I, Michael Snook, hereby submit this original work as part of the requirements for the degree of Doctor of Philosophy in Mathematical Sciences.

It is entitled:
Quantum Resistant Authenticated Key Exchange from Ideal Lattices

Student's name: Michael Snook

This work and its defense approved by:

Committee chair: Jintai Ding, Ph.D.
Committee member: Chris Christensen, Ph.D.
Committee member: Daniel Smith-Tone, PH.D.
Committee member: Ying Wang, Ph.D.
Committee member: Ning Zhong, Ph.D.
Quantum Resistant Authenticated Key Exchange from Ideal Lattices

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Department of Mathematical Sciences
Abstract

We examine a collection of key exchange protocols based on the ring learning with errors lattice problem. The main protocol we present achieves higher efficiency than its immediate predecessor, as well as a security proof that avoids the random oracle model used by the previous protocol. This key exchange protocol looks to be a promising tool for secure communication in a post-quantum world.
© 2016 by Michael Snook.

Released under a Creative Commons Attribution–NonCommercial–NoDerivatives 4.0 International License.
Acknowledgments

I would like to thank my family and friends for their unwavering love and support.

I am indebted my adviser, Dr. Jintai Ding, for his patience and leadership these past years. I would not have gotten to where I am without his guidance.

Thanks to my fellow cryptography students for helping to look over my proofs and testing implementations. Without their assistance, this work could not have been completed.
# Contents

Abstract iii
Copyright iv
Acknowledgments v
List of Tables viii
List of Figures ix
List of Algorithms x

1. Introduction 1
   1.1. Motivations 2

2. Classical Key Exchange 5
   2.1. Diffie–Hellman 6
       2.1.1. Elliptic Curves 7
   2.2. HMQV 8
   2.3. Shor’s Algorithm 10

3. Lattices 15
   3.1. Lattice Basics 15
   3.2. Lattice Reduction 20
       3.2.1. Gram–Schmidt 21
       3.2.2. Gauss Reduction 22
       3.2.3. LLL Reduction 23
   3.3. Lattice Problems 24
       3.3.1. Shortest Vector Problem 26
       3.3.2. Closest Vector Problem 28
       3.3.3. Shortest Independent Vectors Problem 30
List of Tables

A.1. Performance Comparison .............................. 95
List of Figures

2.1. Diffie–Hellman Key Exchange ........................................ 7
2.2. HMQV Key Exchange ...................................................... 10
3.1. The Lattice $\mathbb{Z}^2$ ...................................................... 16
3.2. Skewed Lattice ............................................................. 17
3.3. Two Fundamental Regions ................................................. 18
5.1. Ring Learning with Errors Key Exchange ............................... 46
5.2. Authenticated RLWE Key Exchange ..................................... 50
5.3. Authenticated RLWE Key Exchange II .................................. 54
5.4. Matrix Form ............................................................... 59
5.5. One-Pass Protocol ......................................................... 62
List of Algorithms

1. Factorization to Order Reduction .................................................. 11
2. Shor’s Order Finder ................................................................. 13
3. Generalized Shor’s Algorithm ..................................................... 14
4. Gauss Reduction ................................................................. 22
5. LLL Reduction ................................................................. 24
Chapter 1

Introduction

The world in which we live is increasingly dominated by the presence of the internet in people’s daily lives. Online banking, shopping, and various social media platforms all rely on traditional security concepts of secrecy, authenticity, and integrity to protect their clients from online threats. When an individual shopping at an online retailer submits an order, they typically include some form of payment and shipping information. The buyer needs a secure communications channel to prevent a third party from impersonating the retailer, stealing the payment information in transit, or modifying the shipping information to steal the order.

A number of ciphers have been used historically for securely transmitting information over an insecure communications channel. The communicating parties would use some secret information only they had, called a key to achieve this. The sender would use the key to their encrypt their messages, rendering them unintelligible, before transmission, and the receiver would use the key decrypt the received message, recovering the original, human-readable form. The problem of establishing a secure communications channel between two parties then becomes the problem of generating a key for the parties to use.

We then come to the question of how two parties can establish keys over an insecure channel; this is the goal of the field of public key cryptography. The first breakthrough published in
this area was the Diffie–Hellman key exchange protocol [DH76], which allows two parties to generate a cryptographic key over an insecure communications channel.

