Implementation and Extension of a Post-Quantum Anonymous Credential

Sherry Wang

Thesis submitted to the University of Ottawa
in partial fulfillment of the requirements for the
Master of Computer Science[1]

Department of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

[1]The M.Sc. program is a joint program with Carleton University, administered by the Ottawa-Carleton Institute of Mathematics and Statistics

Abstract

With the development of quantum computers and quantum algorithms, many of our classical public-key cryptographic systems will be compromised. To circumvent this, one approach is to take our classical cryptographic systems and design them on mathematically hard problems for which no known quantum algorithms can provide a speedup. Our work revolves around one such cryptographic system, the anonymous credential protocol.

In this thesis, we implement a post-quantum anonymous credential scheme, and extend it to include additional features like the disclosure of properties of attributes. We implement the protocol in Python, run a series of correctness tests, and then compare our work with implementations from the literature. We also outline a more complete picture of what adversarial models are needed for post-quantum security. The motivation behind our work is to build more usable, feature-intensive anonymous credential systems that are resistant to adversaries with quantum computing capabilities. We present our work as a step towards providing more easily accessible software for a larger audience to extend.

Acknowledgement

I am deeply grateful for my advisors, Dr. Anne Broadbent and Dr. Carlisle Adams. Over the past few years, they have been incredibly kind, supportive, patient, and generous with their time and mentorship. I could not have asked for better advisors, and will always remain thankful for their expertise, discussions, and guidance on this research journey. Thank you, Dr. Carlisle Adams and Dr. Anne Broadbent!

Along this research journey, I have met a host of wonderful people including some new and past lab mates. Thank you to Archishna, Bennett, Christine, Daniel, Denis, Haitian, Kieran, Linh, Pierre, Peng, Peter, Nagisa, Samia, Sohrab, and Upendra. The last stretch of my research was challenging but many thanks to Upendra for providing a helping hand. To the rest of my lab mates, it has been fun sharing a lab with all of you.

Finally, I am grateful for the support and love of my family, friends, and very beloved fluffy bundle of joy, Loki.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Anonymous credential systems are a privacy-enhancing cryptographic protocol that consists of three main categories of parties: certificate authorities, provers, and verifiers. A certificate authority is a trusted third party that can issue anonymous credentials for provers. The provers can then use their anonymous credentials to disclose attributes during transactions with the verifiers.

With the rise of quantum computing, attackers may one day have access to full-scale quantum computers, compromising current public-key cryptographic systems. Quantum algorithms such as Grover's algorithm and Shor's algorithm provide quadratic and exponential speed-up compared to the best-known classical algorithms.

Lattice-based cryptography is considered a potential candidate for post-quantum cryptographic systems. Currently, there are no known polynomial-time quantum algorithms that can solve lattice-based problems. The popularity of lattice-based cryptography is also reflected in the candidate schemes for the US National Institute of Standards and Technology (NIST) post-quantum cryptographic standardization call for proposals. Two of the three finalized federal information processing standards are based on the module lattice problem[1].

In this thesis, we focus on the implementation, extension, and discussion of the security of an anonymous credential scheme based on commit-transferable signatures. The commit-transferable signature is a signature scheme that has two unique properties: it is able to compute a signature directly on a commitment, and its signature can be transferred into a new signature where the underlying message in the commitment is still the same. This lends itself very well to anonymous credentials that use pseudonyms to preserve anonymity in interactions, since transferred signatures can be thought of as pseudonyms.

Anonymous credentials have many real-life applications. They can be considered as a digitized and privacy-enhancing equivalent of a driver's license, passport, or

[1] NIST announcement on three approved Federal Information Processing Standards, two of which are lattice-based.

another identification document.

A prover, Alice, could obtain an anonymous credential issued for her personal attributes by a trusted certificate authority such as a government agency. If Alice then wanted to purchase a bottle of sherry at a liquor store, she could present her anonymous credentials and show the appropriate verifiers (i.e. the liquor store employee) her birth date, without disclosing any other attributes.

There are extensions of anonymous credentials where provers can choose to disclose properties of their attributes well. An example scenario is if, continuing the liquor store scenario above, the liquor store employee wants to check that Alice's credential is not on a list of revoked licenses. This exact scenario would allow Alice to engage in an interaction with the employee to disclose this property of her attribute, without revealing the exact ID number of her credential. Another scenario is if Alice is being considered for a delivery job position. Her potential employer may need to verify whether she possesses a provisional driver's license or a full unrestricted license. Alternatively, perhaps Alice wants to prove that her province of residence is either Ontario or Quebec. In another instance, for promotional deals at her local grocery store, perhaps Alice may need to disclose that her age is below a certain threshold (e.g., her age is between 0 and 31 years of age). There are a wide range of uses for disclosing properties of attributes, and some of these examples will later be explored in our implementation. We show examples visually in Figure 1.1. In the figure, the credential has a list of attributes that have been committed beforehand. The owner of the credential can engage with the verifiers to disclose information about her attributes.

Anonymous credentials have been used in a diverse range of real-world applications, including products such as IBM's Idemix and Microsoft's UProve. More broadly, there has been a growing number of governments interested in implementing digital credential ecosystems. Although digital credentials and anonymous credentials were once different technologies, nowadays they are often used to mean the same protocol. In Canada, the federal government is in the planning stages of creating an anonymous credential ecosystem[2]. In the European Union, a pilot project was launched for their EU Digital Identity, an anonymous credential system for their citizens and residents[3].

Anonymous credentials are a cryptographic system that has a wide range of uses and provides increased privacy for its users. There is also a growing audience of people and governments around the world that are interested in adopting this technology.

[2]https://www.canada.ca/en/government/system/digital-government/digital-government-innovations/digital-credentials.html

[3]https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/europe-fit-digital-age/european-digital-identity_en

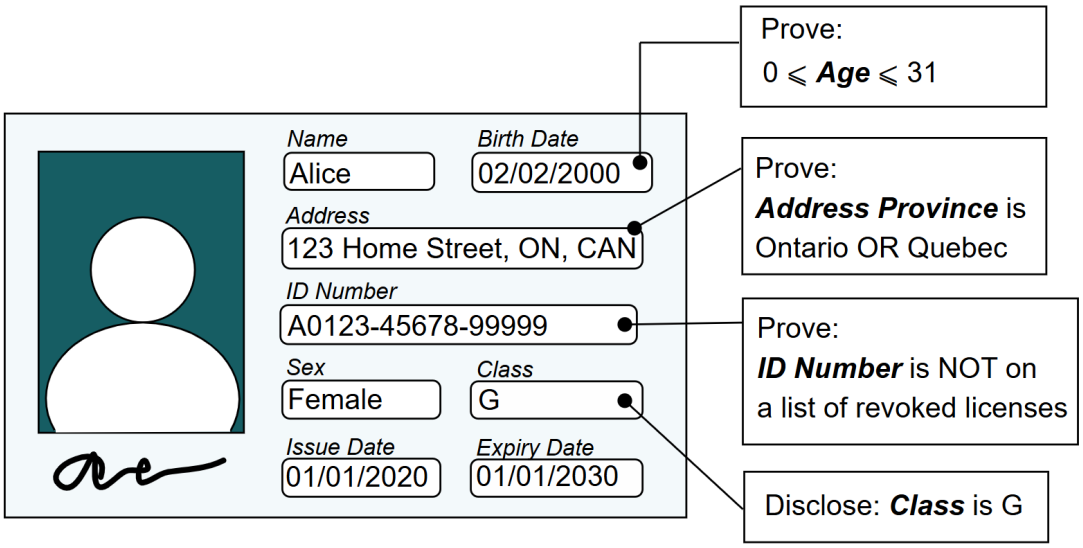

Figure 1.1: An example of an anonymous credential that is equivalent to a driver's license. Some example scenarios of the disclosure of property of attributes (NOT, OR, RANGE) and disclosure of attributes are shown. The class attribute refers to the Ontario driver's license class, where the G class signifies a full unrestricted license.

1.1 Contributions

The main purpose behind our work is to provide an implementation result that will further the development of more feature-complete post-quantum anonymous credentials.

Thus, our main contribution is the implementation of a post-quantum lattice-based anonymous credential scheme based on the work of [28]. We chose to use Python because it is a high-level language that would potentially be more accessible to a larger audience. The implementation can be found on our public GitHub repository:

https://github.com/wsherry/Post_Quantum_Anonymous_Credential.

In addition to the base implementation, we have extended the protocol to be able to disclose properties of attributes without revealing the attribute itself. These extensions include proving that an attribute is NOT a value, that an attribute belongs in a non-arbitrary RANGE of values, and that the attribute is either one value OR another. We evaluated the efficiency of the base implementation (i.e. the issuing of the credentials and the verification of the validity of the credentials) against existing work in the literature.

We also ran completeness tests that simulated some real-life scenarios with the disclosure of attribute properties functionality of our implementation. Some of these

real-life scenarios include disclosing information of a "driver license class" attribute, checking the range of the age attribute, and more.

Furthermore, we discuss the security of the scheme against classical and quantum adversaries. We do so by introducing a list of adversaries that post-quantum anonymous schemes should take into consideration.

Our work is meant as a step towards a feature-comprehensive implementation. Some of these additional properties and features are a next step toward furthering the development of highly practical and efficient post-quantum anonymous credentials. We have made our code public for the express purpose of encouraging other researchers and developers to extend upon and adapt our work.

1.2 Outline

The thesis is organized as follows.

In Chapter 2, the relevant preliminaries and mathematical notations are introduced. Some of the preliminary elements include abstract algebra basics and cryptographic primitives.

In Chapter 3, post-quantum anonymous credentials are introduced. We provide definitions, security notions, as well as a survey of relevant literature.

In Chapter 4, we discuss our implementation of the post-quantum anonymous credential protocol. This section includes information on the design of the protocol, additional features added, and specifications of the technologies used.

Next, in Chapter 5, we evaluate the implementation. Part of the evaluation includes discussion of adversarial security models and tests, the setup of the evaluation, results, as well as comparisons of our implementation against other implementations in the literature.

Finally, in Chapter 6, we discuss future work.

The thesis was written for a wide variety of readers, from those with extensive backgrounds of the relevant material to those without much background. Here are a few recommended ways of reading the thesis based on the reader's interests:

- Readers interested in learning lattice-based cryptography can review the preliminaries in Chapter 2.
- Readers with some basic lattice-based cryptography background, interested in the anonymous credential scheme should review the Mathematical Notations in Chapter 2.1 before skipping ahead to the start of Chapter 3.
- Readers interested in learning about post-quantum security can refer to Chapter 5.

- Readers interested in knowing how our anonymous credential scheme compares to other works can refer to Chapter 3.2, and also to Chapter 5 for a comparison with other implemented works.

Chapter 2

Preliminaries

In this chapter, we review the basic mathematical concepts and cryptographic background of the thesis. We first discuss the mathematical notations used throughout the thesis, followed by relevant information about post-quantum cryptography, lattices, gaussian distributions, zero-knowledge proofs, commitment schemes, and blind signatures.

2.1 Mathematical Notations

Let $\mathbb{Z}$ represent the set of integers, $\mathbb{R}$, the set of real numbers, and $\mathbb{N}$, the set of natural numbers. We denote a set of integers modulo q by $\mathbb{Z}_q$.

The two norms that are used in the thesis are as follows:

$$||x|| = \sqrt{\sum_i (x_i)^2} \text{ and } ||x||_\infty = \max_i |x_i|.$$

Our work heavily involves the use of cyclotomic rings. We represent them by:

$$\mathcal{R} = \mathbb{Z}[x]/(x^N + 1) \text{ and } \mathcal{R}_q = \mathbb{Z}_q[x]/(x^N + 1), \text{ where } N \in \mathbb{Z}.$$

By default, the ring $\mathcal{R}$ is over degree N. For another degree d, we specify the ring to be: $\mathcal{R}_d = \mathbb{Z}[X]/(X^d + 1)$ and $\mathcal{R}_{d,q} = \mathbb{Z}_q[X]/(X^d + 1)$.

We represent a set of polynomials with an infinity norm of less than or equal to β as:

$$S_\beta = \{a \in \mathcal{R} \mid ||a||_\infty \leq \beta\}, \text{ where } \beta \in \mathbb{R} \text{ and } \beta > 0.$$

The convention we use throughout the thesis is a bold lower case letter (e.g., $\mathbf{x}$) to represent a column vector with elements in a cyclotomic ring and bold upper case letter (e.g., $\mathbf{A}$) to represent a matrix with elements in a cyclotomic ring. Non-bolded lower case letters and upper case letters represent vectors and matrices in integer rings, or as otherwise stated.

2.2 Group Theory

Let us first discuss some basic elements of group theory.

Definition 2.2.1. ***Group*** *[24]. A group G consists of a set of elements and an operator * for combining two elements a,b ∈ G. The operator * must satisfy the following four laws at minimum:*

- *Closure Law: For $a, b \in G$, a*b ∈ G*
- *Identity Law: There is an element e ∈ G such that e*a = a*e = a*
- *Inverse Law: For every a ∈ G, there is a unique inverse $a^{-1} \in G$ satisfying $a * a^{-1} = a^{-1} * a = e$*
- *Associative Law: a*(b*c) = (a*b)*c for all a,b,c ∈ G*

As well, there can be the following law:

- *Commutative Law: a*b = b*a for all a,b ∈ G. If this property is satisfied, the group is also considered as a commutative or an abelian group.*

The group operator * can be addition, multiplication, or another operation, and the elements can be integers, functions, or polynomials, for example.

Definition 2.2.2. ***Ring*** *[24]. A ring $\mathcal{R}$ is a set that has two operations, addition and multiplication. Let us denote addition and multiplication by + and ·, respectively. They satisfy the following properties:*

- *The + operation forms an abelian group.*
- *The · operation is closed under multiplication. In other words, multiplying any two elements in $\mathcal{R}$ will give us another element in $\mathcal{R}$.*
- *The · operation satisfies the associative law and the identity law with the multiplicative identity $1 \in \mathcal{R}$.*
- *Both operations satisfy the distributive law with respect to addition, e.g., $a \cdot (b + c) = a \cdot b + a \cdot c$*
- *If the · operation also satisfies the commutative law, then $\mathcal{R}$ would be considered a commutative ring.*

In lattice-based cryptography, polynomial rings are often used.
For a ring $\mathcal{R}$, the polynomial ring can be denoted by:

$$\mathcal{R}[x] = \{a_0 + a_1x + ... + a_nx^n : n \geq 0 \text{ and } a_0, a_1, ..., a_n \in \mathcal{R}\}. \tag{2.2.1}$$

The polynomial quotient ring is a class of rings of the form:

$$\mathcal{R}[x]/f(x), \tag{2.2.2}$$

where $f(x)$ is some polynomial.

We are interested in a special kind of polynomial quotient ring, called the cyclotomic polynomial ring, which we will discuss in the next section.

2.3 Cyclotomic Polynomials

Cyclotomic polynomials are a special kind of polynomial with primitive roots of unity.

Definition 2.3.1. ***Roots of Unity.*** *For a positive integer n, the nth roots of unity are complex solutions to equation $x^n = 1$.*

The set of all nth roots of unity is given by $\{\zeta_n^k = e^{2k\pi i/n} | k = 0, ..., n-1\}$.

Geometrically, the nth roots of unity can be interpreted as the points spread evenly on the unit circle in the complex plane, where the x-axis represents the real part of the complex number, and the y-axis represents the imaginary part. For example, the 1st root of unity is 1, the second roots of unity are 1 and -1, and so on [30]. A visualization of the roots of unity can be seen in Figure 2.1.

Now, let us consider what it means for a root of unity to be primitive.

Definition 2.3.2. ***Primitive Roots of Unity.*** *The nth root of unity ζ is considered primitive if there are no roots of unity for any integer d smaller than n, i.e. $\zeta^n = 1$ and $\zeta^d \neq 1$ for $d < n$).*

Putting all the pieces together, we have the following definition:

Definition 2.3.3. ***Cyclotomic Polynomial [30].*** *The nth cyclotomic polynomial is a polynomial with nth primitive roots of unity, $\{\zeta_n^k = e^{2k\pi i/n} | k = 0, ..., n-1\}$. They take on the form:*

$$\Phi_n(x) = \prod_{\substack{0 \leq k < n \\ \gcd(k,n)=1}} \left(x - \zeta_n^k\right). \tag{2.3.1}$$

In special cases of the cyclotomic polynomial, when n is a power of 2, the nth cyclotomic polynomial can be represented as $\Phi_n(x) = x^m + 1$, where $m = 2^{k-1}$. We consider this special case for the underlying ring in the Module Learning with Errors (MLWE) problem which is discussed in Section 2.5.1.

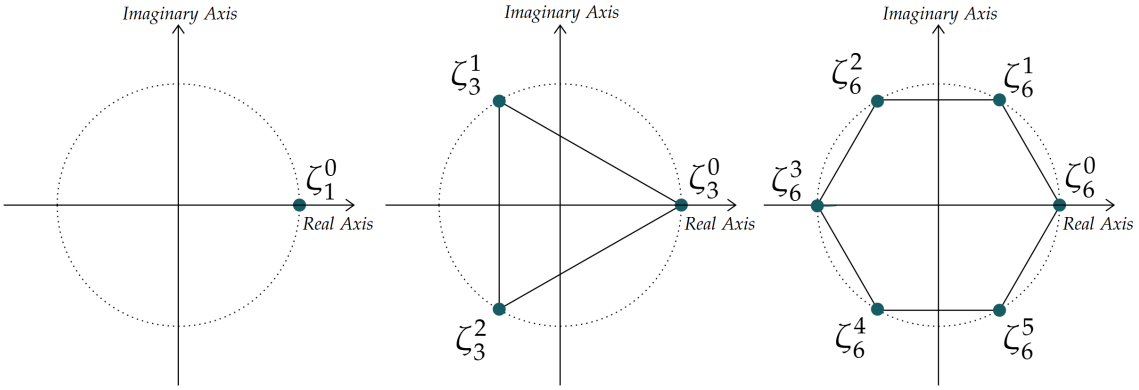

Figure 2.1: From left to right: the 1st, 3rd, and 6th roots of unity displayed on a unit circle.

For lattice-based cryptography, and specifically the MLWE problem, the quotient polynomial rings we are interested in look like the following, where N is a positive integer:

$$\mathcal{R} = \frac{\mathbb{Z}[x]}{(x^N + 1)}. \tag{2.3.2}$$

Similarly, the ring of cyclotomic polynomials (modulo q) is:

$$\mathcal{R}_q = \frac{(\mathbb{Z}/q\mathbb{Z})[x]}{(x^N + 1)}. \tag{2.3.3}$$

Every element in $\mathcal{R}$ and $\mathcal{R}_q$ has the form of $a_0 + a_1 x + ... + a_{N-1}x^{N-1}$ with the coefficients being in $\mathbb{Z}$ and $\mathbb{Z}/q\mathbb{Z}$, respectively. Every polynomial defined in $\mathcal{R}$ and $\mathcal{R}_q$ is of degree less than N.

The advantage of using the MLWE problem over the Learning with Errors (LWE) problem comes from the use of these cyclotomic polynomial rings. The cyclotomic polynomials have unique algebraic properties that provide extra structure. The extra structure is considered hard to exploit but improves the computational efficiency and lowers the key size [30].

Since we will eventually be using a sub-ring, we show how an element of a ring can be represented by its sub-ring. Let $\mathcal{R}_d = \mathbb{Z}[X]/(X^d + 1)$ and $\mathcal{R}_N = \mathbb{Z}[X]/(X^N + 1)$, where $N = dt$ for some integer t. We will show how one can map elements in $\mathcal{R}_d^t$ to $\mathcal{R}_N$. We can represent polynomial $a \in \mathcal{R}_N$ using $b \in \mathcal{R}_d^t$:

$$a = \sum_{i=0}^{t-1} b_i(X^t) \underset{\mathcal{R}_N}{\otimes} X^i, \tag{2.3.4}$$

where $\underset{\mathcal{R}_N}{\otimes}$ is the multiplication operator in $\mathcal{R}_N$ [33].

2.4 Post-quantum Cryptography Primer

Cryptography was first considered an art before becoming a mathematical discipline in the 1970s and 1980s [27]. In its earlier years, cryptography was primarily used by the military; however, it has now been widely adopted for use by the public today [27]. Cryptography consists of algorithms and protocols. Algorithms define single tasks, while protocols usually involve active participation between participants [1]. Some examples of algorithms and protocols include the exchanging of secret keys, user authentication, digital cash, and much more [27]. Post-quantum cryptography is an extension of cryptography. It is similarly comprised of classical algorithms and protocols with the property of also being robust against quantum attackers. Let us first explore some basics of cryptography before diving into post-quantum cryptography.

2.4.1 Goals of cryptography

This section closely follows [1].

The primary objective when designing cryptographic systems is to satisfy a set of cryptographic properties that include confidentiality, integrity, and authenticity.

Confidentiality is about keeping information secret, so that only the intended parties have access to the secret information. The secret information consists of the secret message itself as well as its metadata. To ensure confidentiality in today's society, there are two classes of encryption and decryption algorithms that are used: symmetric and asymmetric algorithms. Symmetric key systems (e.g., stream ciphers and block ciphers) use the same key for both encryption and decryption. Some common examples of symmetric key systems include the one-time pad and the widely adopted Advanced Encryption Standard (AES). On the other hand, asymmetric key systems, also known as public key systems (e.g., the RSA algorithm), use a key pair: a public key for encryption and a private key for decryption.

Integrity defines who can make changes to the data. Changes to the data can occur in both storage and in transit. However, only authorized parties should have access. The general technique for ensuring integrity is to create a tag, sometimes also known as a checksum or a fingerprint, along with the ciphertext. If the data is changed in any way, the tag should reflect an alteration with high probability, and the recipient would disregard the data.

Finally, authenticity is about knowing who the parties are in a transaction, in order to reduce impersonation attacks. Digital signatures are one such authentication technique that is commonly used. We will explore the topic of blind digital signatures in Section 2.9.

Ultimately, building and designing good cryptographic systems requires understanding and fulfilling these goals [1].

2.4.2 Cryptographic Strength

This section continues to follow [1].

It is also important to consider the strength of a cryptographic system. We can achieve good cryptographic strength by selecting relevant adversaries and using computational complexity theory, with respect to the cryptographic system and the design goals in question.

When defining adversaries, we need to take into consideration what their capabilities are as well as the resources that are available to them (i.e. time, computational power, and storage capacities). The attack types and resources of an adversary help to define an adversarial model. Adversaries are commonly classified as one of the following types: passive, active, man-in-the-middle (MITM), honest-but-curious, fail-stop, malicious, and byzantine [1]. The descriptions of each of these types can be seen in Table 2.1.

To measure cryptographic strength, we need to consider the resources and computational power that an adversary has access to. To do so, we use computational complexity theory. Computational complexity theory is often used to estimate the amount of effort needed to perform a task.

Big-O notation defines and categorizes resource usage into several classes, including linear, polynomial, and exponential, which are represented by $O(n)$, $O(poly(n))$, and $O(2^{poly(n)})$, respectively. These classes indicate the amount of resources used, in terms of time and storage based on the input size (e.g., a polynomial function would use a polynomial amount of resources in terms of the input size). Big-O notation provides an abstraction of measuring resources and computations, independent of actual hardware and operating systems [1].

We now consider some approaches for selecting adversaries to achieve good cryptographic strength.

One approach is to assume a computationally powerful adversary such as a polynomial-bounded adversary with an impressively large but finite amount of resources (i.e. lots of money, lots of employees, and state of the art supercomputers). In this case, cryptographic protocols and tools are selected based on the belief and assumption that an adversary cannot find the solution with their polynomial-bounded resources. This is also known as computational security.

Table 2.1: Adversary types and descriptions.

Adversary Type	**Description**
Passive	Can read, copy, and analyze data. The goal is to learn confidential information
Active	Can read, modify, delete, and inject data.
Man-in-the-middle (MITM)	Joins invisibly in a conversation between recipients. Can be passive and actively modifying conversation.
Honest-but-curious	Performs protocol correctly and attempts to learn confidential information.
Fail-stop	Can halt protocol at any time.
Malicious	Deviates from protocol entirely.
Byzantine	Coordinated malicious agents in a distributed system.

The second approach is to assume that there are no computational limitations on an adversary. This is also known as a computationally unbounded adversary, where the adversary is assumed to have unlimited computing power, time, and memory. This is also known as information-theoretic security. In this case, all possible solutions to the underlying mathematical problem are equally likely, so the adversary cannot determine which solution is the correct one. In other words, an adversary with infinite resources will have no better strategy than guessing the answer randomly. Zero-knowledge proof of knowledge (ZKPoK) protocols tend to use such a model [1]. On the other hand, Pedersen's commitment uses both models, and its hiding protocol is information-theoretically secure, while its binding protocol is only computationally secure. Both the ZKPoK and Pedersen's commitment are important components of our protocol.

While it is important to understand how to achieve cryptographic strength, it is also important to convince others that a particular cryptographic protocol is indeed strong.

For cryptographic protocols whose security is based on hard mathematical problems, security proofs can be used. A security proof is used to show that breaking the algorithm is equivalent to solving its underlying mathematical problem. In a security proof, we first select a theoretical model of the environment in which the protocol will

operate. Common models include the standard model and the random oracle model (ROM).

Definition 2.4.1. ***Standard model.*** *The standard model is a model centered on the adversary, where it is only limited by computational resources and time. Though resource-bounded, adversaries still have more resources than honest parties.*

The ROM model is an augmentation of the standard model.

Definition 2.4.2. ***Random Oracle Model [8].*** *The random oracle model uses a black box that, given a unique input, provides as output, a random response chosen uniformly. If a query is repeated, the same output is provided. All parties are given access to the same random oracle.*

Often, the standard model is more convincing and is more widely accepted.

The second step in a security proof is making any relevant assumptions relevant to the cryptographic protocol and its underlying mathematical problem. Assumptions are often made about the difficulty of performing certain mathematical computations, when the difficulty is not rigorously proved or known. For this reason, one might hear or read that a protocol is secure if the learning with errors (LWE) assumption is true. These assumptions are not known to be true with absolute certainty, and it is an active area of research to disprove or prove these assumptions.

Finally, the last step of the security proof involves showing a reduction from the mathematical computation to the cryptographic algorithm. A reduction is defined as an efficient transformation from one class of problems to another. The reduction essentially says that breaking the cryptographic problem is synonymous with breaking the underlying mathematical computation. Some examples of reduction proofs include showing that a given cipher is secure in the sense that it has indistinguishable encryptions under chosen plaintext attacks (IND-CPA) [1].

As we have seen, achieving good cryptographic strength and then proving it are important components of the overall security of a cryptographic scheme. To conclude this section, we share some common definitions that are relevant and often used when discussing security of a cryptographic protocol.

The security parameter denoted by λ, reflects the input size of the computational problem. Conventionally, the security parameter is in the unary format, 1^{λ} [30].

The negligible function is often used to describe the probability of success of adversarial attacks.

Definition 2.4.3. ***Neglible Function [30].*** *A function, $\mu : \mathbb{N} \to \mathbb{R}$ is considered neglible if for every positive integer c, there exists an integer N_c such that for all $n > N_c, |\mu(n)| < n^{-c}$.*

An encryption scheme is considered secure if any probabilistic polynomial-time adversary (PPT) has a negligible probability of success in breaking the scheme. Probabilistic refers to the attack being randomized, which often runs faster than deterministic algorithms.

2.4.3 Post-Quantum Elements

Post-quantum cryptography (PQC) is a term coined for cryptographic systems that are considered secure against attacks from quantum computers. Public-key cryptographic systems are especially vulnerable to full-scale quantum computers, as current classical cryptosystems based on integer factorization (e.g., RSA), discrete logarithm (e.g., Diffie-Helman key exchange), elliptic curves (e.g., ECDH), and others will be broken with full-scale quantum computers. Post-quantum cryptography is important to consider now due to "store now, decrypt later" adversaries. These types of adversaries store ciphertexts now, for decryption once full-scale quantum computers are made available. This might be used by large intelligence agencies for example, with some historical precedence of this kind of adversary in World War II and the Cold War. During this time, the United States' National Security Agency ran the VENONA Project which led to the decryption of Soviet Union messages sent years earlier [37]. It is also imperative that we make preparations for a future where quantum computers are widely used. Additionally, the roll-out of post-quantum cryptographic systems is likely to require a lot of time and latency. There are many steps in turning a mathematical construct into a functional system.

To understand why access to full-scale quantum computers would break public-key cryptographic systems, we turn our attention to a few quantum algorithms, one of which is Grover's search algorithm. Grover's search algorithm can be used to attack symmetric cryptographic systems [37]. It can find the unique input given a particular output value of a black box function and can do so in $O(\sqrt{n})$ evaluations [23]. This means that Grover's algorithm could brute force a 256-bit key in approximately 2^{128} operations. It offers a quadratic speedup compared to some of the best known classical algorithms. In practice, increasing the key length can help deter quantum attacks. Thus, many modern applications require AES-256 so that they are better protected against quantum computer attacks in the future [37]. However, for asymmetric cryptosystems, the threat is much greater with Shor's algorithm, which is able to efficiently factorize large integers [40]. Discrete logarithm problems that many of our current classical cryptosystems rely on would also be broken in polynomial time using Shor's algorithm [40].

IBM's current best quantum computer, Heron, has 156 qubits. For context, thousands to millions of fault-tolerant qubits would be necessary for running some of these quantum algorithms [37].

Thus, for post-quantum algorithms, we turn to a class of mathematical problems

that would not be affected by the exponential speed-up offered by quantum computers. In other words, there are currently no known quantum algorithms that can solve these mathematical problems more efficiently than classical algorithms.

The three most promising PQC algorithms in the NIST competition are lattice-based, code-based, and hash-based. A table of the families of PQC algorithms is shown in Table 2.2, which has been partially derived from [16, 37, 41].

Table 2.2: Comparison of different post-quantum cryptography algorithms.

Post-quantum cryptography algorithms	**Description [17]**	**Pros and Cons**	**Supported Services**	**NIST Candidate cryptosystems (3rd/4th round submissions)**
Lattice-based	A lattice of infinite points is generated from linear integer combinations of a basis. Messages are represented by vectors, and a public key in the form of a matrix is made available. The ciphertext is a multiplication of the message and the public key, with some addition of noise.	It has strong security guarantees with small key sizes which makes it suitable for a range of applications. However, there may be implementation challenges due to the underlying complex mathematical constructs.	Key transport, digital signatures	KYBER, FrodoKEM, SABER, DILITHIUM, FALCON NTRU Prime NTRU
Hash-based	Based on the security of hash functions. Hash functions convert a variable length message into a fixed-length block. Signatures combine a one-time signature with a Merkle tree. Hash functions are used to compute the nodes of the tree.	Hash functions are foundational in cryptography, and have been carefully evaluated. However, it often requires large key signatures or increased processing time. Also, digital signatures often are one-time use.	Digital signatures	SPHINCS+ PICNIC
Code-based	Algorithms are based on error-correcting codes. Often, the messages are randomized by adding random errors.	It has well understood security assumptions. However, its large key size may cause bandwidth and memory challenges.	Key transport	Classic McEliece, HQC, BIKE
Multivariate	These algorithms are based on multivariable polynomials over a finite field.	It has small signature sizes but is vulnerable against attacks.	Digital signatures	Rainbow, GeMSS

2.5 Lattices Primer

A lattice is a set of points in a m-dimensional space with periodic structure. Let $v_1, \ldots, v_n \in \mathcal{R}^m$ be a set of independent vectors. A lattice Λ is defined as a set of linear integer combinations of $v_1, \ldots, v_n$:

$$\Lambda = \{a_1 v_1 + a_2 v_2 + \cdots + a_n v_n | a_1, a_2, \ldots, a_n \in \mathbb{Z}\}. \tag{2.5.1}$$

An example of a two-dimensional lattice can be seen in Figure 2.2. There are two example bases in the figure, indicated by the straight and wavy arrows, respectively. The wavy arrow basis is a considerably better basis for lattice computations than the straight-line arrows. A better basis is determined by its vectors being closer to orthogonal with one another, and also having minimal lengths. As we consider high-dimensional lattices, this will become even more apparent.

Definition 2.5.1. ***Fundamental Domain.*** *Let Λ be an n-dimensional lattice with basis $v_1, ..., v_n$. The fundamental domain, also known as the fundamental parallelpiped of Λ, is a region defined as:*

$$F(v_1, ..., v_n) = \{t_1 v_1 + \cdots + t_n v_n \mid t_i \in [0, 1)\}. \tag{2.5.2}$$

The fundamental domain is the convex region surrounded by the given basis vectors and nearby lattice points. An example of the fundamental domain can be seen in Figure 2.2, which is represented by the shaded area.

An arbitrary vector $t \in \mathbb{R}^n$ in the span of a lattice Λ can be uniquely identified by a lattice vector v and a translation of a vector $w \in F$, where F is the fundamental domain of the lattice.

The coordinates of the basis vectors in a lattice are taken from an integer ring $\mathbb{Z}_q$, where q is often a prime or a power of two and denotes the modulus [37].

Definition 2.5.2. ***Dual Lattice.*** *Given a full-rank lattice L, its dual lattice Λ^* is:*

$$\Lambda^* = \{y \in span(\Lambda) \mid \forall x \in \Lambda, x \cdot y \in \mathbb{Z}\}. \tag{2.5.3}$$

The dual lattice is also a lattice, and they are used in a variety of places including in the smoothing parameter for discrete Gaussian distributions, as we shall see in the next subsection.

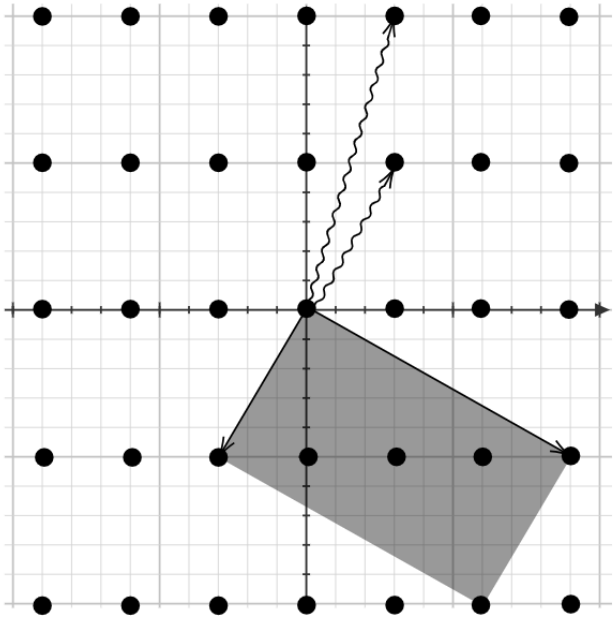

Figure 2.2: A 2-dimensional lattice, where the vectors are the basis of the lattice. The straight arrow pair is a basis, and the wavy arrow pair is another basis. The shaded area represents the fundamental domain.

2.5.1 Hard Lattice Problems

There are a number of computationally hard problems that rely on lattices. Two search problems include the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP). Cryptosystems are often based on the decision versions or approximate variants of these problems [30].

Definition 2.5.3. ***Shortest Vector Problem (SVP).*** *Given a lattice basis B, find the shortest non-zero vector in the lattice $\Lambda(B)$.*

Definition 2.5.4. ***Closest Vector Problem (CVP)*** *. Given a lattice basis B and a target vector t that is not in the lattice $\Lambda(B)$, find a vector in $\Lambda(B)$ that is closest to t.*

The average-case hardness problem is known as the short integer solution (SIS) problem.

Definition 2.5.5. ***Short Integer Solution (SIS).*** *Let $a_i \in \mathbb{Z}_q^n$ be an n-dimensional vector with entries that have been sampled uniformly from $\mathbb{Z}_q$. Let $A = [a_1, ..., a_m]$ be a matrix and β be a positive integer. Find a non-zero vector $x \in \mathbb{Z}^m$ such that $\|x\| \leq \beta$ and $Ax = 0$.*

A variant of the SIS problem that we are interested in is the module short integer solution (MSIS). In MSIS, instead of working over integers, we work with modules. Modules are a generalization of a vector space with a ring structure. The use of this algebraic structure in MSIS helps to decrease key sizes and is considered more efficient than the SIS problem.

Definition 2.5.6. ***Module Short Integer Solution Problem, M-SIS*** $_{q,\ell,m,\beta}$ ***[29].*** *Given a uniformly random matrix* $\mathbf{A} \in \mathcal{R}_q^{\ell \times m}$*, output vector* $\mathbf{z} \in \mathcal{R}^m$ *such that* $\mathbf{Az} = 0$ *and* $0 < \|\mathbf{z}\| \leq \beta$*.*

The learning with errors (LWE) is another computationally difficult problem for building lattice-based cryptosystems.

We first consider the starting point of such a problem. We take a set of basis vectors $v_1, \ldots, v_n \in \mathbb{Z}_q^m$ and multiply them by secret integer coefficients $s_1, \ldots, s_n \in \mathbb{Z}$ to get a value t. The problem is to find what the values for $s_1, \ldots, s_n$ are, if t and the basis vectors are given:

$$s_1 v_1 + s_2 v_2 + \cdots + s_n v_n = t. \tag{2.5.4}$$

In the form of a matrix, this problem can be thought of as $As = t$, where A is the matrix of all the basis vectors, and s denotes the secret set of integers $s_1, ..., s_n$. This problem is easily solvable since it is a system of equations. The LWE problem makes a slight modification to this problem, by introducing error terms. To do so, we assume that t is not a point on the lattice but a point close to a lattice point. Essentially, the LWE problem is trying to solve for s in the following equation:

$$As + e = t', \tag{2.5.5}$$

where e is some error vector and t' is a point close to a point on the lattice. Small values of e are important for ensuring that intended recipients should still be able to uniquely uncover the closest lattice point t from t' [37].

The formal definition is as follows for LWE:

Definition 2.5.7. ***Learning with Errors (LWE).*** *For a matrix* A*, a set of basis vectors* $v_i \in \mathbb{Z}_q^m$*, an unknown error vector of small integers* e*, some positive integer* q*, and a point* $t \in \mathbb{Z}_q^m$*, determine the secret coefficient,* s *such that* $As + e = t \bmod q$*.*

Choosing sufficiently large rank and dimension, n and m, respectively, helps deter and minimize the effectiveness of brute-force attacks.

We are interested in a variant of the LWE problem called the module learning with errors problem defined below. A module is an algebraic structure that is a generalization of rings and vector spaces. Module lattices are a generalization of arbitrary lattices and ideal lattices.

Definition 2.5.8. ***Decision module learning with error ($MLWE_{q,\ell,m,S_t}$) problem*** *[29]: Let $\mathcal{R}$ be a ring, and $\mathcal{S}_t$ denote the set of all polynomials of infinity norm less than t, i.e. $\mathcal{S}_t = \{a \in \mathcal{R} \mid \|a\|_\infty \leq t\}$. Let $\ell \in \mathbb{Z}$. For $\mathcal{R}$, $\mathbf{s} \leftarrow \mathcal{S}_t^\ell$, use $A_{q,\boldsymbol{s}}$ to denote the distribution of $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + \mathbf{e}) \in \mathcal{R}_q^\ell \times \mathcal{R}_q$, where $\mathbf{a} \leftarrow \mathcal{R}_q^\ell$ and $\mathbf{e} \leftarrow \mathcal{S}_t$. The goal is to distinguish m samples from either $A_{q,\mathbf{s}}$ or $U(\mathcal{R}_q^\ell, \mathcal{R}_q)$.*

2.6 Gaussian Distribution

We source information from [30, 36, 42] for this section.

The discrete Gaussian distribution behaves similarly to the continuous Gaussian distribution but provides discrete lattice support.

Gaussian functions are continuous functions defined in the following way:

$$f(x) = a \cdot \exp\left(-\frac{(x-c)^2}{2\sigma^2}\right), \tag{2.6.1}$$

where c denotes the mean, a defines the scaling factor, and σ defines the standard deviation. Visually, the variables c, a, and σ represent where the peak of the function is, the height of the function, and the width of the function, respectively. This can be seen in Figure 2.3.

