EPK}[i'], \mathsf{ML}) \rangle$ and $\langle \mathsf{SignTPM}(\mathsf{HSK}[i'], \mathsf{m}, \mathsf{bsn}), \mathsf{SignH}(\mathsf{CRE}[i'], \mathsf{m}, \mathsf{bsn}) \rangle$ to obtain σ.
- $\mathsf{CL} \leftarrow \mathsf{CL} \cup \{(i, i', \mathsf{bsn})\}$
- **Return** σ.

$\underline{\mathsf{Exp}^{\text{ror-Anon-b}}_{\mathsf{DAA}, \mathcal{A}_{\text{ror}}}(\lambda) :}$

- $\mathsf{sp} \leftarrow \mathsf{Setup}(1^\lambda)$; $\mathsf{HP}, \mathsf{EPK}, \mathsf{ESK}, \mathsf{CRE}, \mathsf{HSK}, \mathsf{SL}, \mathsf{CL} := \emptyset$.
- $(\mathsf{ipk}, \mathsf{ML}, \mathsf{state}) \leftarrow \mathcal{A}(\mathsf{sp})$.
- **If** $\mathcal{A}$'s output is not well formed, then **Return** $\perp$.
- $d \leftarrow \mathcal{A}(\mathsf{ipk}, \mathsf{isk}, \mathsf{state}; \mathsf{ChalO}^{ror}(\cdot, \cdot, \cdot; b), \mathsf{AddHonO}(\cdot, \mathcal{P}), \mathsf{JoinO}(\cdot; \mathcal{P}), \mathsf{SignO}(\cdot, \cdot, \cdot, \cdot; \mathcal{P}))$.
- **If** $\exists (i, \mathsf{m}, \mathsf{bsn}, \sigma) \in \mathsf{SL}$ s.t. $\mathsf{bsn} \neq \perp$ and $(i, *, \mathsf{bsn}) \in \mathsf{CL}$ or $(*, i, \mathsf{bsn}) \in \mathsf{CL}$, then **Return** 0.
- **If** $d = b$, then **Return** 1.
- **Return** 0.

FIGURE A.2: Challenge oracle and security experiment for 'real-or-random' anonymity.

In the following assume $\mathcal{A}_1$ is taking part in the LOR-ANON experiment with challenge bit b, i.e., $\mathcal{A}_1$ has access to $\mathsf{ChalO}^{lor}(\cdot,\cdot,\cdot,\cdot;b)$. $\mathcal{A}_1$ will (implicitly) simulate the ROR-ANON experiment with challenge bit b for $\mathcal{A}_2$ as follows: When $\mathcal{A}_2$ queries $\mathsf{AddHonO}(\cdot,\mathcal{P})$, $\mathsf{JoinO}(\cdot;\mathcal{P})$, $\mathsf{SignO}(\cdot,\cdot,\cdot,\cdot;\mathcal{P})$, $\mathcal{A}_1$ simply relays them to its own oracle and returns the output to $\mathcal{A}_2$. When $\mathcal{A}_2$ queries $\mathsf{ChalO}^{ror}(i,\mathsf{m},\mathsf{bsn};b)$, $\mathcal{A}_1$ does the following.

1. Create a fresh TPM i' (unknown to $\mathcal{A}_2$) by calling $\mathsf{JoinO}(i',\mathcal{P})$ thus creating a set of signing credentials for this new TPM. Note that $\mathcal{A}_1$ can do this since it controls $\mathcal{I}$.
2. It then queries its challenge oracle $\mathsf{ChalO}^{lor}(\cdot,\cdot,\cdot,\cdot;b)$ with input $(i,i',\mathsf{m},\mathsf{bsn})$ and receives back the signature σ.
3. Finally, it returns σ to $\mathcal{A}_2$.

At some point of the experiment, if $\mathcal{A}_2$ outputs a bit d as its ROR guess, $\mathcal{A}_1$ outputs d as his LOR guess.

It is easy to check that $\mathcal{A}_1$ perfectly simulates the ROR-ANON experiment with challenge bit b to $\mathcal{A}_2$ when it is taking part in the LOR-ANON experiment with challenge bit b. Moreover, since the winning condition (i.e., the check regarding the challenge list CL) of $\mathcal{A}_2$ is the same as $\mathcal{A}_1$, $\mathcal{A}_1$ wins whenever $\mathcal{A}_2$ wins. Therefore, we have

$$\begin{aligned}\mathsf{Adv}^{\text{lor-anon}}_{\mathsf{DAA},\mathcal{A}_1}(\lambda) &= \left|\Pr\left[\mathsf{Exp}^{\text{ror-Anon-1}}_{\mathsf{DAA},\mathcal{A}_2}(\lambda)=1\right]-\Pr\left[\mathsf{Exp}^{\text{ror-Anon-0}}_{\mathsf{DAA},\mathcal{A}_2}(\lambda)=1\right]\right| \\ &= \mathsf{Adv}^{\text{ror-anon}}_{\mathsf{DAA},\mathcal{A}_2}(\lambda).\end{aligned}$$

This completes the proof. □

Of Lemma A.1.4. Let $\mathcal{A}_1$ be an adversary that has non-negligible advantage ϵ in the LOR-ANON experiment. Then, we construct an adversary $\mathcal{A}_2$ with advantage $\epsilon/2-\mathsf{negl}(\lambda)$ in the ROR-ANON experiment which runs $\mathcal{A}_1$ as a subroutine.