Shortly after Diffie and Hellman published their key exchange protocol, Rivest, Shamir, and Adleman published an asymmetric cryptosystem [RSA78] in which the key used for encrypt messages was separate from that used to decrypt them. Then the encryption key can be openly published without allowing parties other than the key’s owner to read encrypted messages. The mathematics behind the RSA encryption scheme are also used to construct a signature scheme, a way to guarantee that the sender of a message is who they claim to be. Since Diffie–Hellman and RSA, a number of additional public-key cryptographic primitives and protocols have been proposed based on number-theoretic problems similar to these, including ElGamal encryption and signatures [ElG84]; the Digital Signature Algorithm (DSA) based off of ElGamal; Goldwasser–Micali, which introduced the concept of semantic security [GM84]; and the Secure Remote Password (SRP) Protocol [Wu+98].

### 1.1. Motivations

Many of the constructions used today in public-key cryptography are based on number-theoretic problems such as integer factorization or finding logarithms in finite groups (discrete logarithms). Peter Shor showed that these two number-theoretic problems are susceptible to efficient attack by quantum computers [Sho97]. For a time, such weaknesses were mainly academic, as a general purpose quantum computer large enough to break real-world key sizes was seen as purely theoretical.

In recent years, however, research has advanced towards the construction of a practical general-purpose quantum computer on the scale capable of breaking modern security measures. The threat quantum computers provide against many of the currently existing cryptographic standards has gained more attention as a result. While no sufficiently large quantum computer capable of using Shor’s algorithm to attack secure communications exists as of yet, sufficient progress has been made to render them worth consideration for future security. Fur-
thermore, even if current systems are not under *immediate* threat of quantum attack, the time needed to smoothly transition to new, quantum-resistant schemes means that the transition cannot wait until a quantum threat is imminent.

In August 2015, the National Security Agency announced they would be pursuing post-quantum algorithms for their next suite of algorithms, to eventually replace the current algorithms in Suite B. In February 2016, at the PQCrypto 2016 conference, the National Institute of Standards and Technology informally launched a call for proposals for post-quantum algorithms, with a formal announcement planned for fall of 2016. Based on these announcements, the need for quantum resistant algorithms is hard to overstate.

At present, there are four main hard mathematical problems that have been used as the basis of proposed post-quantum asymmetric cryptographic primitives: hash-based, multivariate, lattice-based, and code-based; isogeny groups of supersingular elliptic curves have shown recent promise as an up-and-coming fifth branch. Hash-based signature schemes—such as the Lamport one-time signature scheme [Lam79] and the Merkle-tree construction [M+79; Mer89] that allows multiple one-time signatures to be based of a single private key—were originally proposed in the late 1970s but received little attention at the time compared to number-theoretic schemes like RSA and DSA. Interest has been revived for hash-based signatures in recent years because their security relies only on the existence of a secure hash function, and so the scheme remains secure against quantum adversaries as long as the hash-function used is quantum resistant.

A variety of signature schemes have been based on multivariate polynomial equations and error correcting codes as well; these two mathematical problems have also been the basis for a number of encryption schemes. We will discuss a variety of lattice-based primitives later in section 3.4. Key exchange protocols based on post-quantum problems, however, have been studied somewhat less than public-key encryption and signature schemes. Key exchange based on isogenies of supersingular elliptic curves has also been proposed [JD11]; a previous method
based on ordinary (i.e., not supersingular) elliptic curves [RS06] has been shown to be weak to a subexponential quantum attack [CJS14].

We present here in some detail a number of key exchange schemes designed to be resistant to quantum attacks. The basic scheme we present as our jumping-off point is a lattice-based analogue of Diffie–Hellman key exchange, with two independently published variations, one due to Ding et al. [DL12] and one from Peikert [Pei14]. The security of these schemes is based off of a particular hard lattice problem known as the learning with errors problem, which we detail in chapter 4. We also describe in section 5.2 an authenticated scheme due to Zhang et al. similar to HMQV [Zha+15], which represents the beginning of the work done. Our main result, in section 5.3 is an optimized version of the authenticated protocol, together with an analysis of the optimized version’s security in chapter 6. The security analysis makes up the bulk of the work done for this dissertation. We also examine a number of variations we can make to this streamlined version and how they affect the working of the protocol. We also present some performance comparisons of an implementation of this protocol to similar Diffie–Hellman style protocols based on the hardness of the learning with errors problem.
Chapter 2

Classical Key Exchange

The general purpose of cryptographic protocols is to allow two (or more) parties to securely communicate over an insecure communications channel. One of the critical aspects of a secure communications protocol is the secure generation and transmission of cryptographic keys. Since these keys are used to establish a secure channel, they must be established between the parties using the insecure lower-level channel.