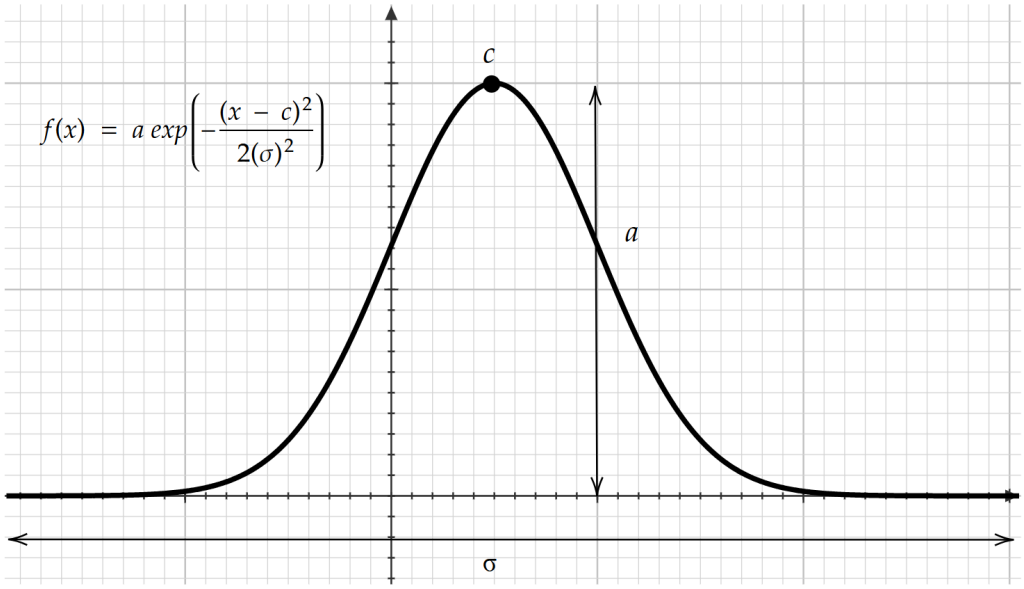

Figure 2.3: Continuous Gaussian function in $\mathbb{R}^2$.

Generalizing to a higher dimension, we can set a scale parameter $s = \sqrt{2\pi}\sigma$ and substitute it as σ to get an expression of the Gaussian measure:

$$\rho_{s,\mathbf{c}}(\mathbf{x}) = \exp\left(-\frac{-\pi\|\mathbf{x}-\mathbf{c}\|^2}{s^2}\right). \tag{2.6.2}$$

Normalizing the function with derivative $\rho(x)dx = s^n$ gives us an n-dimensional Gaussian distribution [42].

The total measure, involves integrating the measure over $\mathbb{R}^n$:

$$\int_{\mathbf{x}\in\mathbb{R}^n} \rho_{s,\mathbf{c}}(\mathbf{x})\,\mathrm{d}\mathbf{x} = s^n, \tag{2.6.3}$$

The Gaussian probability density function can be represented as:

$$D_{s,\mathbf{c}}(\mathbf{x}) = \frac{\rho_{s,\mathbf{c}}(\mathbf{x})}{s^n}. \tag{2.6.4}$$

Now, we can use the above equations to define the discrete Gaussian distribution in a similar way. Since a lattice Λ is a countable set, we can consider the $\mathbf{x}$ to be a non-continuous lattice vector [30]. The discrete Gaussian distribution over the lattice Λ for all lattice vectors that belong to Λ is the following:

$$D_{\Lambda,s,c}(\mathbf{x}) = \frac{\rho_{s,c}(\mathbf{x})}{\rho_{s,c}(\Lambda)}. \tag{2.6.5}$$

This assigns a value 0 to all $\mathbf{x} \notin \Lambda$ and the values above for every $\mathbf{x} \in \Lambda$ [42].

The discrete Gaussian distribution is used as a noise model for LWE. The Gaussian distribution is useful for smoothing a distribution over a space. Given a Gaussian distribution, $\rho_{s,c}(c)$ whose center is a lattice point, if random samples from this distribution are modulo the lattice's fundamental domain, the resulting samples will induce a fundamental domain distribution. The larger the s is, the closer the distribution is to uniform.

The usefulness of discrete Gaussian distributions is due to two of its key properties.

The first lemma proves that the discrete and continuous Gaussian distributions share similar characteristics when the scale of the discrete Gaussian is sufficiently large.

Lemma 2.6.1. ***Similar to Continuous Gaussian*** *[36]. Let $D_{\Lambda,s,c}$ be a discrete Gaussian distribution over an n-dimensional lattice Λ with arbitrary scale $s \geq 2\eta_\varepsilon(\Lambda)$ and center $c \in \mathbb{R}^n$. For $0 < \varepsilon < 1$, the following are satisfied:*

$$\|E_{x\sim D_{\Lambda,s,c}}[x-c]\|^2 \leq \left(\frac{\varepsilon}{1-\varepsilon}\right)^2 s^2 n,$$

$$E_{x\sim D_{\Lambda,s,c}}\left[\|x-c\|^2\right] \leq \left(\frac{1}{2\pi} + \frac{\varepsilon}{1-\varepsilon}\right)^2 s^2 n.$$

The first inequality of the lemma tells us that on average, the random samples of $D_{\Lambda,s,c}$ are close to the distribution center. The second inequality suggests that the discrete Gaussian has almost the same variance as a continuous Gaussian which has a variance of $\frac{ns^2}{2\pi}$.

Since a vector $t \in \mathbb{R}^n$ can be identified by a lattice vector v and a vector w in its fundamental domain, we can reduce any arbitrary vector in $\mathbb{R}^n$ to a vector within F by simply taking $w = t \bmod F$, giving rise to lemma 2.6.2 that one can sample uniformly in the fundamental domain from a discrete Gaussian distribution.

Lemma 2.6.2. ***Near Uniformity [36].*** *Let Λ be an n-dimensional lattice and $D_{s,c}$ be a Gaussian distribution with arbitrary scale $s \geq \eta_\varepsilon(\Lambda)$ and center $c \in \mathbb{R}^n$. The statistical distance between $D_{s,c} \bmod F$ and a uniform distribution $U(F)$ over the fundamental domain F is*

$$\Delta(D_{s,c} \mod F, U(F)) \leq \frac{\varepsilon}{2}.$$

This lemma motivates the definition of the smoothing parameter, which is the minimum Gaussian noise magnitude such that it blurs the lattice to almost a uniform distribution over $\mathbb{R}^n$ [30].

2.7 Zero-Knowledge Proofs

A zero-knowledge proof is a proof that reveals nothing other than its validity. This primitive was first introduced by Goldwasser, Micali, and Rackoff in 1985. The main motivation behind zero-knowledge proofs arose due to the question of how mutually distrustful parties could reveal parts of their secrets to each other [22]. For example, suppose that all users have their course grades available online in their password-protected student accounts. If Alice wanted to disclose to Bob the grade she received in her Cryptography 101 class, she could text or tell him her grade value. Bob, however, cannot actually verify that Alice is telling the truth. One approach Alice can take is to share her school account password with Bob so that he can verify that she had indeed sent the correct grade value. This is undesirable for Alice because she is also giving him access to view all of her grades and any accompanying sensitive data. The point of zero-knowledge proofs is thus to allow Alice to convince Bob that she has in fact revealed the correct information without having actually revealed more information. In this scenario, Alice might consider taking a screenshot of her course grade to send to Bob, with some relevant verifiable information (e.g., the domain of the student portal URL, name of account, design of the website, date and course title). We assume doctored screenshots are not permissible in this particular scenario.

Firstly, in a zero-knowledge proof system, the two parties are often denoted by a prover and a verifier. The prover provides a proof for a statement to convince the verifier of its validity.

In a proof system, there are two main tasks, firstly a proof is produced, and secondly, the proof is verified for its validity. These proof systems can be interactive and non-interactive. In an interactive proof system, these tasks take place between the verifier and the prover, and they communicate back and forth with messages. The interaction between these two parties may be complex. The prover in this case may want to prove to the verifier that some element belongs to a language $\mathcal{L}$: $x \in \mathcal{L}$. A secure zero-knowledge proof should satisfy the properties: completeness, soundness, and zero-knowledge. The soundness property ensures that the verifiers will not be tricked into accepting false statements. The completeness property ensures that the prover can convince the verifier of their true statements. The formal definitions are given below.

Definition 2.7.1. ***Interactive Machine*** *[22].* *An interactive machine is a multi-tape Turing machine with the following tapes :*

- *read-only input tape*
- *read-only random tape*
- *read-and-write tape*
- *write-only output tape*
- *a pair of communication tapes (one is read-only, the other is write-only)*
- *read-and-write switch tape*

Each interactive machine has a single bit $\sigma \in \{0, 1\}$ called its identity. The machine is active if its identity is equal to the content of the switch tape. Otherwise, the machine is idle. While in idle state, the machine cannot make any modifications to its state or any of its tapes. When a machine is active and content is written on its write-only communication tape, this is equivalent to having the message sent. The read-only communication tape, on the other hand, is called the message received. The movement of the pair of communication tapes is unidirectional.

Definition 2.7.2. ***Interactive Proof*** *[22].* *The generalized interactive proof is defined as follows. An interactive pair $(\mathcal{P}, \mathcal{V})$ is called an interactive proof system for a language $\mathcal{L}$, with completeness bound $c(n)$ and soundness bound $s(n)$ if:*

- *(Completeness): for every $x \in \mathcal{L}$: $Pr[(\mathcal{P}, \mathcal{V})(x) = 1] \geq c(|x|)$*

- *(Soundness): for every $x \notin \mathcal{L}$ and every interactive machine $\mathcal{B}$:*

$$Pr[(\mathcal{B}, \mathcal{V})(x) = 1] \leq s(|x|)$$

The property of zero-knowledge captures the idea that no additional knowledge is revealed during the protocol. In other words, whatever the verifier computed after interacting with the prover on input $x \in \mathcal{L}$ can also be efficiently computed from the input x without interaction [22]. We now take a look at perfect zero-knowledge and computational zero-knowledge.

Definition 2.7.3. ***Perfect Zero-Knowledge [22].*** *Let $(\mathcal{P}, \mathcal{V})$ be an interactive proof system for some language $\mathcal{L}$. We say that $(\mathcal{P}, \mathcal{V})$ is perfect zero-knowledge if for every probabilistic polynomial-time interactive machine $\mathcal{V}^*$ there exists a probabilistic polynomial-time algorithm M^* such that for every $x \in \mathcal{L}$ the following two conditions hold:*

1. $\Pr[M^*(x) = \perp] \leq \frac{1}{2}$, *where $\perp$ is just a special symbol,*
2. *Let $m^*(x)$ be a random variable describing the distribution of $M^*(x)$ conditioned on $M^*(x) \neq \perp$ (i.e. $\Pr[m^*(x) = \alpha] = \Pr[M^*(x) = \alpha \mid M^*(x) \neq \perp]$ for every $\alpha \in \{0,1\}^*$). Then the following random variables are identically distributed:*
   - *$\langle \mathcal{P}, \mathcal{V}^* \rangle(x)$, (i.e. the output of the interactive machine $\mathcal{V}^*$ after interacting with the interactive machine $\mathcal{P}$ on input x), and*
   - *$m^*(x)$ (i.e. the output of machine M^* on input x, conditioned on not being $\perp$).*

Machine M^* is called a perfect simulator for the interaction of $\mathcal{V}^*$ with $\mathcal{P}$.

Often times, we can apply a relaxation and use the following definition instead.

Definition 2.7.4. ***Computational Zero-Knowledge [22].*** *Let $(\mathcal{P}, \mathcal{V})$ be an interactive proof system for some language $\mathcal{L}$. We say that $(\mathcal{P}, \mathcal{V})$ is computational zero-knowledge (or just zero-knowledge) if for every probabilistic polynomial-time interactive machine $\mathcal{V}^*$ there exists a probabilistic polynomial-time algorithm M^* such that the following two are computationally indistinguishable:*

- *$\{\langle \mathcal{P}, \mathcal{V}^* \rangle\}_{x \in \mathcal{L}\rangle}$ (i.e. the output of the interactive machine $\mathcal{V}^*$ after it interacts with the interactive machine $\mathcal{P}$ on input x), and*
- *$\{M^*(x)\}_{x \in \mathcal{L}}$ (i.e. the output of machine M^* on input x).*

The machine M^* is called a simulator for the interaction of $\mathcal{V}^*$ with $\mathcal{P}$.

2.7.1 Rejection Sampling

Rejection sampling is a useful tool in lattice-based cryptography.

In lattice-based zero-knowledge proofs, the prover wants to output a vector whose distribution is independent of a secret message vector. The reason behind this is so that no additional information can be gleaned from the secret message. Oftentimes, during an interaction with a prover and verifier, the prover will compute a vector $z = cr + y$, where c is a challenge sent by the verifier, r is a secret random vector, and y is a masking vector. To hide r in z, rejection sampling is used. A short y is chosen from a narrow distribution D, so that the secret r is not leaked when y is added [20].

In other words, we need to ensure that z is ultimately within the distribution D. Hence, in rejection sampling, we only accept samples that fall within the target distribution D. If z is greater than the target distribution, it is rejected. Some advantages of rejection sampling include flexibility and less memory usage. These are attributed to the fact that the parameters of the discrete Gaussian distribution can be changed and the memory used is often independent of the parameters chosen [19].

In particular, we are interested in the following rejection sampling lemma and its corresponding algorithms, which can be seen in Table 2.3.

Lemma 2.7.5 (Lemma 2.14 from [32]). *Let $\mathcal{V} \subseteq \mathcal{R}^\ell$ be a set of polynomials with norm at most T and $\rho : \mathcal{V} \to [0,1]$ be a probability distribution. Fix the standard deviation $s = \gamma T$. Then, the following statements hold.*

1. *Let $M = \exp(14/\gamma + 1/(2\gamma^2))$. Now, sample $\mathbf{v} \leftarrow \rho$ and $\mathbf{y} \leftarrow D_s^\ell$, set $\mathbf{z} = \mathbf{y} + \mathbf{v}$, and run $b \leftarrow Rej_1(\mathbf{z}, \mathbf{v}, s)$ as defined in Table 2.3. Then, the probability that $b = 0$ is at least $(1 - 2^{-128})/M$ and the distribution of $(\mathbf{v}, \mathbf{z})$, conditioned on $b = 0$, is within statistical distance of 2^{-128} of the product distribution $\rho \times D_s^\ell$.*

2. *Let $M = \exp(1/(2\gamma^2))$. Now, sample $\mathbf{v} \leftarrow \rho$ and $\mathbf{y} \leftarrow D_s^\ell$, set $\mathbf{z} = \mathbf{y} + \mathbf{v}$, and run $b \leftarrow Rej_2(\mathbf{z}, \mathbf{v}, s)$ as defined in Table 2.3. Then, the probability that $b = 0$ is at least $1/(2M)$ and the distribution of $(\mathbf{v}, \mathbf{z})$, conditioned on $b = 0$, is identical to the distribution $\mathcal{F}$ where $\mathcal{F}$ is defined as follows: sample $\mathbf{v} \leftarrow \rho$, $\mathbf{z} \leftarrow D_s^\ell$ conditioned on $\langle \mathbf{v}, \mathbf{z} \rangle \geq 0$ and output $(\mathbf{v}, \mathbf{z})$.*

3. *Let $M = \exp(1/(2\gamma^2))$. Now, sample $\mathbf{v} \leftarrow \rho$, $\beta \leftarrow \{0,1\}$ and $\mathbf{y} \leftarrow D_s^\ell$, set $\mathbf{z} = \mathbf{y} + (-1)^\beta \mathbf{v}$, and run $b \leftarrow Rej_0(\mathbf{z}, \mathbf{v}, s)$ as defined in Table 2.3. Then, the probability that $b = 0$ is at least $1/M$ and the distribution of $(\mathbf{v}, \mathbf{z})$, conditioned on $b = 0$, is identical to the product distribution $\rho \times D_s^\ell$.*

Table 2.3: [**Fig. 1 and Fig. 2** [32]] Three rejection sampling algorithms.

Rej$_1(\mathbf{z}, \mathbf{v}, s)$

$u \leftarrow [0, 1)$
If $u > \frac{1}{M} \cdot \exp\left(\frac{-2\langle \mathbf{z}, \mathbf{v}\rangle + \|\mathbf{v}\|^2}{2s^2}\right)$
  return 1 (i.e. *reject*)
Else
  return 0 (i.e. *accept*)

Rej$_2(\mathbf{z}, \mathbf{v}, s)$

If $\langle \mathbf{z}, \mathbf{v}\rangle < 0$
  return 1 (i.e. *reject*)
$u \leftarrow [0, 1)$
If $u > \frac{1}{M} \cdot \exp\left(\frac{-2\langle \mathbf{z}, \mathbf{v}\rangle + \|\mathbf{v}\|^2}{2s^2}\right)$
  return 1 (i.e. *reject*)
Else
  return 0 (i.e. *accept*)

Rej$_0(\vec{z}, \vec{v}, s)$

$u \leftarrow [0, 1)$
If $u > \frac{1}{M \cdot \exp\left(\frac{-\|\mathbf{v}\|^2}{2s^2}\right) \cdot \cosh\left(\frac{\langle \mathbf{z}, \mathbf{v}\rangle}{s^2}\right)}$
  return 1 (i.e. *reject*)
Else
  return 0 (i.e. *accept*)

2.8 Commitment Schemes

A commitment scheme allows one party to commit to a message while satisfying the hiding and binding properties. The hiding property states that the commitment should not reveal anything about the message. The binding property states that the commitment cannot be later opened to reveal two different messages. In other words, it can only reveal the original message that had been committed. Often, the commitment scheme can be thought of as analogous to a sealed envelope containing a letter. The contents of the envelope are safe from prying eyes until it is opened. In addition, the letter is also binding since it has been placed in the sealed envelope and cannot be changed [27].

The basic protocol consists of two algorithms:

- $\texttt{Gen}(1^\lambda)$: outputs the parameters of the protocol, *params*
- $\texttt{Com}(params, m; r)$: outputs a commitment *comm* given a randomness r and a message m

In Figure 2.4, the basic commitment scheme is shown with the algorithms discussed. The sender is able to make a commitment and send it to the receiver. To allow an opening of the commitment, the sender can send m and r to the receiver. Then, the receiver can verify that $\texttt{Com}(params, m; r)$ equals the commitment *comm* that was given to the receiver [27].

Although there are a number of different commitment schemes in the literature, there are two of great importance to us: the BDLOP commitment and the ABDLOP commitment. The ABDLOP commitment combines the Ajtai commitment with the BDLOP commitment [32].

Ajtai's commitment scheme commits to a message m using randomness r where both $\|m\|$ and $\|r\|$ are small. Let $\mathbf{A}_1$ and $\mathbf{A}_2$ be public matrices, and $\mathbf{t}$ be the commitment [2]:

$$\mathbf{A}_1 \cdot \mathbf{m} + \mathbf{A}_2 \cdot \mathbf{r} = \mathbf{t} \bmod \mathrm{q}. \tag{2.8.1}$$

The opening to Ajtai's commitment is m and r. The opening essentially allows a verifier to check if the commitment was made honestly or not. Creating a second valid opening $(\mathbf{m}', \mathbf{r}')$ is equivalent to solving the SIS problem over $\mathcal{R}_q$. This commitment scheme is also hiding because this commitment relies on the fact that the two matrices, $\mathbf{A}_2$ and $\mathbf{A}_2 \cdot \mathbf{r}$ are indistinguishable from uniform.

The BDLOP commitment scheme, on the other hand commits to a message $\mathbf{n}$ using randomness $\mathbf{s}$, where $\|\mathbf{s}\|$ is small. Matrices $\mathbf{A}$ and $\mathbf{B}$ are publicly known, and the commitment is $\mathbf{t}$ [6]:

$$\begin{bmatrix} \mathbf{A} \\ \mathbf{B} \end{bmatrix} \cdot \mathbf{s} + \begin{bmatrix} 0 \\ \mathbf{n} \end{bmatrix} = \begin{bmatrix} \mathbf{t}_A \\ \mathbf{t}_B \end{bmatrix} = \mathbf{t} \mod q. \tag{2.8.2}$$

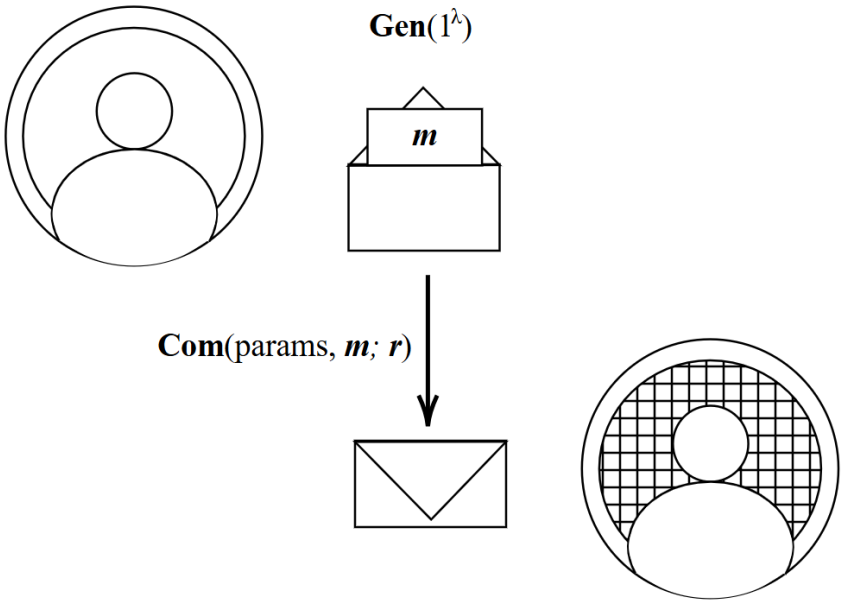

Figure 2.4: Basic commitment scheme with the sender on the top left and the receiver on the bottom right. Let m represent the message, r represent the randomness and *params* represent the parameters.

The $\mathbf{s}$ is the opening for the BDLOP scheme, since it can uniquely determine what $\mathbf{n}$ is. Similarly to the Ajtai's commitment scheme, the hiding property of this commitment relies on the fact that the following two matrices are indistinguishable from uniform:

$$\left(\begin{bmatrix} A \\ B \end{bmatrix}, \begin{bmatrix} A \\ B \end{bmatrix} \cdot s \right).$$

The relevant commitment scheme which our implementation uses, combines both Ajtai's commitment scheme and the BDLOP scheme to form the ABDLOP commitment scheme:

$$\begin{bmatrix} \mathbf{A}_1 \\ 0 \end{bmatrix} \cdot \mathbf{m} + \begin{bmatrix} \mathbf{A}_2 \\ \mathbf{B} \end{bmatrix} \cdot \mathbf{r} + \begin{bmatrix} 0 \\ \mathbf{n} \end{bmatrix} = \begin{bmatrix} \mathbf{t}_A \\ \mathbf{t}_B \end{bmatrix} = \mathbf{t} \mod q \,, \tag{2.8.3}$$

We do not require knowledge of both the randomness values $\mathbf{r}$ from Equation 2.8.1 and $\mathbf{s}$ from Equation 2.8.2 for the opening [32]. The hiding property follows from $\mathbf{r}$ being chosen from a distribution such that the following matrices are indistinguishable from uniform:

$$\left(\begin{bmatrix} \mathbf{A}_2 \\ \mathbf{B} \end{bmatrix}, \begin{bmatrix} \mathbf{A}_2 \\ \mathbf{B} \end{bmatrix} \cdot \mathbf{r} \right). \tag{2.8.4}$$

2.9 Blind Signatures

Blind signatures were first introduced in 1983 by David Chaum [14]. In his work, he outlined an untraceable payment system that uses a blind signature.

Blind signatures are a variation of digital signatures. They allow a signer to blindly sign a recipient's message without having seen what the message is. One of the analogies given in Chaum's work describes a ballot counting system in the form of a carbon lined envelope. The idea is that the voter seals their vote in one of these carbon-lined envelopes, places this envelope within another larger envelope, inscribes her return address on it, and mails it off to the trusted signer. Upon receiving this envelope, the trusted signer opens the larger envelope and signs the outside of the carbon-lined envelope, without having seen the contents of the inner envelope. The signer places the signed envelope within another envelope and mails it back to the voter. The voter is now allowed to open both envelopes to access their vote. The vote, now certified and signed, can be later sent to the polling station without a return address, where the ballots are verified against the public key of the trusted signer and the ballots are tallied.

The main properties of this protocol are that the trusted signer can sign a hidden message without knowing what the message reads. In addition to this, this message once signed, can be revealed in its raw form and is untraceable to the original owner of the message.

Chapter 3

Post-Quantum Anonymous Credentials

An anonymous credential system is a privacy-enhancing scheme with three main parties: a certificate authority, a prover, and a verifier. There are also two main algorithms as part of the scheme: the issuing and verification algorithms. The anonymous credential system can be seen in Figure 3.1. In an anonymous credential scheme, provers initially meet with the certificate authority offline to prove ownership over the attributes they would like to commit to their credential. In the real world, provers can be thought of as customers that may require a credential to access services. Certificate authorities are trusted parties, often government agencies, that can issue an anonymous credential to provers through the issuing protocol. Once the anonymous credential is issued, provers can participate in a showing protocol with the verifiers. The showing protocol enables provers to share the validity of their anonymous credentials as well as a subset of their credential attributes with verifiers. Verifiers can then verify the validity of the credentials and any subset of attributes that were revealed in the process. In particular, provers often will share a pseudonym with the verifiers instead of their original credential. When these pseudonyms are generated honestly, verifiers should be convinced of its validity with respect to the original credential, while still preserving the privacy of the prover. In some extensions of the protocol, verifiers can also verify that the properties of certain attributes are valid (i.e. if the age attribute falls within some range) without learning more information during their interaction. Verifiers can be thought of as institutions, businesses, or some other service-provider entity. They primarily validate and verify the correctness of credentials in order to proceed with the services they are offering.

Anonymous credentials and digital credentials have had slightly different definitions in the literature since their origin, but over the years, they have commonly been referred to under the umbrella term, anonymous credential system. Credentials were first introduced by Chaum in 1985 [15]. In 2000, Stefan Brands introduced the notion

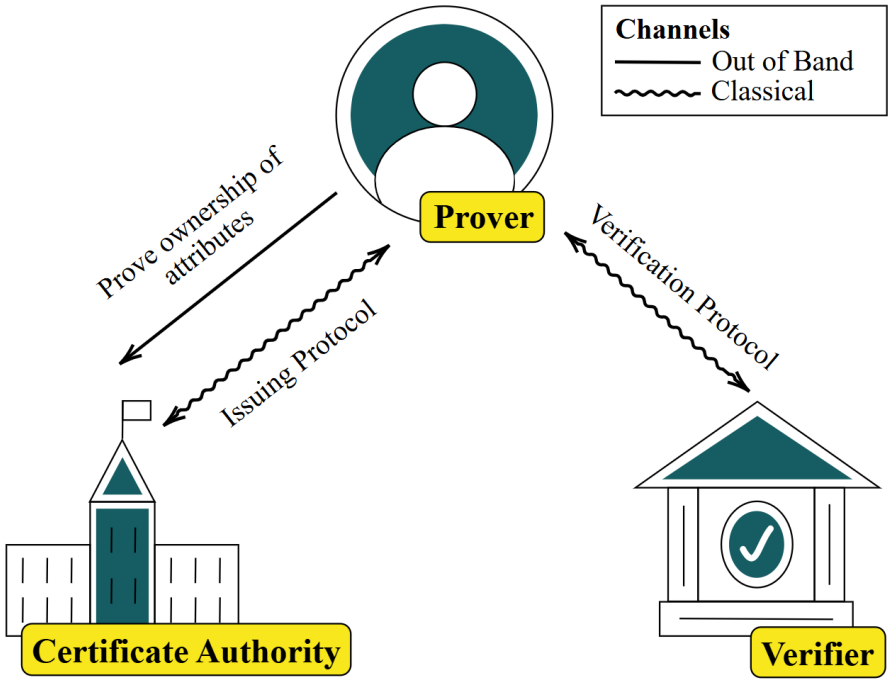

Figure 3.1: The parties and interactions involved in an honest anonymous credential scheme.

of digital credentials, a primarily single-use extension of Chaum's previous technology which introduced the disclosure of attribute properties [11]. Around the same time, Camenisch and Lysyanskaya built a scheme called anonymous credentials that was multi-use because users could prove knowledge of a valid credential without having to reveal it [12]. Since then, numerous adaptations have surfaced based on both types of technologies.

More recently, some post-quantum anonymous credential work has surfaced. In particular, various lattice-based frameworks have been proposed [4, 9, 10, 13, 28, 34, 35, 38]. Among these proposed post-quantum schemes, we have chosen to base our main implementation on the work of [28]. Their scheme focuses on using what is called a commit-transferrable signature. This scheme was chosen for its low key sizes and more efficient computations, compared to some of the other schemes. More information on the relevant works and why we chose [28] is given in Section 3.2.

3.1 Definitions

Anonymous credentials are multi-use. The multi-use comes from giving the provers the ability to create pseudonyms of valid credentials and then using a different pseudonym during different interactions with verifiers. We are interested in multi-use

credential schemes.

Anonymous credentials often satisfy a subset of the properties: anonymity, unforgeability, and unlinkability. Informally, these properties are described as follows.

The anonymity property ensures that the prover stays anonymous during interactions with the certificate authority and verifiers. Anonymity with the certificate authorities ensures that the certificate authorities cannot distinguish between two provers with two different inputs in the issuing stage. The anonymity with verifiers property is satisfied when a prover interacts with a verifier, and no additional information is revealed beyond whether the credential is valid or not.

The unforgeability property ensures that an adversary cannot prove that they have a valid credential if they have never been issued a valid credential from the certificate authority.

Unlinkability ensures that verifiers and certificate authorities cannot track the actions of a specific prover. In other words, an adversary cannot distinguish if two different pseudonyms belong to the same prover or not. If a prover discloses uniquely identifying information during their interaction with the verifier, this property does not hold.

These three properties are discussed more formally in Section 3.4 on security notions.

3.2 Literature Review on Lattice-based Anonymous Credentials

We will now discuss some post-quantum anonymous credential schemes from the literature. We compare the works in the literature against each other, and we ultimately decide on using the commit-transferrable signature-based scheme [28] which we provide the reasoning for in Section 3.2.6.

3.2.1 Lattice Signature with Efficient Protocols, Application to Anonymous Credentials

Jeudy, Roux-Langlois, and Sanders construct lattice-based anonymous credentials in their work [26]. Their scheme is based on a lattice-based signature scheme which efficiently allows commitments of messages to be signed. Additionally, their scheme proves in zero-knowledge the possession of a message-signature pair. The main properties of this protocol are anonymity, unlinkability, and unforgeability. The underlying hard problems of their system are M-LWE and M-ISIS, the inhomogeneous variation of M-SIS. The soundness of the zero-knowledge argument of knowledge (ZKAoK) used in their scheme comes from the EUF-CMA security.

The M-ISIS problem is defined as follows.

Definition 3.2.1. ***M-ISIS***$_{d,m,q,\beta}$**.** *Let U denote the uniform distribution. Let d, m, q be positive integers and $\beta > 0$. The Module Inhomogeneous Short Integer Solution problem M-ISIS$_{d,m,q,\beta}$ asks to find $\mathbf{s} \in \mathcal{R}^m$ such that $\mathbf{Ds} = \mathbf{t} \mod q\mathcal{R}$ and $\|\mathbf{s}\|_2 \leq \beta$, given $\mathbf{D}$ sampled from the distribution $U(\mathcal{R}_q^{d\times m})$ and $\mathbf{t}$ sampled from $U(\mathcal{R}_q^d)$.*

Definition 3.2.2. ***EUF-CMA security model.*** *This model is the Existential Unforgeability against Chosen Message Attacks security model. This security model captures the idea that attackers capable of obtaining signatures on messages of its choosing are unable to forge a signature on a new message.*

There have been some further extensions and developments on this scheme.

Chathurangi, Li, Foo, and Zhang extended the above scheme to include the property of traceability [13]. The traceability property allows for the de-anonymization of the credentials if misuse of a credential is suspected. In addition to a certificate authority, prover, and verifier, there are two additional actors in their scheme—a tracer and a judge. The tracer has the capability to generate deanonymization data when credential misuse is suspected. The tracer then forwards this data to the judge who can recover the identity associated to the credential.

In another paper, Argo, Güneysu, Jeudy, Land, Roux-Langlois, and Sanders design and implement a similar anonymous credential scheme, using signature with efficient protocols (SEP)[4]. They designed a SEP which comes with two main protocols: an issuing protocol on committed messages and an efficient proof of knowledge of a signature. The two properties of their anonymous credential implementation are anonymity and unforgeability. In their work, they implemented a proof of concept using the C language[1].

3.2.2 A Framework for Practical Anonymous Credentials from Lattices

Bootle, Lyubashevsky, Nguyen, and Sorniotti introduced a lattice-based anonymous credential based on a new variant of the SIS problem, called the ISIS$_f$ problem [10].

Definition 3.2.3. ***ISIS***$_f$ ***Problem.*** *Let $[N] = \{1, 2, \ldots, N\}$. Given a matrix $A \in \mathbb{Z}_p^{n\times m}$, a function $f : [N] \to \mathbb{Z}_p^n$, and access to an oracle that chooses a random input $x \in [N]$, we get as outputs, x together with a vector $\|s\| \leq \beta$, satisfying $A \cdot s = f(x)$. The game is won by coming up with a new tuple $(x', s') \in [N] \times \mathbb{Z}^m$ where $\|s'\| \leq \beta'$ and $As' = f(x')$. Note that x' can be equal to one of the x, as long as $s' \neq s$.*

[1] Their code can be found here: https://github.com/Chair-for-Security-Engineering/lattice-anonymous-credentials.

The function f is chosen from a class of fairly simple functions such as linear or slightly higher degree functions. This in turn gives efficient zero-knowledge proofs.

The properties of the protocol satisfy anonymity and unforgeability.

The hardness of the problem, ISIS_f, instantiated with linear functions or slightly higher degree functions f still requires further investigation and remains an open problem.

Their work has been the basis for two open-source implementations. Lyubashevsky, Seiler, and Steuer provided an implementation of this anonymous credential system using their library LaZer for zero-knowledge and succinct proofs [34]. The LaZer library consists of C code underneath a Python interface, and they provided a demonstration of an anonymous credential in the library based on the above scheme[2].

Another implementation is from Margaria, Pino, Vesco, D'Alconzo, Di Scala, Guglielmino, and Sanna [35]. They implemented the [10] scheme in C++ and C, and have made their work open source[3].

3.2.3 Lattice-based Commit-Transferrable Signatures and Applications to Anonymous Credentials

Lai, Chen, Liu, Lysyanskaya, and Wang developed an anonymous credential scheme using a novel signature called commit-transferrable signatures (CTS) [28]. From a high-level perspective, CTS can do two things. Firstly, one of its properties includes allowing signatures to be computed on top of a commitment. Additionally, signatures can be transformed into new signatures with respect to new commitments on the same underlying secret attributes. The anonymous credential scheme takes CTS paired with a NIZKPoK system to create the anonymous credential scheme. The properties that the credential scheme meets are anonymity, unlinkability, and unforgeability. The scheme relies on the hardness assumptions of M-SIS and M-LWE. We provide an in-depth look at this construction in Section 3.3.

3.2.4 Post-Quantum Privacy Pass via Post-Quantum Anonymous Credentials

Policharla, Westerbaan, and Faz-Hernández constructed a post-quantum anonymous credential based on zkDilithium [38]. The zkDilithium scheme is a STARK-friendly variation on Dilithium2. A Scalable Transparent ARgument of Knowledge (STARK) is a non-interactive argument of knowledge system, while Dilithium is a lattice-based post-quantum digital signature scheme which has been selected for standardization by NIST. Dilithium offers three different parameter sets, and Dilithium2 is one such

[2]Their library can be found here: https://github.com/lazer-crypto/lazer

[3]Their open-source code can be found here: https://github.com/Cybersecurity-LINKS/pqzk-blns

parameter set that has the smallest public key, secret key, and signature sizes. The properties of their anonymous credential scheme: anonymity, unlinkability, and unforgeability follow from the use of Dilithium. Additionally, Dilithium is based on the hard problems: M-LWE, M-SIS, and SelfTargetMSIS. The SelfTargetMSIS problem is based on the hardness of both the M-SIS problem and hash function.

Definition 3.2.4. ***SelfTargetMSIS (Definition 3 [25]).*** *Let $\tau, m, k, \gamma \in \mathbb{N}$ and $H : \{0,1\}^* \to B_\tau$, where $B_\tau \subseteq \mathcal{R}_q$ is the set of polynomials with exactly τ coefficients in $\{-1, 1\}$ and all remaining coefficients zero. The advantage of an algorithm $\mathcal{A}$ for solving $\mathsf{SelfTargetMSIS}_{H,\tau,m,k,\gamma}$ is defined as:*

$$\mathrm{Adv}^{\mathsf{SelfTargetMSIS}}_{H,\tau,m,k,\gamma}(\mathcal{A}) := \Pr\big[\, H([I_m \mid A] \cdot y \,\|\, M) = y_{m+k} \;\wedge\; \|y\|_\infty \leq \gamma \mid A \leftarrow \mathcal{R}_q^{m\times k},\ (y, M) \leftarrow \mathcal{A}^H(A) \,\big].$$

Note that the $\|$ notation refers to a string concatenation and $\mathcal{A}^H(A)$ denotes $\mathcal{A}$ with quantum query access to H.

The main motivation in their paper was to construct a post-quantum anonymous credential for the simpler problem of building a post-quantum privacy pass. In privacy passes, clients can interact with an attester and an issuer to obtain a token that can later be redeemed to a website upon being challenged. Privacy passes are considered better for the end-user experience compared to the alternative of having users repeatedly answer CAPTCHAs [18]. Their implementation is benchmarked against some post-quantum blind-signature schemes and thus only a subset of its anonymous credential capabilities are actually demonstrated. The anonymous credential is implemented in Rust[4].

3.2.5 Efficient Implementation of a Post-Quantum Anonymous Credential Protocol

The last post-quantum credential protocol that we will discuss is the construction and implementation of an anonymous credential system by Blazy et al. [9]. Their system is constructed from a signature scheme, a verifiable encryption scheme, as well as a non-interactive zero-knowledge proof [9]. The authors used a relaxed version of some of these schemes. The properties captured by the anonymous credential are anonymity, unforgeability, and traceability. The traceability property allows for signature tracebacks in certain scenarios. If signatures have been computed by colluding provers and verifiers, the signature can be traced back to a member of the forging coalition. The main actors include a certificate authority, provers, and verifiers. In addition, they

[4]Their open-source code can be found here: https://github.com/guruvamsi-policharla/zkdilithium

also introduce a fourth entity named inspector, who is a trusted authority who can deanonymize the presentation tokens (think of this as the credential) under special circumstances. Finally, the scheme is based on the hardness of the M-SIS, M-LWE, and NTRU problems. This paper offers a concrete implementation of their work in the programming language C[5].

Definition 3.2.5. ***NTRU$_{q,r}$ Problem.*** *The $NTRU_{q,r}$ problem over an implicit ring $\mathcal{R}$ is defined in the following way. The distribution $\mathcal{A}$ is defined by sampling ring elements $f, g \xleftarrow{\$} \mathcal{D}_r$ and outputting $h = f/g$, if g is invertible in $\mathcal{R}_q$. Otherwise, g is resampled. The problem $NTRU_{q,r}$ is to distinguish h from a random element in $\mathcal{R}_q$.*

3.2.6 Paper Selection

The comparisons between the different works can be seen in Tables 3.1 and 3.2.

Table 3.1: The properties and hardness problems of the different anonymous credential schemes from the literature.

	[26]	[4]	[10]	[28]	[38]	[9]
Anonymity	✓	✓	✓	✓	✓	✓
Unforgeability	✓	✓	✓	✓	✓	✓
Traceability	extended by [13]					✓
Unlinkability	✓			✓	✓	
Hardness Problem	M-ISIS, M-LWE	M-SIS, M-LWE	ISIS$_f$	M-SIS M-LWE	M-SIS, M-LWE, SelfTargetMSIS	M-SIS, M-LWE, NTRU

[5]We were unable to find access to their code online.

Table 3.2: The sizes and bit-security of the anonymous credential protocols from the literature.

	[26]	[4]	[10]	[28]	[38]	[9]
Public Parameters Size	0.27MB	-	-	24.66 MB	-	-
Public Key Size	9.56 MB	49.78 KB	-	440 KB	-	-
Secret Key Size	10.59 MB	10.25 KB	-	15 KB	-	-
Pseudonym Size	-	-	-	2.01 MB	-	1.92 MB
Signature Size	317 KB	6.81 KB	-	236.56 KB	112 KB	1.92 MB
Credential Size	724 KB	79.58 KB	122 KB	372.56 KB	-	-
Bit Security	128	128	128	128	115	-
Implementation	No	Yes	Yes [34, 35]	Yes (our contribution)	Yes	Yes

Ultimately, we chose the [28] protocol to implement and extend, due to its efficiency in terms of the smaller sizes of its various parameters compared to other works. In addition, it relies on hardness problems that are fairly well studied compared to ISIS_f or even SelfTargetMSIS. We also decided to choose a scheme that had not already been implemented for further extension because we wanted to diversify the anonymous credential schemes available.