In the following assume $\mathcal{A}_2$ is taking part in the ROR-ANON experiment with challenge bit b, i.e., $\mathcal{A}_2$ has access to $\mathsf{ChalO}^{ror}(\cdot,\cdot,\cdot;b)$. $\mathcal{A}_2$ flips a random bit $b' \leftarrow \{0,1,\}$ and executes $\mathcal{A}_1$ as follows: When $\mathcal{A}_1$ queries $\mathsf{AddHonO}(\cdot,\mathcal{P})$, $\mathsf{JoinO}(\cdot;\mathcal{P})$, $\mathsf{SignO}(\cdot,\cdot,\cdot,\cdot;\mathcal{P})$, $\mathcal{A}_2$ simply relays them to its own oracle and returns the output to $\mathcal{A}_1$. When $\mathcal{A}_1$ queries $\mathsf{ChalO}^{lor}(i_0,i_1,\mathsf{m},\mathsf{bsn};b')$, $\mathcal{A}_2$ queries own challenge oracle $\mathsf{ChalO}^{ror}(i_{b'},\mathsf{m},\mathsf{bsn};b)$ and returns the output to $\mathcal{A}_1$. At some point of the experiment, if $\mathcal{A}_1$ outputs a bit d as its LOR guess, $\mathcal{A}_2$ outputs d as its ROR guess.

We analyze the advantage of $\mathcal{A}_2$. In case $b = 0$, $\mathcal{A}_2$ perfectly simulates the view of the LOR experiment to $\mathcal{A}_1$; when $b' = 0$, it simulates the LOR experiment with challenge bit b'. Moreover, as long as $\mathcal{A}_1$ does not make any signing query on the identity (i') chosen randomly within the challenge oracle $\mathsf{ChalO}^{ror}(\cdot,\cdot,\cdot;b)$, $\mathcal{A}_2$ wins when $\mathcal{A}_1$. Although the winning condition of $\mathcal{A}_1$ is not influenced even if $\mathcal{A}_1$ queries $i' \neq \{ i_0, i_1 \}$ with $\mathsf{bsn} \neq \perp$, this may influence the winning condition of $\mathcal{A}_2$. However, assuming that the TPM identity space is exponentially large (as is usually done) the event that $\mathcal{A}_1$ accidentally querying a signature for identity i' can happen with all but a negligible probability since i' is information theoretically hidden from $\mathcal{A}_1$. On the other hand, in case $b = 1$, $\mathcal{A}_2$ simulates an experiment where $\mathcal{A}_1$ runs identically regardless of the choice of b' since the output of $\mathsf{ChalO}^{ror}(i_{b'}, \mathsf{m}, \mathsf{bsn}; 1)$ is independent of b'. Namely, we have $\Pr\left[\mathsf{Exp}^{\text{ror-Anon-1}}_{\mathsf{DAA},\mathcal{A}_2}(\lambda) = 1\right] = 1/2$.

Piecing everything together, we can compute $\mathcal{A}_2$'s advantage as follows.

$$
\begin{aligned}
\mathsf{Adv}^{\text{ror-anon}}_{\mathsf{DAA},\mathcal{A}_2}(\lambda) &= \left|\Pr\left[\mathsf{Exp}^{\text{ror-Anon-1}}_{\mathsf{DAA},\mathcal{A}_2}(\lambda) = 1\right] - \Pr\left[\mathsf{Exp}^{\text{ror-Anon-0}}_{\mathsf{DAA},\mathcal{A}_2}(\lambda) = 1\right]\right| \\
&= \left|\frac{1}{2} - \Pr\left[\mathsf{Exp}^{\text{ror-Anon-0}}_{\mathsf{DAA},\mathcal{A}_2}(\lambda) = 1\right]\right| \\
&= \Big|\frac{1}{2} - \Big(\frac{1}{2}\Pr\left[\mathsf{Exp}^{\text{lor-Anon-0}}_{\mathsf{DAA},\mathcal{A}_1}(\lambda) = 1\right] \\
&\qquad + \frac{1}{2}\Pr\left[\mathsf{Exp}^{\text{lor-Anon-1}}_{\mathsf{DAA},\mathcal{A}_1}(\lambda) = 0\right] - \mathsf{negl}(\lambda)\Big)\Big| \\
&= \Big|\frac{1}{2} - \Big(\frac{1}{2}\Pr\left[\mathsf{Exp}^{\text{lor-Anon-0}}_{\mathsf{DAA},\mathcal{A}_1}(\lambda) = 1\right] \\
&\qquad + \frac{1}{2}\Big(1 - \Pr\left[\mathsf{Exp}^{\text{lor-Anon-1}}_{\mathsf{DAA},\mathcal{A}_1}(\lambda) = 1\right]\Big)\Big)\Big| - \mathsf{negl}(\lambda) \\
&= \Big|\frac{1}{2}\Big(\Pr\left[\mathsf{Exp}^{\text{lor-Anon-1}}_{\mathsf{DAA},\mathcal{A}_1}(\lambda) = 1\right] \\
&\qquad - \Pr\left[\mathsf{Exp}^{\text{lor-Anon-0}}_{\mathsf{DAA},\mathcal{A}_1}(\lambda) = 1\right]\Big)\Big| - \mathsf{negl}(\lambda) \\
&= \frac{1}{2}\mathsf{Adv}^{\text{lor-anon}}_{\mathsf{DAA},\mathcal{A}_1}(\lambda) - \mathsf{negl}(\lambda).
\end{aligned}
$$