We note that the authors of the works shown in Table 3.2 claim 128-bit security despite their different key sizes. We do some additional bit-security testing for the selected design [28] in Subsection 4.2.1.

To the best of our knowledge, there are five existing implementations of a post-quantum anonymous credential protocol [4, 9, 34, 35, 38]. We will be using their works as benchmarks in Section 5.3.

3.3 CTS-based Anonymous Credentials

We will now explore the selected work of [28] in more detail. Their anonymous credential scheme is built from three building blocks, the BDLOP commitment scheme, a commit-transferrable signature, and finally a non-interactive zero-knowledge proof of knowledge.

3.3.1 BDLOP Commitment Scheme

One of the building blocks used for the anonymous credential is the BDLOP commitment scheme. The algorithms that define the BDLOP commitment scheme can be seen in Table 3.3.

Table 3.3: The **BDLOP** algorithms: `CKeyGen, Commit, Open, Combine,` and `Randomize` as depicted in [28].

`CKeyGen`(1^λ) — given security parameter λ as input, output the public parameters, *params*.

1. Set the parameters $n, k, \ell, q_1, q_2 \in \mathbb{Z}$, and ring $\mathcal{R} = \mathbb{Z}[x]/\langle x^N + 1\rangle$ where N is a power of 2.
2. Choose random matrices $\mathbf{A}_1' \xleftarrow{\$} \mathcal{R}_{q_1}^{n\times(k-n)}$ and $\mathbf{A}_2' \xleftarrow{\$} \mathcal{R}_{q_2}^{\ell\times(k-n-\ell)}$.
3. Output the public parameters: $params := \mathbf{A}_0 = \begin{bmatrix}\mathbf{A}_1\\ \mathbf{A}_2\end{bmatrix}$, with $\mathbf{A}_1 := [\mathbf{I}_n, \mathbf{A}_1'] \in \mathcal{R}_{q_1}^{n\times k}$, $\mathbf{A}_2 := [\mathbf{0}^{\ell\times n}, \mathbf{I}_\ell, \mathbf{A}_2'] \in \mathcal{R}_{q_2}^{\ell\times k}$.

`Commit`$(params, m; \mathbf{r})$ — given public parameters *params*, message $m \in \mathcal{R}_{q_2}^\ell$ and randomness $\mathbf{r} \xleftarrow{\$} \mathcal{S}_\beta^k$ as inputs, output a commitment $\boldsymbol{comm}$ on message m.

1. Output: $\boldsymbol{comm} := \begin{bmatrix}\mathbf{t}_1\\ \mathbf{t}_2\end{bmatrix} = \begin{bmatrix}\mathbf{A}_1\\ \mathbf{A}_2\end{bmatrix}\mathbf{r} + \begin{bmatrix}\mathbf{0}\\ m\end{bmatrix}$.

`Open`$(params, \boldsymbol{comm})$ — given the public parameters *params* and commitment $\boldsymbol{comm}$ as inputs, provide a valid opening $(\mathbf{r}, m)$.

1. Outputs the exact valid opening of $\boldsymbol{comm} := (\mathbf{t}_1^\top, \mathbf{t}_2^\top)^\top \in \mathcal{R}_{q_1}^n \times \mathcal{R}_{q_2}^\ell$ consisting of a message $m \in \mathcal{R}_{q_2}^\ell$ and a short vector $\mathbf{r} = (r_1, \ldots, r_k)^\top \in \mathcal{R}^k$, such that:

$$\begin{bmatrix}\mathbf{t}_1\\ \mathbf{t}_2\end{bmatrix} = \begin{bmatrix}\mathbf{A}_1\\ \mathbf{A}_2\end{bmatrix}\mathbf{r} + \begin{bmatrix}\mathbf{0}\\ m\end{bmatrix}, \quad \text{where for all } i,\ \|\mathbf{r}_i\|_\infty \le \beta.$$

`Combine`$(\mathbf{r}, \mathbf{r}')$ — given randomness vectors $\mathbf{r} \in \mathcal{S}_\beta^k$ and $\mathbf{r}' \in \mathcal{S}_\beta^k$ as inputs, output a combined random value $\hat{\mathbf{r}}$.

1. Output $\hat{\mathbf{r}} = \mathbf{r} + \mathbf{r}' \in \mathcal{S}_{2\beta}^k$.

`Randomize`$(params, \boldsymbol{comm}, \mathbf{r}')$ — given the public parameters *params*, commitment $\boldsymbol{comm}$, and random vector $\mathbf{r}' \in \mathcal{S}_\beta^k$ as inputs, output a randomized commitment $\boldsymbol{comm}'$.

1. Output: $\boldsymbol{comm}' = \boldsymbol{comm} + \mathbf{A}_0 \cdot \mathbf{r}'$.

To prove the well-formedness of the BDLOP commitment, the proof framework from [32] is used.

3.3.2 Commit-Transferable Signatures

Commit-transferable signatures (CTS) are a special type of signature with two key properties. The first key property allows a signature to be computed directly on a commitment, simplifying the process. The prover need only perform a ZKPoK of the opening of the commitment. Once the signer verifies the proof, the signer can then directly compute the signature on the commitment. The second property is that the prover can compute new signatures from their old honestly-generated signature. Provers do so by creating valid signatures on a new commitment to the same underlying message committed to in the old honestly-generated signature. This property lends itself very well to the idea of pseudonyms for anonymous credentials, which we shall see shortly.

Commit-transferrable signatures use the **BDLOP** algorithms as well as a NIZKPoK system we will denote as $\Pi^{(1)}$. The concrete language $\mathcal{L}_{\gamma',q_2,\bar{\mathcal{C}}}$ for $\Pi^{(1)}$ is shown below:

$$\mathcal{L}_{\gamma',q_2,\bar{\mathcal{C}}} = \Big\{ (\mathbf{F}_{\text{comm}'}, \mathbf{u}) \in \mathcal{R}^{\ell\times(\ell\cdot(2\tau+1)+\hat{\ell}+k-n)} \times \mathcal{R}^{\ell}_{N,q_2} : \exists \mathbf{x} \in \mathcal{R}^{\ell(2\tau+1)+\hat{\ell}+k-n} \quad (3.3.1)$$

$$\text{and } f \in \bar{\mathcal{C}} \text{ such that } 0 < \|\mathbf{x}\| \leq \gamma' \text{ and } \mathbf{F}_{\text{comm}'} \cdot \mathbf{x} = f \cdot \mathbf{u} \Big\}.$$

$$\text{Note: } \bar{\mathcal{C}} \text{ is the set of differences } \mathcal{C} - \mathcal{C} \text{ except } 0,$$

$$\text{where } \mathcal{C} = \{ c \in \mathcal{R} : \|c\|_1 = \kappa \ \wedge \ \|c\|_\infty = 1 \}.$$

The algorithms used in the CTS protocol are described in Tables 3.4, 3.5, 3.6, and 3.7, which define the algorithms *Setup, Commit, Randomize, Combine, KeyGen, Sign, Transfer,* and *Verify*.

Table 3.4: The **CTS** algorithms: ***Setup***, ***Commit***, and ***Randomize*** as depicted in [28].

Setup (1^λ) — given security parameter λ as input, output the parameters *params* of the commit-transferrable signature scheme. The message space is defined as:

$$\mathcal{M} := \{m(\mathbf{X}^i) \in \mathcal{R}_{N,q_2} : m \in \{0,1\}^d, m \neq 0^d, \text{ and } \|m\|_1 = w\}.$$

1. Run BDLOP.`CKeyGen` to get $\mathbf{A} := \begin{bmatrix} \mathbf{I}_n & \mathbf{A}_1 \\ \mathbf{0}^{\ell \times n} & \mathbf{A}_2 \end{bmatrix}$ where $[\mathbf{I}_n, \mathbf{A}_1] \in \mathcal{R}_{N,q_1}^{n \times k}$ and $[\mathbf{0}^{\ell \times n}, \mathbf{A}_2] \in \mathcal{R}_{N,q_2}^{\ell \times k}$, with $\mathbf{A}_1 \in \mathcal{R}_{N,q_1}^{n \times (k-n)}$ and $\mathbf{A}_2 = (\mathbf{I}_\ell, \mathbf{A}_2') \in \mathcal{R}_{N,q_2}^{\ell \times (k-n)}$.
2. Sample a random matrix $\mathbf{D} \xleftarrow{\$} \mathcal{R}_{N,q_2}^{\ell \times (\ell + \hat{\ell})}$.
3. Set parameters κ, γ, γ', and a Gaussian parameter α.
4. Run $\Pi^{(1)}$.Setup(1^λ) to get a common reference string crs.
5. Output $\mathit{params} := (\mathbf{A}, \mathbf{D}, q_1, q_2, N, \kappa, \gamma, \gamma', \alpha, \mathcal{M}, \mathcal{R}, \text{crs})$.

Commit $(\mathit{params}, m; \mathbf{R})$ — given public parameters *params*, message $m \in \mathcal{R}_{q_2}^\ell$ and randomness $\mathbf{R} \in S_1^{k \times (\ell \cdot \tau)}$ as inputs, output a commitment ***comm*** on m.

1. Set $\mathbf{G} = \mathbf{I}_\ell \otimes g^\top$, where $\in \mathcal{R}_N^{\ell \times (\ell \cdot \tau)}$, where $\mathbf{I}_\ell \in \mathcal{R}_N^{\ell \times \ell}$ is the identity matrix, $g^\top = (1, \delta, \ldots, \delta^{\tau-1})$, and $\delta = \lfloor q_2^{1/\tau} \rceil$.
2. Compute and output the commitment ***comm*** =BDLOP.`Commit`$(\mathbf{A}, m \cdot \mathbf{G}; \mathbf{R})$, i.e. $\boldsymbol{comm} := \mathbf{C} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{bmatrix} \cdot \mathbf{R} + \begin{bmatrix} \mathbf{0}^{n \times (\ell \cdot \tau)} \\ m \cdot \mathbf{G} \end{bmatrix} \begin{matrix} \in \mathcal{R}_{N,q_1}^{n \times (\ell \cdot \tau)} \\ \in \mathcal{R}_{N,q_2}^{\ell \times (\ell \cdot \tau)} \end{matrix}$.

Randomize $(\mathit{params}, \boldsymbol{comm}, \mathbf{R}')$ — given the public parameters *params*, commitment ***comm***, and random vector $\mathbf{R}' \in \mathcal{S}_1^{k \times (\ell \cdot \tau)}$ as inputs, output randomized commitment ***comm***$'$.

1. Computes ***comm***$'$ =BDLOP.`Randomize`$(\mathbf{A}, \boldsymbol{comm}, \mathbf{R}')$, i.e.

$$\boldsymbol{comm}' := \mathbf{C}' = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{bmatrix} \cdot \mathbf{R} + \begin{bmatrix} \mathbf{0}^{n \times (\ell \cdot \tau)} \\ m \cdot \mathbf{G} \end{bmatrix} + \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{bmatrix} \cdot \mathbf{R}' \begin{matrix} \in \mathcal{R}_{N,q_1}^{n \times (\ell \cdot \tau)} \\ \in \mathcal{R}_{N,q_2}^{\ell \times (\ell \cdot \tau)} \end{matrix}.$$

Table 3.5: The **CTS** algorithms: *Combine*, *KeyGen*, and *Sign* as depicted in [28].

Combine$(\mathbf{R}, \mathbf{R}')$ — given randomness matrices $\mathbf{R}, \mathbf{R}' \in \mathcal{S}_1^{k\times(\ell\cdot\tau)}$ as inputs, output new randomness matrix $\mathbf{R}'' \in \mathcal{S}_2^{k\times(\ell\cdot\tau)}$.

1. Output $\mathbf{R}'' := \mathbf{R} + \mathbf{R}'$.

KeyGen$(params)$ — given public parameters $params$, output secret key sk and public key pk.

1. Sample $\mathbf{T} \xleftarrow{\$} S_1^{(\ell+\hat{\ell})\times(\ell\cdot\tau)}$ and set $\mathbf{A}_0 = \mathbf{D} + \mathbf{T}\cdot\mathbf{G} \in \mathcal{R}_{N,q_2}^{\ell\times(\ell\cdot\tau)}$.
2. Sample $\mathbf{B} \xleftarrow{\$} \mathcal{R}_{N,q_2}^{\ell\times(\ell\cdot\tau)}$ and a non-zero $\mathbf{u} \xleftarrow{\$} \mathcal{R}_{N,q_2}^{\ell}$.
3. Output $pk := (\mathbf{A}_0, \mathbf{B}, \mathbf{u})$, and $sk := \mathbf{T}$.

Sign$(params,\ pk, sk,\ \boldsymbol{comm})$ — given public parameters $params$, public key pk, secret key sk and commitment ***comm*** as inputs, output a signature Sig_{comm} with respect to ***comm***.

1. Parse $\boldsymbol{comm} := \mathbf{C} = \begin{bmatrix} \mathbf{C}_1 \\ \mathbf{C}_2 \end{bmatrix} \begin{matrix} \in \mathcal{R}_{N,q_1}^{n\times(\ell\cdot\tau)} \\ \in \mathcal{R}_{N,q_2}^{\ell\times(\ell\cdot\tau)} \end{matrix}$.
2. Set $F_{\text{comm}} = \Big[[\mathbf{D} \,|\, \mathbf{A}_0] \,|\, \mathbf{B}_{\text{comm}} \,|\, \mathbf{A}_2\Big] = \Big[[\mathbf{D} \,|\, \mathbf{A}_0] \,|\, [\mathbf{B} + \mathbf{C}_2] \,|\, \mathbf{A}_2\Big]$, and sample $Sig_{\text{comm}} \leftarrow \mathbf{SamplePre}(F_{comm}, \mathbf{T}, \mathbf{u}, \alpha)$. Note that the algorithm **SamplePre** is a pre-image sampling algorithm that finds a short pre-image Sig_{comm} satisfying $F_{comm}\cdot Sig_{comm} = \mathbf{u}$, with the help of the trapdoor $\mathbf{T}$, and a gaussian parameter α.
3. Output $Sig_{comm} := \begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \mathbf{s}_3 \end{bmatrix}$ as the signature of ***comm***, where $\mathbf{s}_1 = \begin{bmatrix} \mathbf{s}_{1,1} \\ \mathbf{s}_{1,2} \end{bmatrix}$, and $\mathbf{s}_{1,1} \in \mathcal{R}_N^{\ell+\hat{\ell}}$, $\mathbf{s}_{1,2} \in \mathcal{R}_N^{\ell\cdot\tau}$, $\mathbf{s}_2 \in \mathcal{R}_N^{\ell\cdot\tau}$, $\mathbf{s}_3 \in \mathcal{R}_N^{k-n}$.

Table 3.6: The **CTS** algorithm: *Transfer* as depicted in [28].

Transfer$(params,\ pk, Sig_{comm}, m, (\mathbf{R}, \mathbf{R}'))$ — given public parameters $params$, public key pk, signature Sig_{comm}, message m, and randomness matrices $\mathbf{R}, \mathbf{R}'$ as inputs, output a randomized signature Sig'_{comm}.

1. Parse $Sig_{comm} = [\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3]^\top$where $\mathbf{s}_1 \in \mathcal{R}_N^{(1+\tau)\cdot\ell+\hat{\ell}}$, $\mathbf{s}_2 \in \mathcal{R}_N^{\ell\cdot\tau}$, and $\mathbf{s}_3 \in \mathcal{R}_N^{k-n}$.
2. Run *Commit*$(params, m, \mathbf{R})$ to obtain: $\boldsymbol{comm} := \mathbf{C} = \begin{bmatrix} \mathbf{C}_1 & \in \mathcal{R}_{N,q_1}^{n\times(\ell\cdot\tau)} \\ \mathbf{C}_2 & \in \mathcal{R}_{N,q_2}^{\ell\times(\ell\cdot\tau)} \end{bmatrix}$.
3. Run *Randomize*$(params, \boldsymbol{comm}, \mathbf{R}')$ to obtain:
   $\boldsymbol{comm}' := \mathbf{C}' = \begin{bmatrix} \mathbf{C}'_1 & \in \mathcal{R}_{N,q_1}^{n\times(\ell\cdot\tau)} \\ \mathbf{C}'_2 & \in \mathcal{R}_{N,q_2}^{\ell\times(\ell\cdot\tau)} \end{bmatrix}$.
4. Compute a temporary signature Sig'_{comm} as:
   $$Sig_{comm'} := \begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \mathbf{s}_3 - \widetilde{\mathbf{R}}_2 \cdot \mathbf{s}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{s}_{1,1} \\ \mathbf{s}_{1,2} \\ \mathbf{s}_2 \\ \mathbf{s}_3 - \widetilde{\mathbf{R}}_2 \cdot \mathbf{s}_2 \end{bmatrix} \in \mathcal{R}^{\ell\cdot(2\tau+1)+\hat{\ell}+k-n},$$
   where we denote $\widetilde{\mathbf{R}} := \mathbf{R} + \mathbf{R}' = \begin{bmatrix} \widetilde{\mathbf{R}}_1 \\ \widetilde{\mathbf{R}}_2 \end{bmatrix} \in \mathcal{R}_N^{k\times(\ell\cdot\tau)}$, with $\widetilde{\mathbf{R}}_1 \in \mathcal{R}_N^{n\times(\ell\cdot\tau)}$ and $\widetilde{\mathbf{R}}_2 \in \mathcal{R}_N^{(k-n)\times(\ell\cdot\tau)}$.
5. Compute $F_{comm'} := \Big[[\mathbf{D} \,|\, \mathbf{A}_0] \,|\, \mathbf{B}_{\text{comm}'} \,|\, \mathbf{A}_2\Big] = \Big[[\mathbf{D} \,|\, \mathbf{A}_0] \,|\, [\mathbf{B} + \mathbf{C}'_2] \,|\, \mathbf{A}_2\Big]$.
6. Run the prove algorithm and output
   $$Sig_{comm'} := \pi \leftarrow \Pi^{(1)}.\text{Prove}(\text{crs}, (F_{comm'} \cdot \mathbf{u}), Sig_{comm'}),$$
   proving that Sig'_{comm} is a short ℓ_2 norm vector and satisfies $F_{comm'} \cdot Sig_{comm'} = \mathbf{u}$, through using the NIZKPoK system $\Pi^{(1)}$ with the relaxed language $\mathcal{L}_{\gamma',q_2,\bar{C}}$.

Table 3.7: The **CTS** algorithm: ***Verify*** as depicted in [28]

Verify$(params, pk, \boldsymbol{comm}, Sig_{comm})$ — given public parameters $params$, public key pk, commitment $\boldsymbol{comm}$ as inputs, and signature Sig_{comm} as input, output accept or reject.

1. Parse $\boldsymbol{comm} := \mathbf{C} = \begin{bmatrix} \mathbf{C}_1 & \in \mathcal{R}_{N,q_1}^{n\times(\ell\cdot\tau)} \\ \mathbf{C}_2 & \in \mathcal{R}_{N,q_2}^{\ell\times(\ell\cdot\tau)} \end{bmatrix}$.
2. If Sig_{comm} is a non-zero short vector within ℓ_2 norm γ, then:
   (a) Set matrix $F_{comm} := \Big[[\mathbf{D} \,|\, \mathbf{A}_0] \,|\, [\mathbf{B} + \mathbf{C}_2] \,|\, \mathbf{A}_2^\top\Big]$.
   (b) Check whether Sig_{comm} satisfies

$$F_{comm} \cdot \begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \mathbf{s}_3 \end{bmatrix} = \mathbf{u} \in \mathcal{R}_{N,q_2}.$$

   Otherwise, Sig_{comm} is a proof of the NIZKPoK system $\Pi^{(1)}$:
   (a) Set matrix $F_{comm} := \Big[[\mathbf{D} \,|\, \mathbf{A}_0] \,|\, [\mathbf{B} + \mathbf{C}_2] \,|\, \mathbf{A}_2^\top\Big]$.
   (b) Run the verify algorithm (with respect to the language $\mathcal{L}'_{\gamma',q_2,\bar{C}}$):

$$\Pi^{(1)}.\text{VerifyProve}(\text{crs}, (F_{comm}, \mathbf{u}), Sig_{comm}),$$

   and output its result.

Commit-transferable signatures are secure because they satisfy the following properties: correctness, simulatability (which implies unlinkability) and unforgeability. These properties are defined below. The definitions and proofs have been taken from [28]. The details of the scheme's security are here for completeness.

Informally, the correctness property generally states that if the algorithms in the signature protocol are used honestly, the signatures and transferred signatures are valid and should pass the verification check done by the verifier.

Definition 3.3.1. ***CTS Correctness (Definition 3.1 [28]).***

Let `Setup`, `Commit`, `Randomize`, `Combine`, `KeyGen`, `Sign`, `Verify`, *and* `Transfer` *be efficient algorithms for the CTS.*

These algorithms define a correct randomizable commitment scheme if for all parameters output by `Setup`, *all messages* $m \in \mathcal{M}$, *random vectors* $\boldsymbol{r} \in \mathcal{R}$ *and* $\boldsymbol{r}' \in \mathcal{R}'$,

$$\texttt{Randomize}(\texttt{Commit}(params, m; \boldsymbol{r}), \boldsymbol{r}') =$$

$$\texttt{Commit}(params, m; \texttt{Combine}(\boldsymbol{r}, \boldsymbol{r}')).$$

Furthermore, the algorithms define a correct commit-transferrable signature scheme if for all parameters output by `Setup`, *all messages* $m \in \mathcal{M}$, *random vectors* $\boldsymbol{r} \in \mathcal{R}$ *and* $\boldsymbol{r}' \in \mathcal{R}'$, *the following accepts:*

$(sk, pk) \leftarrow \texttt{KeyGen}(params)$,
$Sig_{comm} \leftarrow \texttt{Sign}(params, pk, sk, \texttt{Commit}(params, m, \boldsymbol{r}))$,
$Sig_{comm'} \leftarrow \texttt{Transfer}(params, pk, \sigma, m, (\boldsymbol{r}, \boldsymbol{r}'))$,
both $\texttt{Verify}(params, pk, \texttt{Commit}(params, m, \boldsymbol{r}), Sig_{comm})$ *and*
$\texttt{Verify}(params, pk, \texttt{Commit}(params, m, \texttt{Combine}(\boldsymbol{r}, \boldsymbol{r}')), Sig_{comm'})$

Next, let us look at a lemma that uses the correctness definition. In our implementation, we choose appropriate parameters to uphold the correctness as stated in the lemma.

Lemma 3.3.2 (**Lemma B.1 from [28]**). *For the specific construction outlined in [28], as can be seen in Tables 3.4, 3.5, 3.6, 3.7, the CTS scheme is considered correct if it satisfies the correctness definition 3.3.1 with respect to the parameters:* N, q_2, α, γ, *and the NIZKPoK system* $\Pi^{(1)}$ *of the relaxed language* $L_{\gamma', q_2, \bar{C}}$, *where*

$$\gamma = \alpha \sqrt{2 \cdot \left(\ell \cdot (2\tau + 1) + \hat{\ell} + k - n\right) \cdot N} \text{ and}$$

$$\gamma' \geq \beta = \left(\left(\sqrt{k - n} + \sqrt{\ell \cdot \tau}\right) \cdot N \cdot \alpha \sqrt{2 \cdot \ell \cdot \tau} + \alpha \sqrt{\left(\ell \cdot (2\tau + 1) + \hat{\ell} + k - n\right) \cdot N}\right).$$

Let us now examine the security of CTS through the simulatability and unforgeability properties.

Informally, the simulatability property says that a transferred signature does not leak information about its input or randomness. This property is shown through the use of a simulator that simulates the creation of signatures. The distribution of signatures output by the simulator needs to be statistically close to the signatures output by the `Transfer` algorithm. Simulatability implies unlinkability, which says that a transferred signature and its underlying commitment are unlinkable to the original signature and commitment. Unlinkability is an important concept for security when we discuss how the anonymous credential system is implemented.

Definition 3.3.3. ***Simulatability (Definition 3.2 [28]).*** *The* `Transfer` *algorithm is simulatable if there exists a PPT simulator S which can simulate* `Transfer` *in an indistinguishable way. The two-stage simulation process is as follows:*

- *S generates parameters params and some trapdoor information Trap*
- *S is given params and Trap, and arbitrary public key pk and commitment* $\boldsymbol{comm}$*. S can generate a simulated transferred signature* $\widetilde{\boldsymbol{Sig}_{comm'}}$.

The simulatability requires that for $t = poly(\lambda)$, $\{m_i\}_{i\in[t]} \in \mathcal{M}$, *randomness* $\{\boldsymbol{r}_i, \boldsymbol{r}'_i\}_{i\in[t]}$, *no PPT distinguisher* $\mathcal{D}$ *can distinguish the following with better than negligible advantage:*

- $(params, pk, \{\boldsymbol{comm}'_i\}_{i\in[t]}, \{\boldsymbol{Sig}_{comm'}.\}_{i\in[t]})$, *where params and pk are sampled honestly, and*
  $\boldsymbol{comm}_i = \texttt{Commit}(params, m_i; \boldsymbol{r}_i)$,
  $\boldsymbol{Sig}_{comm_i} \leftarrow \texttt{Sign}(params, pk, sk, \boldsymbol{comm}_i)$,
  $\boldsymbol{comm}'_i = \texttt{Randomize}(params, \boldsymbol{comm}_i, \boldsymbol{r}'_i$, *and*
  $\boldsymbol{Sig}_{comm'}. \leftarrow \texttt{Transfer}(params, pk, \sigma_i, m_i, (\boldsymbol{r}_i, \boldsymbol{r}'_i))$
- $(params, pk, \{\boldsymbol{comm}'_i\}_{i\in[t]}, \{\ \widetilde{\boldsymbol{Sig}_{comm'}}\}_{i\in[t]})$, *where params is generated by the simulator, pk is sampled honestly,* $\boldsymbol{comm}'_i$ *is generated as above, and* $\widetilde{\boldsymbol{Sig}_{comm'}}$. *is generated by the simulator*

Lemma 3.3.4 (**Lemma 4.5 from [28]**). *Simulatability: Suppose* $\Pi^{(1)}$ *is a NIZKPoK system, the algorithm* `Transfer` *is simulatable.*

Additionally, CTS satisfies the unforgeability property.

Definition 3.3.5. ***Unforgeability (Definition 3.3 [28]).*** *An unforgeable commit-transferable signature is such that for all PPT adversaries* $\mathcal{A}$*, the probability that* $\mathcal{A}$ *wins the following game is negligible:*

- ***Input generation phase:*** *On input* 1^λ*, the challenger generates* $params \leftarrow \texttt{Setup}(1^\lambda)$*,* $(sk, pk) \leftarrow \texttt{KeyGen}(params)$*.*
- ***Query phase:*** *Given* $(params, pk)$ *as inputs, the adversary* $\mathcal{A}$ *can make queries to an oracle in the form of* $(\boldsymbol{comm}_i, m_i, \boldsymbol{r}_i)$*, and gets as responses* $\boldsymbol{Sig}_{comm_i} = \texttt{Sign}(params, pk, \boldsymbol{comm}_i)$ *if* $\boldsymbol{comm}_i = \texttt{Commit}(params, m_i; \boldsymbol{r}_i)$*, or else,* $\perp$ *.*
- ***Challenge phase****: The adversary* $\mathcal{A}$ *outputs* $(m^*, \boldsymbol{r}, \sigma)$*. Let*

$$\boldsymbol{comm}^* = \texttt{Commit}(params, m^*;\ \boldsymbol{r}),$$

and $\mathcal{A}$ *wins the game if*

$$\texttt{Verify}(params, pk, \boldsymbol{comm}^*, \boldsymbol{Sig}_{comm})$$

accepts, and m^* *has never been queried in the query phase.*

The scheme is selectively secure if the adversary needs to commit to the challenge message m^* *before the input generation phase, and is adaptively secure if this condition is not required.*

For our implementation, we focus on the selectively secure version.

Before we go into the unforgeability of the specific construction and its specified parameters, we first introduce another lemma which the unforgeability proof will rely on.

Lemma 3.3.6 (**Lemma A.7 from [28]**)**.** *Let Rot be a map:* $R \to \mathbb{Z}^{N\times N}$*, where Rot(a) is a rotation matrix. There exists an efficient algorithm that on input ring vectors* $\mathbf{a}_1 \in \mathcal{R}_q^{\ell_1}$*,* $\mathbf{a}_2 \in \mathcal{R}_q^{\ell_2}$ *such that Rot*$([\mathbf{a}_1^\top | \mathbf{a}_2^\top]) \in \mathbb{Z}^{N\times N(\ell_1+\ell_2)}$ *is full-rank, elements* x*,* $c \in \mathcal{R}_q^*$*,* $u \in \mathcal{R}_q$ *with* $\|c\|_2 \leq \tau$ *and matrices* $\mathbf{R}_1 \in \mathcal{R}_q^{\ell_1\times k}$*,* $\mathbf{R}_2 \in \mathcal{R}_q^{\ell_2\times k}$*, outputs a random sample* $\mathbf{r} \in \mathcal{R}_q^{\ell_1+\ell_2+k}$ *from a distribution that is statistically close to*

$$D_\sigma(\Lambda_q^u([\mathbf{a}_1^\top | \mathbf{a}_2^\top | \mathbf{R}_1 + \mathbf{a}_2^\top \mathbf{R}_2 + x \cdot \mathbf{g}_\delta^\top | \mathbf{a}_2^\top])), \text{ where } \sigma \geq 2\sqrt{\delta^2+1}\left(s_1\left(\begin{bmatrix}\mathbf{R}_1\\ \mathbf{R}_2\end{bmatrix}\right)+1\right).$$

We employ the use of Lemma 3.3.6 during the setup of our construction.

Lemma 3.3.7 (**Lemma 4.6 from [28]**)**.** *Suppose* $\Pi^{(1)}$ *is a rewinding-extractable NIZKPoK system in the random oracle model, assume that M-SIS*$_{q_2,\ell,\ell(\tau+1)+\hat{\ell}+k-n+1,\nu}$ *problem and M-SIS*$_{q_2,\ell,\ell(\tau+1)+\hat{\ell}+k-n+1,\nu'}$ *problem are hard with*

$$\nu = \alpha\sqrt{2\cdot\left(\ell(\tau+1)+\hat{\ell}+k-n\right)\cdot N} + \left(\sqrt{\ell+\hat{\ell}}+\sqrt{k-n}+2\sqrt{\ell\cdot\tau}\right)\cdot\alpha\cdot N\sqrt{2\cdot\ell\cdot\tau}+1,$$

$$\nu' = \alpha' \sqrt{2 \cdot \left(\ell(\tau+1) + \hat{\ell} + k - n\right) \cdot N} + \left(\sqrt{\ell + \hat{\ell}} + \sqrt{k-n} + 2\sqrt{\ell \cdot \tau}\right) \cdot \alpha' \cdot N\sqrt{2 \cdot \ell \cdot \tau} + 2\sqrt{\kappa},$$

where $\alpha' = \gamma' / \sqrt{2 \cdot \left(\ell \cdot (2\tau+1) + \hat{\ell} + k - n\right) \cdot N}$.

Then the lattice-based commitment-transferable signature construction as outlined in tables 3.4, 3.5, 3.6, and 3.7, is partially selectively unforgeable for the exact commitment relation $\tilde{\mathcal{L}}_{q_1,q_2}$*, i.e. the advantage of any PPT adversary* $\mathcal{A}$ *against the partially selective unforgeability game of CTS is at most:*

$$Adv_{\mathcal{A}}^{unforge}(\lambda) \leq 2Adv_{\mathcal{A}}^{M\text{-}LWE}(\lambda) + Adv_{\mathcal{A}}^{unforge^*}(\lambda).$$

where $Adv_{\mathcal{A}}^{M\text{-}LWE}$ *denotes the advantage* $\mathcal{A}$ *has under the M-LWE assumption and* $Adv_{\mathcal{A}}^{unforge^*}$ *denotes the advantage* $\mathcal{A}$ *has in the selective unforgeability game for the relation* $\hat{\mathcal{L}}_{q_1,q_2}$ *(see Equation 3.3.2).*

Lemma 3.3.8 (Lemma B.4 from [28]). *Let* $\mathcal{A}$ *be a PPT adversary with advantage* ϵ *in the selective unforgeability game for the exact commitment relation* $\hat{\mathcal{L}}_{q_1,q_2}$ *(see Equation 3.3.2), i.e.* $Adv_{\mathcal{A}}^{unforge^*}(\lambda) = \epsilon$.

Let h *be a bound on the number of random oracle queries made by* $\mathcal{A}$*. Let*

$$\nu = \alpha \sqrt{2 \cdot \left(\ell(\tau+1) + \hat{\ell} + k - n\right) \cdot N} + \left(\sqrt{\ell + \hat{\ell}} + \sqrt{k-n} + 2\sqrt{\ell \cdot \tau}\right) \cdot \alpha \cdot N\sqrt{2 \cdot \ell \cdot \tau} + 1,$$

$$\nu' = \alpha' \sqrt{2 \cdot \left(\ell(\tau+1) + \hat{\ell} + k - n\right) \cdot N} + \left(\sqrt{\ell + \hat{\ell}} + \sqrt{k-n} + 2\sqrt{\ell \cdot \tau}\right) \cdot \alpha' \cdot N\sqrt{2 \cdot \ell \cdot \tau} + 2\sqrt{\kappa},$$

where $\alpha' = \gamma' / \sqrt{2 \cdot \left(\ell \cdot (2\tau+1) + \hat{\ell} + k - n\right) \cdot N}$.

Then, there exists a reduction algorithm $\mathbf{R}$ *for* $M\text{-}SIS_{q_2,\ell,\ell(\tau+1)+\hat{\ell}+k-n+1,\nu}$ *or* $M\text{-}SIS_{q_2,\ell,\ell(\tau+1)+\hat{\ell}+k-n+1,\nu'}$ *such that*

$$Adv_{\mathbf{R}}^{M\text{-}SIS}(\lambda) \geq \epsilon \left(\frac{\epsilon}{h} - 2^{-\lambda}\right).$$

3.3.3 Non-interactive Zero-Knowledge Proof of Knowledge

The third ingredient needed for the anonymous credential scheme is a multi-theorem straight-line extractable NIZKPoK for proving the well-formedness of a commitment ***comm*** output by `CTS.Commit`. The NIZKPoK is based on the work of [32] and is an important part of the issuing protocol for the anonymous credential scheme we

use [28]. The authors of [28] chose a multi-theorem straight-line extractable NIZKPoK to avoid the security proof of the anonymous credential system incurring an exponential loss. Notably, the anonymous credential scheme of [28] uses an encrypt-and-prove paradigm. The paradigm consists of the use of the Regev encryption scheme for encrypting a witness and then proving that the encrypted message under the ciphertext satisfies the well-formedness of a commitment relation. Though we outsource this component of our implementation to a package, we explain some of the basic concepts behind the NIZKPoK used.

Multi-theorem means that multiple witnesses can be extracted from an adversary that generates multiple valid proofs, and straight-line refers to the fact that the extraction does not require rewinding. A formal definition is given below.

Definition 3.3.9. ***Multi-Theorem Extractability (Definition A.21 [28]).*** *Let $\mathfrak{R}$ be a relation, and a non-interactive proof system for $\mathfrak{R}$ is a tuple of PPT algorithms: Setup, Prove, Verify, $\mathsf{SimSetup}$. A NIZK system is multi-theorem straight-line extractable, if there exists a PPT oracle simulator $\mathsf{SimSetup}$ and a PPT extractor Ext with the following properties:*

CRS indistinguishability. *For any PPT adversary $\mathcal{A}$, we have*

$$\begin{aligned} Adv(\mathcal{A}) := \; & \Pr[\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) : \mathcal{A}(\mathsf{crs}) = 1] \\ & - \Pr[(\widetilde{\mathsf{crs}}, \mathsf{tk}) \leftarrow \mathsf{SimSetup}(1^\lambda) : \mathcal{A}(\widetilde{\mathsf{crs}}) = 1] \leq \mathsf{negl}(\lambda). \end{aligned}$$

Straight-Line Extractability. *Let Q_S represnt the number of signature queries. There exists constants c, e_1, e_2 and polynomial $p(\lambda)$ such that for any $Q_H = poly(\lambda)$ and PPT adversary $\mathcal{A}$ that makes at most Q_H random oracle queries with*

$$\Pr\left[\begin{array}{l} (\widetilde{\mathsf{crs}}, \mathsf{tk}) \leftarrow \mathsf{SimSetup}(1^\lambda), \{(x_i, \pi_i)\}_{i\in[Q_s]} \leftarrow \mathcal{A}(\widetilde{\mathsf{crs}}) : \\ \forall i \in [Q_s], \;\; \mathsf{Verify}(\widetilde{\mathsf{crs}}, x_i, \pi_i) = 1 \end{array}\right] \geq \mu(\lambda),$$

we have

$$\Pr\left[\begin{array}{l} (\widetilde{\mathsf{crs}}, \mathsf{tk}) \leftarrow \mathsf{SimSetup}(1^\lambda), \;\; \{(x_i, \pi_i)\}_{i\in[Q_s]} \leftarrow \mathcal{A}(\widetilde{\mathsf{crs}}), \\ \{w_i \leftarrow \mathsf{Ext}(1^\lambda, Q_H, Q_s, 1/\mu, \mathsf{tk}, x_i, \pi_i)\}_{i\in[Q_s]} : \\ \forall i \in [Q_s], \;\; (x_i, \pi_i) \in \mathfrak{R} \wedge \mathsf{Verify}(\widetilde{\mathsf{crs}}, x_i, \pi_i) = 1 \end{array}\right] \geq \frac{1}{2} \cdot \mu(\lambda) - \mathsf{negl}(\lambda).$$

Moreover, the running time of Ext is upper bounded by

$$Q_H^{e_1} \cdot Q_s^{e_2} \cdot \frac{1}{\mu^c} \cdot p(\lambda).$$

The NIZKPoK system is shown in a series of figures in the work of [32]. For completeness, we show Tables 3.8, 3.9, 3.10, and 3.11 which are copied over (with

some modification) from Figures 10, 8, 7, and 6 in [32], respectively. Table 3.8 (Figure 10 from [32]) shows a general protocol to prove approximate range proofs and quadratic functions. The NIZKPoK starts with 3.8 and continues in 3.9, 3.10, and 3.11 (respectively, Figure 8, Figure 7, Figure 6 in [32]). We will discuss the importance and use of each component.

The purpose of the PoK system is to prove that the commitment ***comm*** is well-formed. The exact commitment relation that we would like to prove well-formedness for is:

$$\hat{\mathcal{L}}_{q_1,q_2} := \{\boldsymbol{comm} : \exists(m, q_1, q_2, \mathbf{R}), \text{ where } m \in \mathcal{M}, \mathbf{R} \in \mathcal{S}_1^{k\times(\ell\cdot\tau)} \tag{3.3.2}$$

$$\text{and } \boldsymbol{comm} = \text{CTS.}\mathit{Commit}\,(\text{params}, m \cdot \mathbf{G}; \mathbf{R})\},$$

where $\mathcal{M} := \{m(\mathbf{X}^i) \in \mathcal{R}_{N,q_2} : m \in \{0,1\}^d,\ m \neq 0^d, \text{ and } \|m\|_1 = w\}$ is the message space.

This relation is equivalent to the following equations that we are trying to prove well-formedness of:

$$\boldsymbol{comm} := \mathbf{C} = \begin{bmatrix}\mathbf{C}_1\\ \mathbf{C}_2\end{bmatrix} = \begin{bmatrix}\mathbf{A}_1\\ \mathbf{A}_2\end{bmatrix} \cdot \mathbf{R} + \begin{bmatrix}\mathbf{0}^{n\times(\ell\cdot\tau)}\\ m\cdot\mathbf{G}\end{bmatrix} \begin{matrix}\in \mathcal{R}_{N,q_1}^{n\times(l\cdot\tau)}\\ \in \mathcal{R}_{N,q_2}^{\ell\times(\ell\cdot\tau)}\end{matrix}. \tag{3.3.3}$$

The above equation can be transferred into the following two equations:

$$\mathbf{C}_1 = \mathbf{A}_1 \cdot \mathbf{R} \mod q_1, \quad \mathbf{C}_2 = \mathbf{A}_2 \cdot \mathbf{R} + m \cdot \mathbf{G} \mod q_2, \tag{3.3.4}$$

where $\mathbf{R} \in \mathcal{S}_1^{k\times(\ell\cdot\tau)}$ and $m \in \mathcal{M}$.