This completes the proof. □

□

A.2 CONCURRENT ORACLE DEFINITIONS

It is crucial that oracles simulating a DAA protocol to an adversary allow him the same concurrent access as he would have in real life. For example,

an adversary in control of two host machines might begin a join session using one of its honest TPMs whilst using the other to commence a signing session. This example demonstrates that oracles simulating part of an interactive protocol (joining or signing) can be interleaved with other oracles by the adversary. In other words, he should be allowed to make a query that corresponds to the first move of an interactive protocol and receive the corresponding reply. Then, he should be able to query an entirely different oracle with no obligation to complete the protocol he started with the first. Only oracles defined in this way capture, with sufficient granularity, the true nature of a real life adversary's power in the DAA setting. In the main body of this work Figure 3.2 presents simplified, non-concurrent versions of these oracles. We choose this presentation structure so as to provide intuition on the function of these oracles. However, we reiterate the important considerations one must bare in mind when referring to the simplified oracles of Figure 3.2. In particular, one must allow an adversary to interleave calls to interactive oracles so that he is not obliged to finish an interaction with one before beginning one with another. Note that the full definitions in Figure A.3 cover this concurrency explicitly.

Of equal importance is that concurrent adversarial access is the default setting in the UC framework used to define DAA security in [CDL16]. Thus, in order that our property-based definition implies the same level of security, we must define oracles that give the adversary equivalent power. We follow the standard notation for defining such oracles as used in group signatures [BMW03], [BCC+16]. The algorithms comprising a protocol are modelled as interactive Turing machines (ITIs) which take as input a protocol message M_{in} and an internal state state and output a message M_{out}, a new internal state state', and a decision $\mathsf{dec} \in \{\mathsf{cont}, \mathsf{accept}, \mathsf{reject}\}$. We cut the protocol into pieces where each 'piece' is defined by the receiving of a message and then the sending of a response message. We say that an ITI (algorithm) run as part of a protocol returns cont if the protocol is still going on, reject if some check failed or an invalid input was received, and accept if everything runs as expected and there is no next step for that ITI in the protocol.

Figure A.3 provides formal definition for these oracles.

InitO–P(i)
- If $i \notin \mathsf{HP}$, then Return $\perp$.
- $\mathsf{CRE}[i] := \perp$, $\mathsf{dec}^i_{\mathsf{JoinTPM}} := \mathsf{cont}$, $\mathsf{dec}^i_{\mathsf{Issue}} := \mathsf{cont}$.
- $\mathsf{st}^i_{\mathsf{JoinTPM}} := (\mathsf{ESK}[i])$
- $\mathsf{st}^i_{\mathsf{JoinH}} := (\mathsf{EPK}[i], \mathsf{ipk})$
- $\mathsf{st}^i_{\mathsf{Issue}} := (\mathsf{isk}, \mathsf{EPK}[i], \mathsf{ML})$
- $(\mathsf{st}^i_{\mathsf{JoinH}}, \mathsf{M}_{\mathsf{Issue}}, \mathsf{dec}^i_{\mathsf{JoinH}}) \leftarrow \mathsf{JoinH}(\mathsf{st}^i_{\mathsf{JoinH}}; \perp)$.
- While $(\mathsf{dec}^i_{\mathsf{JoinTPM}} \wedge \mathsf{dec}^i_{\mathsf{JoinH}} \wedge \mathsf{dec}^i_{\mathsf{Issue}} = \mathsf{cont})$ do
  - $(\mathsf{st}^i_{\mathsf{Issue}}, \mathsf{M}_{\mathsf{JoinH}}, \mathsf{dec}^i_{\mathsf{Issue}}) \leftarrow \mathsf{Issue}(\mathsf{st}^i_{\mathsf{Issue}}; \mathsf{M}_{\mathsf{Issue}})$.
  - $(\mathsf{st}^i_{\mathsf{JoinH}}, \mathsf{M}_{\mathsf{JoinTPM}}, \mathsf{dec}^i_{\mathsf{JoinH}}) \leftarrow \mathsf{JoinH}(\mathsf{st}^i_{\mathsf{JoinH}}; \mathsf{M}_{\mathsf{JoinH}})$.
  - $(\mathsf{st}^i_{\mathsf{JoinTPM}}, \mathsf{M}_{\mathsf{JoinH}}, \mathsf{dec}^i_{\mathsf{JoinTPM}}) \leftarrow \mathsf{JoinTPM}(\mathsf{st}^i_{\mathsf{JoinTPM}}; M_{\mathsf{JoinTPM}})$.
  - $(\mathsf{st}^i_{\mathsf{JoinH}}, \mathsf{M}_{\mathsf{Issue}}, \mathsf{dec}^i_{\mathsf{JoinH}}) \leftarrow \mathsf{JoinH}(\mathsf{st}^i_{\mathsf{JoinH}}; \mathsf{M}_{\mathsf{JoinH}})$.
- If $(\mathsf{dec}^i_{\mathsf{JoinTPM}} \vee \mathsf{dec}^i_{\mathsf{JoinH}} = \mathsf{accept})$:
  - If $\mathsf{dec}^i_{\mathsf{JoinH}} = \mathsf{accept}$, then $\mathsf{CRE}[i] := \mathsf{st}^i_{\mathsf{JoinH}}$.
  - If $\mathsf{dec}^i_{\mathsf{JoinTPM}} = \mathsf{accept}$, then $\mathsf{HSK}[i] := \mathsf{st}^i_{\mathsf{JoinTPM}}$.
- Return 1.

JoinO(i, M_{in}; party)

– If party $= \mathcal{P}$:
- If $i \notin \mathsf{HP}$ or $\mathsf{CRE}[i] \neq \perp$, then Return $\perp$.
- If not defined, set $\mathsf{dec}^i_{\mathsf{JoinH}} = \mathsf{dec}^i_{\mathsf{JoinTPM}} = \mathsf{cont}$.
- If $\mathsf{dec}^i_{\mathsf{JoinH}} \neq \mathsf{cont}$, then Return $\perp$.
- If $\mathsf{st}^i_{\mathsf{JoinH}}$ is undefined
  - $\mathsf{st}^i_{\mathsf{JoinH}} := (\mathsf{EPK}[i], \mathsf{ipk})$.
- If $\mathsf{st}^i_{\mathsf{JoinTPM}}$ is undefined:
  - $\mathsf{st}^i_{\mathsf{JoinTPM}} := (\mathsf{ESK}[i])$.
- $(\mathsf{st}^i_{\mathsf{JoinH}}, \mathsf{M}_{\mathsf{JoinTPM}}, \mathsf{dec}^i_{\mathsf{JoinH}}) \leftarrow \mathsf{JoinH}(\mathsf{st}^i_{\mathsf{JoinH}}; \mathsf{M}_{in})$.
- While $(\mathsf{dec}^i_{\mathsf{JoinTPM}} \wedge \mathsf{dec}^i_{\mathsf{JoinH}} = \mathsf{cont})$ do
  - $(\mathsf{st}^i_{\mathsf{JoinTPM}}, \mathsf{M}_{\mathsf{JoinH}}, \mathsf{dec}^i_{\mathsf{JoinTPM}}) \leftarrow \mathsf{JoinTPM}(\mathsf{st}^i_{\mathsf{JoinTPM}}; \mathsf{M}_{\mathsf{JoinTPM}})$
  - $(\mathsf{st}^i_{\mathsf{JoinH}}, \mathsf{M}_{\mathsf{Issue}}, \mathsf{dec}^i_{\mathsf{JoinH}}) \leftarrow \mathsf{JoinH}(\mathsf{st}^i_{\mathsf{JoinH}}; \mathsf{M}_{\mathsf{JoinH}})$.
- If $(\mathsf{dec}^i_{\mathsf{JoinTPM}} \vee \mathsf{dec}^i_{\mathsf{JoinH}} = \mathsf{accept})$:
  - If $\mathsf{dec}^i_{\mathsf{JoinTPM}} = \mathsf{accept}$, then $\mathsf{HSK}[i] := \mathsf{st}^i_{\mathsf{JoinTPM}}$.
  - If $\mathsf{dec}^i_{\mathsf{JoinH}} = \mathsf{accept}$, then $\mathsf{CRE}[i] := \mathsf{st}^i_{\mathsf{JoinH}}$.
- Return $(\mathsf{M}_{\mathsf{Issue}}, \mathsf{dec}^i_{\mathsf{JoinH}})$.

– If party $= \mathcal{M}$:
- If $i \notin \mathsf{HT}$, then Return $\perp$.
- If $\mathsf{dec}^i_{\mathsf{JoinTPM}}$ is undefined, set $\mathsf{dec}^i_{\mathsf{JoinTPM}} := \mathsf{cont}$.
- If $\mathsf{dec}^i_{\mathsf{JoinTPM}} \neq \mathsf{cont}$, then Return $\perp$.
- If $\mathsf{st}^i_{\mathsf{JoinTPM}}$ is undefined:
  - $\mathsf{st}^i_{\mathsf{JoinTPM}} := (\mathsf{ESK}[i])$.
- $(\mathsf{st}^i_{\mathsf{JoinTPM}}, \mathsf{M}_{\mathsf{JoinH}}, \mathsf{dec}^i_{\mathsf{JoinTPM}}) \leftarrow \mathsf{JoinTPM}(\mathsf{st}^i_{\mathsf{JoinTPM}}; \mathsf{M}_{in})$.
- If $\mathsf{dec}^i_{\mathsf{JoinTPM}} = \mathsf{accept}$, then
  - $\mathsf{HSK}[i] := \mathsf{st}^i_{\mathsf{JoinTPM}}$.
- Return $(\mathsf{M}_{\mathsf{JoinH}}, \mathsf{dec}^i_{\mathsf{JoinTPM}})$.

IssueO(i, M_{in})
- If $i \notin \mathsf{HT} \cup \mathsf{CP}$, then Return $\perp$.
- If $\mathsf{dec}^i_{\mathsf{Issue}}$ is undefined, set $\mathsf{dec}^i_{\mathsf{Issue}} = \mathsf{cont}$.
- If $\mathsf{dec}^i_{\mathsf{Issue}} \neq \mathsf{cont}$, then Return $\perp$.
- If $\mathsf{st}^i_{\mathsf{Issue}}$ is undefined
  - $\mathsf{st}^i_{\mathsf{Issue}} := (\mathsf{isk}, \mathsf{EPK}[i], \mathsf{ML})$.
- $(\mathsf{st}^i_{\mathsf{Issue}}, \mathsf{M}_{\mathsf{JoinH}}, \mathsf{dec}^i_{\mathsf{Issue}}) \leftarrow \mathsf{Issue}(\mathsf{st}^i_{\mathsf{Issue}}; \mathsf{M}_{in})$.
- If $i \in \mathsf{CP}$, then $\mathsf{TS}[i] \leftarrow \mathsf{TS}[i] \cup \{(\mathsf{M}_{in}, \mathsf{M}_{\mathsf{JoinH}})\}$.
- If $\mathsf{dec}^i_{\mathsf{Issue}} = \mathsf{accept}$, then
  - $\mathsf{CRE}[i] := \mathsf{M}_{\mathsf{JoinH}}$.
- Return $(\mathsf{M}_{\mathsf{JoinH}}, \mathsf{dec}^i_{\mathsf{Issue}})$.