Table 3.8 shows an interactive proof that validates and verifies a number of important equations. The figure relies on a number of other proofs shown in some of the other tables. Firstly, it relies on the interactive proof of knowledge (PoK) of a single quadratic equation with automorphisms which can be seen in Table 3.11.

For completeness, we show the relevant equations defined in the NIZKPoK protocol in Table 3.8 below. First, we define a map $\mathcal{T} : \mathbb{Z}^{kd} \times \mathbb{Z}^{kd} \to \mathcal{R}_d$. The map outputs: $\mathcal{T}(\vec{a}, \vec{b}) := \sum_{i=0}^{k-1} \sigma_{-1}\left(\sum_{j=0}^{d-1} a_{id+j}X^j\right) \cdot \left(\sum_{j=0}^{d-1} b_{id+j}X^j\right) \in \mathcal{R}_d$.

$$\forall i \in \{d, e\}, \quad g^{(i)}(b^{(i)}) = (b^{(i)} - 1)(b^{(i)} + 1) \tag{3.3.5}$$

$$G(\mathbf{x}') = \mathcal{T}(\mathbf{x}', \mathbf{x}' - \mathbf{1}_{(\nu_e+k_{\text{bin}})d}) \tag{3.3.6}$$

$$\forall j \in [256], \quad H_j^{(d)}(\mathbf{s}, \mathbf{y}^{(d)}, b^{(d)}) = z_j^{(d)} - \mathcal{T}(b^{(d)}\mathbf{r}_j^{(d)}, \mathbf{e}^{(d)}) - y_j^{(d)} \tag{3.3.7}$$

$$\forall j \in [256], \quad H_j^{(e)}(\mathbf{x}', \mathbf{s}, \mathbf{y}^{(e)}, b^{(e)}) = z_j^{(e)} - \mathcal{T}(b^{(e)}\ \mathbf{r}_j^{(e)}, \mathbf{e}^{(e)}) - y_j^{(e)} \tag{3.3.8}$$

$$\forall i \in [v_e], \quad I_i(\mathbf{s}, \mathbf{x}) = \mathcal{T}(\mathbf{E}_i\mathbf{s} - \mathbf{v}_i, \mathbf{E}_i\mathbf{s} - \mathbf{v}_i) + \mathcal{T}(\overrightarrow{p_i}, \overrightarrow{x_i}) - (\beta_i^{(e)})^2 \tag{3.3.9}$$

$$\forall i \in \{d, e\}, 1 \leq j \leq d-1, J_j^{(i)}(b^{(i)}) = \mathcal{T}(\delta_j, b^{(i)}) \tag{3.3.10}$$

Next, we look at the general statements to be proven in Table 3.8, which is the knowledge of vector $\mathbf{s} = (\mathbf{s}_1, \sigma(\mathbf{s}_1), \mathbf{m}, \sigma(\mathbf{m})) \in \mathcal{R}_{d,q}^{2m_1} \times \mathcal{R}_{d,q}^{2\ell}$ such that the following equations are satisfied:

$$\forall 1 \leq i \leq \rho, \quad f_i(\mathbf{s}) = 0 \tag{3.3.11}$$

$$\forall 1 \leq i \leq \rho_{\text{eval}}, \quad \tilde{F}_i(\mathbf{s}) = 0 \tag{3.3.12}$$

$$\forall 1 \leq i \leq v_d, \quad \|\mathbf{D}_i\mathbf{s} - \mathbf{v}_i\|_\infty \leq \beta_i^{(d)} \tag{3.3.13}$$

$$\forall 1 \leq i \leq v_e, \quad \|\mathbf{E}_i\mathbf{s} - \mathbf{v}_i\| \leq \beta_i^{(e)} \tag{3.3.14}$$

$$\mathbf{E}_{\text{bin}}\mathbf{s} - \mathbf{v}_{\text{bin}} \in \{0,1\}^{d \cdot k_{\text{bin}}} \tag{3.3.15}$$

In [32], the security properties are that this sub-protocol is sound, complete, and commit-and-prove simulatable. The latter property states that a simulator without access to private information $\mathbf{s}_1, m$ is able to output a simulation of a commitment with a non-aborting transcript of the protocol. For every algorithm $\mathcal{A}$ that has an advantage ϵ in distinguishing the simulated commitment from the real commitment whenever the prover does not abort, there exists an algorithm $\mathcal{A}'$ with the same running time that has an advantage $\epsilon/2 - 2^{-128}$ in distinguishing the Extended-MLWE$_{n+\ell+1,m_2-n-\ell-1,\mathcal{S}_v,\mathcal{C},s_2}$.

The second sub-protocol is generalizing the previous single quadratic PoK to many quadratic equations with automorphism PoK. In this sub-protocol, the N equations can be combined linearly into one quadratic equation. Then, the running of the sub-protocol with the single quadratic PoK suffices. The resultant single quadratic equation of committing to only one garbage polynomial is at the cost of reducing the soundness error by a negligible additive factor. This combination of quadratic equations can be seen in Table 3.10, and involves the verifier sending N challenges to the prover for them to compose their single equation.

The third sub-protocol of interest is about polynomial evaluations with vanishing constant coefficients. This PoK shown in Table 3.9 describes the proof of the N quadratic relations and that the constant coefficient of the quadratic $k(m_1+\ell)$- variate polynomials $F_1, ..., F_M$, (where $F_j\left(\left(\sigma^i(\mathbf{s}_1)\right)_{i\in[k]}, \left(\sigma^i(\mathbf{m})\right)_{i\in[k]}\right), j \in [M]$) is equal to 0.

Finally in Table 3.8, a general protocol is given to prove various quadratic relations on $\mathbf{s} = (\mathbf{s}_1, \mathbf{m}, \sigma(\mathbf{s}_1), \sigma(\mathbf{m}))$, including quadratic equations over $\mathcal{R}_{d,q}$, quadratic relations over $\mathbb{Z}_q$, the approximate bound on the infinity norm, the exact bound on the ℓ_2 norm and that a vector is binary.

Table 3.8: [**Figure 10** [32]] General commit-and-prove protocol Π, for various quadratic relations on messages $(\mathbf{s}_1, \mathbf{m}) \in \mathcal{R}_{d,q}^{m_1+\ell}$ in an ABDLOP commitment.

Public information:

Commitment $\mathbf{t} \in \mathcal{R}_{d,q}^{n+\ell}, \mathbf{A}_1 \in \mathcal{R}_{d,q}^{n\times(m_1+v_e)}, \mathbf{A}_2 \in \mathcal{R}_{d,q}^{n\times m_2}, \mathbf{B} \in \mathcal{R}_{d,q}^{\ell\times m_2}$ such that
$\mathbf{t} = \begin{bmatrix}\mathbf{A}_1\\ 0\end{bmatrix}\begin{bmatrix}\mathbf{s}_1\\ \mathbf{x}\end{bmatrix} + \begin{bmatrix}\mathbf{A}_2\\ \mathbf{B}\end{bmatrix}\mathbf{s}_2 + \begin{bmatrix}\mathbf{0}\\ \mathbf{m}\end{bmatrix}$
$\mathbf{B}^{(d)} \in \mathcal{R}_{d,q}^{(256/d)\times m_2}, \mathbf{B}^{(e)} \in \mathcal{R}_{d,q}^{(256/d)\times m_2}, \mathbf{b}^{(d)} \in \mathcal{R}_{d,q}^{m_2}, \mathbf{b}^{(e)} \in \mathcal{R}_{d,q}^{m_2}$
For $i \in [\rho]$, quadratic functions $f_i : \mathcal{R}_{d,q}^{2(m_1+\ell)} \to \mathcal{R}_{d,q}$
For $i \in [\rho_{\text{eval}}]$, quadratic functions $F_i : \mathcal{R}_{d,q}^{2(m_1+\ell)} \to \mathcal{R}_{d,q}$
For $i \in [v_d]$, matrix $(\mathbf{D}_i) \in \mathcal{R}_{d,q}^{k_i\times(m_1+\ell)}$, vector $\mathbf{u}_i \in \mathcal{R}_{d,q}^{k_i}$, bound $\beta_i^{(d)}$
For $i \in [v_e]$, matrix $(\mathbf{E}_i) \in \mathcal{R}_{d,q}^{p_i\times(m_1+\ell)}$, vector $\mathbf{v}_i \in \mathcal{R}_{d,q}^{p_i}$, bound $\beta_i^{(e)}$
Matrix $\mathbf{E}_{\text{bin}} \in \mathcal{R}_{d,q}^{k_{\text{bin}}\times 2(m_1+\ell)}$
Bounds $\alpha^{(d)}, \alpha^{(e)}$ such that $\|\mathbf{e}^{(d)}\| \le \alpha^{(d)}, \|\mathbf{e}^{(e)}\| \le \alpha^{(e)}$
Standard deviations $\mathfrak{s}^{(d)} = \gamma^{(d)}\sqrt{337}\alpha^{(d)}, \mathfrak{s}^{(e)} = \gamma^{(e)}\sqrt{337}\alpha^{(e)}$, acceptance coefficient $t \in \mathbb{R}$
Challenge dimensions $c^{(d)} = d\sum_{i=1}^{v_e} k_i, \quad c^{(e)} = d(k_{\text{bin}} + \sum_{i=1}^{v_e}(p_i+1))$
Input vectors of $\Pi_{\texttt{eval}}^{(2)}$ protocol: $\phi = (f_1, \ldots, f_\rho, g^{(d)}, g^{(e)})$, and
$\Psi = \left((F_i)_{i\in\rho_{\text{eval}}}, G, \left(H_j^{(d)}\right)_{j\in[256]}, \left(H_j^{(e)}\right)_{j\in[256]}, (I_i)_{i\in v_e}, \left(J_j^{(i)}\right)_{i\in\{d,e\},j\in[d]}\right)$, where the matrices are defined in Equations 3.3.5 - 3.3.10.

Private information:

Randomness $\mathbf{s}_2 \leftarrow \mathcal{S}_v^{m_2}$, message $\mathbf{s} = (\mathbf{s}_1, \mathbf{m}) \in \mathcal{R}_{d,q}^{m_1+t}$ such that Equations 3.3.11 to 3.3.15 hold.
Binary decomposition $x_i \in \mathcal{R}_{d,q}$ of $(\beta_i^{(e)})^2 - \|\mathbf{E}_i\mathbf{s} - \mathbf{v}_i\|^2$.
Vectors $\mathbf{e}^{(d)} = (\mathbf{D}_1\mathbf{s} - \mathbf{u}_1||\cdots||\mathbf{D}_{v_d}\mathbf{s} - \mathbf{u}_{v_d})$, $\mathbf{e}^{(e)} = (\mathbf{E}_1\mathbf{s} - \mathbf{v}_1||\cdots||\mathbf{E}_{v_e}\mathbf{s} - \mathbf{v}_{v_e}||\mathbf{x})$.

Prover		**Verifier**
$b^{(d)}, b^{(e)} \leftarrow \{-1, 1\} \subset \mathcal{R}_{d,q}$		
$\mathbf{y}^{(d)} \leftarrow D_{\mathfrak{s}^{(d)}}^{256/d}$, $\mathbf{y}^{(e)} \leftarrow D_{\mathfrak{s}^{(e)}}^{256/d}$		
$\mathbf{t}^{(d)} := \mathbf{B}^{(d)}\mathbf{s}_2 + \mathbf{y}^{(d)}$		
$\mathbf{t}^{(e)} := \mathbf{B}^{(e)}\mathbf{s}_2 + \mathbf{y}^{(e)}$		
$t^{(d)} := (\mathbf{b}^{(d)})^\top\mathbf{s}_2 + b^{(d)}$		
$t^{(e)} := (\mathbf{b}^{(e)})^\top\mathbf{s}_2 + b^{(e)}$	$\xrightarrow{\mathbf{t}^{(d)},\mathbf{t}^{(e)},t^{(d)},t^{(e)}}$	
		$R^{(d)} \leftarrow \text{Bin}_1^{256\times c^{(d)}}$
	$\xleftarrow{R^{(d)},R^{(e)}}$	$R^{(e)} \leftarrow \text{Bin}_1^{256\times c^{(e)}}$
$\vec{z}^{(d)} := b^{(d)}R^{(d)}\vec{e}^{(d)} + \vec{y}^{(d)}$		
$\vec{z}^{(e)} := b^{(e)}R^{(e)}\vec{e}^{(e)} + \vec{y}^{(e)}$		
If $\text{Rej}_0(\vec{z}^{(d)}, b^{(d)}R^{(d)}\vec{e}^{(d)}, \mathfrak{s}^{(d)})$		
and $\text{Rej}_0(\vec{z}^{(e)}, b^{(e)}R^{(e)}\vec{e}^{(e)}, \mathfrak{s}^{(e)})$		
Then continue, Else abort		
$\mathbf{s}^* := (\mathbf{s}_2, (\mathbf{s}_1, \mathbf{x}), (\mathbf{m}, \mathbf{y}^{(d)}, \mathbf{y}^{(e)}, b^{(d)}, b^{(e)}))$		
	$\xrightarrow{\vec{z}^{(d)},\vec{z}^{(e)}}$	
Run $\Pi_{\text{eval}}^{(2)}(\mathbf{s}^*, \sigma, \phi, \Psi)$ (See Table 3.9)		Accept iff: Π verifies $\|\vec{z}^{(d)}\|_\infty \le 14\mathfrak{s}^{(d)}$ $\|\vec{z}^{(e)}\| \le t\sqrt{256}\mathfrak{s}^{(e)}$

Table 3.9: [**Figure 8** [32]] Commit-and-prove sub-protocol $\Pi^{(2)}_{\text{eval}}((\mathbf{s}_2, \mathbf{s}_1, \mathbf{m}), \sigma, (f_1, \ldots, f_N), (F_1, \ldots, F_M))$.

Private information:
$(\mathbf{s}_1, \mathbf{m}) \in \mathcal{R}_{d,q}^{m_1+\ell}$ so that $\|\mathbf{s}_1\| \leq \alpha, \mathbf{s}_2 \leftarrow \mathcal{S}_v^{m_2}$
Public information:
$\mathbf{A}_1 \in \mathcal{R}_{d,q}^{n\times m_1}, \quad \mathbf{A}_2 \in \mathcal{R}_{d,q}^{n\times m_2}, \quad \mathbf{B} \in \mathcal{R}_{d,q}^{\ell\times m_2}, \quad \mathbf{B}_g \in \mathcal{R}_{d,q}^{\lambda\times m_2}, \quad \mathbf{b} \in \mathcal{R}_{d,q}^{m_2}$
$\begin{bmatrix}\mathbf{t}_A \\ \mathbf{t}_B\end{bmatrix} = \begin{bmatrix}\mathbf{A}_1 \\ \mathbf{0}\end{bmatrix} \cdot \mathbf{s}_1 + \begin{bmatrix}\mathbf{A}_2 \\ \mathbf{B}\end{bmatrix} \cdot \mathbf{s}_2 + \begin{bmatrix}\mathbf{0} \\ \mathbf{m}\end{bmatrix},$
$f_1, \ldots, f_N, F_1, \ldots, F_M : \mathcal{R}_{d,q}^{k(m_1+\ell)} \to \mathcal{R}_{d,q}, \sigma \in \text{Aut}(\mathcal{R}_{d,q})$
Note that for an element $h = h_0 + h_1 X + \cdots + h_{d-1}X^{d-1} \in \mathcal{R}_{d,q}$, we will write $\widetilde{h}$ to represent the constant coefficient h_0.

Prover		**Verifier**
$\mathbf{s} := \begin{bmatrix}(\sigma^i(\mathbf{s}_1))_{i\in[k]} \\ (\sigma^i(\mathbf{m}))_{i\in[k]}\end{bmatrix}$ $\mathbf{g} := (g_1, \ldots, g_\lambda) \leftarrow \{x : \mathcal{R}_{d,q} : \tilde{x} = 0\}^\lambda$ $\mathbf{t}_g := \mathbf{B}_g\mathbf{s}_2 + \mathbf{g}$		
	$\xrightarrow{\mathbf{t}_g}$	
	$\xleftarrow{(\gamma_{i,j})_{i\in[\lambda],j\in[M]}}$	$\Gamma = (\gamma_{i,j}) \leftarrow \mathbb{Z}_q^{\lambda\times M}$
For $i \in [\lambda]$: $h_i := g_i + \sum_{j=1}^M \gamma_{i,j} F_j(\mathbf{s})$		
	$\xrightarrow{h_1,\ldots,h_\lambda}$	
Define $f_1, \ldots, f_{N+\lambda}$, where for $j \in [N]$, $f_j = f_j((\sigma^i(\mathbf{s}_1))_{i\in[k]}, (\sigma^i(\mathbf{m}))_{i\in[k]})$ and for $i \in [\lambda]$, $f_{N+i} := g_i + \sum_{j=1}^M \gamma_{i,j} F_j\left((\sigma^i(\mathbf{s}_1), (\sigma^i(\mathbf{m}))_{i\in[k]})\right) - h_i$, Run $\Pi^{(2)}_{\text{many}}((\mathbf{s}_2, \mathbf{s}_1, \mathbf{m}\|\mathbf{g}), \sigma, (f_i)_{i\in[N+\lambda]})$ (See Table 3.10).		Accept iff: $\Pi^{(2)}_{\text{many}}$ verifies and $\tilde{h}_1 = \cdots = \tilde{h}_\lambda = 0$.

Table 3.10: [**Figure 7** [32]] Commit-and-prove sub-protocol $\Pi^{(2)}_{\text{many}}((\mathbf{s}_2, \mathbf{s}_1, \mathbf{m}), \sigma, (f_1, \ldots, f_N))$.

Private information:
$(\mathbf{s}_1, \mathbf{m}) \in \mathcal{R}_{d,q}^{m_1+\ell}$ so that $\|\mathbf{s}_1\| \leq \alpha, \mathbf{s}_2 \leftarrow \mathcal{S}_v^{m_2}$
Public information:
$\mathbf{A}_1 \in \mathcal{R}_{d,q}^{n \times m_1}, \quad \mathbf{A}_2 \in \mathcal{R}_{d,q}^{n \times m_2}, \quad \mathbf{B} \in \mathcal{R}_{d,q}^{\ell \times m_2}, \quad \mathbf{b} \in \mathcal{R}_{d,q}^{m_2},$
$$\begin{bmatrix} \mathbf{t}_A \\ \mathbf{t}_B \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{0} \end{bmatrix} \cdot \mathbf{s}_1 + \begin{bmatrix} \mathbf{A}_2 \\ \mathbf{B} \end{bmatrix} \cdot \mathbf{s}_2 + \begin{bmatrix} \mathbf{0} \\ \mathbf{m} \end{bmatrix},$$
$f_1, \ldots, f_N : \mathcal{R}_{d,q}^{k(m_1+\ell)} \to \mathcal{R}_{d,q}, \sigma \in \text{Aut}(\mathcal{R}_{d,q})$

Prover		**Verifier**
	$\xleftarrow{\mu_1, \ldots, \mu_N}$	$\mu_1, \ldots, \mu_N \leftarrow \mathcal{R}_{d,q}$
$f := \sum_{j=1}^{N} \mu_j f_j$ Run $\Pi^{(2)}((\mathbf{s}_2, \mathbf{s}_1, \mathbf{m}), \sigma, f)$ (See Table 3.11)		

Table 3.11: [**Figure 6** [32]] Commit-and-prove sub-protocol $\Pi^{(2)}((\mathbf{s}_2, \mathbf{s}_1, \mathbf{m}), \sigma, f)$.

Private information:
$(\mathbf{s}_1, \mathbf{m}) \in \mathcal{R}_{d,q}^{m_1+\ell}$ so that $\|\mathbf{s}_1\| \leq \alpha, \mathbf{s}_2 \leftarrow \mathcal{S}_v^{m_2}$
Public information:
$\mathbf{A}_1 \in \mathcal{R}_{d,q}^{n \times m_1}, \quad \mathbf{A}_2 \in \mathcal{R}_{d,q}^{n \times m_2}, \quad \mathbf{B} \in \mathcal{R}_{d,q}^{\ell \times m_2}, \quad \mathbf{b} \in \mathcal{R}_{d,q}^{m_2},$
$\begin{bmatrix} \mathbf{t}_A \\ \mathbf{t}_B \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{0} \end{bmatrix} \cdot \mathbf{s}_1 + \begin{bmatrix} \mathbf{A}_2 \\ \mathbf{B} \end{bmatrix} \cdot \mathbf{s}_2 + \begin{bmatrix} \mathbf{0} \\ \mathbf{m} \end{bmatrix},$
$r_0 \in \mathcal{R}_{d,q}, \ \mathbf{r}_1 \in \mathcal{R}_{d,q}^{k(m_1+\ell)}, \ \mathbf{R}_2 \in \mathcal{R}_{d,q}^{k(m_1+\ell) \times k(m_1+\ell)}, \ \sigma \in \mathrm{Aut}(\mathcal{R}_{d,q}),$
$\mathbf{s}^\top \mathbf{R}_2 \mathbf{s} + \mathbf{r}_1^\top \mathbf{s} + r_0 = 0$

Prover		**Verifier**
$\mathbf{s} := \begin{bmatrix} (\sigma^i(\mathbf{s}_1))_{i \in [k]} \\ (\sigma^i(\mathbf{m}))_{i \in [k]} \end{bmatrix}$ $\mathbf{y}_1 \leftarrow D_{\mathfrak{s}_1}^{m_1}$ $\mathbf{y}_2 \leftarrow D_{\mathfrak{s}_2}^{m_2}$ $\mathbf{w} := \mathbf{A}_1\mathbf{y}_1 + \mathbf{A}_2\mathbf{y}_2$ $\mathbf{y} := \begin{bmatrix} (\sigma^i(\mathbf{y}_1))_{i \in [k]} \\ -(\sigma^i(\mathbf{B}\mathbf{y}_2))_{i \in [k]} \end{bmatrix}$ $g_1 := \mathbf{s}^\top \mathbf{R}_2 \mathbf{y} + \mathbf{y}^\top \mathbf{R}_2 \mathbf{s} + \mathbf{r}_1^\top \mathbf{y}$ $t := \mathbf{b}^\top \mathbf{s}_2 + g_1$ $v := \mathbf{y}^\top \mathbf{R}_2 \mathbf{y} + \mathbf{b}^\top \mathbf{y}_2$		
	$\xrightarrow{\mathbf{w}, t, v}$	
		$c \in \mathcal{C}$
$\mathbf{z}_1 := c\mathbf{s}_1 + \mathbf{y}_1$ $\mathbf{z}_2 := c\mathbf{s}_2 + \mathbf{y}_2$ for $i = 1, 2$: if $\mathrm{Rej}_i(\mathbf{z}_i, c\mathbf{s}_i, \mathfrak{s}_i) = 1$ then $\mathbf{z}_1, \mathbf{z}_2 := \perp$	$\xleftarrow{c}$	
	$\xrightarrow{\mathbf{z}_1, \mathbf{z}_2}$	$\mathbf{z} := \begin{bmatrix} (\sigma^i(\mathbf{z}_1))_{i \in [k]} \\ (\sigma^i(c\mathbf{t}_B - \mathbf{B}\mathbf{z}_2))_{i \in [k]} \end{bmatrix}$ $f := ct - \mathbf{b}^\top \mathbf{z}_2$ Accept iff $\|\mathbf{z}_1\| \leq \mathfrak{s}_1 \sqrt{2m_1 d}$ and $\|\mathbf{z}_2\| \leq \mathfrak{s}_2 \sqrt{2m_2 d}$ $\mathbf{A}_1\mathbf{z}_1 + \mathbf{A}_2\mathbf{z}_2 = \mathbf{w} + c\mathbf{t}_A$and $\mathbf{z}^\top \mathbf{R}_2 \mathbf{z} + c\mathbf{r}_1^\top \mathbf{z} + c^2 r_0 - f = v$

The Regev public key encryption scheme can be used to encrypt the witness (i.e. the opening of the commitment). This scheme is comprised of three algorithms that are shown in Table 3.12.

The zero-knowledge proof in [32] is used for proving the witness satisfying the relation in Equation 3.3.4.

- We need to prove the knowledge of $\mathbf{r}_i \in \mathcal{R}_N \cong \mathcal{R}_d^t$ with small norm where the $\mathbf{r}_i$ represent columns in the randomness matrix $\mathbf{R}$ (i.e. $\mathbf{R} = (\mathbf{r}_1, \ldots, \mathbf{r}_{\ell \cdot \tau})$, $t = N/d$, and $m \in \mathcal{R}_N$ is a binary polynomial with a specified ℓ_1-norm in equation 3.3.4.

- In addition, we also need to prove the well-formedness of ***cipher*** which is the ciphertext generated by the Regev Encryption algorithm (defined in table 3.12). This is equivalent to proving the existence of a small $\mathbf{r}_{\text{PKE}} \in \mathcal{R}_d^{m_{\text{PKE}}}$ that satisfies the Regev Encryption scheme for the message $\mu = (\mathbf{r}_1 \| \ldots \| \mathbf{r}_{\ell \cdot \tau} \| m)$ with $\mathbf{R} = (\mathbf{r}_1, \ldots, \mathbf{r}_{\ell \cdot \tau})$.

To do this concretely, we can first commit to the vector $(\mathbf{r}_{\text{PKE}} \| \mathbf{r}_1 \| \ldots \| \mathbf{r}_{\tau \cdot \ell} \| m)$ through using ABDLOP commitment in [32]. Then, we prove through several linear relations including equation 3.3.4, Regev's Encryption algorithm, that m is a binary polynomial, as well as the following bounds:

$$(i) \quad \|\mathbf{r}_{\text{PKE}}\|_2 \leq 2\sqrt{d \cdot m_{\text{PKE}}}; \quad (ii) \quad \|\mathbf{r}_i\|_2 \leq \sqrt{kN}; \quad (iii) \quad \|m\|_1 = \omega.$$

The NIZKPoK construction from [32] is considered correct and IND-CPA secure from the M-LWE$_{q_{\text{PKE}}, n_{\text{PKE}}, m_{\text{PKE}}, \hat{S}_2}$ problem, where $\hat{\mathcal{S}}_2 = \{a \in \mathcal{R}_d \mid \|a\|_\infty \leq 2\}$.

Table 3.12: The ring-based Regev encryption scheme algorithms as depicted in [28].

Let $\mathcal{R}_{d,q_{\text{PKE}}} = \mathbb{Z}_{q_{\text{PKE}}}[X]/(X^d+1)$, $\mathcal{R}_d = \mathbb{Z}[X]/(X^d+1)$, and $\hat{\mathcal{S}}_2 = \{a \in \mathcal{R}_d \mid \|a\|_\infty \leq 2\}$.
`KeyGen`(λ) — given security parameter λ as input, output the secret key $sk = \mathbf{S}$ and public key $pk = (\mathbf{A}_{\text{PKE}}, \mathbf{B}_{\text{PKE}})$. 1. Set $d, q_{\text{PKE}}, n_{\text{PKE}}, m_{\text{PKE}}, k_{\text{PKE}} \in \mathbb{Z}$, where d is a power-of-two and q_{PKE} is a prime. 2. Sample $\mathbf{A}_{\text{PKE}} \xleftarrow{\$} \mathcal{R}_{d,q_{\text{PKE}}}^{n_{\text{PKE}} \times m_{\text{PKE}}}$, $\mathbf{S} \leftarrow \hat{S}_2^{k_{\text{PKE}} \times n_{\text{PKE}}}$, $\mathbf{E} \leftarrow \hat{S}_2^{k_{\text{PKE}} \times m_{\text{PKE}}}$. 3. Compute $\mathbf{B}_{\text{PKE}} = \mathbf{S} \cdot \mathbf{A}_{\text{PKE}} + 3 \cdot \mathbf{E} \in \mathcal{R}_{d,q_{\text{PKE}}}^{k_{\text{PKE}} \times m_{\text{PKE}}}$. 4. Output $pk := (\mathbf{A}_{\text{PKE}}, \mathbf{B}_{\text{PKE}})$, $sk := \mathbf{S}$.
`Encrypt`(pk, μ) — given inputs public key pk and secret message $\mu \in \mathcal{R}_{d,q_{\text{PKE}}}^{k_{\text{PKE}}}$, output a ciphertext $\boldsymbol{cipher} = (\mathbf{c}_0, \mathbf{c}_1)$. 1. Sample $\mathbf{r}_{\text{PKE}} \xleftarrow{\$} \hat{\mathcal{S}}_2^{m_{\text{PKE}}}$. 2. Compute $c_0 = \mathbf{A}_{\text{PKE}} \cdot \mathbf{r}_{\text{PKE}} \in \mathcal{R}_{d,q_{\text{PKE}}}^{n_{\text{PKE}}}$, $c_1 = \mathbf{B}_{\text{PKE}} \cdot \mathbf{r}_{\text{PKE}} + \mu \in \mathcal{R}_{d,q_{\text{PKE}}}^{k_{\text{PKE}}}$. 3. Output $\boldsymbol{cipher} = (c_0, c_1)$.
`Decrypt`$(pk, sk, \boldsymbol{cipher})$ — given public key pk, secret key sk, and the ciphertext $\boldsymbol{cipher}$ as inputs, output the message $\mu = \mu' \pmod 3$. 1. Compute $\mu' = c_1 - \mathbf{S} \cdot c_0 \in \mathcal{R}_{d,q_{\text{PKE}}}^{k_{\text{PKE}}}$, where $\boldsymbol{cipher} = (\mathbf{c}_0, \mathbf{c}_1)$. 2. Output $\mu' \pmod 3$.

3.3.4 Anonymous Credentials

The anonymous credential scheme of [28] is built from a secure commit-transferable signature scheme with the algorithms CTS.*Setup*, CTS.*Commit*, CTS.*Randomize*,

CTS.*KeyGen*, CTS.*Sign*, CTS.*Transfer*, CTS.*Verify*, as well as an efficient mult-theorem straight-line extractable NIZKPoK Π comprised of the algorithms, NIZK-Setup, NIZKProve, NIZKVerify, and SimSetup. The CTS.*Transfer* function helps to preserve anonymity, by creating pseudonyms of valid anonymous credentials.

The anonymous credential system's functions can be seen in Tables 3.13 and 3.14.

In [28], an algorithm for revealing attributes is also presented in interactive form. We show this in Table 3.15.

Table 3.13: The anonymous credential scheme algorithms, `Setup` and `Registration` as depicted in [28].

Let $\mathcal{M}_{\text{params}}$ denote the message space, $\mathcal{R}_{\text{params}}$ denote the ring used, and $\mathbf{R}, \mathbf{R}', \mathbf{R}'' \in \mathcal{R}_{\text{params}}$ denote randomness matrices.
`Setup`(1^λ) — given input λ, outputs public parameters $params = (\text{ CTS. } params, \text{NIZK.}params)$, and keys for the prover and certificate authority, usk and (pk_O, sk_O), respectively. 1. Run CTS.*Setup* to obtain CTS.$params$. 2. Run NIZKSetup(CTS.$params$) to obtain NIZK.$params$. 3. An honest prover generates their secret key usk by sampling $\mathcal{M}_{params}$. 4. An honest certificate authority O generates its public and secret keys $(pk_O, sk_O) \leftarrow$ CTS.*KeyGen*(CTS.$params$).
`Registration`$(params; usk)$ — given the public parameters $params$, and the secret key usk as inputs, the prover generates a pseudonym nym and NIZK proof π and sends it to the certificate authority. 1. Prover samples randomness $\mathbf{R} \leftarrow \mathcal{R}_{\text{params}}$ and generates a commitment $\boldsymbol{comm} \leftarrow$ CTS.`Commit`$(params, usk, \mathbf{R})$. 2. Prover generates an NIZK proof $\pi \leftarrow$NIZKProve$(params, \boldsymbol{comm}, usk, \mathbf{R})$. 3. Prover sends $nym = (\boldsymbol{comm}, \pi)$ as the pseudonym to certificate authority O. 4. O runs NIZKVerify to check whether nym is well-formed.

Table 3.14: The anonymous credential scheme algorithms: `Issue`, `Prove` and `Verify`, as depicted in [28].

Let $\mathcal{M}_{\text{params}}$ denote the message space, $\mathcal{R}_{\text{params}}$ denote the ring used, and $\mathbf{R}, \mathbf{R}', \mathbf{R}'' \in \mathcal{R}_{\text{params}}$ denote randomness matrices.
`Issue`($comm$, π, sk_O) — given the prover's commitment $comm$ and associated proof π, and the certificate authority's secret key sk_O as inputs, the certificate authority issues a credential Sig_{comm} to a prover. 1. Prover is known to certificate authority O as $nym = (\boldsymbol{comm}, \pi)$. 2. O computes and gives $Sig_{comm} \leftarrow CTS.\texttt{Sign}(params, sk_O, \boldsymbol{comm})$ to the prover.
`Prove`(nym_2, pk_O; usk, nym_1, Sig_{comm}) — the prover has a credential Sig_{comm} from certificate authority O. The prover is known to the certificate authority as nym_1 and to a verifier as nym_2, and proves to the verifier that they have a valid $Sig_{comm'}$ issued by O. 1. Prover samples $\mathbf{R}'$ and runs $\boldsymbol{comm}' \leftarrow$CTS.$\texttt{Randomize}(\boldsymbol{comm}, \mathbf{R}, \mathbf{R}')$. 2. Prover computes signature: $Sig_{comm'} = CTS.\texttt{Transfer}(\text{params}, pk_O, usk, \mathbf{R}, \mathbf{R}', Sig_{comm})$. 3. Prover gives the verifier $Sig_{comm'}$ and $nym' = (\mathbf{comm}', \pi')$, where π' is an NIZK proof that $\boldsymbol{comm}'$ is well-formed.
`Verify`(nym', $Sig_{comm'}$) — the verifier verifies if the prover possesses a credential $Sig_{comm'}$ from the certificate authority with respect to the prover's pseudonym nym'. 1. Verifier runs CTS.$\texttt{Verify}(params, pk_O, \boldsymbol{comm}', Sig_{comm'}$ and the NIZK verifier of π' on input $(Sig_{comm'}, nym' = (\mathbf{comm}', \pi'))$ to verify the prover's credential on the pseudonym nym'.

Table 3.15: [**Table 17** [28]] The interactive version of $\Pi_{\text{Disclosure}}$. $\mathit{\Pi}_{\text{Disclosure}}$ discloses that certain attributes are in the committed message m.

Parameters:
Some of the matrices used are from the CTS construction seen in Section 3.3.2 (e.g., $\mathbf{A}_1, \mathbf{a}_2^\top, \boldsymbol{comm}, \mathbf{r}_1, m, \mathbf{G}$). The rejection sampling algorithm, Rej_1, can be found in Table 2.3.

Public parameters:
$\mathbf{A}_1, \mathbf{a}_2, B = \sigma \cdot \sqrt{2k \cdot N}$, $\mathbf{B}^\top = (\mathbf{b}_i) \in \mathcal{R}_q^{k \times \hat{k}}$
Commitment: $\boldsymbol{comm} := \begin{bmatrix} \mathbf{t}_{1,1} \\ \mathbf{t}_{2,1} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{a}_2^\top \end{bmatrix} \cdot \mathbf{r}_1 + \begin{bmatrix} 0 \\ m \cdot \mathbf{G} \end{bmatrix}$, with $m = \sum_{i \in [N]} m_i X^{i-1}$ and $\mathbf{G} = \mathbf{I}_\ell \otimes (1, \delta, \ldots, \delta^{\tau-1})$.

Let $\mathcal{I} \subseteq [N]$ denote the subset of attribute indices the prover wants to disclose.
Let $m_{\mathcal{I}} = \{m_i\}_{i \in \mathcal{I}}$ represent the attributes to be disclosed.
Set $m_{\text{att}} = \sum_{i \in \mathcal{I}} m_i X^{i-1}$ and $m' = \sum_{i \in [N] \setminus \mathcal{I}} m_i X^{i-1}$.
The prover discloses m_{att} publicly to the verifier.

Prover		**Verifier**
$D = \{f \in \mathcal{R}_{q_2} : f_i = 0 \text{ for } i \in \mathcal{I}\}^{\hat{k}}$ $\mathbf{g} := (g_1, \ldots, g_{\hat{k}})^\top \xleftarrow{\$} D$ $\mathbf{t}_g = (t_{g,i}) = \mathbf{B} \cdot \mathbf{r}_1 + \mathbf{g}$	$\xrightarrow{\mathbf{t_g}}$ $\xleftarrow{(\gamma_i)_{i \in [\hat{k}]}}$	$(\gamma_i) \leftarrow \mathbb{Z}_{q_2}^{\hat{k}}$
$\forall i \in [\hat{k}], h_i = g_i + \gamma_i \cdot m'$ $\mathbf{y} \leftarrow D_\xi^k, \mathbf{w} = \mathbf{A}_1 \cdot \mathbf{y}$ $\forall i \in [\hat{k}], w_i = (\gamma_i \cdot \mathbf{a}_2^\top + \mathbf{b}_i^\top) \cdot \mathbf{y}$	$\xrightarrow{\mathbf{w},\ h_i,\ w_i}$	
	$\xleftarrow{d}$	$d \leftarrow \mathcal{C}$
$\mathbf{z} = \mathbf{y} + d \cdot \mathbf{r}_1$ $\text{Rej}_1(\mathbf{z}, d \cdot \mathbf{r}_1, \sigma)$	$\xrightarrow{\mathbf{z}}$	**Checks:** 1. $\lVert\mathbf{z}\rVert \overset{?}{\leq} B, \quad \mathbf{A}_1 \cdot \mathbf{z} \overset{?}{=} \mathbf{w} + d \cdot \mathbf{t}_{1,1}$ 2. $\forall i \in [\hat{k}]$, whether the coefficients with respect to $\mathcal{I}$ in h_i are zero, and $(\gamma_i \cdot \mathbf{a}_2^\top + \mathbf{b}_i^\top) \cdot \mathbf{z} \overset{?}{=} w_i + d \cdot (\gamma_i \cdot (\mathbf{t}_{2,1} - \overline{m_{\text{att}}}) + t_{g,i} - h_i)$.

3.4 Security Notions

The security descriptions are taken from [7], which are the same security descriptions used in the construction of the anonymous credential scheme from [28]. In particular, anonymity, unlinkability, and unforgeability are properties the anonymous credential scheme satisfies.

For anonymity, we consider privacy against verifiers and certificate authorities. In the former case, the proof of a credential shown by the prover should not leak any information other than its validity. In the latter case, certificate authorities cannot distinguish between two different users with different private inputs in the registration process.

Unlinkability says that adversaries cannot distinguish whether (nym_1, π_1) and (nym_2, π_2) are from the same prover or not. Here, the proof of credentials for each respective pseudonym nym_1 and nym_2 are π_1 and π_2.

Finally, unforgeability is the notion that adversaries cannot provide a valid proof of credential Sig_{comm^*} with respect to a pseudonym nym^* of some secret key usk^* if the adversary has never received a valid credential from a certificate authority.

3.4.1 Secure Scheme

The following theorem from [28] states that the anonymous credential scheme based on CTS is secure.

Theorem 3.4.1 (**Theorem 6.2 from [28]**). *Assuming that CTS is secure for the exact commitment relation $\hat{\mathcal{L}}$ (seen in Equation 3.3.2), and $\Pi^{(2)}$ is a secure multi-theorem straight-line extractable NIZKPoK system for $\hat{\mathcal{L}}$, the anonymous credential system depicted in Tables 3.13 and 3.14 is secure.*

Proof: A proof sketch follows.

Anonymity: Anonymity against the organization follows from the security of NIZKPoK and the hiding property of the commitment scheme. Anonymity against the verifier follows from the simulatability of CTS since the transferred signature does not leak information beyond its validity.

Unlinkability: The unlinkability follows from the hiding property of the re-randomized commitments and the simulatability of the CTS, so that any verifier cannot link two pairs of pseudonym-proofs.

Unforgeability: For proving unforgeability, the NIZPoK extractor of the commitment relation is used as well as the unforgeability of the CTS. If there is an adversary $\mathcal{A}$ that forges a valid proof of the anonymous credential, then we can construct a reduction $\mathcal{B}$ that breaks CTS unforgeability:

- $\mathcal{B}$ simulates the NIZKPoK and extracts $\mathcal{A}$'s private values: message m, and randomness $\mathbf{R}$ in the commitment of the registration queries from the ZKPoK proof provided.

- When $\mathcal{A}$ makes an issue query, $\mathcal{B}$ makes a signing query to the CTS challenger,
- $\mathcal{B}$ extracts the witness from the commitment and makes a signing query,
- As long as $\mathcal{A}$ can forge a valid proof, $\mathcal{B}$ can break the CTS unforgeability.

It is easy to verify that as long as $\mathcal{A}$ can forge a valid proof, $\mathcal{B}$ can break the CTS unforgeability. ∎

Chapter 4

Implementation

To build the anonymous credentials scheme, we will need to build a commit-transferable signature. As we saw in Chapter 3, the commit-transferable signature relies on the BDLOP commitment scheme and a non-interactive proof of knowledge scheme. In this chapter, we discuss the implementation process of these components. We build and implement the functions outlined in Tables 3.13 and 3.14. We also discuss technologies used, modifications made to the scheme, features added, and parameter selection.

We chose to use a high-level programming language, Python, to implement the system, which is a departure from the low-level languages used in implementations of existing work. The purpose of our work is to create an open source code base that not only helps to diversify implementation approaches, but also to provide greater accessibility and ease of implementation for users who may want to adapt the protocol further. Due to our own familiarity with the language as well as its popularity within the coding community, we chose to use Python. The disadvantage of using a language like Python is that we could not truly optimize our code to run more efficiently compared to a low-level language. Despite this, the benefits of accessibility and diverging from previous implementation approaches seemed to outweigh the disadvantages.

The main contribution of our work, aside from a full implementation, is also the extension of new features. These features are notably the disclosure of properties of attributes, which include the NOT property, OR property, and RANGE property.

4.1 Design and Methodology

The main building blocks for the anonymous credential algorithm include the commit-transferable signatures with an interactive zero-knowledge proof of knowledge (ZKPoK). For our ease of implementation, we have kept the ZKPoK interactive [28]. The non-interactive version requires the use of hash functions which adds a layer of difficulty and complexity in the implementation process. We leave this as future work.

Notably, we use the ZKPoK from [32] for both the commitment well-formedness proof (Equation 3.3.2) and the credential validity proof (Equation 3.3.1). The credential validity proof system was not specified in [28]. However, the ZKPoK system from [32] can easily accomodate the credential validity proof as well. We opted to use the same proof system for code re-usability purposes.

The file structure and user-friendly description of all functions and how to use the code are provided in the Appendix A.

We use the *lattice-zk* package by Florian Lugstein [31] for the interactive proof of knowledge portion of our implementation[1]. Initially, we implemented an interactive proof of knowledge using mainly the Python package, *numpy*. However, *numpy* does not have built-in functions for dealing with multiplication between matrices over polynomial rings. As the degrees of our polynomial rings were initially quite high, the runtime of our code was also very high. Thus, we found and started using the *lattice-zk* package based on the interactive proof of knowledge of [32]. The *lattice-zk* package implemented by Lugstein is mostly dependent on *SageMath* functions, which are equipped to handle multiplication between matrices over polynomial rings and much more.

4.2 Modifications and Features

4.2.1 Modifications

Due to the long run-time of our code, we looked into minimizing the parameter values for the scheme. When the protocol was initially run with the original parameters given in [28], one of the biggest bottlenecks in runtime was matrix multiplication over high-degree polynomial rings. Often times, running one iteration of the protocol would take several hours, ultimately terminating due to insufficient memory. To combat this problem, many of the parameters were recalculated to smaller values. The security of the scheme with the adapted values is 128-bit security. We discuss bit-security calculations later in this section.

Calculation of Parameters

The values were recalculated based on the information obtained in [28, 32]. Table 4.1 highlights the general parameters of the anonymous credential scheme. There are two IZKPoK schemes in the protocol—the well-formedness of a commitment IZKPoK (from Equation 3.3.2) can be initiated by the parameters in Table 4.2 and 4.3, while the validity of the credential IZKPoK (from Equation 3.3.1) can be initiated with the parameters from Table 4.4 and 4.5. We note that the ring degrees of these two IZKPoKs are different. We set $d = 128$ for the commitment well-formedness proof

[1] The *lattice-zk* code can be found here: https://lattice-zk.isec.tugraz.at/

and $d = 512$ for the credential validity proof. Finally, concrete sizes of the scheme are shown in Table 4.6.

We chose to minimize the degrees of the anonymous credential ring by a factor of 4, and thus set $N = 512$. In addition, we set $\tau = 2$ and $\omega = 22$. We used these parameter values as input to calculate the other parameters of interest in the anonymous credentials scheme. We calculated and chose values that would ultimately give us 128-bit security of the scheme.

For the parameters related to the interactive proof of knowledge protocol associated with commitment well-formedness, we adapted and ran the provided code from [32] to obtain the values $q_1, q_2, q_{\text{LNP}}, m_2$, and the Root Hermite factors[2]. We ran the code from [32] with the ring degree we had chosen as input, and we obtained parameters that had been calculated to meet the security requirement (128-bit security). The Root Hermite factor calculated from the code for the anonymous credentials scheme in terms of the M-SIS problem is 1.003218. The Root Hermite factors for the proof system in terms of M-SIS and M-LWE problems respectively are 1.004389 and 1.004434. We kept the number of factors that $X^d + 1$ splits into q_{LNP} as $l = 2$. These factors represent prime factors of q_{LNP} and are fitting for our scheme since our commitment message uses two prime moduli for our CTS-based anonymous credential which are q_1 and q_2. In other words, $q_{\text{LNP}} = q_1 \cdot q_2$.

We used some of the same values provided in [28] for the dimensions of matrices used in the BDLOP commitment (i.e. n, ℓ, k) as those were already minimized.

The size of the message space was determined by $|\mathcal{M}| = \binom{d}{\omega} = \binom{128}{22} \approx 2^{81}$.

For the encryption parameters, the modulus, as well as the height and width of the public matrices used were calculated as follows:

- Set $n_{\text{PKE}} = 1$
- Set $k_{\text{PKE}} = \ell \cdot \tau \cdot N/d \cdot k + (N/d) = 36$
- Set $m_{\text{PKE}} = 2 \cdot n_{\text{PKE}} + k_{\text{PKE}} = 38$
- Set a modulus $q_{\text{PKE}} > 2(12 \cdot m_{\text{PKE}} \cdot d + 1)$. Thus, we chose the lowest prime greater than $2(12 \cdot m_{\text{PKE}} \cdot d + 1) = 58370$, which is $q_{\text{PKE}} = 58373$.

The challenge set used in the ZKPoK is denoted:

$$\mathcal{C} = \{c \in \mathcal{R} : ||c||_1 = \kappa^*, ||c||_\infty = 1\}.$$

The maximum coefficient of a challenge in $\mathcal{C}$, $\kappa^* = 2$ remained the same.

For the ABDLOP portion of the ZKPoK, we set the height (i.e. the number of rows) of the matrices of $\mathbf{A}_1$ and $\mathbf{A}_2$ to $n^* = 1$. The length of the message $\mathbf{s}_1$ is $m_1 = m_{\text{PKE}} + (\ell \cdot \tau \cdot k) \cdot N/d + 4 = 74$. We kept the length of the message m

[2] https://github.com/khanhcrypto/LBZKP

Table 4.1: [**Adapted from Table 10** [28]] Parameter selection for the **CTS**-based anonymous credentials.

Variable	Description	Parameter Value
q_1	moduli for the **BDLOP** commitment	$2^{32} - 299$
q_2		$2^{26} - 27$
n	number of rows of $\mathbf{A}_1$ the **BDLOP** commitment	1
k	number of columns of $\mathbf{A}_1$ the **BDLOP** commitment	4
ℓ	number of rows of $\mathbf{A}_2$ in the **BDLOP** commitment	1
$\hat{\ell}$	number of columns of $\mathbf{D}$ in **CTS** scheme	2
τ	dimension of $\mathbf{g} = (1, \delta, \ldots, \delta^{\tau-1})$ with $\delta = q_2^{1/\tau}$	2
N	dimension of ring for **CTS**	512
ω	number of 1's in the identity $m \in \mathcal{M}$	22
$\lvert\mathcal{M}\rvert$	size of the user space	$\approx 2^{81}$
η	$\sigma = \eta \cdot T$ is the standard deviation for rejection sampling	7.6
α	parameter used in SamplePre	1064960
	abort time for rejection sampling	6

to $\ell^* = 0$ since the prover's secret attributes will be committed in the $\mathbf{s}_1$ vector instead. For the length of $\mathbf{s}_2$, we determined the value by running the code from [32] as discussed earlier. To calculate the gaussian parameter used in SamplePre, we use $\alpha = 2\sqrt{\delta^2 + 1} \cdot \left(\left(\sqrt{2\ell} + \sqrt{\ell \cdot \tau}\right)\sqrt{N} + 1\right)$ as in [28].

In terms of instantiating the other variables for the ZKPoK (such as the repetition rates), we have taken to using the original variables from [32], as can be seen in Table 4.3.

The second IZKPoK for proving the validity of a credential follows similarly in terms of calculation as the first IZKPoK. We refer to the parameter instantiations in Section 6.2 in [32] which detail the proof of knowledge of an M-LWE secret. This application can be easily transferred to prove the M-SIS secret, which is relevant for

Table 4.2: [**Adapted from Table 11** [28]] Parameter selection for the commitment well-formedness **IZKPoK**, where the root-hermite factor is 1.003687.

Variable	Description	Parameter Value
q_{LNP}	modulus for the proof system (i.e. $q_{\text{LNP}} = q_1 \cdot q_2$)	$\approx 2^{58}$
d	ring dimension of proof system	128
q_{PKE}	encryption modulus	58373
n_{PKE}	height(# rows) of $\mathbf{A}_{\text{PKE}}$	1
m_{PKE}	width (# columns) of $\mathbf{A}_{\text{PKE}}$	38
k_{PKE}	height (# rows) of $\mathbf{B}_{\text{PKE}}$	36
l	# factors $X^d + 1$ splits into mod q_{LNP}	2
γ_1	rejection sampling constant for $c\mathbf{s}_1$	17
γ_2	rejection sampling constant for $c\mathbf{s}_2$	1.2
$\gamma^{(e)}$	rejection sampling constant exact **ARP**	25
$\gamma^{(d)}$	rejection sampling constant non-exact **ARP**	12
η^*	upper bound of $\sqrt[2k]{\|\|c^{2k}\|\|}$ for $k = 32$	59
κ^*	maximum coefficient of a challenge in $\mathcal{C}$	2
n^*	height (number of rows) of matrices $\mathbf{A}_1, \mathbf{A}_2$ in **ABDLOP**	1
m_1	length of the message s_1 in the Ajtai commitment	74
ℓ^*	length of the message m in the BDLOP commitment	0
m_2	length of the message s_2 in **ABDLOP**	32
ν^*	randomness s_2 is sampled from $\mathcal{S}_{\nu^*}^{m_2}$	1
γ^*	parameters to cut low-order bits from w	979498
D^*	number of low-order bits cut from t_A	12
$\hat{k}$	repetition times for attribute disclosure	2
	repetition rate	7
γ'	ℓ_2 norm bound on the credential	43543104798560.02

the credential validity proof. To calculate the bound γ' on the credential Sig_{comm} (i.e. the relation represented by Equation 3.3.1), we use the following equation from Theorem 4.3 in [32]:

$$\gamma' = 2\sqrt{2 \cdot (\ell \cdot (2\tau + 1) + \hat{\ell} + k - n) \cdot N} \cdot \eta \cdot \kappa^* \cdot \tag{4.2.1}$$
$$\left((\sqrt{k-n} + \sqrt{\ell \cdot \tau}) \cdot N \cdot \alpha\sqrt{2 \cdot \ell \cdot \tau} + \alpha\sqrt{2 \cdot (\ell \cdot (2\tau + 1) + \hat{\ell} + k - n) \cdot N} \right).$$

Finally, we calculated the sizes for each IZKPoK in Table 4.6 according to the equations given in [28]:

- $|\text{para}_{\text{ABDLOP}}| = \left(n^* \cdot (m_1 + m_2 + v_e) + (\ell^* + 2) \cdot m_2\right) \cdot d \cdot \lceil \log(q_{\text{LNP}}) \rceil + 512 \cdot m_2 \cdot \lceil \log(q_{\text{LNP}}) \rceil$ bits
- $|\text{para}_{\text{PKE}}| = (n_{\text{PKE}} + k_{\text{PKE}}) \cdot m_{\text{PKE}} \cdot d \cdot \lceil \log(q_{\text{PKE}}) \rceil$ bits
- $|\text{para}_{\text{Disclosure}}| = k \cdot \hat{k} \cdot N \cdot \lceil \log q_2 \rceil$ bits

Table 4.3: Instantiation of Table 3.8 for the commitment well-formedness **IZKPoK**.

Variable	Description	Instantiation
t	factor of d in N (i.e. $N = t \cdot d$)	4
ρ	# of equations to prove	$t_0 = \ell \cdot \tau \cdot t \cdot (n + \ell) = 16$
ρ_{eval}	# of evaluations with constant coefficient zero	1
v_{e}	# of exact norm proofs	$\ell \cdot \tau + 1 = 3$
v_{d}	# of non-exact norm proofs	1
k_{bin}	Length of the binary vector to prove	1
$\mathbf{s}_1$	Committed message in the Ajtai commitment	$(\boldsymbol{r}_{\text{PKE}}, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{\ell\cdot\tau}, m)$
$\mathbf{m}$	Committed message in the BDLOP commitment	$\emptyset$ (no message)
$f_1, \ldots, f_{t_0}$	Equations to prove	$\begin{bmatrix}\mathbf{C}_1\\ \mathbf{C}_2\end{bmatrix} = \begin{bmatrix}\mathbf{A}_1\\ \mathbf{A}_2\end{bmatrix} \cdot \mathbf{R} + \begin{bmatrix}\mathbf{0}^{n\times(\ell\cdot\tau)}\\ m \cdot \mathbf{G}\end{bmatrix} \begin{matrix}\in \mathcal{R}_{d,q_2}^{t\cdot n\times(\ell\cdot\tau)}\\ \in \mathcal{R}_{d,q_2}^{t\cdot\ell\times(\ell\cdot\tau)}\end{matrix}$
F_1	Evaluation to prove constant coefficient zero	$\sigma_{-1}(\sum_{i=0}^{d-1} X^i) \cdot m - \omega$
x	dimension for $\mathbf{E}$ matrix	$k \cdot t = 16$
$\mathbf{E}_1$	Public matrix for proving $\|\mathbf{E}_1\mathbf{s} - \mathbf{v}_1\| \leq \beta_1^{(e)}$	$[\mathbf{I}_{m_{\text{PKE}}} \quad \mathbf{0}^{m_{\text{PKE}}\times x} \ldots \mathbf{0}^{m_{\text{PKE}}\times x} \quad 0]$
$\mathbf{v}_1$	Public vector for proving $\|\mathbf{E}_1\mathbf{s} - \mathbf{v}_1\| \leq \beta_1^{(e)}$	$\mathbf{0}$
$\beta_1^{(e)}$	Upper bound on $\|\mathbf{E}_1\mathbf{s} - \mathbf{v}_1\| \leq \beta_1^{(e)}$	$2\sqrt{d \cdot m_{\text{PKE}}} = 185.2$
$\mathbf{E}_i$	Public matrix for proving $\|\mathbf{E}_i\mathbf{s} - \mathbf{v}_i\| \leq \beta_i^{(e)}$, where $i \in [2, ..., \ell \cdot \tau + 1]$	$[\mathbf{0}^{x\times m_{\text{PKE}}} \quad \mathbf{0}^{x\times(x\cdot(i-2))} \quad \mathbf{I}_x \quad \mathbf{0}^{x\times(x\cdot(\ell\cdot\tau+1-i))} \quad 0]$
$\mathbf{v}_i$	Public vector for proving $\|\mathbf{E}_i\mathbf{s} - \mathbf{v}_i\| \leq \beta_i^{(e)}$ where $i \in [2, ..., \ell \cdot \tau + 1]$	$\mathbf{0}$
$\beta_i^{(e)}$	Upper bound on $\|\mathbf{E}_i\mathbf{s} - \mathbf{v}_i\| \leq \beta_i^{(e)}$ where $i \in [2, ..., \ell \cdot \tau + 1]$	$\sqrt{N \cdot k} = 45.25$
$\mathbf{D}_1$	Public matrix for proving $\|\mathbf{D}_1\mathbf{s} - \mathbf{u}_1\| \leq \beta_1^{(d)}$	$q_{\text{PKE}}^{-1} \cdot \begin{bmatrix}\mathbf{A}_{\text{PKE}}, \mathbf{0}\\ \mathbf{B}_{\text{PKE}}, \mathbf{I}_{k_{\text{PKE}}}\end{bmatrix}$
$\mathbf{u}_1$	Public vector for proving $\|\mathbf{D}_1\mathbf{s} - \mathbf{u}_1\| \leq \beta_1^{(d)}$	$q_{\text{PKE}}^{-1} \cdot \begin{bmatrix}\boldsymbol{t}_0\\ \boldsymbol{t}_1\end{bmatrix}$
$\beta_1^{(d)}$	Upper bound on $\|\mathbf{D}_1\mathbf{s} - \mathbf{u}_1\| \leq \beta_1^{(d)}$	$(d \cdot m_{\text{PKE}} + 1)\sqrt{(n_{\text{PKE}} + 1) \cdot d} = 137232$
$\mathbf{E}_{\text{bin}}$	Matrix for proving binary	$[\mathbf{0} \quad \ldots \quad \mathbf{0} \quad 1]$
$\mathbf{v}_{\text{bin}}$	Vector for proving binary	$\mathbf{0}$

Table 4.4: [**Adapted from Table 11 in** [32]] Parameter selection for credential validity **IZKPoK**.

Variable	**Description**	**Parameter Value**
q_{LNP}	modulus for the proof system (i.e. $q_{\text{LNP}} = q_1 \cdot q_2$)	$\approx 2^{58}$
d	ring dimension of proof system	512
l	# factors $X^d + 1$ splits into mod q_{LNP}	2
γ_1	rejection sampling constant for $c\mathbf{s}_1$	19
γ_2	rejection sampling constant for $c\mathbf{s}_2$	1
$\gamma^{(e)}$	rejection sampling constant exact **ARP**	6
η^*	upper bound of $\sqrt[2k]{\|\|c^{2k}\|\|}$ for $k = 32$	59
κ^*	maximum coefficient of a challenge in $\mathcal{C}$	2
n^*	height (number of rows) of matrix $\mathbf{F}_{comm}$	1
m^*	width of matrix F_{comm}	10
m_1	length of the message s_1 in the Ajtai commitment	10
ℓ^*	length of the message m in the BDLOP commitment	0
m_2	length of the message s_2 in **ABDLOP**	25
ν^*	randomness s_2 is sampled from $\mathcal{S}^*_{\nu^*}$	1
γ^*	parameters to cut low-order bits from w	131052
D^*	number of low-order bits cut from t_A	9
$\hat{k}$	repetition times for attribute disclosure	2
	repetition rate	7

Table 4.5: Instantiation of Table 3.8 for the credential validity **IZKPoK**.

Variable	**Description**	**Instantiation**
ρ	# of equations to prove	0
ρ_{eval}	# of evaluations with constant coefficient zero	0
v_{e}	# of exact norm proofs	1
v_{d}	# of non-exact norm proofs	0
k_{bin}	Length of the binary vector to prove	0
$\mathbf{s}_1$	Committed message in the Ajtai commitment	Sig_{comm}
$\mathbf{m}$	Committed message in the BDLOP commitment	$\emptyset$ (no message)
$\mathbf{E}_1$	Public matrix for proving $\|\mathbf{E}_1\mathbf{s} - \mathbf{v}_1\| \leq \beta_1^{(e)}$	$\begin{bmatrix} \mathbf{I}_{m^*} \\ F_{\text{comm}} \end{bmatrix}$
$\mathbf{v}_1$	Public vector for proving $\|\mathbf{E}_1\mathbf{s} - \mathbf{v}_1\| \leq \beta_1^{(e)}$	$\begin{bmatrix} \mathbf{0} \\ \mathbf{u} \end{bmatrix}$
$\beta_1^{(e)}$	Upper bound on $\|\mathbf{E}_1\mathbf{s} - \mathbf{v}_1\| \leq \beta_1^{(e)}$	γ' (from equation 4.2.1)

Table 4.6: [**Adapted from Table 11** [28]] Public and private parameter sizes for the **IZKPoKs**.

Variable	Description	Size
Commitment Well-formedness IZKPoK		
$\lvert\text{para}_{\text{ABDLOP}}\rvert$	public parameter size of **ABDLOP** commitment	2902.8 KB
$\lvert\text{para}_{\text{PKE}}\rvert$	public parameter size of **PKE** encryption	9056.3 KB
$\lvert\text{para}_{\text{Disclosure}}\rvert$	public parameter size of **Disclosure** encryption	106.5 KB
$\lvert\boldsymbol{cipher}_{\text{PKE}}\rvert$	ciphertext size of **PKE** encryption	0.5 KB
Credential Validity IZKPoK		
$\lvert\text{para}_{\text{ABDLOP}}\rvert$	public parameter size of **ABDLOP** commitment	3286.3 KB

Pre-image Sampling

The pre-image sampling function is used to generate a short pre-image Sig_{comm} given public matrices F_{comm} and u in the `ISSUE` function shown in Table 3.14. Due to difficulties in implementing the pre-image sampling function, we implemented a workaround. The workaround involves generating the secret Sig_{comm} during the initialization of F_{comm}, and then multiplying the two matrices to get u. This alternative way of generating Sig_{comm} does not impact the underlying M-SIS assumption, as finding a short Sig_{comm} is still a difficult feat for adversaries. Given that our focus in this section of the thesis is to assess parameter sizes, we leave the proof of security to future work.

The pre-image sampling algorithm over $\mathbb{Z}_q$ is as follows [21]:

- Find a short basis for the dual lattice, using the trapdoor
- Find an arbitrary solution x_0, to $F_{\text{comm}} \cdot x_0 = u \bmod q$
- Use a variant of Babai's "nearest-plane" [5] to sample a vector v from the dual lattice shifted to $-x_0$. Babai's "nearest-plane" algorithm iteratively chooses a plane at random by sampling from a discrete Gaussian
- Output the short pre-image $t = x_0 + v$.

To use this algorithm, we first map our elements from $\mathcal{R}_q$ to $\mathbb{Z}_q$. A ring element $a \in \mathcal{R}$ can be mapped to a matrix in $\mathbb{Z}^{N\times N}$ by $\text{Rot} : \mathcal{R} \to \mathbb{Z}^{N\times N}$:

$$\text{Rot}(a) := \begin{bmatrix} \text{Coeffs}(a)^\top \\ \text{Coeffs}(xa \bmod (X^N+1))^\top \\ \vdots \\ \text{Coeffs}(x^{N-1}a \bmod (X^N+1)^\top \end{bmatrix},$$

where an element $a = \sum_{i\in[N]} a_i x^i \in \mathcal{R}$, where $[N] = \{0, 1, 2, \ldots, N-1\}$ can be represented by $\text{Coeffs}(a) = (a_0, \ldots, a_{N-1})$.

The main difficulty arose when using *sagemath*'s *DiscreteGaussianLatticeSampler* for the Babai "nearest-plane" variant algorithm. We mapped our matrices $F_{\text{comm}} \in \mathcal{R}_{N,q}^{n\times m}$ into $\mathbb{Z}_q^{n\cdot N\times m\cdot N}$, the dual lattice represented by $F_{\text{comm}}^{\perp} \in \mathcal{R}_{N,q}^{m\times \ell}$ into $\mathbb{Z}_q^{m\cdot N\times \ell\cdot N}$ and $u \in \mathcal{R}_{N,q}$ into $\mathbb{Z}_q^{N\times N}$. Sampling from *DiscreteGaussianLatticeSampler* produced vectors $\hat{v} \in \mathbb{Z}^{m\cdot N}$ that were not from the dual lattice. In our attempt to resolve the issue, we partitioned the input parameters—the basis and center matrices and later tried to reconstruct the matrix outputted by the function. We used this approach since the Rot map introduced a lot of redundancy in the values. This technique allowed us to successfully sample singular vectors in the dual lattice, however, the dimension required of $\hat{v}$ remained mismatched. Attempting to reconstruct it into

the proper dimensions $\hat{v} \in \mathbb{Z}^{m \cdot N \times N}$ ended up being very challenging. We exhausted the ways in trying different partitioning and reconstructing techniques, but to no avail.

Bit-security of the scheme

We followed the work of [3] and used their provided script to calculate the bit-security[3].

For the bit-security calculation, we consider two main forms of attacks that adversaries may use, primal attack and dual attack. Both attacks use the BKZ algorithm, which is a lattice basis reduction algorithm [39]. We primarily look at attacks that use the BKZ algorithm since it is commonly used in cryptanalysis of lattice-based schemes. The primal attack constructs a unique-SVP instance from the LWE problem and uses the BKZ algorithm. The dual attack, on the other hand, consists of finding a short vector in the dual lattice.

We inputted the following parameters of our scheme into the script:

- $q = q_2 = 2^{26} - 27$
- $n = N = 512$
- $\sigma = \eta \cdot T \approx 1041$

From the results in Table 4.7, where the attacks costs (in $\log_2$) are given, we can deduce that our scheme with the relevant parameters has (at least) a bit-security of 128.

Table 4.7: The cost of primal and dual attacks on the **CTS**-based anonymous credential scheme.

Attack	**Number of Used Samples**	**Block Dimension of the BKZ algorithm**	**Classical cost (in $\log_2$)**	**Quantum cost (in $\log_2$)**
Primal	1016	527	154	139
Dual	1020	525	153	139

[3]Their bit-security calculation script can be found here: https://github.com/tpoeppelmann/newhope

4.2.2 Features

The original protocol allows provers to disclose an attribute or a subset of attributes to the verifier. An extension of attribute-based anonymous credentials that we have designed and implemented is the disclosure of attributes' properties. We note that the anonymity property may not hold if the user decides to disclose an attribute or a property of an attribute that has identifying information (e.g., their home address). We do not consider these cases of revealing identifying information in our anonymity property.

We have extended the anonymous credential scheme to include disclosure of properties of attributes. This enables provers to disclose information about their attributes without revealing the attributes themselves to the verifier. We have added three algorithms to respectively allow provers to disclose that:

- an attribute is NOT a value (e.g., the age attribute is not 16),
- an attribute is one value OR another (e.g., the Canadian driver's license class is G1 or G2), and
- an attribute is within a non-arbitrary range (e.g., the age attribute is between 0 and 7). The main limitation with the range disclosure of the attribute is that it can only provide range information from $0 \leq attribute \leq 2^n - 1$, where n represents the number of bits in the attribute in binary form. A combination of these algorithms can actually provide arbitrary ranges, although in a less efficient way.

We have designed and implemented three property disclosure algorithms that can be seen in Tables 4.8, 4.9, and 4.10. These figures are an extension of the disclosure of attributes as seen in Table 3.15.

We note that for the OR property disclosure feature, we randomize the true and false values each time the prover engages with the verifier to accommodate any ordering of statements. For example, if Alice is 20, and wants to prove that her age is 20 or 21, an ordering is randomly chosen before the interaction (i.e. either "Alice is 20 OR 21" or "Alice is 21 OR 20").

Table 4.8: The interactive version of $\Pi_{\text{Discloses\_NOT}}$. $\Pi_{\text{Discloses\_NOT}}$ discloses that certain attributes are not in the committed message m.

Some of the matrices used are from the CTS construction seen in Section 3.3.2 (e.g., $\mathbf{A}_1, \mathbf{a}_2^\top, \boldsymbol{comm}, \mathbf{r}_1, m, \mathbf{G}$). The rejection sampling algorithm, Rej_1, can be found in Table 2.3.

Public parameters:
$\mathbf{A}_1, \mathbf{a}_2, B = \sigma \cdot \sqrt{2k \cdot N}$, $\mathbf{B}^\top = (\mathbf{b}_i) \in \mathcal{R}_q^{k \times \hat{k}}$, and $\mathbf{G} = \mathbf{I}_\ell \otimes (1, \delta, \ldots, \delta^{\tau-1})$.
Commitment: $\boldsymbol{comm} := \begin{bmatrix} \mathbf{t}_{1,1} \\ \mathbf{t}_{2,1} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{a}_2^\top \end{bmatrix} \cdot \mathbf{r}_1 + \begin{bmatrix} 0 \\ m \cdot \mathbf{G} \end{bmatrix}$, with $m = \sum_{i \in [N]} m_i X^{i-1}$.

Let $\mathcal{I} \subset [N]$ denote the subset of attribute indices where the prover wants to apply the NOT property on.
Let $m_\mathcal{I} = \{m_i\}_{i \in \mathcal{I}}$ denote a subset of attributes in which the NOT property will be applied.
Let $\overline{m_\mathcal{I}} = \{\overline{m_i}\}_{i \in \mathcal{I}}$ denote a subset of false attribute values.
Set $\overline{m_{\text{att}}} = \sum_{i \in \mathcal{I}} \overline{m_i} X^{i-1}$, and $m' = \sum_{i \in \mathcal{I}} (m_i - \overline{m_i}) X^{i-1} + \sum_{i \in [N] \setminus \mathcal{I}} m_i X^{i-1}$.
The prover discloses $\overline{m_{\text{att}}}$ publicly.

Prover		**Verifier**
$D = \{f \in \mathcal{R}_{q_2} : f_i = f_i \cdot (\overline{m_i} - m_i) \text{ for } i \in \mathcal{I}\}^{\hat{k}}$		
$\mathbf{g} := (g_1, \ldots, g_{\hat{k}})^\top \xleftarrow{\$} D$		
$\mathbf{t}_g = (t_{g,i}) = \mathbf{B} \cdot \mathbf{r}_1 + \mathbf{g}$	$\xrightarrow{\mathbf{t_g}}$	
	$\xleftarrow{(\gamma_i)_{i \in [\hat{k}]}}$	$(\gamma_i) \leftarrow \mathbb{Z}_{q_2}^{\hat{k}}$
$\forall i \in [\hat{k}], h_i = g_i + \gamma_i \cdot m'$		
$\mathbf{y} \leftarrow D_\xi^k, \mathbf{w} = \mathbf{A}_1 \cdot \mathbf{y}$		
$\forall i \in [\hat{k}], w_i = (\gamma_i \cdot \mathbf{a}_2^\top + \mathbf{b}_i^\top) \cdot \mathbf{y}$	$\xrightarrow{\mathbf{w},\ h_i,\ w_i}$	
	$\xleftarrow{d}$	$d \leftarrow \mathcal{C}$
$\mathbf{z} = \mathbf{y} + d \cdot \mathbf{r}_1$		
if $\text{Rej}_1(\mathbf{z}, d \cdot \mathbf{r}_1, \sigma) = 1$		
then $\mathbf{z} := \perp$	$\xrightarrow{\mathbf{z}}$	
		Checks:
		1. $\lVert\mathbf{z}\rVert \overset{?}{\leq} B, \quad \mathbf{A}_1 \cdot \mathbf{z} \overset{?}{=} \mathbf{w} + d \cdot \mathbf{t}_{1,1}$
		2. $\forall i \in [\hat{k}]$,
		whether the coefficients with respect to $\mathcal{I}$ in h_i
		are nonzero, and
		$(\gamma_i \cdot \mathbf{a}_2^\top + \mathbf{b}_i^\top) \cdot \mathbf{z} \overset{?}{=} w_i + d \cdot (\gamma_i \cdot (\mathbf{t}_{2,1} - \overline{m_{\text{att}}}) + t_{g,i} - h_i)$.

Table 4.9: The interactive version of $\Pi_{\text{Discloses\_OR}}$. $\Pi_{\text{Discloses\_OR}}$ discloses that certain coefficients are one value or another in the committed message m.

Some of the matrices used are from the CTS construction seen in Section 3.3.2 (e.g., $\mathbf{A}_1, \mathbf{a}_2^\top, \boldsymbol{comm}, \mathbf{r}_1, m, \mathbf{G}$). The rejection sampling algorithm, Rej_1, can be found in Table 2.3.

Public parameters:
$\mathbf{A}_1, \mathbf{a}_2, B = \sigma \cdot \sqrt{2k \cdot N}$, $\mathbf{B}^\top = (\mathbf{b}_i) \in \mathcal{R}_q^{k \times \hat{k}}$, and $\mathbf{G} = \mathbf{I}_\ell \otimes (1, \delta, \ldots, \delta^{\tau-1})$.
Commitment: $\boldsymbol{comm} := \begin{bmatrix} \mathbf{t}_{1,1} \\ \mathbf{t}_{2,1} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{a}_2^\top \end{bmatrix} \cdot \mathbf{r}_1 + \begin{bmatrix} 0 \\ m \cdot \mathbf{G} \end{bmatrix}$, with $m = \sum_{i \in [N]} m_i X^{i-1}$.

Let $\mathcal{I} \subset [N]$ denote the subset of attribute indices the prover wants to apply the OR property on.
Let $m_\mathcal{I} = \{m_i\}_{i \in \mathcal{I}}$ represent a subset of attributes in which the OR property will be applied.
Let $\overline{m_\mathcal{I}} = \{\overline{m_i}\}_{i \in \mathcal{I}}$ denote a subset of false attribute values.
Set $m_{\text{att}} = \sum_{i \in \mathcal{I}} m_i X^{i-1}$, $m' = \sum_{i \in [N] \setminus \mathcal{I}} m_i X^{i-1}$.
Set $\overline{m_{\text{att}}} = \sum_{i \in \mathcal{I}} \overline{m_i} X^{i-1}$, $\overline{m'} = \sum_{i \in \mathcal{I}} (m_i - \overline{m_i}) X^{i-1} + \sum_{i \in [N] \setminus \mathcal{I}} m_i X^{i-1}$.
The prover discloses $\overline{m_{\text{att}}}$ and m_{att} publicly.

Prover		**Verifier**
$(\gamma_i') \leftarrow \mathbb{Z}_{q_2}^{\hat{k}}$,		
$D = \{f \in \mathcal{R}_{q_2} : f_i = 0 \text{ for } i \in \mathcal{I}\}^{\hat{k}}$		
$\overline{D} = \{f \in \mathcal{R}_{q_2} : f_i = -\gamma_i' \cdot (m_i - \overline{m_i}) \text{ for } i \in \mathcal{I}\}^{\hat{k}}$		
$\mathbf{g} := (g_1, \ldots, g_{\hat{k}})^\top \xleftarrow{\$} D$		
$\overline{\mathbf{g}} := (\overline{g_1}, \ldots, \overline{g_{\hat{k}}})^\top \xleftarrow{\$} \overline{D}$		
$\mathbf{t}_g = (t_{g,i}) = \mathbf{B} \cdot \mathbf{r}_1 + \mathbf{g}$		
$\overline{\mathbf{t}_g} = \overline{(t_{g,i})} = \mathbf{B} \cdot \mathbf{r}_1 + \overline{\mathbf{g}}$	$\xrightarrow{\mathbf{t}_g, \overline{\mathbf{t}_g}}$	
	$\xleftarrow{(\gamma_i)_{i \in [\hat{k}]}}$	$(\gamma_i) \leftarrow \mathbb{Z}_{q_2}^{\hat{k}}$
$(\gamma'')_{i \in [\hat{k}]} = (\gamma)_{i \in [\hat{k}]} - (\gamma')_{i \in [\hat{k}]}$		
$\forall i \in [\hat{k}], h_i = g_i + \gamma_i'' \cdot m'$		
$\forall i \in [\hat{k}], \overline{h_i} = \overline{g_i} + \gamma_i' \cdot \overline{m'}$		
$\mathbf{y} \leftarrow D_\xi^k, \mathbf{w} = \mathbf{A}_1 \cdot \mathbf{y}$		
$\forall i \in [\hat{k}], w_i = (\gamma_i'' \cdot \mathbf{a}_2^\top + \mathbf{b}_i^\top) \cdot \mathbf{y}$		
$\forall i \in [\hat{k}], \overline{w_i} = (\gamma_i' \cdot \mathbf{a}_2^\top + \mathbf{b}_i^\top) \cdot \mathbf{y}$	$\xrightarrow{\mathbf{w}, (h_i, w_i, \overline{h_i}, \overline{w_i})_{i \in [\hat{k}]}}$	
		$d \leftarrow \mathcal{C}$
	$\xleftarrow{d}$	
$\mathbf{z} = \mathbf{y} + d \cdot \mathbf{r}_1$		
if $\text{Rej}_1(\mathbf{z}, d \cdot \mathbf{r}_1, \sigma) = 1$		
then $\mathbf{z} := \perp$	$\xrightarrow{\mathbf{z}, (\gamma'')_{i \in [\hat{k}]}, (\gamma')_{i \in [\hat{k}]}}$	
		Checks:
		1. $\|\mathbf{z}\| \overset{?}{\leq} B, \quad \mathbf{A}_1 \cdot \mathbf{z} \overset{?}{=} \mathbf{w} + d \cdot \mathbf{t}_{1,1}$
		2. $\forall i \in [\hat{k}]$,
		whether the coefficients with respect to $\mathcal{I}$ in h_i are nonzero, and
		$\gamma_i'' + \gamma_i' = \gamma_i$,
		$(\gamma_i'' \cdot \mathbf{a}_2^\top + \mathbf{b}_i^\top) \cdot \mathbf{z} \overset{?}{=} w_i + d \cdot (\gamma_i'' \cdot (\mathbf{t}_{2,1} - m_{\text{att}}) + t_{g,i} - h_i)$,
		$(\gamma_i' \cdot \mathbf{a}_2^\top + \mathbf{b}_i^\top) \cdot \mathbf{z} \overset{?}{=} \overline{w_i} + d \cdot (\gamma_i' \cdot (\mathbf{t}_{2,1} - \overline{m_{\text{att}}}) + \overline{t_{g,i}} - \overline{h_i})$.

Table 4.10: The interactive version of $\Pi_{\text{Discloses\_RANGE}}$. $\Pi_{\text{Discloses\_RANGE}}$ discloses that certain attributes in the committed message m belong in a range (between and inclusive of 0 and $2^n - 1$ where n is the length of the binary string of the attribute).

Some of the matrices used are from the CTS construction seen in Section 3.3.2 (e.g., $\mathbf{A}_1, \mathbf{a}_2^\top, \boldsymbol{comm}, \mathbf{r}_1, m, \mathbf{G}$). The rejection sampling algorithm, Rej_1, can be found in Table 2.3.

Public parameters:
$\mathbf{A}_1, \mathbf{a}_2, B = \sigma \cdot \sqrt{2k \cdot N}$, $\mathbf{B}^\top = (\mathbf{b}_i) \in \mathcal{R}_q^{k \times \hat{k}}$, and $\mathbf{G} = \mathbf{I}_\ell \otimes (1, \delta, \ldots, \delta^{\tau-1})$.
Commitment: $\boldsymbol{comm} := \begin{bmatrix} \mathbf{t}_{1,1} \\ \mathbf{t}_{2,1} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1^\top \\ \mathbf{a}_2 \end{bmatrix} \cdot \mathbf{r}_1 + \begin{bmatrix} 0 \\ m \cdot \ \mathbf{G} \end{bmatrix}$, with $m = \sum_{i \in [N]} m_i X^{i-1}$.

Let $\mathcal{I} \subset [N]$ denote the subset of attribute indices where the prover wants to apply the RANGE property on.
Let $m_{\mathcal{I}} = \{m_i\}_{i \in \mathcal{I}}$ denote the subset of attributes on which the RANGE property will be performed.
Let $\overline{m_{\mathcal{I}}} = \{\overline{m_i}\}_{i \in \mathcal{I}}$ denote a subset of the complement attribute values (i.e. if m_i is 0, then $\overline{m_i}$ is 1 and vice versa).
Set $m_{\text{att}} = \sum_{i \in \mathcal{I}} m_i X^{i-1}$, $m' = \sum_{i \in [N] \setminus \mathcal{I}} m_i X^{i-1}$.
Set $\overline{m_{\text{att}}} = \sum_{i \in \mathcal{I}} \overline{m_i} X^{i-1}$, $\overline{m'} = \sum_{i \in \mathcal{I}} (m_i - \overline{m_i}) X^{i-1} + \sum_{i \in [N] \setminus \mathcal{I}} m_i X^{i-1}$.
Let $m_{overflow\_att} \subset m_{att}$ represent the overflow bits for an attribute in binary (e.g., if **age** attribute is represented by $m_2, \ldots, m_{10}$ and we want to check **age** $\leq$7, we need to check that each overflow bit of the age attribute, $m_2, \ldots, m_7$, is NOT 1.)