IdentifyO($i, \sigma, \mathsf{m}, \mathsf{bsn}$)
- If $i \notin \mathsf{HT} \cup \mathsf{HP}$, then Return $\perp$.
- $b \leftarrow \mathsf{Identify}(\sigma, \mathsf{m}, \mathsf{bsn}, \mathsf{HSK}[i])$.
- Return b.

AddHonO(i; party)
- If $i \in \mathsf{HT} \cup \mathsf{HP} \cup \mathsf{CT} \cup \mathsf{CP}$, then Return $\perp$.
- If party $= \mathcal{M}$, then $\mathsf{HT} \leftarrow \mathsf{HT} \cup \{i\}$.
- If party $= \mathcal{P}$, then $\mathsf{HP} \leftarrow \mathsf{HP} \cup \{i\}$.
- $(\mathsf{esk}, \mathsf{epk}) \leftarrow \mathsf{TPM.Kg}(\mathsf{param})$.
- $(\mathsf{ESK}[i], \mathsf{EPK}[i]) := (\mathsf{esk}, \mathsf{epk})$.
- Return $\mathsf{EPK}[i]$.

AddCrptO(i, epk; party)
- If $i \in \mathsf{HT} \cup \mathsf{HP} \cup \mathsf{CT} \cup \mathsf{CP}$, then Return $\perp$.
- If party $= \mathcal{M}$ Then $\mathsf{CT} \leftarrow \mathsf{CT} \cup \{i\}$.
- If party $= \mathcal{P}$ Then $\mathsf{CP} \leftarrow \mathsf{CP} \cup \{i\}$.
- $(\mathsf{ESK}[i], \mathsf{EPK}[i]) := (\perp, \mathsf{epk})$.
- Return accept.

SignO($i, \mathsf{m}, \mathsf{bsn}, \mathsf{M}_{in}$; party)

– If party $= \mathcal{P}$:
- If $i \notin \mathsf{HP}$ or $\mathsf{CRE}[i] = \perp$ or $\mathsf{HSK}[i] = \perp$, then Return $\perp$.
- $\mathsf{st}^i_{\mathsf{SignH}} := (\mathsf{CRE}[i], \mathsf{m}, \mathsf{bsn})$.
- $\mathsf{st}^i_{\mathsf{SignTPM}} := (\mathsf{HSK}[i], \mathsf{m}, \mathsf{bsn})$.
- $\mathsf{dec}^i_{\mathsf{SignH}} := \mathsf{cont}$.
- $\mathsf{dec}^i_{\mathsf{SignTPM}} := \mathsf{cont}$.
- $\mathsf{M}_{\mathsf{SignH}} := \perp$.
- While $(\mathsf{dec}^i_{\mathsf{SignH}} = \mathsf{cont})$ do
  - $(\mathsf{st}^i_{\mathsf{SignH}}, \mathsf{M}_{\mathsf{SignTPM}}, \mathsf{dec}^i_{\mathsf{SignH}}) \leftarrow \mathsf{SignH}(\mathsf{st}^i_{\mathsf{SignH}}; \mathsf{M}_{\mathsf{SignH}})$.
  - $(\mathsf{st}^i_{\mathsf{SignTPM}}, \mathsf{M}_{\mathsf{SignH}}, \mathsf{dec}^i_{\mathsf{SignTPM}}) \leftarrow \mathsf{SignTPM}(\mathsf{st}^i_{\mathsf{SignTPM}}; \mathsf{M}_{\mathsf{SignTPM}})$.
- If $\mathsf{dec}^i_{\mathsf{SignH}} = \mathsf{accept}$
  - $\sigma \leftarrow \mathsf{st}^i_{\mathsf{SignH}}$.
  - $\mathsf{SL} \leftarrow \mathsf{SL} \cup \{i, \mathsf{m}, \mathsf{bsn}, \sigma\}$.
- Return σ.

– If party $= \mathcal{M}$:
- If $i \notin \mathsf{HT}$ or $\mathsf{HSK}[i] = \perp$, then Return $\perp$.
- If undefined, $\mathsf{dec}^i_{\mathsf{SignTPM}} := \mathsf{cont}$.
- If $\mathsf{dec}^i_{\mathsf{SignTPM}} \neq \mathsf{cont}$, then Return $\perp$.
- If $\mathsf{st}^i_{\mathsf{SignTPM}}$ is undefined
  - $\mathsf{st}^i_{\mathsf{SignTPM}} := (\mathsf{HSK}[i], \mathsf{m}, \mathsf{bsn})$.
- $(\mathsf{st}^i_{\mathsf{SignTPM}}, \mathsf{M}_{\mathsf{SignH}}, \mathsf{dec}^i_{\mathsf{SignTPM}}) \leftarrow \mathsf{SignTPM}(\mathsf{st}^i_{\mathsf{SignTPM}}; \mathsf{M}_{in})$.
- If $\mathsf{dec}^i_{\mathsf{SignTPM}} = \mathsf{accept}$
  - $\mathsf{SL} \leftarrow \mathsf{SL} \cup \{i, \mathsf{m}, \mathsf{bsn}, \perp\}$.
- Return $(\mathsf{M}_{\mathsf{SignH}}, \mathsf{dec}^i_{\mathsf{SignTPM}})$.