Prover	**Verifier**
For each bit in m_{att}, run $\Pi_{\text{Discloses\_OR}}$ For each bit $m_{overflow\_att}$, run $\Pi_{\text{Discloses\_NOT}}$ to check they are NOT 1	Checks that $\Pi_{\text{Discloses\_OR}}$ passes for all the bits in $m_{att} \backslash m_{overflow\_att}$ and $\Pi_{\text{Discloses\_NOT}}$ passes for all the bits in $m_{overflow\_att}$

We first consider disclosing that an attribute is not a specified value. The details of how this $\Pi_{\text{Discloses\_NOT}}$ interaction takes place can be seen in Figure 4.8. We provide a concrete scenario to help the reader better understand the inner workings of $\Pi_{\text{Discloses\_NOT}}$. First, consider that we may have a prover, Alice, who wants to disclose to a verifier, Bob, that she is not 20 years old. She begins an interactive proof with Bob to disclose that her age attribute on her anonymous credential is not 20. First, she starts by generating a masking vector g which she uses to hide her false age. She achieves this by multiplying a random value with the age she claims she is not, subtracted by her actual age (e.g., 21). Note that if Alice is lying and she is indeed 20 years old, the constant coefficient of her masking vector will be 0 and the final conditions that Bob will ultimately check will not pass. She then creates a masked vector $\mathbf{t}_g$ using the masking vector g which hides both her actual age and a secret randomness value from her committed anonymous credential. Alice proceeds to send $\mathbf{t}_g$ to Bob. In turn, Bob sends some challenges to Alice. With the challenges, Alice computes and generates new vectors h_i that hide all other attributes of her anonymous credential scheme. She computes yet other vectors $\mathbf{w}$ and w_i to hide values from her commitment. The vectors Alice generates on this turn are also hidden by randomness values she generates. Alice sends the vectors she creates to Bob, and Bob proceeds to send an additional set of challenges her way. Alice uses these newly received challenges to hide additional information she has in the form of a vector $\mathbf{z}$, which she checks passes a rejection sampling test. Alice sends $\mathbf{z}$ to Bob, and he performs a series of checks on some conditions. Bob checks that the norm of $\mathbf{z}$ is below a certain bound, B. He checks that the values Alice generated in her interaction with him are consistent with part of Alice's commitment being correct (i.e. Alice's $\mathbf{t}_{1,1}$ is consistent with the values he received). In the second half of his checks, Bob verifies that the coefficient with respect to Alice's age attribute in h_i is nonzero. Bob finally checks to see that all the secret and random values generated by Alice match his information about the second part of her commitment (i.e. $\mathbf{t}_{2,1}$). By the end of this protocol, Bob should be convinced, if all his checks pass, that Alice indeed was truthful in that she is not 20 years old.

This disclosure of the NOT property can very easily be generalized to multiple attributes.

The $\Pi_{\text{Discloses\_OR}}$ algorithm is very similar to the $\Pi_{\text{Discloses\_NOT}}$ algorithm. The main distinction between these two algorithms is that the masking vector g sampled from the set G varies for each algorithm. The $\Pi_{\text{Discloses\_RANGE}}$ is essentially composed of a number of $\Pi_{\text{Discloses\_OR}}$ algorithms.

4.3 Technologies Used

The computations and the tests were run on Intel(R) Core(TM) i9-9820X CPU @ 3.30GHz and NVIDIA GeForce RTX 2080 Ti (11 GB).

To obtain completeness and the respective running time of each algorithm, we ran a series of tests. The tests included running the issuing and verification algorithms. We also tested the disclosure and disclosure of properties algorithms which can be seen in Figure 4.1. These tests are described in Section 5.2, and the results of the tests can be found in Section 5.3.

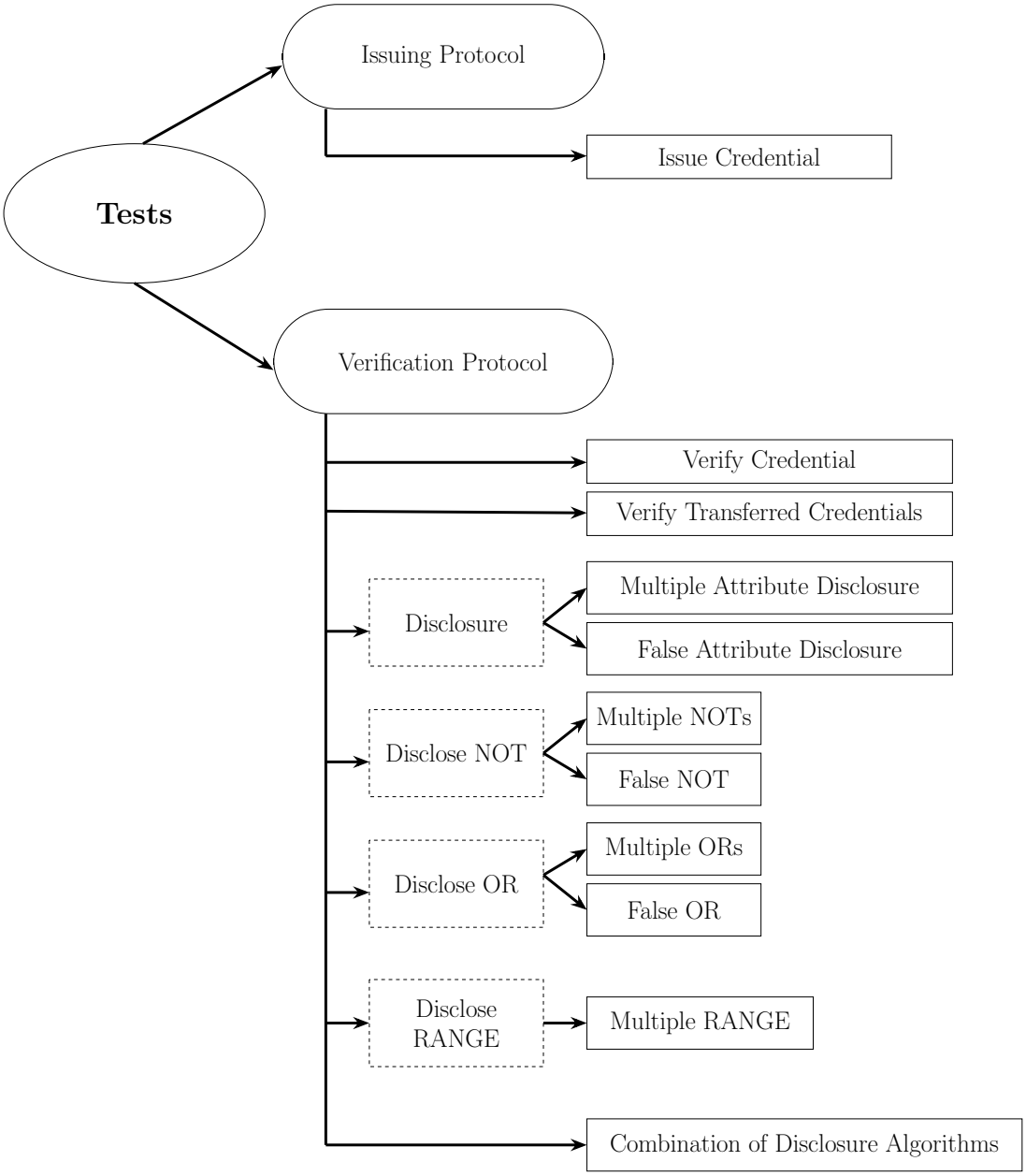

Figure 4.1: A flowchart of all the test cases ran on the anonymous credential algorithms. Descriptions of tests are shown in more detail in Section 5.2.

Chapter 5

Evaluation

In this chapter, we explore the security and efficiency of our implementation. We discuss different security models in Section 5.1 and explore how we evaluated our implementation in Section 5.2. All test results and how our scheme compares with existing implementations can be seen in Section 5.3. We discuss runtime comparisons in Section 5.4.

5.1 Adversarial Security Models and Tests

In this section, we outline some security models with classical and quantum adversaries in. Although we did not directly simulate all these security models in the evaluation of our protocol, the models provided are here for completeness and as a reference for future work. The consolidated list of models provides a starting point for what post-quantum security means and what types of adversaries to be wary of when designing and implementing post-quantum systems. Often times, we notice in the literature that the security of the schemes relies on classical adversaries and difficult mathematical problems where there are no quantum algorithm speedups such as in [38, 9]. For applied work, it is also important to consider a variety of adversaries that might challenge the security of any post-quantum scheme.

We have three main actors in the anonymous credential system: the certificate authority, the prover, and the verifier. For our security models, we introduce a fourth actor: an eavesdropper.

The certificate authority can be honest and honest-but-curious. The prover and verifier can be honest, honest-but-curious, and dishonest. Honest parties follow the protocol step by step correctly. Honest-but-curious parties follow the protocol correctly and attempt to learn additional information from their input and output only. Finally, dishonest parties can deviate from the protocol to achieve their own goals.

Eavesdroppers can be categorized into two main types: passive and active. Passive eavesdroppers will try to learn as much information as possible without interfer-

Table 5.1: The possible types of adversaries for each actor in the anonymous credential scheme. In total, there are $3 \times 5 \times 5 \times 4 = 300$ possible adversarial models.

Type	**Certificate Authority**	**Prover**	**Verifier**	**Eavesdropper**
Honest	✓	✓	✓	
Classical Honest-but-curious	✓	✓	✓	
Quantum Honest-but-curious	✓	✓	✓	
Classical Malicious		✓	✓	
Quantum Malicious		✓	✓	
Classical Passive				✓
Quantum Passive				✓
Classical Active				✓
Quantum Active				✓

ing with the execution of the protocol. Active eavesdroppers will interfere with the protocol to gain as much information as possible.

Every dishonest passive and active adversary can be considered both in the classical case, where they only have access to classical resources, and in the quantum case, where they have access to quantum resources. We show in Table 5.1 all the adversary types for each actor in the protocol. Since the certificate authority is considered and assumed to be trustworthy, we did not consider them in the malicious case.

Therefore, there are $3 \times 5 \times 5 \times 4 = 300$ possible adversarial models.

In the context of our scheme and its properties (notably unforgeability, unlinkability, and anonymity), we have outlined a snapshot of what we consider to be the most relevant adversarial models. These adversarial cases can be seen in Table 5.2. We chose the adversarial models that we thought best highlighted the security of each property. The issuing protocol as listed in Table 5.2 refers to the issuance of anonymous credentials between the certificate authority and a prover. The verification protocol, on the other hand, is all the interactions that take place between a prover and a verifier. This includes scenarios where the verifier verifies the validity of the anonymous credentials provided as well as the validity of any disclosed attributes

or disclosed properties of attributes.

In our evaluation, we test for the following security model where the following adversaries are all classical:

- **Unforgeability (Row 5)** with a malicious prover and an honest verifier in the verification protocol.

At a baseline level, our protocol works with three parties, one of which (we assume the certificate authority) is considered a trusted, honest party. In this type of environment, we thus direct our focus to the adversarial models that are likely to occur in the verification protocol. We chose the model that was one of the most likely scenarios to occur in our given real-life environment. If these attacks are successful, this model also has some of the most adverse and dire consequences. We only chose one model to test because it best highlights the properties of unforgeability during the verification stage. In terms of the anonymity property, we rely on the security theorem presented in [28], and thus do not directly test this property. We leave the testing of the unlinkability property (i.e. **Unlinkability (Row 12)** with an honest prover but malicious verifier in the verification protocol) for future work.

Theis additional test case help to further support that the implemented protocol is secure.

Table 5.2: Outline of classical and quantum adversarial models. Each row represents a security model, and the adversaries can either have classical resources and/or quantum resources.

#	**Property**	**Certificate Authority**	**Prover**	**Verifier**	**Eavesdropper**	**Security Scenario**
1	Unforgeability	Honest	Honest	-	Passive	Issuing
2		Honest	Honest	-	Active	Issuing
3		Honest	Honest-but-curious	-	-	Issuing
4		Honest	Malicious	-	-	Issuing
5		-	Malicious	Honest	-	Verification
6		-	Honest	Honest	Passive	Verification
7		-	Honest	Honest	Active	Verification
8	Unlinkability	Honest-but-curious	Honest	-	-	Issuing
9		Honest	Honest	-	Passive	Issuing
10		Honest	Honest	-	Active	Issuing
11		-	Honest	Honest-but-curious	-	Verification
12		-	Honest	Malicious	-	Verification
13		-	Honest	Honest	Passive	Verification
14		-	Honest	Honest	Active	Verification
15	Anonymity	Honest-but-curious	Honest	-	-	Issuing
16		-	Honest	Honest-but-curious	-	Verification
17		-	Honest	Malicious	-	Verification

5.2 Evaluation Setup

We evaluated all our algorithms by running different tests that check for correctness and security in certain adversarial scenarios.

We measured and recorded the run times of the respective algorithms. We also collected size information for our tests. The results can be seen shortly, in Section 5.3.

As was previewed in Figure 4.1, the tests are as follows:

1. **Issuing Credentials:** We test the issuing protocol. We start by generating a random set of attributes to be committed in a credential. We then execute the issuing protocol between a certificate authority and a prover. During the protocol, the certificate authority verifies if the commitment of attributes is well-formed. At the end of a successful run, we check if a valid anonymous credential was issued to the prover or not. We record run time information.

2. **Verification of Signed Credentials:** We test whether our honestly generated and signed anonymous credentials can be successfully verified by a verifier. We record the run times.

3. **Verification of Transferred Credentials (Pseudonyms):** We test whether all transferred anonymous credentials from an honestly generated and signed credential, can be successfully verified by a verifier. We record the run times.

4. **Disclosure:** We test the disclosure algorithm which discloses a random subset of attributes in a credential without revealing other attributes' information. We create dummy credentials initially for the disclosure procedure. We test this algorithm on up to five attributes and record the run times.

   The next set of tests are for algorithms that we designed and implemented. Since they are correctness tests, we are looking for these tests to pass (i.e. the algorithms work as intended).

5. **Disclosure of OR Property:** In the OR property algorithm, we consider the case where a prover wants to show that an attribute is one of two values, either a value m_1 or m_2 (e.g., if an anonymous credential attribute represents driver license classes, we may want to reveal that this attribute is either an intermediate provisional license or a full unrestricted license). We test the correctness of this test for up to five attributes and collect their respective run times.

6. **Disclosure of NOT Property:** The test was used to check the correctness of a randomly selected attribute not being a specific value. For example, this test can be thought of as simulating that a prover's birthday month is not *August*. Like the previous tests, we recorded run time information for disclosing the NOT property on up to five randomly selected attributes.

7. **Disclosure of RANGE Property:** This test was used to disclose that an attribute m_i is within a limited range: $0 \leq m_i \leq 2^n - 1$ where $i \in N$ and n belongs to the set [1,2,3,4,5]. The upper bound of the range we set for this test was arbitrarily $2^5 - 1 = 31$. For this test, we randomly selected a range and performed the interactive proof on this. We wanted to use this test to simulate revealing whether the age of the prover is between a pair of values. Run times were collected.

8. **Combination of all Disclosure algorithms:** We also evaluate run time performance in the case where a combination of all of these disclosure and disclosure of properties on a credential are being used in a verification interaction.

   To further test for correctness and security, we set up tests with bad-acting, malicious provers. These tests are for testing the unforgeability property. We simulated provers who wanted to disclose false information about their attributes. We perform and gather data for all but the RANGE property algorithm, since the RANGE property algorithm leaks information about the underlying attribute value, penalizing them for being dishonest.

9. **False Disclosure:** We looked at disclosing a false value. A particular scenario in real life might include for example, the prover wanting to disclose that they are 22 years of age when in fact their age is 19 years of age. In our tests, these scenarios should always fail (i.e. the verifier does not accept the credential and the proof). Run times were collected.

10. **False NOT Disclosure:** Next, we tested the NOT property with a false value. We consider the scenario in which the prover might claim in an interaction that their attribute is not a value when, in fact, it is. For example, if the anonymous credential represents an identification document of some kind with an expiry date, we set the fake value to today's value and try and prove that it is NOT today's value. These tests should always fail (i.e. the verifier does not accept the credential and the proof). Run time information was collected.

11. **False OR Disclosure:** Finally, we tested the OR property with a false value. In this case, we took two false values and ran the algorithm on these two values. One can think of this as the scenario of disclosing that a prover lives in Ontario or Quebec, when in fact they live in British Columbia. Run time information was collected.

We compare the issuing credentials test and the verification of credentials test with the results of the implementations of [4, 9, 34, 35, 38].

5.3 Results

We ran 15 iterations of the issuing tests and 100 iterations of the verification tests (Tests 1 to 3 in Section 5.2). We ran 100 iterations of the disclosure tests (Tests 4 to 11 in Section 5.2). For each of our tests, we recorded the average, median, minimum, and maximum run times. For the disclosure algorithms tests, we not only recorded the amount of time it takes for the specific disclosure algorithms to run, we also recorded separately, the setup times associated with them. The setup included setting up the credentials with mock attributes. Finally, we recorded the average, median, minimum, and maximum sizes for the disclosure algorithms tests. We obtained the sizes of vectors and matrices generated and needed on the prover and verifier sides for the disclosure algorithms tests. We also recorded the sizes of messages sent back and forth between prover and verifier in all their interactions.

In Table 5.4, our implementation of anonymous credentials is compared to the implementations of [4, 9, 34, 35, 38], specifically the issuing and verification algorithms. We have omitted CPU cycle information because this information is not easily accessible for high-level type programming languages like Python. The implementations of [4, 9, 34, 35, 38] take different measurements. In [4, 9, 34, 35], tests were run and a subset of minimum, median, average, and maximum run times was extracted. In [38], the authors tested the issuing and verification protocol for three different cases: better time, better size, and balanced. In each of the aforementioned cases, the implementations were time-optimized, size-optimized, and finally, balanced between time and size.

Our runtime results for the issuing and verification protocol can be seen in Table 5.3. For the issue protocol, the success rate is fairly low, likely due to a bug in the code. Despite extensive and detailed analysis, we need to leave the remainder of this debugging to future work.

The long run times are in part due to the rejection sampling repetition rate as initiated in Subsection 4.2.1. For the issuing and verification protocol, the repetition rate was calculated to be 7 times (as seen in Tables 4.3 and 4.5 respectively). In other words, the estimated success rate of these algorithms is 14.3%. We can see that the values converge to roughly 14.3% for the verification protocol. Due to time limitations, we ran a small number of iterations for the issuing protocol. However, with more iterations, the success rate should also converge to 14.3%.

To make meaningful comparisons between our work and the other implementations, we first need to ensure that the metrics we are comparing are the same. Thus, we opted to test our protocol and record the run times for the lowest, average, and highest times. The better time, balanced, and better size metrics used in [38] are somewhat analogous with the lowest, average, and highest run times, respectively, in [4, 9, 34, 35].

The information we have gathered is the running time in seconds. We used the

Table 5.3: Our issuing and verification protocol run times. The issuing and verification protocol was run 15 and 100 times, respectively. The minimum (Min), average (Avg), median (Med), and maximum (Max) run times were extracted.

	Time(s)					
	Min	**Avg**	**Med**	**Max**	**Success Rate**	**Avg Repetition Rate**
Issuing Protocol	5574.9	15320.9	16773.0	39389.5	6/15	2.7
Verification Protocol	44.2	318.5	225.8	1563.5	100/100	7.1
Pseudonym Verification Protocol	44.5	310.3	228.6	1463.4	100/100	7.0

function *perf_counter()* from the ***time*** package to gather and time our tests.

Next, we evaluated our additional disclosure features. It is not possible to compare our work with the literature because the other implementations do not have these features. The results can be seen in Table 5.5 and 5.6.

We plot many of our results in box plots. We provide guidelines for reading the box plots in Figure 5.1, and our actual data can be seen in Figures 5.2 through 5.10.

As can be seen in Figure 5.6, the run times for the disclosure of attributes and the disclosure of the NOT property are fairly constant and do not depend on the number of attributes. This is due to the fact that we do not require additional setup or computations for these procedures since the attributes can all be disclosed (or have the NOT property applied on them) simultaneously without information leakage. On the other hand, the number of attributes does affect the run times of the disclosure of the OR property and the RANGE property. To ensure that no information leaks are taking place, we require additional setup for each attribute, and thus the overall run time increases. Since the RANGE property uses the OR property, it naturally follows that the runtime increases with the number of attributes as well.

All our tests passed successfully in convincing the verifier of the respective disclosures and disclosures of properties of attributes.

Table 5.4: Our issuing and verification protocol run times compared with other implementations. Each of our protocols were run 10 times. The row headings, Min, Avg, Med, and Max represent minimum, average, median, and maximum run times, respectively.

	Implementation Time (s)							
	Ours		[4]	[9]	[34]	[35]	[38]†	
Issuing Protocol	Min	5574.9	0.287	0.67297	-	1.185	Better Time	0.304
	Avg	15320.9	0.383	1.84276	0.117	3.472	Balanced	0.660
	Med	16773.0	0.328	1.28102	-	2.582	Med	-
	Max	39389.5	0.815	-	-	18.935	Better Size	4.822
Verification Protocol	Min	44.2	0.346	0.16554	-	0.896	Better Time	0.0314
	Avg	318.5	0.504	0.17187	0.198	1.342	Balanced	0.0220
	Med	225.8	0.430	0.17174	-	1.186	Med	-
	Max	1563.5	1.174	-	-	0.896	Better Size	0.0198

† Policharla et al. combined both the issuing and verification protocols together for obtaining each variant of their scheme (e.g., the better time variant is optimized over the sum of both the issuing and verification protocol). This explains why in the verification protocol, upon initial inspection, the time-optimized variant is slower than the size-optimized variant (0.0314s $>$ 0.0198s).

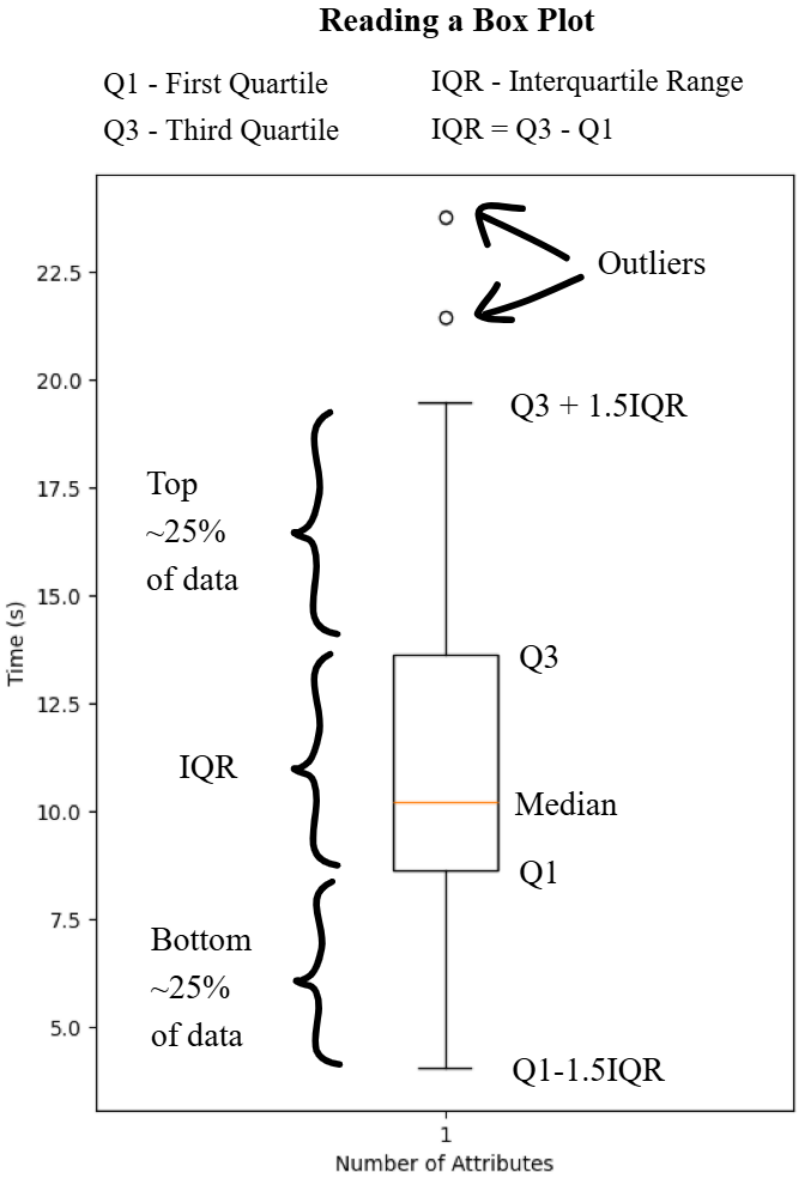

Figure 5.1: Box plot reading guidelines.

Table 5.5: Correctness test run times of disclosure of attributes and disclosure of properties of attributes. The average (Avg), median (Med), minimum (Min), and maximum (Max) run times are shown for each algorithm. Each algorithm was run a total of 100 iterations.

	Number of Attribute(s)	**Our implementation Time (s)**				**Our implementation Time with Setup (s)**			
		Avg	**Med**	**Min**	**Max**	**Avg**	**Med**	**Min**	**Max**
Disclosing Attribute(s)	1	1.649	1.352	0.590	5.433	1.767	1.474	0.709	5.559
	2	1.974	1.367	0.631	8.494	2.092	1.487	0.741	8.613
	3	1.901	1.386	0.650	6.772	2.021	1.506	0.759	6.896
	4	2.022	1.375	0.610	7.941	2.136	1.496	0.729	8.057
	5	1.719	1.374	0.668	5.521	1.838	1.494	0.791	5.640
Disclosing NOT Property	1	1.706	1.332	0.596	5.982	1.819	1.434	0.712	6.100
	2	1.731	1.360	0.592	6.483	1.839	1.477	0.681	6.574
	3	2.008	1.380	0.614	7.262	2.124	1.502	0.707	7.379
	4	2.037	1.377	0.625	7.934	2.155	1.497	0.749	8.056
	5	1.867	1.375	0.690	8.594	1.984	1.495	0.808	8.714
Disclosing OR Property	1	2.462	2.329	0.852	12.02	2.579	2.447	0.946	12.136
	2	4.538	4.126	1.748	17.457	4.650	4.235	1.865	17.545
	3	7.282	6.778	2.839	21.182	7.396	6.881	2.957	21.299
	4	8.994	8.287	4.411	20.268	9.106	8.405	4.499	20.357
	5	11.452	10.769	5.693	26.469	11.567	10.871	5.811	26.587
Disclosing RANGE Property	1	8.985	8.796	1.737	20.533	9.107	8.921	1.826	20.662
	2	15.402	14.993	4.033	31.256	15.521	15.107	4.155	31.385
	3	21.903	21.996	7.512	38.192	22.019	22.117	7.631	38.315
	4	30.723	30.418	12.817	57.084	30.841	30.538	12.936	57.208
	5	35.314	34.121	16.257	65.537	35.432	34.242	16.375	65.661
Combination	-	8.955	8.619	3.430	21.090	11.251	10.234	4.055	23.785

Table 5.6: Correctness test run times of disclosure of false attributes and disclosure of false properties of attributes. Each algorithm was run a total of 100 iterations.

	Our implementation Time (s)					**Our implementation Time with Setup (s)**				
	Avg	**Med**	**Min**	**Max**	**True Negative Rate**	**Avg**	**Med**	**Min**	**Max**	**True Negative Rate**
Disclosing a Fake Attribute	1.869	1.457	0.658	8.002	100%	1.989	1.578	0.771	8.125	100%
Disclosing False NOT Property	1.760	1.279	0.538	6.057	100%	1.880	1.401	0.664	6.182	100%
Disclosing False OR Property	2.233	1.712	0.874	7.551	100%	2.362	1.838	1.004	7.675	100%

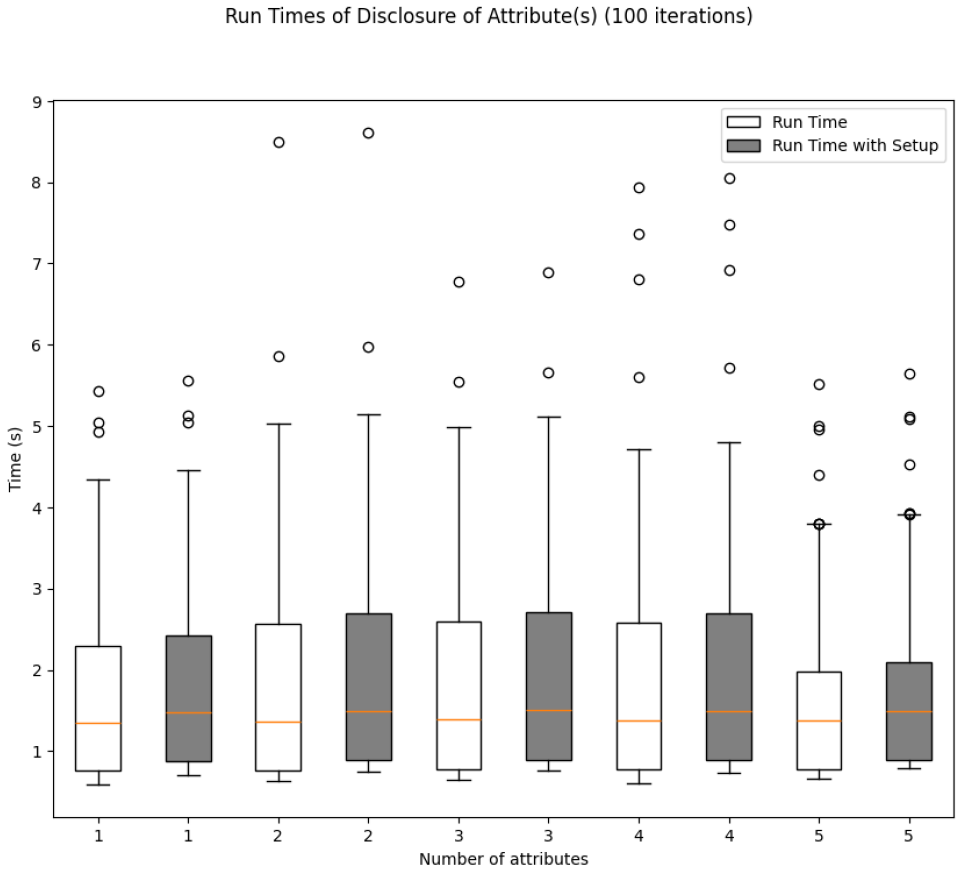

Figure 5.2: Run times of disclosure of attributes plotted on a box plot. Each test was run 100 times. The run time with setup refers to the algorithms run time including any relevant setup time.

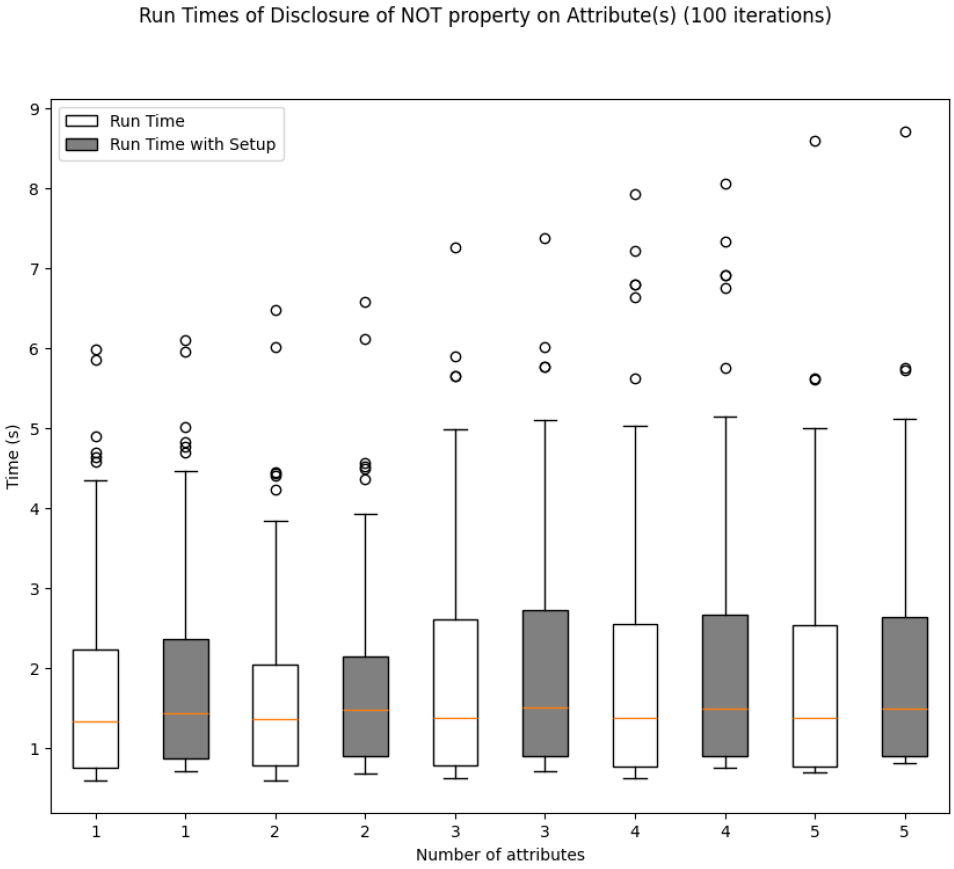

Figure 5.3: Run times of disclosure of NOT property on attributes plotted on a box plot. Each test was run 100 times. The run time with setup refers to the algorithms run time including any relevant setup time.

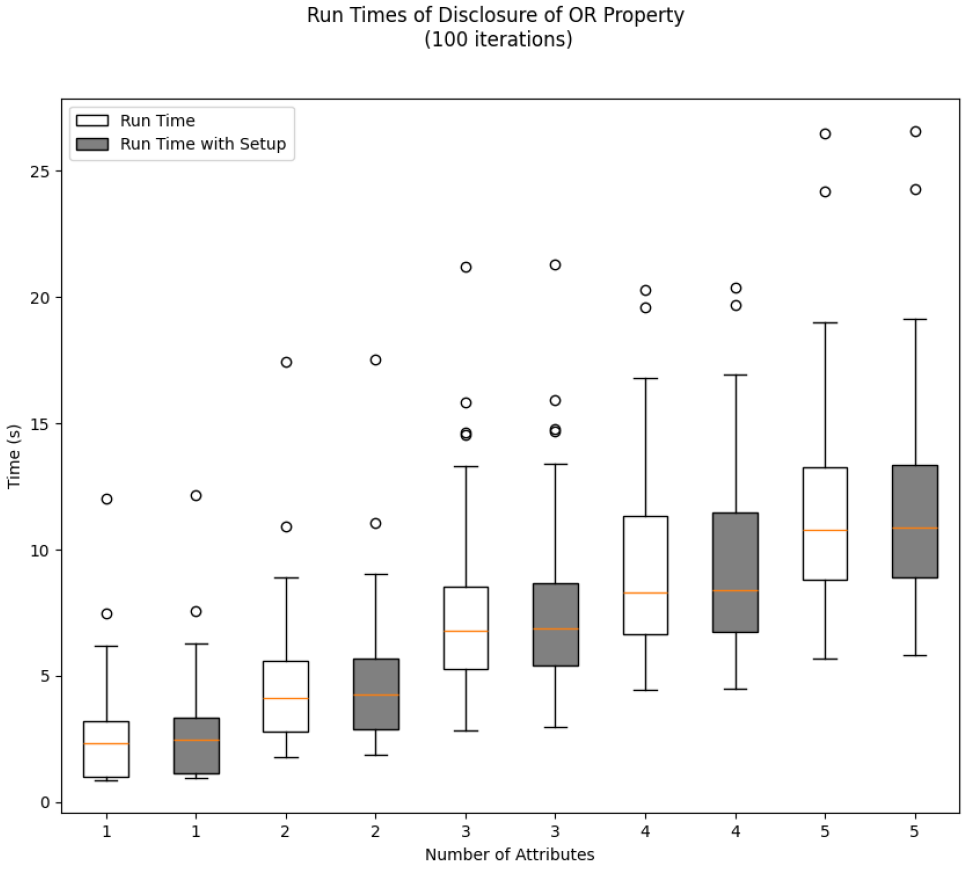

Figure 5.4: Run times of disclosure of OR property on attributes plotted on a box plot. Each test was run 100 times. The run time with setup refers to the algorithms run time including any relevant setup time.

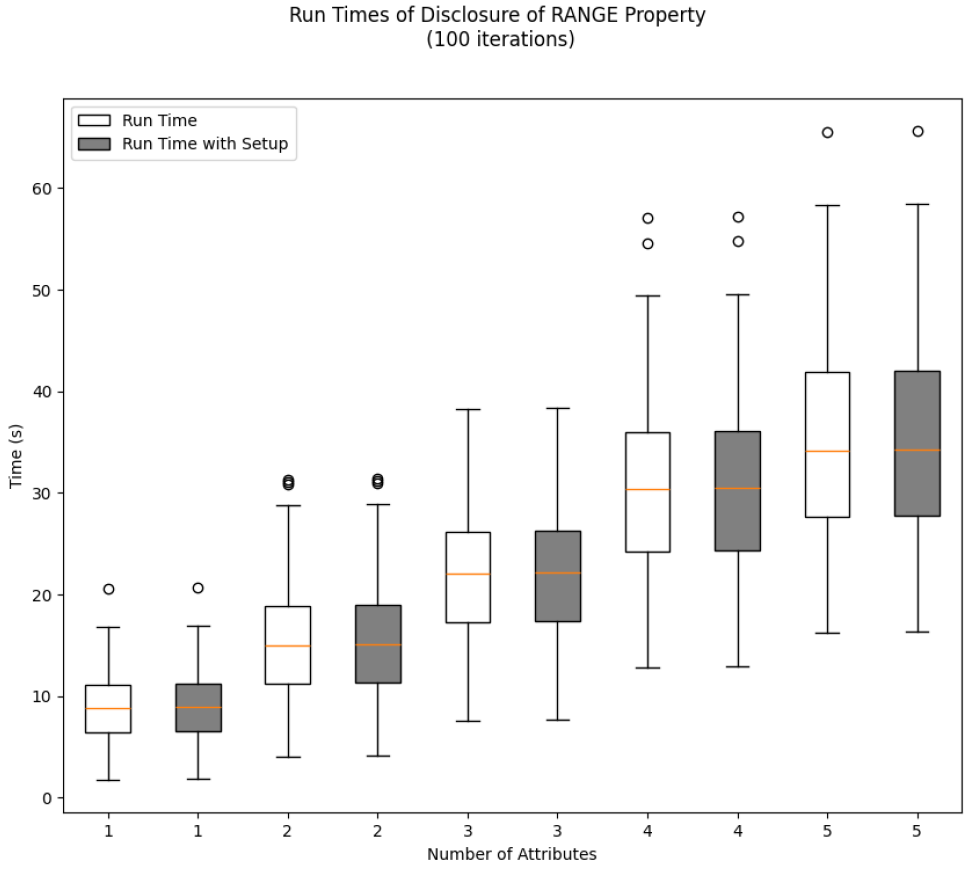

Figure 5.5: Run times of disclosure of RANGE property on attributes plotted on a box plot. Each test was run 100 times. The run time with setup refers to the algorithms run time including any relevant setup time.

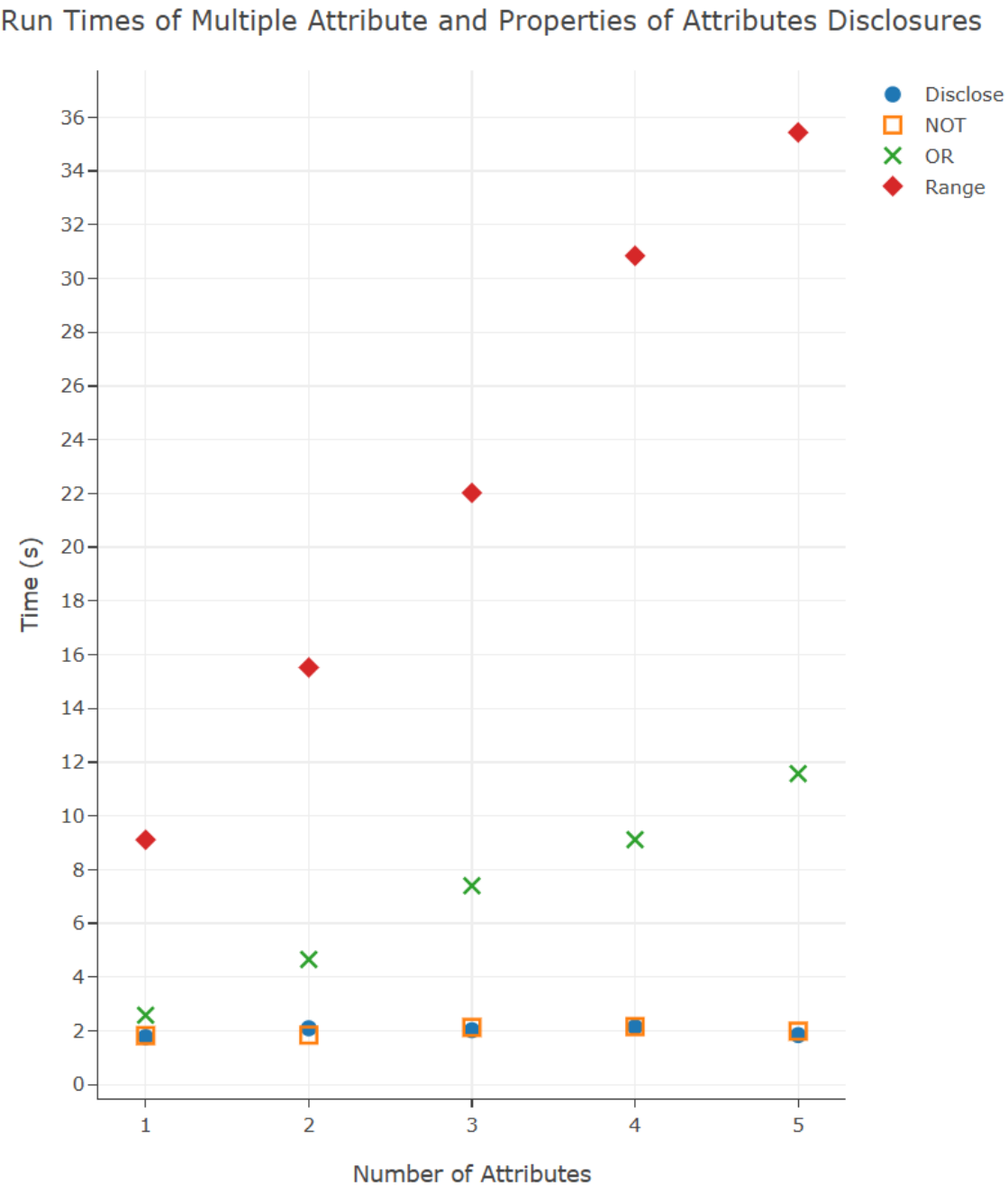

Figure 5.6: The comparison of run times of multi-attribute disclosure and multi-attribute disclosure of properties. Each test was run 100 times. The run time with setup refers to the algorithms run time including any relevant setup time.

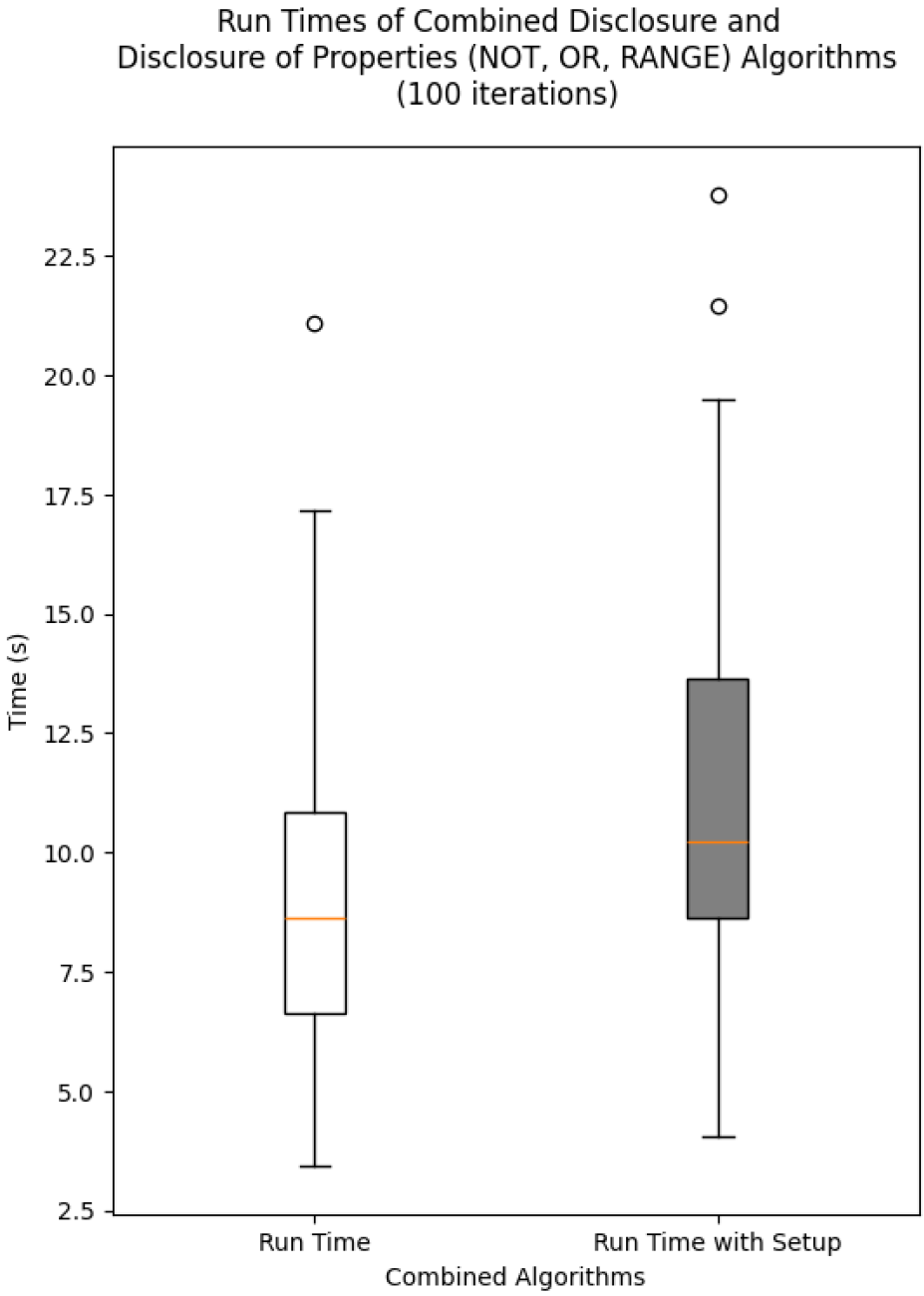

Figure 5.7: Run times of a combination of all disclosure and disclosure of properties algorithms on attributes plotted on a box plot. Each test was run 100 times. The run time with setup refers to the algorithms run time including any relevant setup time.

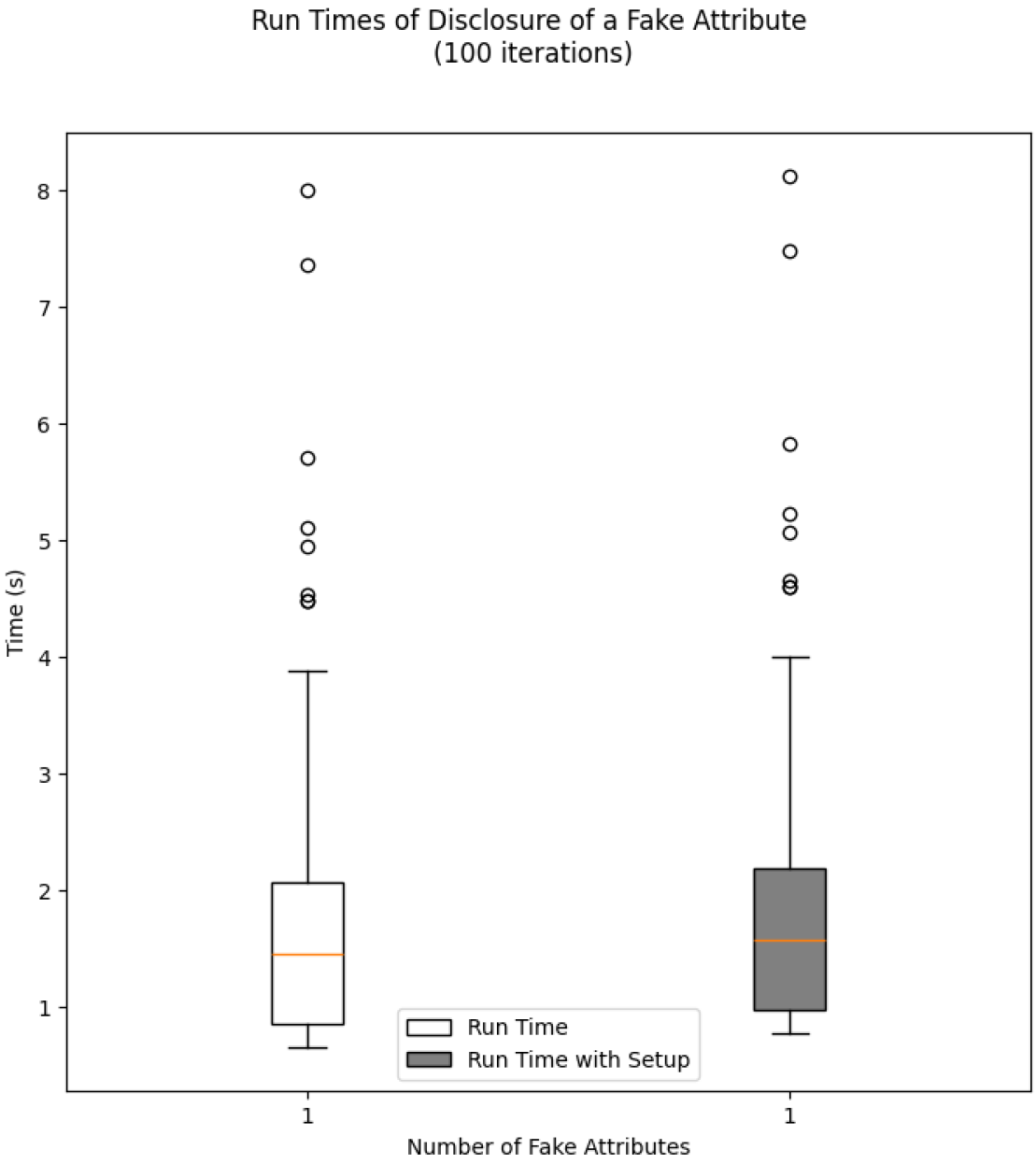

Figure 5.8: Run times of disclosure of a fake attribute plotted on a box plot. Each test was run 100 times. The run time with setup refers to the algorithms run time including any relevant setup time.

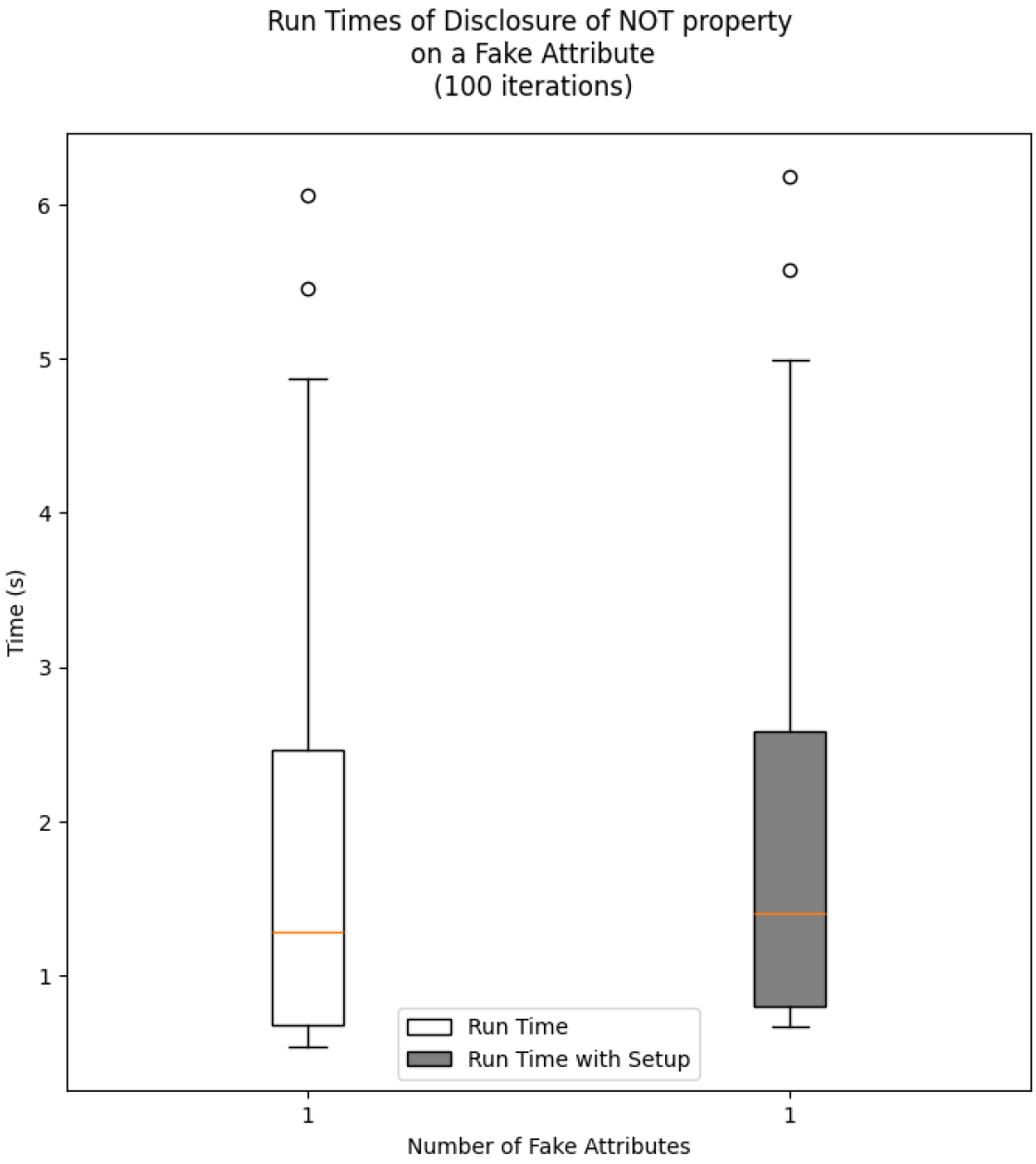

Figure 5.9: Run times of disclosure of NOT property on a fake attribute plotted on a box plot. Each test was run 100 times. The run time with setup refers to the algorithms run time including any relevant setup time.

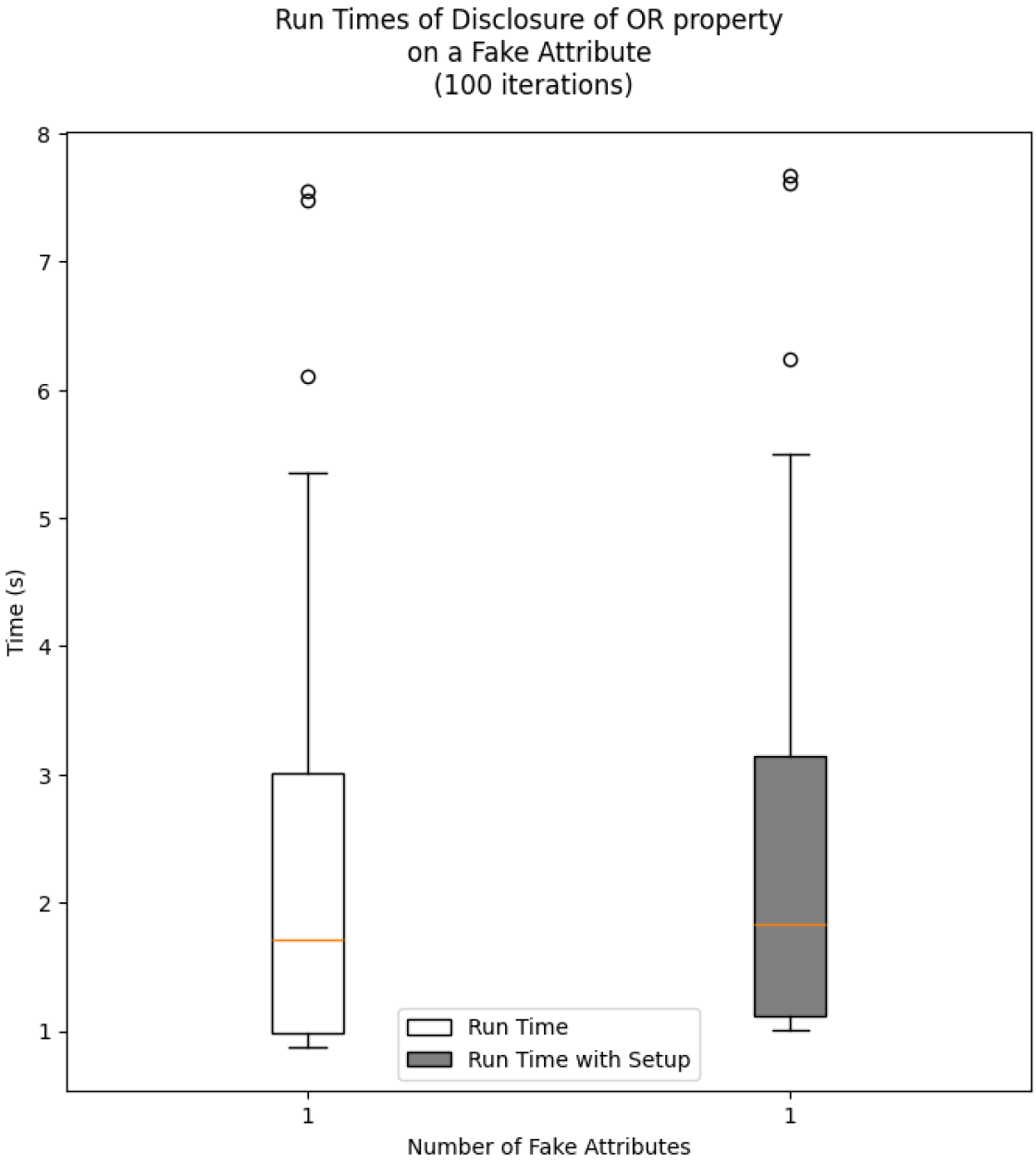

Figure 5.10: Run times of disclosure of OR property on a fake attribute plotted on a box plot. Each test was run 100 times. The run time with setup refers to the algorithms run time including any relevant setup time.

Finally, we look at the respective sizes for the disclosure algorithms. For the disclosure of attributes and the disclosure of properties of attributes, we record the sizes (in KB) for the prover and verifier sides of the interaction, as well as the overall message size. In this case, the message refers to the underlying committed attributes of the anonymous credentials. For the prover and the verifier, we sum up all the required information sizes that will be exchanged. Our results can be seen in Table 5.7. In addition, we have plotted the size results in box plots, which can be seen in Figures 5.11 through 5.15.

Table 5.7: Size information of all prover information, verifier information, and the message size. Each algorithm was run a total of 100 iterations.

	Number of Attributes	**Our Implementation Sizes**								
		Prover Size (KB)				**Verifier Size (KB)**				**Message Size (B)**
		Avg	**Med**	**Min**	**Max**	**Avg**	**Med**	**Min**	**Max**	
Disclosing Attribute(s)	1	258.6	180.2	90.1	1.3 MB	47.1	32.8	16.4	229.6	4096
	2	275.7	180.2	90.1	1.4 MB	50.1	32.8	16.4	246.0	4096
	3	266.7	180.2	90.1	991.2	48.5	32.8	16.4	180.4	4096
	4	218.0	180.2	90.1	1.4 MB	39.7	32.8	16.4	262.4	4096
	5	232.5	180.2	90.1	901.1	42.3	32.8	16.4	164.0	4096
Disclosing NOT Property	1	263.1	180.2	90.1	991.2	47.9	32.8	16.4	180.4	4096
	2	240.6	180.2	90.1	811.0	43.8	32.8	16.4	147.6	4096
	3	214.5	180.2	90.1	811.0	39.0	32.8	16.4	147.6	4096
	4	208.2	180.2	90.1	901.1	37.9	32.8	16.4	164.0	4096
	5	283.9	270.3	90.1	1.1 MB	51.7	49.2	16.4	196.8	4096
Disclosing OR Property	1	416.5	278.6	139.3	1.3 MB	49.0	32.8	16.4	147.6	4096
	2	746.6	696.5	278.6	2.3 MB	87.9	82.0	32.8	262.4	4096
	3	1.2 MB	1.1 MB	417.9	3.3 MB	138.9	131.2	49.2	393.6	4096
	4	1.5 MB	1.4 MB	557.2	3.3 MB	174.5	164.0	65.6	393.6	4096
	5	2.0MB	1.8 MB	835.8	3.8 MB	229.8	213.2	98.4	442.8	4096
Disclosing RANGE Property	1	1.4 MB	1.2 MB	229.4	5.9 MB	180.2	164.0	32.8	688.8	4096
	2	2.7 MB	2.7 MB	458.8	5.7MB	332.6	344.4	65.6	672.4	4096
	3	3.8 MB	3.8 MB	1.4 MB	7.0 MB	467.1	467.4	180.4	820.0	4096
	4	4.9 MB	4.7 MB	2.1 MB	9.4 MB	595.6	574.0	262.4	1.1 MB	4096
	5	6.2 MB	5.8 MB	2.5 MB	13.9 MB	740.5	705.2	311.6	1.6 MB	4096
Combination	-	1.9 MB	1.7 MB	688.2	3.6 MB	262.7	254.2	98.4	524.8	4096

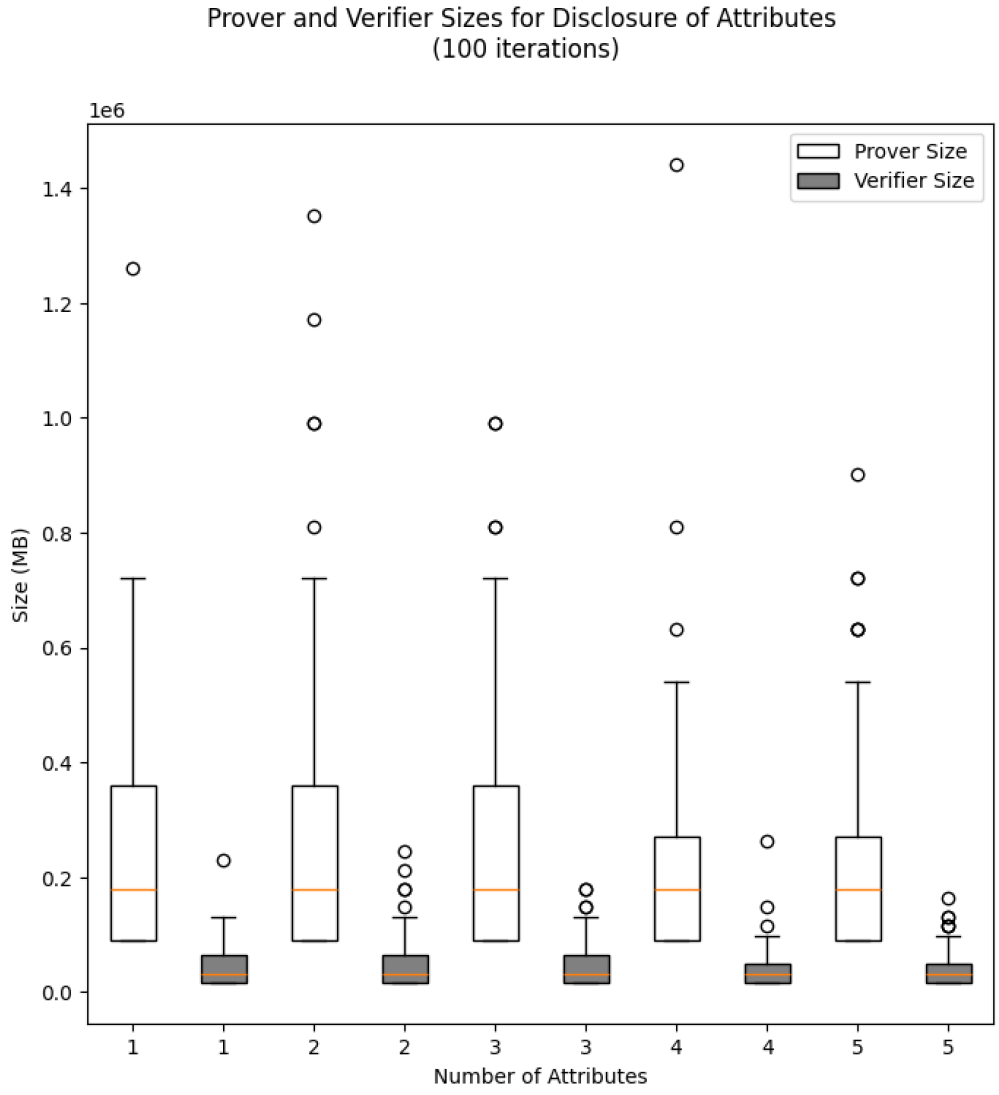

Figure 5.11: Prover and verifier sizes of disclosure of attributes plotted on a box plot. Each algorithm was run a total of 100 iterations.

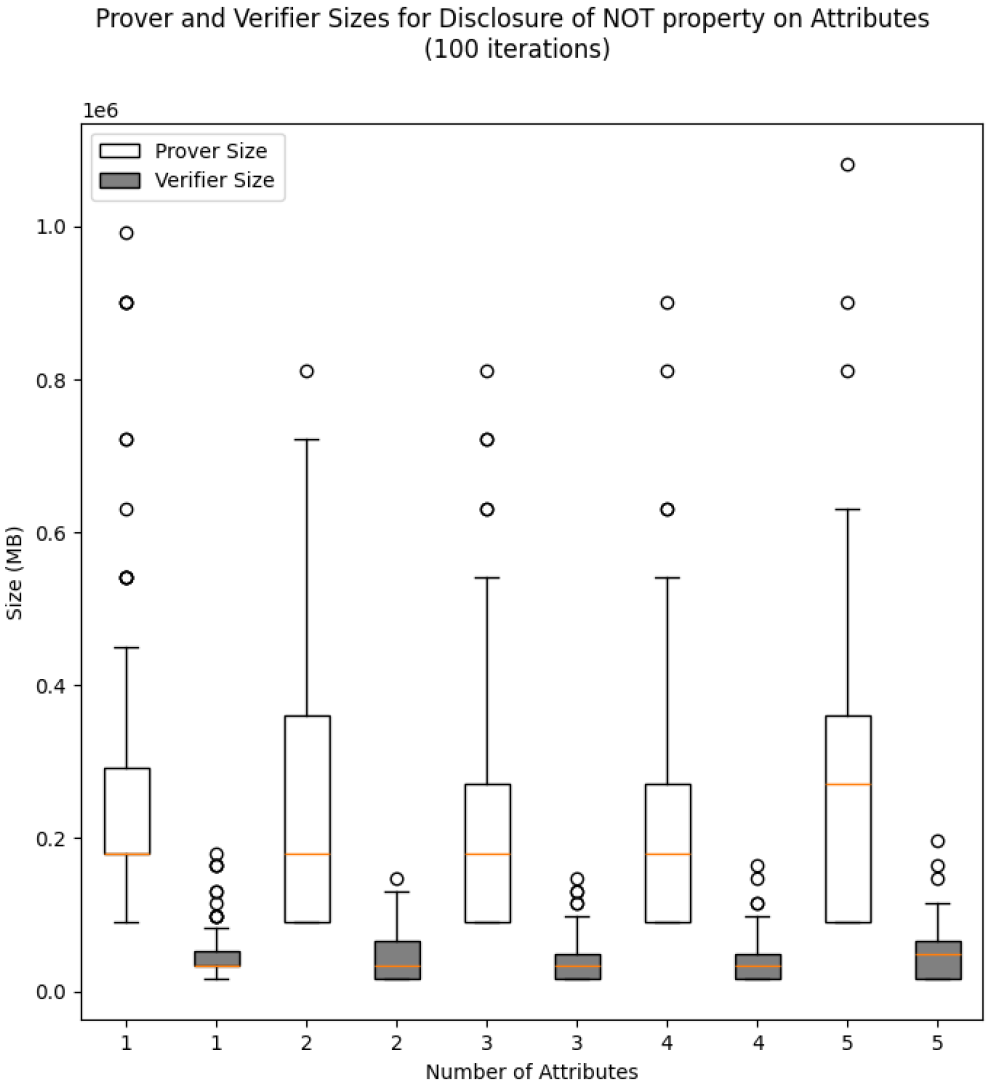

Figure 5.12: Prover and verifier sizes of disclosure of NOT property on attributes plotted on a box plot. Each algorithm was run a total of 100 iterations.

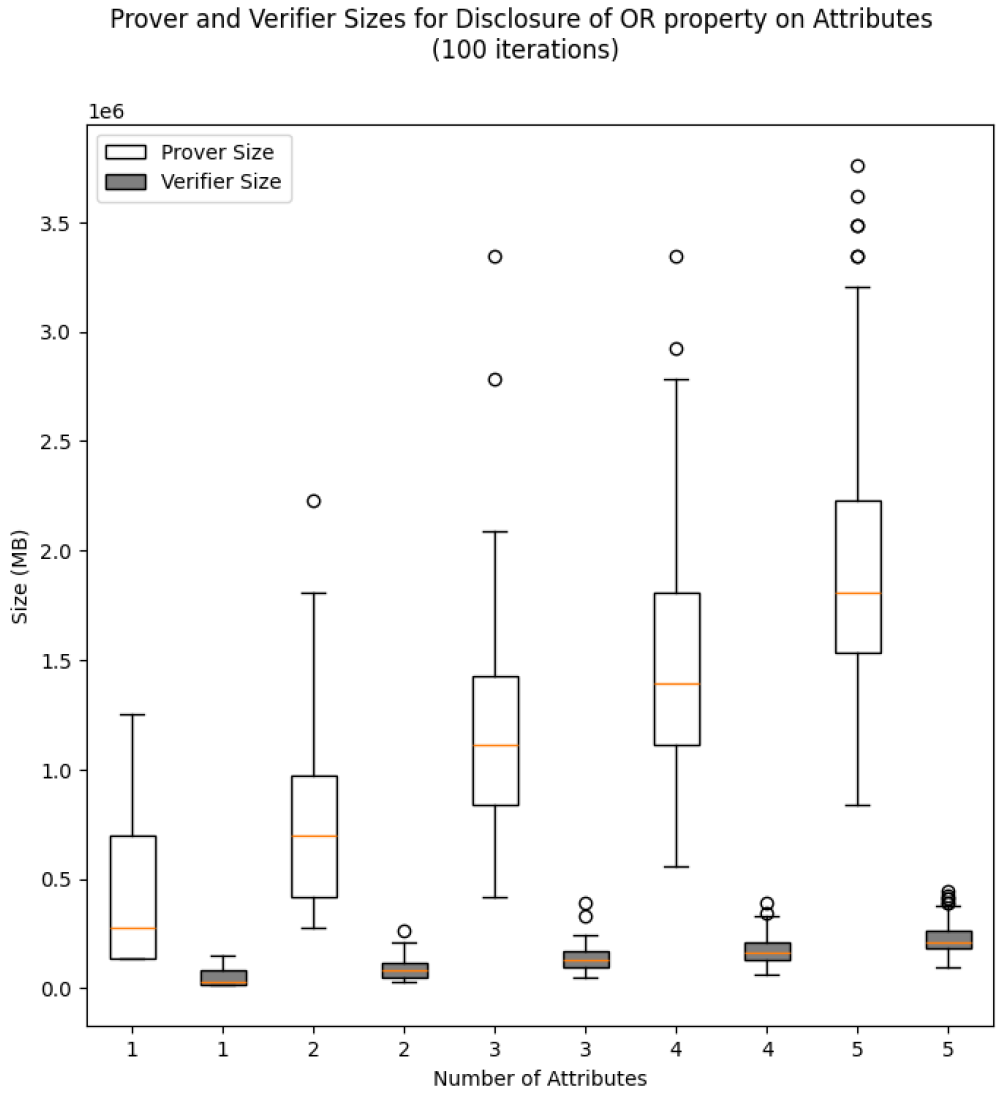

Figure 5.13: Prover and verifier sizes of disclosure of OR property on attributes plotted on a box plot. Each algorithm was run a total of 100 iterations.

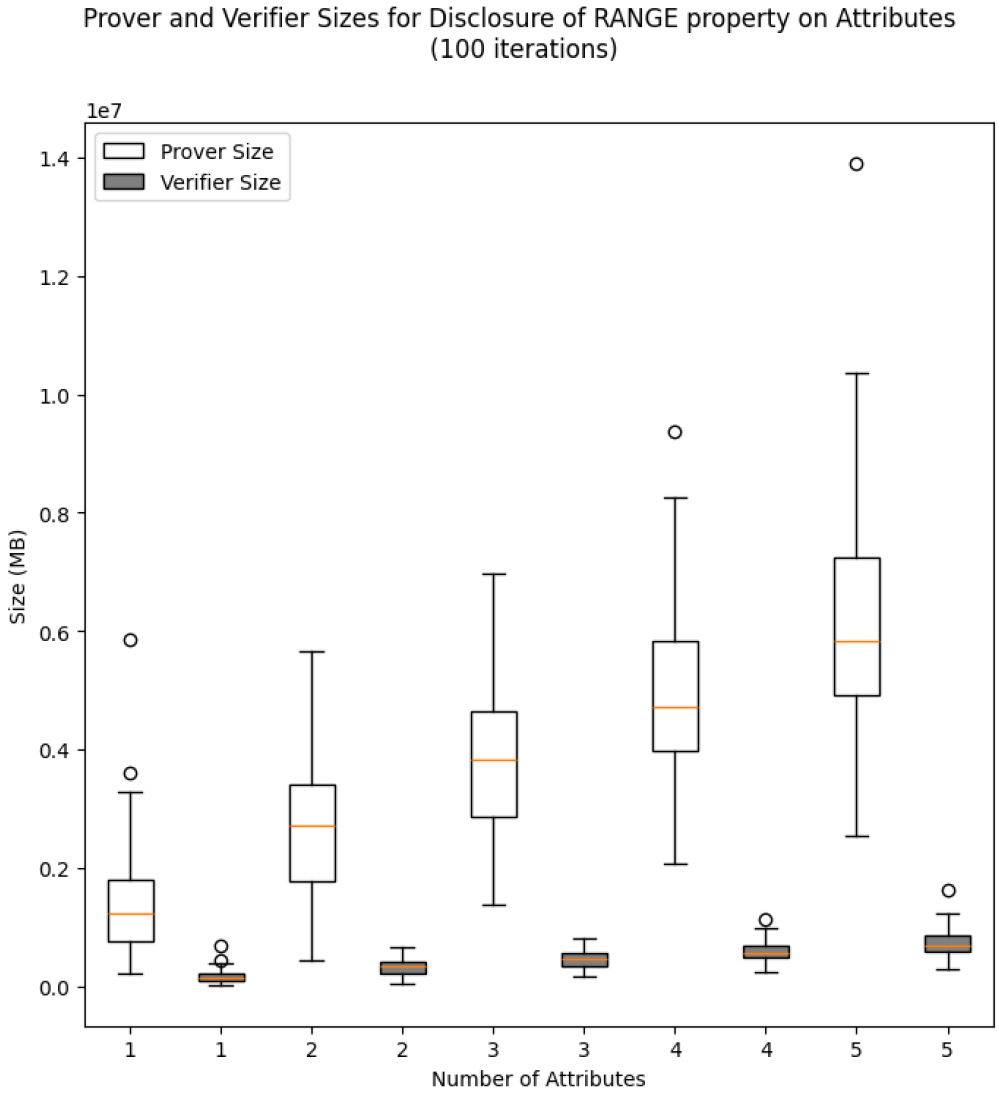

Figure 5.14: Prover and verifier sizes of disclosure of RANGE property on attributes plotted on a box plot. Each algorithm was run a total of 100 iterations.

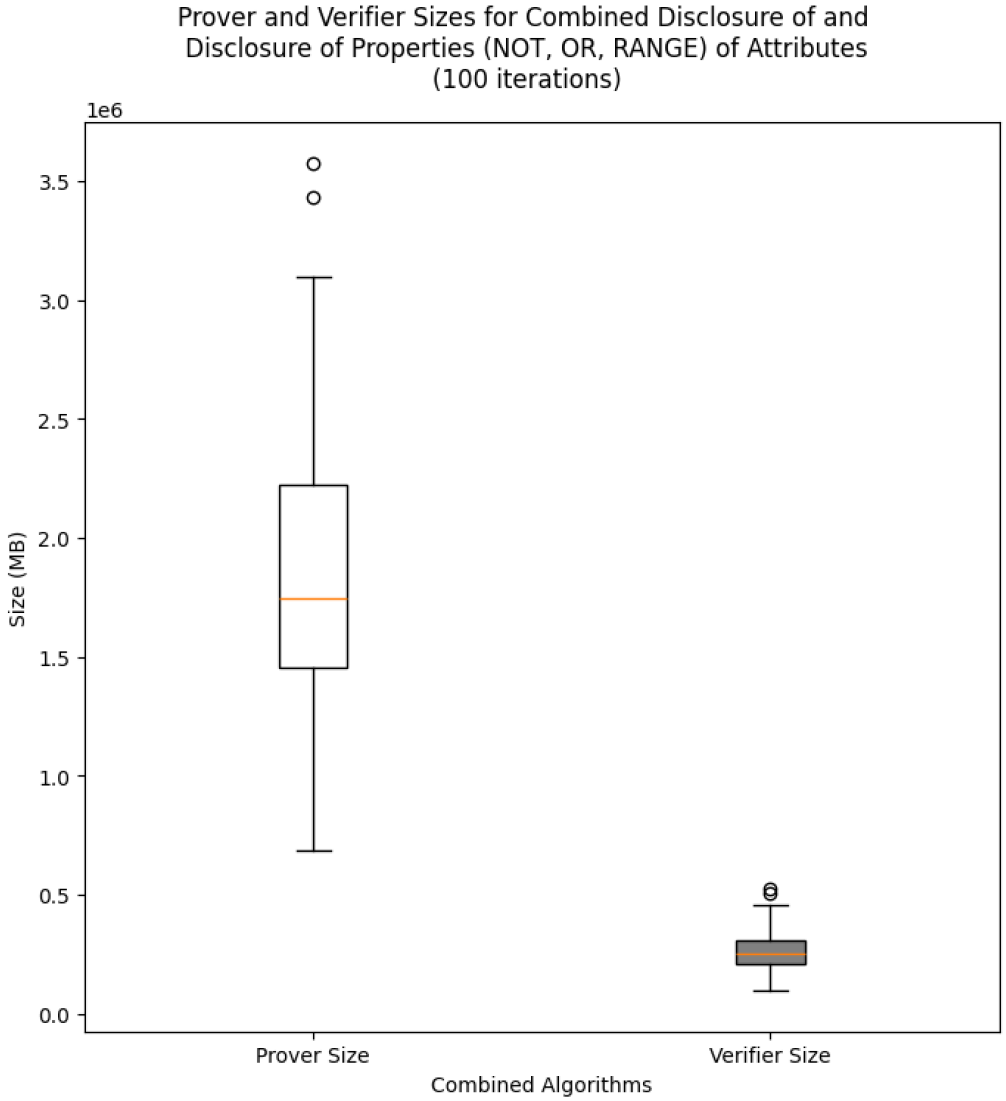

Figure 5.15: Prover and verifier sizes of combined disclosure of and disclosure of properties (NOT, OR, RANGE) on attributes plotted on a box plot. Each algorithm was run a total of 100 iterations.

5.4 Discussion

Every implementation used different processing cores and hardware which we also show in Table 5.8:

- Our protocol was run on an Intel i9-9820X CPU with a clockspeed of 3.3 GHz;
- The Argo et al. [4] implementation was run on a laptop featuring an Intel Core i7-12800H CPU running at 4.6GHz;
- The Blazy et al. [9] implementation involved running their protocol on AMD Ryzen 7 5800H CPU which has a clock speed of 3.2 GHz and 8 Cores;
- We did not find any information on hardware specifications for the Lyubashevsky et al. [34] implementation;
- The Margaria et al. [35] implementation was run on a laptop with Intel Core i7-1225U CPU, running at 1.70 GHz, 24 GB RAM, and on the Ubuntu 22.04 OS;
- Policharla et al. [38] ran their protocol on an Intel Core i9 CPU with a clock speed of 2.4 GHz.

Table 5.8: Programming language and hardware information for existing implementations.

	Implementations					
	Ours	**[4]**	**[9]**	**[34]**	**[35]**	**[38]**†
Language Used	Python	C	C	C (with Python interface)	C++	Rust
CPU	Intel i9-9820X at 3.3 GHz	Intel i7-12800H at 4.6GHz	AMD Ryzen 7 5800H at 3.2 GHz	-	Intel i7-1225U at 1.70 GHz	Intel i9 at 2.4 GHz
RAM	32 GB	-	-	-	24 GB	16 GB

The main difference in the results of Table 5.4 is due to the IZKPoK component that we used. The *lattice-zk* package uses Sagemath for its three dimensional matrix

multiplications, which uses the basic $O(n^3)$ multiplication algorithm. Due to the high volume of large matrix multiplications the *lattice-zk* package uses, there is a significant run time cost. The difference in programming languages and matrix multiplication algorithms is also a factor.