ChalO($i_0, i_1, \mathsf{bsn}, \mathsf{m}; b$)
- If $i'_b \notin \mathsf{HP}$ or $\mathsf{CRE}[i'_b] = \perp$ or $\mathsf{HSK}[i'_b] = \perp$ for $b' \in \{0, 1\}$ then Return $\perp$.
- If $|\mathsf{CL}| \geq 1$ and $(i_0, i_1, *) \notin \mathsf{CL}$, then Return $\perp$.
- $\mathsf{st}^{i_b}_{\mathsf{SignH}} := (\mathsf{CRE}[i_b], \mathsf{m}, \mathsf{bsn})$.
- $\mathsf{st}^{i_b}_{\mathsf{SignTPM}} := (\mathsf{HSK}[i_b], \mathsf{m}, \mathsf{bsn})$.
- $\mathsf{dec}^{i_b}_{\mathsf{SignTPM}}, \mathsf{dec}^{i_b}_{\mathsf{SignH}} := \mathsf{cont}$.
- $\mathsf{M}_{\mathsf{SignH}} := (\mathsf{CRE}[i_b], \mathsf{m}, \mathsf{bsn})$.
- While $(\mathsf{dec}^{i_b}_{\mathsf{SignTPM}} \wedge \mathsf{dec}^{i_b}_{\mathsf{SignTPM}} = \mathsf{cont})$ do
  - $(\mathsf{st}^{i_b}_{\mathsf{SignH}}, \mathsf{M}_{\mathsf{SignTPM}}, \mathsf{dec}^{i_b}_{\mathsf{SignH}}) \leftarrow \mathsf{SignH}(\mathsf{st}^{i_b}_{\mathsf{SignH}}; \mathsf{M}_{\mathsf{SignH}})$.
  - $(\mathsf{st}^{i_b}_{\mathsf{SignTPM}}, \mathsf{M}_{\mathsf{SignH}}, \mathsf{dec}^{i_b}_{\mathsf{SignTPM}}) \leftarrow \mathsf{SignTPM}(\mathsf{st}^{i_b}_{\mathsf{SignTPM}}; \mathsf{M}_{\mathsf{SignTPM}})$.
- If $\mathsf{dec}^{i_b}_{\mathsf{SignH}} = \mathsf{accept}$
  - $\sigma \leftarrow \mathsf{st}^{i_b}_{\mathsf{SignH}}$.
- $\mathsf{CL} \leftarrow \mathsf{CL} \cup \{(i_0, i_1, \mathsf{bsn})\}$.
- Return σ.

FIGURE A.3: Details of the oracles used in the security games.

BIBLIOGRAPHY

[ABD16] Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. pages 153–178, 2016.

[ABG+21] Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, Tjerand Silde, and Thor Tunge. Lattice-based proof of shuffle and applications to electronic voting. pages 227–251, 2021.

[ABGS22] Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, and Tjerand Silde. Verifiable mix-nets and distributed decryption for voting from lattice-based assumptions. Cryptology ePrint Archive, Report 2022/422, 2022. `https://eprint.iacr.org/2022/422`.

[ABGS23] Diego F. Aranha, Carsten Baum, Kristian Gjøsteen, and Tjerand Silde. Verifiable mix-nets and distributed decryption for voting from lattice-based assumptions. ACM CCS, 2023. `https://eprint.iacr.org/2022/422`.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). pages 99–108, 1996.

[AKSY22] Shweta Agrawal, Elena Kirshanova, Damien Stehlé, and Anshu Yadav. Practical, round-optimal lattice-based blind signatures. pages 39–53, 2022.

[APS15] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[BBC+18] Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. pages 669–699, 2018.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, and Jens Groth. Foundations of fully dynamic group signatures. pages 117–136, 2016.

[BCL08] Ernie Brickell, Liqun Chen, and Jiangtao Li. Simplified security notions of direct anonymous attestation and a concrete scheme from pairings. Cryptology ePrint Archive, Report 2008/104, 2008. https://eprint.iacr.org/2008/104.

[BDJR97] Mihir Bellare, Anand Desai, Eric Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. pages 394–403, 1997.

[BDL+18] Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. More efficient commitments from structured lattice assumptions. pages 368–385, 2018.

[BE17] Rachid El Bansarkhani and Ali El Kaafarani. Direct anonymous attestation from lattices. *IACR Cryptol. ePrint Arch.*, page 1022, 2017.

[Bel20] Mihir Bellare. Lectures on nizks: A concrete security treatment. Lecture Notes, 2020. Available at https://cseweb.ucsd.edu/ mihir/cse208-Wi20/main.pdf.

[BFG+11] D. Bernhard, G. Fuchsbauer, E. Ghadafi, N.P. Smart, and B. Warinschi. Anonymous attestation with user-controlled linkability. Cryptology ePrint Archive, Report 2011/658, 2011. https://eprint.iacr.org/2011/658.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). pages 103–112, 1988.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. pages 309–325, 2012.

[BHM20] Xavier Boyen, Thomas Haines, and Johannes Müller. A verifiable and practical lattice-based decryption mix net with external auditing. pages 336–356, 2020.

[BIP+22] Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. FINAL: Faster FHE instantiated with NTRU and LWE. pages 188–215, 2022.

[BL11] E. Brickell and J. Li. Enhanced privacy ID from bilinear pairing for hardware authentication and attestation. *IJIPSI*, 1(1):3–33, 2011.

[BLNS23] Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Alessandro Sorniotti. A framework for practical anonymous credentials from lattices. pages 384–417, 2023.

[BLS19] Jonathan Bootle, Vadim Lyubashevsky, and Gregor Seiler. Algebraic techniques for short(er) exact lattice-based zero-knowledge proofs. pages 176–202, 2019.

[BMW03] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. pages 614–629, 2003.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. pages 62–73, 1993.

[BS04] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. pages 168–177, 2004.

[BS23] Katharina Boudgoust and Peter Scholl. Simple threshold (fully homomorphic) encryption from LWE with polynomial modulus. Cryptology ePrint Archive, Report 2023/016, 2023. `https://eprint.iacr.org/2023/016`.

[BSMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6):1084–1118, 1991.

[BSZ05] Mihir Bellare, Haixia Shi, and Chong Zhang. Foundations of group signatures: The case of dynamic groups. pages 136–153, 2005.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. pages 136–145, 2001.

[Can03] Ran Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003. `https://eprint.iacr.org/2003/239`.

[CCD+17] Jan Camenisch, Liqun Chen, Manu Drijvers, Anja Lehmann, David Novick, and Rainer Urian. One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation. pages 901–920, 2017.

[CDE+] J. Camenisch, M. Drijvers, A. Edgington, A. Lehmann, R. Lindemann, and R. Urian. FIDO ECDAA algorithm, implementation draft.

[CDL16] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Universally composable direct anonymous attestation. pages 234–264, 2016.

[CDL17] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation with subverted TPMs. pages 427–461, 2017.

[Che10] Liqun Chen. A DAA scheme requiring less TPM resources. Cryptology ePrint Archive, Report 2010/008, 2010. `https://eprint.iacr.org/2010/008`.

[CHK+06] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: Efficient periodic n-times anonymous authentication. pages 201–210, 2006.

[CHL05] Jan Camenisch, Susan Hohenberger, and Anna Lysyanskaya. Compact e-cash. pages 302–321, 2005.

[CJL16] Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for ntru problems and cryptanalysis of the ggh multilinear map without a low-level encoding of zero. *LMS Journal of Computation and Mathematics*, 19(A):255–266, 2016.

[CKLL19] Liqun Chen, Nada El Kassem, Anja Lehmann, and Vadim Lyubashevsky. A framework for efficient lattice-based DAA. In Liqun Chen, Chris J. Mitchell, Thanassis Giannetsos, and Daniele Sgandurra, editors, *Proceedings of the 1st ACM Workshop on Workshop on Cyber-Security Arms Race, CYSARM@CCS 2019, London, UK, November 15, 2019*, pages 23–34. ACM, 2019.

[CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. pages 78–96, 2006.

[CLNS17] J. Camenisch, A. Lehmann, G. Neven, and K. Samelin. UC-secure non-interactive public-key encryption. In *CSF, 2017*, pages 217–233, 2017.

[CMM19] Núria Costa, Ramiro Martínez, and Paz Morillo. Lattice-based proof of a shuffle. pages 330–346, 2019.

[CPS10] L. Chen, D. Page, and N. P. Smart. On the design and implementation of an efficient DAA scheme. In *CARDIS, 2010*, pages 223–237, 2010.

[CPS+20] Chitchanok Chuengsatiansup, Thomas Prest, Damien Stehlé, Alexandre Wallet, and Keita Xagawa. ModFalcon: Compact signatures based on module-NTRU lattices. pages 853–866, 2020.

[CSS+22] Siddhartha Chowdhury, Sayani Sinha, Animesh Singh, Shubham Mishra, Chandan Chaudhary, Sikhar Patranabis, Pratyay Mukherjee, Ayantika Chatterjee, and Debdeep Mukhopadhyay. Efficient threshold FHE with application to real-time systems. Cryptology ePrint Archive, Report 2022/1625, 2022. `https://eprint.iacr.org/2022/1625`.

[DDP06] Ivan Damgård, Kasper Dupont, and Michael Østergaard Pedersen. Unclonable group identification. pages 555–572, 2006.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

[dK22] Rafaël del Pino and Shuichi Katsumata. A new framework for more efficient round-optimal lattice-based (partially) blind signature via trapdoor sampling. pages 306–336, 2022.

[dLNS17] Rafaël del Pino, Vadim Lyubashevsky, Gregory Neven, and Gregor Seiler. Practical quantum-safe voting from lattices. pages 1565–1581, 2017.

[DTGW17] Jintai Ding, Tsuyoshi Takagi, Xinwei Gao, and Yuntao Wang. Ding Key Exchange. Technical report, National Institute of Standards and Technology, 2017. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions`.

[Dv21] Léo Ducas and Wessel P. J. van Woerden. NTRU fatigue: How stretched is overstretched? pages 3–32, 2021.

[ECE+18] Nada El Kassem, Liqun Chen, Rachid El Bansarkhani, Ali El Kaafarani, Jan Camenisch, and Patrick Hough. L-DAA: Lattice-based direct anonymous attestation. Cryptology ePrint Archive, Report 2018/401, 2018. https://eprint.iacr.org/2018/401.

[EE17] Rachid El Bansarkhani and Ali El Kaafarani. Direct anonymous attestation from lattices. Cryptology ePrint Archive, Report 2017/1022, 2017. https://eprint.iacr.org/2017/1022.

[Fis05] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. pages 152–168, 2005.

[FKMV12] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the Fiat-Shamir transform. pages 60–79, 2012.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. pages 186–194, 1987.

[FWK21a] Valeh Farzaliyev, Jan Willemson, and Jaan Kristjan Kaasik. Improved lattice-based mix-nets for electronic voting. Cryptology ePrint Archive, Report 2021/1499, 2021. https://eprint.iacr.org/2021/1499.

[FWK21b] Valeh Farzaliyev, Jan Willemson, and Jaan Kristjan Kaasik. Improved lattice-based mix-nets for electronic voting. In *Information Security and Cryptology – ICISC 2021*. Springer International Publishing, 2021.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. pages 169–178, 2009.