Argo et al. [4] and Blazy et al. [9] implemented their code in C, Margaria et al. [35] used C++, Lyubashevsky et al. [34] used C underneath a Python interface, and Policharla et al. [38] wrote their code in Rust. The languages used by these different implementations are generally more efficient than using an interpreted language like Python. Additionally, one of the largest bottlenecks in our work is our matrix multiplication computation. Our implementation relies on many such computations, and thus substantially slows down our overall runtime. Our matrix multiplication of polynomials was implemented naively and thus takes $\mathrm{O}(n^3)$ time, while some of the other implementations use much faster matrix multiplication such as the Number Theoretic Transform (NTT) algorithm which has a time complexity of $\mathrm{O}(n \log(n))$. Since the issuing protocol is a one-time procedure at the very beginning, the overall practicality of the anonymous credentials scheme is not hindered. As we can see from our results, the verification procedures take much less time and would be very acceptable for regular transactional use of the anonymous credentials.

For the disclosure of attributes, as was mentioned, the number of attributes does not affect the overall run-time, hence taking one to two seconds to run. In terms of our extension, the full transaction between prover and verifier takes one to two seconds for the NOT property, and the rest of the algorithms take longer since the run-time is dependent on the number of attributes. The RANGE property algorithm uses both the OR property and NOT property algorithms, and is thus a summation of the running of those two algorithms. The OR property algorithm takes a longer time as well since every attribute is represented by a subset of bits (e.g., the age attribute might be 25, and the binary value would be 00011001). Essentially, the number of iterations of the algorithm is the number of binary bits in an attribute multiplied by the total number of attributes to be disclosed. The low run-times for these algorithms are very promising for regular daily interactions between prover and verifier.

To ensure that false attribute data would fail as intended in these algorithms, we also ran a series of false attribute tests. For these tests, we ran the disclosure, NOT property, and OR property algorithms on just one false attribute, and the run-times are very reasonable, taking less than two seconds for the disclosure and NOT property algorithms, and just over two seconds for the OR property algorithm.

In terms of the size data collected, for the disclosure of attributes and the disclosure of properties of attributes, the sizes obtained from the prover, verifier, and message remained very consistent in a number of algorithms. These algorithms, the disclosure, and the NOT property had constant sizes regardless of the number of attributes. Our size data for the OR property and the RANGE property algorithms predictably generally grows with the number of attributes. Since the prover has a lot

of information to send to the verifier, the prover size is substantially larger than the verifier size.

Our issuing protocol success rate is 6/15 which suggests that there is likely a bug in the implementation. As mentioned in Section 5.3, we leave this debugging for future work.

Overall, the results of our issuing protocol take substantially longer to run and are incomparable with the results of the implementations in the literature. However, the redeeming quality of this result is that the issuing protocol is a one-time procedure with a potentially long lifespan of an anonymous credential. Thus, it would not have a large effect on the main bulk of its use, that is, for the daily transactions between a prover and a verifier. As we saw in the results, the disclosure of attributes as well as our extended features is carried out in a matter of seconds. In terms of practicality, we do not anticipate provers to reveal many attributes at any single time due to the potential loss of privacy that may result. The risk associated with multiple-attribute exposure is that verifiers may be able to identify the prover if the collectively revealed attribute information can uniquely filter for the identity of the prover. Thus, our work can be practical, although there is certainly room for improving efficiency, especially in the issuing algorithm.

Chapter 6

Conclusion and Future Work

We have surveyed many post-quantum anonymous credential schemes in the literature and have implemented and extended one anonymous credential scheme in particular, that of the scheme in [28].

The [28] scheme uses their commit-transferrable signatures as a base for the anonymous credential. We made some design choices in terms of making the proof of knowledge parts interactive and reducing parameter sizes to increase the computation speed of the overall protocol.

In addition to this, we extended the protocol to include the disclosure of properties of attributes. The disclosure of properties extension is based on the interaction between a prover and a verifier. We include three of these properties, which include the NOT property, the OR property, and the Range property. The NOT property proves to the verifier that an attribute is NOT a particular value. The OR property proves to a verifier that a prover's attribute is one value or another. Finally, the Range property proves to a verifier that a prover's attribute is between some limited range of values. These algorithms can also be used in combination with each other to create more complicated statements about attributes. We implement the anonymous credential scheme and the extension using *Python*, *SageMath*, and *lattice-zk* by Lugstein [31].

For the analysis of our anonymous credential scheme, we consider a number of adversarial models. To ensure that a cryptographic scheme is post-quantum, we present in Table 5.2 a list of adversaries that have quantum and classical resources. We implemented a test out of the list and left the rest for future work. We also test the correctness of our scheme. We recorded the run times of each test and the data sizes of some tests.

After implementing the anonymous credential scheme and the various tests, we next compare our results with the implementations of the anonymous schemes from the literature, specifically the works of [4, 9, 34, 35, 38]. We compare our issuing and verification protocols with theirs. Our run times are substantially slower. The main

reason for such a discrepancy in results is primarily the IZKPoK component from the *lattice-zk* package used. There are also some secondary factors which include some hardware differences and language differences. Our implementation uses an interpreted language, while theirs uses low-level languages which are more speed optimized. Our results are substantially slower than theirs, however, we suggest that our issuing protocol is a one-time procedure for a prover that would not impact the overall practicality needed for regular use. The bulk of the use cases for anonymous credentials are between the prover and verifier for disclosing attributes or properties of attributes.

6.1 Future Work

Our implementation of a post-quantum anonymous scheme with disclosure of properties is a proof of concept but nonetheless a step towards building more usable, feature-intensive anonymous credential systems. Our work is less efficient than some of the other implementations in compiled languages due to a difference in matrix multiplication algorithms and programming languages. Since this scheme is one of the first implementations in an interpreted programming language, we hope that this offers more accessibility and ease of use to a larger audience. We have outlined some adversarial models that will also help create more dialogue around the security of our classical cryptographic schemes in a post-quantum world.

In terms of future work, the pre-image sampling function still needs to be implemented. An approach may be to custom implement a new Gaussian Sampling function instead of using the *SageMath* function.

Due to time constraints, we also leave the debugging of the issuing protocol to future work.

There are also opportunities to add more extensions to the anonymous credential system. Some of these extensions include disclosing whether an attribute is a member of a set or not and extending the range disclosure to accommodate arbitrary intervals. The arbitrary range disclosure can be done in our protocol, albeit inefficiently by using a combination of the properties of attribute proofs (i.e. the range disclosure combined with a NOT disclosure proof).

There are also many areas for improving efficiency in our implementation. One of the major bottlenecks we have seen in our work is related to matrix multiplication over elements of a polynomial ring. Our code relies heavily on the use of matrix multiplication. As a result, our code is severely affected. One way of improving the efficiency of these computations is to offload these computations to GPUs. Adapting the code in such a way that the code can be run on a GPU should provide a substantial speed-up. Some parts of the code use *SageMath*, but to the best of our knowledge, there are no Python packages that can directly speed up *SageMath* functions and data types. A potential method is to convert *SageMath* data types to *numpy* arrays and

then run Python GPU packages like *Numba* and *Cuda* on the code. In addition, there is opportunity to explore using AVX2 (or AVX512) instructions to further optimize and parallelize data calculations. The use of (Number Theoretic Transform) NTT multiplications, which are more efficient for matrix multiplication over polynomials should also be considered.

As we saw in Table 3.1, some anonymous credential systems also have a traceability property, useful for tracking malicious actors in the system. A future step could be to incorporate this property in the scheme.

Our work uses interactive zero-knowledge proofs for our verification procedures in the protocol. Future work can consist of making these proofs non-interactive. The use of non-interactive protocols would speed up the protocol and remove additional communication between the prover and the verifier.

Furthermore, our scheme was based on 128-bit security, but work can be done for increased bit-security such as for 256-bit security and higher.

Finally, while we discussed some security models for the system, proving some of these security models would be the next step. This can be done by running additional tests with simulated adversaries, as shown in Table 5.2. As the security models list is quite long, we would approach the implementation by prioritizing certain tests such as tests with malicious actors, tests with the presence of an eavesdropper, and tests simulating honest-but-curious adversaries. The choice of the test implementation order would also require consideration.

Bibliography

[1] Carlisle Adams. *Introduction to Privacy Enhancing Technologies*. Springer Cham, 1st edition, 2021.

[2] M. Ajtai. Generating hard instances of lattice problems. In *STOC '96: Proceedings of the twenty-eighth ACM symposium on Theory of computing*, pages 99–108, 1996.

[3] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key Exchange—A new hope. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 327–343, Austin, TX, August 2016. USENIX Association.

[4] Sven Argo, Tim Güneysu, Corentin Jeudy, Georg Land, Adeline Roux-Langlois, and Olivier Sanders. Practical post-quantum signatures for privacy. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 1523–1537, New York, NY, USA, 2024. Association for Computing Machinery.

[5] L Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, January 1986.

[6] Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. More efficient commitments from structured lattice assumptions. In *Security and Cryptography for Networks*, pages 368–385, 2018.

[7] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. P-signatures and noninteractive anonymous credentials. In Ran Canetti, editor, *Theory of Cryptography*, pages 356–374, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[8] Mihir Bellare and Phillip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, page 62–73, New York, NY, USA, 1993. Association for Computing Machinery.

[9] Olivier Blazy, Céline Chevalier, Guillaume Renaut, Thomas Ricosset, Eric Sageloli, and Hugo Senet. Efficient implementation of a post-quantum anonymous credential protocol. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*, New York, NY, USA, 2023. Association for Computing Machinery.

[10] Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Alessandro Sorniotti. A framework for practical anonymous credentials from lattices. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 384–417, Cham, 2023. Springer Nature Switzerland.

[11] Stefan Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. The MIT Press, 08 2000.

[12] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology — EUROCRYPT 2001*, pages 93–118, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[13] Madusha Chathurangi, Qinyi Li, Ernest Foo, and Leo Yu Zhang. Post-quantum traceable anonymous credentials from lattices. *IACR Communications in Cryptology*, 2(4), 2026.

[14] David Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of CRYPTO 83*, pages 199–203, 1983.

[15] David Chaum. Security without identification: transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, October 1985.

[16] Kanza Cherkaoui Dekkaki, Igor Tasic, and Maria-Dolores Cano. Exploring post-quantum cryptography: Review and directions for the transition process. *Technologies*, 12(12), 2024.

[17] Duc-Thuan Dam, Thai-Ha Tran, Van-Phuc Hoang, Cong-Kha Pham, and Trong-Thuc Hoang. A survey of post-quantum cryptography: Start of a new race. *Cryptography*, 7(3), 2023.

[18] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. In *Proceedings on Privacy Enhancing Technologies Symposium*, pages 164–180, 2018.

[19] Jiaxin Deng, Simin Chen, Jiageng Chen, and Weizhi Meng. A survey on discrete gaussian samplers in lattice based cryptography. In Jiageng Chen, Debiao He, and Rongxing Lu, editors, *Emerging Information Security and Applications*, pages 87–107, Cham, 2022. Springer Nature Switzerland.

[20] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 40–56, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[21] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC '08: Proceedings of the fourtieth ACM symposium on Theory of computing*, pages 197–206, 2008.

[22] Oded Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, 2001.

[23] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC '96: Proceedings of the twenty-eighth ACM symposium on Theory of computing*, pages 212–219, 1996.

[24] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. *An Introduction to Mathematical Cryptography*. Springer New York, NY, 2nd edition, 2014.

[25] Kelsey A. Jackson, Carl A. Miller, and Daochen Wang. Evaluating the security of crystals-dilithium in the quantum random oracle model. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 418–446, Cham, 2024. Springer Nature Switzerland.

[26] Corentin Jeudy, Adeline Roux-Langlois, and Olivier Sanders. Lattice signature withnbsp;efficient protocols, application tonbsp;anonymous credentials. In *Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part II*, page 351–383, Berlin, Heidelberg, 2023. Springer-Verlag.

[27] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. CRC Press, 2nd edition, 2015.

[28] Qiqi Lai, Chongshen Chen, Feng-Hao Liu, Anna Lysyanskaya, and Zhedong Wang. Lattice-based commit-transferrable signatures and applications to anonymous credentials. *IACR Cryptol. ePrint Arch.*, 2023:766, 2023.

[29] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 2014.

[30] Yang Li, Kee Siong Ng, and Michael Purcell. A tutorial introduction to lattice-based cryptography and homomorphic encryption, 2022.

[31] Florian Lugstein. Lattice-zk: Implementing lantern - lattice-based zero-knowledge proofs. https://lattice-zk.isec.tugraz.at/, 2023.

[32] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Maxime Plançon. Lattice-based zero-knowledge proofs andnbsp;applications: Shorter, simpler, andnbsp;more general. In *Advances in Cryptology – CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part II*, page 71–101, Berlin, Heidelberg, 2022. Springer-Verlag.

[33] Vadim Lyubashevsky, Ngoc Khanh Nguyen, Maxime Plancon, and Gregor Seiler. Shorter lattice-based group signatures via "almost free" encryption and other optimizations. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, pages 218–248, Cham, 2021. Springer International Publishing.

[34] Vadim Lyubashevsky, Gregor Seiler, and Patrick Steuer. The lazer library: Lattice-based zero knowledge and succinct proofs for quantum-safe privacy. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 3125–3137, New York, NY, USA, 2024. Association for Computing Machinery.

[35] Davide Margaria, Alessandro Pino, Andrea Vesco, Giuseppe D'Alconzo, Antonio J. Di Scala, Enrico Guglielmino, and Carlo Sanna. Implementation of a post-quantum anonymous verifiable credential framework. In *2025 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, 2025.

[36] D. Micciancio and O. Regev. Worst-case to average-case reductions based on gaussian measures. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 372–381, 2004.

[37] Christof Paar, Jan Pelzl, and Tim Güneysu. *Understanding Cryptography From Established Symmetric and Asymmetric Ciphers to Post-Quantum Algorithms*. Springer Berlin, Heidelberg, 2nd edition, 2024.

[38] Guru-Vamsi Policharla, Bas Westerbaan, Armando Faz-Hernández, and Christopher A Wood. Post-quantum privacy pass via post-quantum anonymous credentials. Cryptology ePrint Archive, Paper 2023/414, 2023.

[39] C.P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. In *Mathematical Programming*, pages 181–199, August 1994.

[40] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science (FOCS 1994)*, pages 124–134, 1994.

[41] Jordan Shropshire, Lindsay Nadobny, Campbell Hodge, and Micah Israel. Comparative analysis of conventional and post-quantum cryptographic algorithms. In *SoutheastCon 2025*, pages 114–121, 2025.

[42] Vinod Vaikuntanathan. Lattices, learning with errors and post-quantum cryptography lecture notes. online, 2020.

Appendix A

Code Layout

In this appendix, we provide a guide on how to use our code. We show how to get our code running, the file layout of our code on GitHub, descriptions of the functions that were used, and tests that were run.

A.1 Quick Setup

To use our code:

1. Clone our GitHub repository:
   https://github.com/wsherry/Post_Quantum_Anonymous_Credential.git

2. Install the following required software to run our code:

   - *Python* $\geq$ 3.10
   - *numpy* $\geq$ 2.0.1
   - *SageMath* $\geq$ 10.3

   We ran and tested our code on Ubuntu 20.04.2 LTS through Windows Subsystem for Linux. We have not tested our code with newer versions of the dependencies above but assume functionality if newer versions are backward compatible. Earlier versions have not been tested either and may work as well.

3. Run our ready-to-run test code (available in the `tests/` folder), or modify and extend our scheme.

A.2 File Layout

The file structure of our code can be seen as follows:

```
AC_PROJECT/
|-- src
    |-- ac.py
    |-- latticezk_commitment.py
    |-- latticezk_credential.py
|-- tests/
    |-- helpers.py
    |-- test_disclose_functions.py
    |-- test_issue.py
    |-- test_verify.py
|-- README.md
```

Most of the core functions related to the anonymous credential can be found in `ac.py`. The `latticezk_commitment.py` and `latticezk_credential.py` files contains the interactive proof of knowledge functions for the commitment well-formedness and the credential validity, respectively. The original source code for the proof of knowledge is available at: https://lattice-zk.isec.tugraz.at/. Finally, in the `tests/` folder, we have all the tests that were run for the evaluation portion of the thesis. We discuss all the relevant functions in the next part of the appendix.

A.3 Functions

First, let us define some of the functions that are used in `ac.py`.

A.3.1 Basic Functions

We represent our $\mathcal{R}_q$ and $\mathcal{R}$ matrices using primarily *numpy* arrays. Recall that $\mathcal{R} = \mathbb{Z}[x]/(x^N + 1)$ and $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^N + 1)$. In our descriptions of our functions, we use `R_q` and `R` to denote the rings $\mathcal{R}_q$ and $\mathcal{R}$, respectively. In certain parts of our code, we convert these *numpy* matrices to *SageMath* matrices. Since *numpy* does not support polynomial matrix ring operations such as multiplication and addition, we manually coded some of the operations.

```
# Apply modulus q on all elements in 2D matrix
# Inputs: x (array): matrix,
#         q (int): modulus
# Outputs: Matrix whose elements are in [-q//2 +1, q//2]

mod_2D(x, q)
```

```
# Apply modulus q on all elements in 3D matrix
# Inputs: rows (int): number of rows in matrix
```

```
#         x (array): matrix,
#         q (int): modulus
# Outputs: Matrix whose elements are in [-q//2 +1, q//2]

mod_3D(rows, x, q)
```

```
# Matrix multiplication of ring R_q matrices: A and B
# Inputs: A,B (arrays): matrices,
#         mod (int): modulus,
#         poly_mod (array): cyclotomic ring,
#         N (int): ring degree
# Outputs: Matrix C that is the product of A and B

poly_matmul(A, B, mod, poly_mod, N=512)
```

```
# Matrix multiplication of ring R matrices: A and B
# Inputs: A,B (arrays): matrices,
#         poly_mod (array): cyclotomic ring,
#         N (int): ring degree
# Outputs: Matrix C that is the product of A and B

poly_matmul_no_mod(A, B, poly_mod, N=512)
```

```
# Generate a ring R_q matrix
# Inputs: n (int): number of rows/height,
#         m (int): number of columns/width,
#         mod (int): modulus,
#         N (int): ring degree
# Outputs: A random ring matrix with elements in R_q

gen_matrix_ring(n, m, mod, N)
```

```
# Generate a message vector in R
# Inputs: n (int): number of rows/height,
#         m (int): number of columns/width,
#         w (int): number of 1s,
#         N (int): ring degree
# Outputs: A message vector in R

gen_message(n, m, w, N)
```

```
# Generate a R_q matrix with values bounded by beta
# Inputs: n (int): number of rows/height,
#         m (int): number of columns/width,
#         N (int): ring degree,
#         beta (int): bound
# Outputs: A random matrix of size (n,m,N) whose elements are
   bounded by beta

generate_random(n, m, N, beta)
```

```
# Adding R matrices A and B together
# Inputs: A,B (arrays): matrices,
#         poly_mod (numpy array): cyclotomic ring,
#         N (int): ring degree
# Outputs: A summation of A and B, in the ring R

poly_add_3D_no_mod(A, B, poly_mod, N=512)
```

```
# Adding R_q matrices A and B together
# Inputs: A,B (arrays): matrices,
#         mod (int): modulus,
#         poly_mod (array): cyclotomic ring
#         N (int): ring degree
# Outputs: A summation of A and B, in the ring R_q

poly_add_3D(A, B, mod, poly_mod, N=512)
```

```
# Converts a matrix in ring R to a matrix in subring S
# where S^(N/d) = R
# Inputs: R (array): matrix,
#         n (int): number of rows/height of A,
#         m (int): number of columns/width of A,
#         k (int): factor (i.e. N/d)
#         N (int): ring degree,
# Outputs: A matrix or vector in the S^k subring

convert_R_to_S_k(R, n, m, k, N)
```

```
# Maps a matrix in R_q^(nxm) to to its coefficient embedding (in
   integer ring)
# Inputs: A (array): matrices,
#         n (int): number of rows/height of A,
#         m (int): number of columns/width of A,
#         N (int): ring degree,
#         poly_mod (list): cyclotomic ring
# Outputs: The coefficient embedding of A (i.e. a matrix in ZZ^(n*N
   x m*N))

rot(A, n, m, N, poly_mod)
```

```
# Converts coefficient embedding of a matrix back to its form in R_q
# Inputs: A (array): matrices,
#         n (int): number of rows/height of A,
#         m (int): number of columns/width of A,
#         N (int): ring degree
# Outputs: a matrix in R_q

unrot(A, n, m, N)
```

```
# Transposes a ring matrix
# Inputs: A (array): matrix
# Outputs: a transposed matrix

ring_transpose(A)
```

A.3.2 BDLOP Functions

```
# Key generation for BDLOP scheme
# Inputs: n (int):number of rows/height,
#         k (int):number of columns/width,
#         l (int):dimension of matrix,
#         q1, q2 (int): moduli,
#         N (int): ring degree
# Outputs: Public parameters (matrices) A, A1, A2

bdlop_ckeygen(n=1, k=4, l=1, q1=2^32-299, q2=2^26-27, N=512)
```

```
# Makes a BDLOP commitment
# Inputs: A0 (array): public matrix,
#         message (array): message comprised of attributes,
#         rand (array): secret random vector,
#         q1, q2 (int): modulus,
#         poly_mod (list): cyclotomic ring,
#         n1 (int): dimension of matrix,
#         N (int): ring degree
# Outputs: A commitment in BDLOP

bdlop_commit(A0, message, rand, q1, q2, poly_mod, n1=1, N=512)
```

```
# Combines two random matrices
# Inputs: A, B (array): random matrices,
#         mod (int): modulus,
#         poly_mod (list): cyclotomic ring,
#         N (int): ring degree
# Outputs: A new random matrix that was a combintion of two random
   matrices

bdlop_combine(A, B, mod, poly_mod, N=512)
```

```
# Ramdomizes a commitment
# Inputs: comm (array): commitment,
#         A0 (array): public matrix,
#         rand (array): secret random vector,
#         n1, n2, m (int): dimensions of matrix,
#         q1, q2 (int): moduli,
#         poly_mod (list): cyclotomic ring,
#         N (int): ring degree
```

```
# Outputs: A randomized commitment comm

bdlop_randomize(comm, A0, rand, n1, n2, m, q1, q2, poly_mod, N)
```

A.3.3 CTS Functions

In this section, we look at some additional building blocks of our implemented scheme.

```
# Setup for the CTS scheme
# Inputs: n, k, l, l_prime (int): matrix dimensions,
#         q1, q2 (int): moduli,
#         N (int): ring degree
# Outputs: setup parameters

cts_setup(n, k, l, l_prime, q1, q2, N)
```

```
# Generates a commitment on a message
# Inputs: A, G (array): public matrix,
#         message (array): message containing attributes
#         rand (array): secret random vector,
#         N (int): ring degree,
#         poly_mod (list): cyclotomic ring,
#         q1, q2 (int): moduli,
#         n1 (int): matrix dimensions
# Outputs: A commitment on a message

cts_commit(A, G, message, rand, N, poly_mod, q1, q2, n1)
```

```
# Randomizes a commitment
# Inputs: comm (array): commitment,
#         A (array): public matrix,
#         rand (array): secret random vector,
#         n1 (int): matrix dimensions,
#         q1, q2 (int): moduli,
#         poly_mod (list): cyclotomic ring,
#         N (int): ring degree,
# Outputs: A randomized commitment

cts_randomize(comm, A, rand, n1, q1, q2, poly_mod, N)
```

```
# Combines two randomness vectors
# Inputs: rand, rand_prime (arrays): randomness vectors
# Outputs: A new randomness vector

cts_combine(rand, rand_prime)
```

```
# Generates the public and secret keys for the CTS scheme
# Inputs: l, l_hat, tau (int): matrix dimensions,
#         D, G (arrays): matrices,
```

```
#          N (int): ring degree,
#          q2 (int): modulus,
#          beta (int): norm
# Outputs: A public key and a secret key

cts_keygen(l, l_hat, tau, D, G, N, q2, beta=1)
```

```
# Provides a signature over a commitment
# Inputs: D, A0, B, A2, u (arrays): public matrices,
#          comm: commitment,
#          N (int): ring degree,
#          width, height (int): matrix dimensions,
#          q2 (int): modulus
#          bound (int): norm
#          poly_mod (list): cyclotomic ring,
# Outputs: A signature

cts_sign(D, A0, B, A2, u, comm, N, width, height, q2, bound,
    poly_mod)
```

```
# Transfer a signed anonymous credential into another credential (
    creating pseudonyms)
# Inputs:  randomized_comm: randomized commitment,
#          D, A, A0, B, u (arrays): public matrices,
#          n, l, l_hat, tau (int): dimensions,
#          sig_comm (array): signature,
#          rand, rand_prime (arrays): random vectors,
#          N (int): ring degree,
#          poly_mod (array): cyclotomic ring,
#          q2 (int): modulus
# Outputs: A transferred signature on the same underlying committed
    message

cts_transfer(randomized_comm, D, A, A0, B, u, n, l, l_hat, tau,
    sig_comm, rand, rand_prime, N, poly_mod, q2)
```

```
# Verifies a signature's validity
# Inputs:  u, A0, B, D (arrays): public matrices,
#          n, l (int): matrix dimensions,
#          comm (array): commitment,
#          sig (array): signature,
#          sig_type (str : "proof" or "verify"):signature type,
#          q2 (int): modulus,
#          poly_mod (array): cyclotomic ring,
#          N (int): ring degree,
# Outputs: True or False depending on if signature is valid or not

cts_verify(u, A0, B, D, n, l, comm, sig, sig_type, q2, poly_mod, N)
```

A.3.4 Regev Encryption Scheme Functions

The following are the functions for the Regev Encryption Scheme:

```
# Generates all relevant keys for the Regev Encryption Scheme
# Inputs: poly_mod (array): cyclotomic ring,
#         d (int): ring degree,
#         q (int): modulus,
#         n,m,k (int): dimensions
# Outputs: The public keys (A,B) and secret key (S)

regev_key_gen(poly_mod, d, q, n, m, k)
```

```
# Encryption of a message miu
# Inputs: A, B (array): public keys,
#         miu (array): message,
#         poly_mod (array): cyclotomic ring,
#         m (int): dimensions,
#         d (int): ring degree,
#         q (int): modulus
# Outputs: A ciphertext

regev_encrypt(A, B, miu, poly_mod, m, d, q)
```

```
# Decrypt a ciphertext to reveal the underlying message
# Inputs: S (array): private key,
#         c0, c1 (arrays): ciphertext,
#         q (int): modulus,
#         d (int): ring degree,
# Outputs: Message miu (decrypts the ciphertext)

regev_decrypt(S, c0, c1, q, d)
```

A.3.5 Anonymous Credential Functions

The following functions are comprised of some of the previous functions, and is the main part of our implementation.

```
# Setup Parameters for the Anonymous Credential Scheme
# Input: N (int): ring degree,
# Outputs: Public parameters A, D, q1, q2, N, kappa, gamma,
   gamma_prime, alpha, opk_a0, opk_B, opk_u, osk, usk

AC_setup(N=512)
```

```
# Registering for a credential (usually intiated by the prover)
# Inputs: n, m, n1 (int): random vector dimensions,
#         N (int): ring degree,
#         A, G (array): public matrix,
#         poly_mod (array): cyclotomic ring,
```

```
#         q1, q2 (int): moduli,
#         usk (array): secret attributes,
#         beta (int): random vector norm
# Outputs: A commitment of attributes

AC_registration(n, m, n1, N, A, G, poly_mod, q1, q2, usk, beta)
```

```
# Credential issuance (usually initiated by the certificate
   authority)
# Inputs: D, A2, A0, B, u (arrays): public parameters,
#         comm (array): commitment,
#         N (int): ring degree,
#         poly_mod (array): cyclotomic ring,
#         q2 (int): modulus,
#         bound (float): bound
# Outputs: A valid credential

AC_issue(D, A2, A0, B, u, comm, N, poly_mod, q2, bound)
```

```
# Proving validity of credential (initiated by prover side)
# Inputs: D, A, opk_a0, opk_B, u (arrays): public parameters,
#         usk: secret matrix,
#         n, l, l_hat, tau, n1 (int): dimensions,
#         poly_mod (array): cyclotomic ring,
#         rand, rand_prime (arrays): random matrices,
#         N (int): ring degree,
#         q1, q2 (int): moduli
#         comm (array): commitment,
#         credential (array): credential,
#         beta (int): bound
# Outputs: Proof after undergoing an interactive proof with verifier
    for checking the validity of a credential

AC_prove(D, A, opk_a0, opk_B, u, usk, n, l, l_hat, tau, n1, poly_mod
   , rand, rand_prime, N, q1, q2, comm, credential, beta)
```

```
# Verifying the validity of an issued credential
# Inputs: u, A0, A, B, D (arrays): public parameters,
#         n, l (int): dimensions,
#         comm (array): commitment,
#         sig (array): signature,
#         sig_type (str : "proof" or "verify"):signature type,
#         q2 (int): modulus,
#         poly_mod (array): cyclotomic ring,
#         N (int): ring degree,
#         alpha (float): bound
# Outputs: Whether credential is valid or not

AC_verify(u, A0, A, B, D, n, l, comm, sig, sig_type, q2, poly_mod, N
   , alpha)
```

A.4 *lattice-zk* Functions

We highlight the most relevant functions that we have used in our implementation for the files: `latticezk_commitment.py` and `latticezk_credential.py`. For all additional functions, we direct the reader to the lattice-zk framework documentation which can be found at https://lattice-zk.isec.tugraz.at/. This component of the scheme uses *SageMath* heavily to represent ring matrices. The following rejection algorithms are based on the algorithms in Table 2.3.

```
# rejection sampling algorithm 1
# Inputs: z,v (arrays): vectors,
#         std (float): standard deviation,
#         M (float): repetitition rate
# Outputs: True or False depending on the success of the rejection
   sampling

rej1(z, v, std, M)
```

```
# rejection sampling algorithm 2
# Inputs: z,v (arrays): vectors,
#         std (float): standard deviation,
#         M (float): repetitition rate
# Outputs: True or False depending on the success of the rejection
   sampling

rej2(z, v, std, M)
```

```
# rejection sampling algorithm 0
# Inputs: z,v (arrays): vectors,
#         std (float): standard deviation,
#         M (float): repetitition rate
# Outputs: True or False depending on the success of the rejection
   sampling

rej_bimodal(z, v, std, M)
```

The next four functions are part of the interactive proof of knowledge algorithms that we are interested in. We forgo the input and output descriptions of the following functions due to the number of parameters in each function, which are explained in Lugstein's implementation. The functions `abdlop_toolbox, abdlop_quadratic_poly, abdlop_multiple_quadratic, abdlop_single_quadratic`, respectively, depict Tables 3.8, 3.9, 3.10, and 3.11, and are used for the commitment well-formedness IZKPoK. Meanwhile, the `abdlop_mlwe` function is used for credential validity IZPoK.

```
# Table 3.8
abdlop_toolbox(m1, m2, ell, n, k, N, M, n_bin, Z, n_is, n_p, lambd,
   is_opti, rej_u_1, rej_u_2, rej_u_3, rej_u_4, get_challenge_u,
   std1, rep_M1, std2, rep_M2, std3, rep_M3, std4, rep_M4, roh, s1,
   s2, A1, A2, B_gamma, B_beta, B_ext, b_ext, theta, tA, tB, R2_is,
```

```
    r1_is, r0_is,R2_p_is, r1_p_is, r0_p_is,P_s, f,E_s_is, v_is, B_is,
    D_s, u)

# Table 3.9
abdlop_quadratic_poly(m1, m2, ell, n, k, N, M, lambd, is_sigma_1,
    is_opti, rej_u_1, rej_u_2, get_challenge_u, std1, rep_M1, std2,
    rep_M2, s1, s2, m, A1, A2, B, Bg, b, tA, tB, R2_is, r1_is, r0_is,
     R2_p_is, r1_p_is, r0_p_is)

# Table 3.10
abdlop_multiple_quadratic(m1, m2, ell, n, N, is_sigma_1, rej_u_1,
    rej_u_2, get_challenge_u, std1, rep_M1, std2, rep_M2, s1, s2, m,
    A1, A2, B, b, tA, tB, R2_is, r1_is, r0_is)

# Table. 3.11
abdlop_single_quadratic(m1, m2, ell, n, is_sigma_1, rej_u_1, rej_u_2
    , get_challenge_u, std1, rep_M1, std2, rep_M2, s1, s2, m, A1, A2,
     B, b, tA, tB, R2, r1, r0)

# Credential validity IZPoK
abdlop_mlwe(m1, m2, n, k, Z, n_is, lambd, is_opti, rej_u_1, rej_u_2,
     rej_u_3, get_challenge_u, std1, rep_M1, std2, rep_M2, std3,
    rep_M3, roh, s1, s2, A1, A2, B_gamma, B_beta, B_ext, b_ext, theta
    , tA, E_s_is, v_is, B_is)
```

A.5 Tests

In this section, we discuss the functions related to the disclosure of attributes and the disclosure of properties of attributes. We also highlight the tests we performed for the evaluation component of our thesis. This section discusses the code in the folder `tests/`.

A.5.1 Helper Functions

We list the functions in `helpers.py`. These are common functions used by other test files.

```
# Writes data to file
# Input: file_name (str): path of file,
#        field_names (list): data headers,
#        data (list of dictionaries): results,

write_file(file_name, field_names, data)
```

The following three verifier check functions are used in the disclosure of attributes and disclosure of properties of attributes. The conditions checked are shown at the bottom of Tables 3.15, 4.8, 4.9, and 4.10. Notably, the functions: `verifier_checks_1`,

verifier_checks_2, verifier_checks_3 checks conditions 1,2,3 respectively in the tables.

```
# Post-interactive zero-knowledge proof protocol where verifier
   performs a series of checks (part 3)
# Inputs: k_hat (int): dimension,
#         gammas (list of ints): challenges
#         A2, B (arrays): public matrices,
#         q2 (int): modulus,
#         poly_mod (array): cyclotomic ring,
#         N (int): ring degree,
#         z (array): vector,
#         challenge (array): challenge given by the verifier,
#         t21 (array): a portion of the commitment,
#         m_att (array): disclosed attributes,
#         h, w, t_g (array): vector
# Outputs: True if equations are equal, otherwise False

verifier_checks_3(k_hat, gammas, A2, B, q2, poly_mod, N, z,
   challenge, t21, m_att, h, w, t_g)
```

```
# Post-interactive zero-knowledge proof protocol where verifier
   performs a series of checks (part 2)
# Checks coefficients of a vector h, and whether they are all
   non_zero or they are all zeroes (depending on the check_non_zero
   flag)
# Inputs: k_hat (int): dimension,
#         disclose_indices (array): disclosed attribute indices,
#         check_zero (boolean): determines which condition to check
#                               (i.e, coefficients are zero/non_zero),
#         h (array): vector on which condition will be checked
# Outputs: True if conditions are met, otherwise False

verifier_checks_2(k_hat, disclose_indices, check_zero, h)
```

```
# Post-interactive zero-knowledge proof protocol where verifier
   performs a series of checks (part 1)
# Inputs: z (array): vector,
#         bound (float): bound,
#         A1 (array): public matrix,
#         q1 (int): modulus,
#         poly_mod (array): cyclotomic ring,
#         N (int): ring degree,
#         w_bold (array): vector,
#         challenge (array): challenge given by the verifier,
#         t11 (array): a portion of the commitment
# Outputs: True if equations are equal, otherwise False

verifier_checks_1(z, bound, A1, q1, poly_mod, N, w_bold, challenge,
   t11)
```

```
# Disclosing the OR property of an attribute
# Inputs: p_size (float): prover size,
#         v_size (float): verifier_size,
#         prover_sizes (array): all prover sizes,
#         verifier_sizes (array): all verifier sizes,
#         index (int): index (for tracking data entries purposes),
#         random_index (int): index for which value is the actual
   credential attribute,
#         message (array): message containing all the attributes,
#         m_prime (array): attributes that won't be disclosed,
#         m_att (array): disclosed attributes,
#         val (int): an attribute value,
#         other_val (int): an attribute value,
#         A1, A2, G, B  (arrays): public matrices,
#         q1, q2 (int): moduli,
#         N (int): ring degree,
#         k_hat (int): dimensions,
#         t11, t21 (array): commitment,
#         disclose_indices (array): indices of attributes to be
   disclosed,
#         r1 (array): random vector,
#         bound (float): bound
#         setup_start (float): start time of protocol and prior
   setup,
#         protocol_start (float): time the protocol began,
#         num_attributes (int): number of attributes disclosed,
#         poly_mod (array): cyclotomic ring,
# Outputs: the validity of the OR property on an attribute

or_property_info(p_size, v_size, prover_sizes, verifier_sizes, index
   , random_index, message, m_prime, m_att, val, other_val, A1, A2,
   G, B, q1, q2, N, k_hat, t11, t21, disclose_indices, r1, bound,
   setup_start, protocol_start, num_attributes, poly_mod)
```

A.5.2 Issue Tests

```
# Runs the issuing protocol
# Input: iterations (int): total number of tests to run
# Outputs: The data and run-times of the protocol (average, median,
   minimum, and maximum times)

issue_test(iterations)
```

A.5.3 Verify Tests

```
# Runs the verifying protocol (that the credential is valid)
# Input: iterations (int): total number of tests to run
```

```
# Outputs: The data and run-times of the protocol (average, median,
    minimum, and maximum times)

verify_test(iterations)
```

```
# Runs the verifying protocol that transferred credentials (
    pseudonyms) from a singular credential are valid
# Input: iterations (int): total number of tests to run
# Outputs: The data and run-times of the protocol (average, median,
    minimum, and maximum times)

pseudonym_verify_test(iterations)
```

A.5.4 Disclose Tests

The tests related to the disclosure of attributes and the disclosure of properties of attributes can be found in `all_disclose_tests.py`. The relevant tests are shown below:

```
# Test for disclosing the NOT property on attributes
# Inputs: num_attributes (int): number of attributes to run the NOT
#                               property on
#         iterations (int): total number of tests to run
# Outputs: The data and run times (average, median, lowest, highest)

run_multi_NOT(num_attributes, iterations=100)
```

```
# Test for runs the NOT property with false attribute values (each
    iteration of the test only applies the NOT property to one false
    attribute)
# Inputs: iterations (int): total number of tests to run
# Outputs the data and run times (average, median, lowest, highest)

run_NOT_false(iterations=100)
```

```
# Test for disclosing the OR property
# Inputs: num_attributes (int): number of attributes to run the OR
#                               property on,
#         iterations (int): total number of tests to run
# Outputs: The data and run times (average, median, lowest, highest)

run_multi_OR_random(num_attributes, iterations=100)
```

```
# Test for runs the OR property with false attribute values (each
    iteration of the test only applies the OR property to two false
    attribute values)
# Inputs: iterations (int): total number of tests to run
# Outputs: The data and run times (average, median, lowest, highest)

run_OR_false(iterations=100)
```

```
# Test for disclosing attributes
# Inputs: num_attributes (int): number of attributes to disclose
#         iterations (int): total number of tests to run
# Outputs: The data and run times (average, median, lowest, highest)

run_multi_disclose(num_attributes, iterations=100)
```

```
# Test for disclosing false attribute values (each iteration only
   discloses one attribute)
# Inputs: iterations (int): total number of tests to run
# Outputs: The data and run times (average, median, lowest, highest)

disclose_false(iterations=100)
```

```
# Test for Disclosing the Range property
# Inputs: num_attributes (int): number of attributes to run
#                               the Range property on
#         subset_choices (list): the subset of an attribute's bit
#                               length (<= max_bit_index)
#         max_bit_index (int): the attribute's max bit length,
#         iterations (int): total number of tests to run
# Outputs: The data and run times (average, median, lowest, highest)

run_multi_range(num_attributes, subset_choices=[1,2,3,4,5],
   max_bit_index=5, iterations=100)
```

```
# Test for running a combination of the disclosure of attributes and
    disclosure of properties of attributes algorithms
# Inputs: iterations (int): total number of tests to run
# Outputs: The data and run times (average, median, lowest, highest)

run_combo(iterations=100)
```

A.6 Suggested Usage of the code base

We have provided some documentation of our code to ease the development process for extensions or modifications of the code. We suggest developers look at our GitHub repository for more complete documentation.

For developers interested in learning more about the overall implementation, we suggest adapting and running the tests we have created in our work. The written test code provides information on how provers, verifiers, and certificate authorities interact to complete different functions of our scheme. New tests can be created to check for additional security.

We direct developers to the basic functions of the protocol for further extensions they might like to add.