[GHL22] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. pages 458–487, 2022.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). pages 291–304, 1985.

[GMS87] Oded Goldreich, Yishay Mansour, and Michael Sipser. Interactive proof systems: Provers that never fail and random selection (extended abstract). pages 449–461, 1987.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. 7(1):1–32, December 1994.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. pages 197–206, 2008.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. pages 212–219, 1996.

[H20] Horizon2020 prometheus project. https://www.h2020prometheus.eu/related-projects/future-tpm.

[HMS21] Javier Herranz, Ramiro Martínez, and Manuel Sánchez. Shorter lattice-based zero-knowledge proofs for the correctness of a shuffle. In *International Conference on Financial Cryptography and Data Security*, pages 315–329. Springer, 2021.

[HPS98a] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *Third Algorithmic Number Theory Symposium (ANTS)*, volume 1423, pages 267–288, June 1998.

[HPS98b] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. Ntru: A ring-based public key cryptosystem. In *International Algorithmic Number Theory Symposium*, pages 267–288. Springer, 1998.

[HS22] Audhild Høgåsen and Tjerand Silde. Return codes from lattice assumptions. *E-VOTE-ID*, 2022.

[ISO09] ISO, International Organization for Standardization. ISO/IEC 11889: Information technology - Trusted platform module library, 2009.

[ISO13] ISO, International Organization for Standardization. ISO/IEC 20008-2: Information technology - Security techniques - Anonymous digital signatures - Part 2: Mechanisms using a group public key, 2013.

[ISO15] ISO, International Organization for Standardization. ISO/IEC 11889: Information technology - Trusted platform module library, 2015.

[Kat21] Shuichi Katsumata. A new simple technique to bootstrap various lattice zero-knowledge proofs to QROM secure NIZKs. pages 580–610, 2021.

[KCB+18] Nada El Kassem, Liqun Chen, Rachid El Bansarkhani, Ali El Kaafarani, Jan Camenisch, and Patrick Hough. L-DAA: lattice-based direct anonymous attestation. *IACR Cryptol. ePrint Arch.*, page 401, 2018.

[KCB+19] Nada El Kassem, Liqun Chen, Rachid El Bansarkhani, Ali El Kaafarani, Jan Camenisch, Patrick Hough, Paulo Martins, and Leonel Sousa. More efficient, provably-secure direct anonymous attestation from lattices. *Future Gener. Comput. Syst.*, 99:425–458, 2019.

[KF17] Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched NTRU parameters. pages 3–26, 2017.

[KLL+18] Vireshwar Kumar, He Li, Noah Luther, Pranav Asokan, Jung-Min "Jerry" Park, Kaigui Bian, Martin B. H. Weiss, and Taieb Znati. Direct anonymous attestation with efficient verifier-local revocation for subscription system. pages 567–574, 2018.

[LDK+20] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2020. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions`.

[LDW+13] Michael Z. Lee, Alan M. Dunn, Brent Waters, Emmett Witchel, and Jonathan Katz. Anon-Pass: Practical anonymous subscriptions. pages 319–333, 2013.

[LNP22] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Maxime Plançon. Lattice-based zero-knowledge proofs and applications: Shorter, simpler, and more general. pages 71–101, 2022.

[LNS21] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. Shorter lattice-based zero-knowledge proofs via one-time commitments. pages 215–241, 2021.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. pages 1–23, 2010.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptography*, 75(3):565–599, June 2015.

[LW20] Changmin Lee and Alexandre Wallet. Lattice analysis on MiNTRU problem. Cryptology ePrint Archive, Report 2020/230, 2020. `https://eprint.iacr.org/2020/230`.

[LWW04] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). pages 325–335, 2004.

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. pages 598–616, 2009.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. pages 738–755, 2012.

[MR09] Daniele Micciancio and Oded Regev. *Lattice-based Cryptography*, pages 147–191. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[NF05] Toru Nakanishi and Nobuo Funabiki. Verifier-local revocation group signature schemes with backward unlinkability from bilinear maps. pages 533–548, 2005.

[NISa] NIST pqc effort. https://csrc.nist.gov/projects/post-quantum-cryptography.

[NISb] NIST threshold call. `https://csrc.nist.gov/Projects/Threshold-Cryptography`. Accessed: 2024-01-27.

[PSFK15] J. Petit, F. Schaub, M. Feiri, and F. Kargl. Pseudonym schemes in vehicular networks: A survey. *IEEE Communications Surveys and Tutorials*, 17(1):228–255, 2015.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. pages 84–93, 2005.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. pages 124–134, 1994.

[SPL+17] Minhye Seo, Jong Hwan Park, Dong Hoon Lee, Suhri Kim, and Seung-Joon Lee. EMBLEM and R.EMBLEM. Technical report, National Institute of Standards and Technology, 2017. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions`.

[SS11] Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. pages 27–47, 2011.

[SS13] Damien Stehlé and Ron Steinfeld. Making NTRUEncrypt and NTRUSign as secure as standard worst-case problems over ideal lattices. Cryptology ePrint Archive, Report 2013/004, 2013. `https://eprint.iacr.org/2013/004`.

[Tru04] Trusted Computing Group. TPM main specification version 1.2, 2004.

[Tru14] Trusted Computing Group. TPM library specification, family "2.0", 2014.

[WCG+17] J. Whitefield, L. Chen, T. Giannetsos, S. Schneider, and H. Treharne. Privacy-enhanced capabilities for vanets using direct anonymous attestation. In *2017 IEEE, VNC, 2017*, pages 123–130, 2017.