Before the prevalence of personal computers, when ciphers were performed by hand or using mechanical tools like the German Enigma machine, it was common to exchange keys in person, in anticipation of later use. Unfortunately, this method does not work effectively for modern internet-based communications where two parties that will never meet in real life may establish multiple independent sessions, with separate session keys, over the course of a single day. One major breakthrough in this regard was Diffie and Hellman’s paper [DH76] introducing a key exchange protocol, allowing two parties to generate a shared secret key using a non-confidential communication channel.

Key exchange protocols are distinct from the notion of key encapsulation, an alternative technique with the same goal of establishing a shared key between two parties. Key encapsulation works by having one party generate a shared key, then encrypt it using the other party’s public key for an asymmetric encryption scheme. The primary advantage of key exchange over
key encapsulation, which we discuss in more detail in subsection 6.2.1, is the *forward secrecy* property.

If a shared key is sent using an asymmetric encryption method and the secret key for that method is later compromised, the shared secret can be decrypted, compromising the communications performed with that key. In contrast, a key exchange protocol can have a property known as forward secrecy, where an adversary compromising one party’s secret keys after the fact cannot use this information to recover the session key. The key exchange protocol needs to be properly constructed to actually obtain forward secrecy, but key encapsulation cannot have the forward secrecy property at all.

### 2.1. Diffie–Hellman

The Diffie–Hellman key-exchange protocol is one of the first public-key cryptographic primitives published. It allows two parties to jointly create a shared secret over an insecure communications channel.

The two parties, Alice and Bob, begin by publicly agreeing on a prime modulus $p$ and a primitive root $g$ modulo $p$. Each of the two then generates a secret integer exponent; Alice generates the exponent $a$ and Bob generates $b$. Alice sends the quantity $g^a$ to Bob, who sends $g^b$ back. Then Alice computes $K = (g^b)^a$ using her secret exponent and the quantity she receives from Bob, and Bob computes $K = (g^a)^b$ in a similar manner. Because modular exponentiation commutes under composition, the two calculate the same value for $K$. The protocol is illustrated in Figure 2.1. Typically, the result of this key exchange, $g^{ab}$, is not used directly, but used as the input to a secure key derivation function.

The security of this scheme relies on the difficulty of recovering the secret exponents $a$ and $b$ given the public messages $g^a$ and $g^b$. The *discrete logarithm problem* in a finite group $G$ is to find the exponent $x$ given group elements $g$ and $g^x$. An efficient algorithm that could solve discrete logarithms would render Diffie–Hellman key exchange useless; an adversary that can solve the discrete logarithm problem could recover the secret exponents and use them to reconstruct the
Fix \( p \) and \( g \)

Alice

Choose a random number
\( a \in \{2, \ldots, p - 1\} \).

Send \( g^a \)

Bob

Choose a random number
\( b \in \{2, \ldots, p - 1\} \).

Send \( g^b \)

Compute \( K = (g^b)^a \).

Compute \( K = (g^a)^b \).

Figure 2.1.: Diffie–Hellman Key Exchange

shared secret \( g^{ab} \). A number of algorithms exist that can solve the discrete logarithm problem, such as Shanks’ baby-step giant-step, Pollard’s rho and kangaroo algorithms, and the index calculus. None of these runs in polynomial time, but Shor’s algorithm, which requires a quantum computer and is discussed in further detail in section 2.3, does run in polynomial time.

2.1.1. Elliptic Curves

Note that in the protocol description above, the element \( g \) and public messages \( g^a \) and \( g^b \) could be considered elements of \( \mathbb{Z}_p^\times \), with \( g \) being a generator. However, nothing in the construction of the protocol specifically requires this. More generally, let \( G \) be any finite group and \( g \) any group element. Since \( g \) generates the subgroup \( \langle g \rangle \), assume without loss of generality that \( G \) is cyclic and generated by \( g \). Then Alice and Bob can pick exponents in \( \{2, \ldots, |G|\} \) and transmit \( g^a \) and \( g^b \), respectively, as in the standard Diffie–Hellman case, and will still wind up computing the same \( K = g^{ab} \). The security of this key exchange protocol is then based on the discrete logarithm problem in whichever group is chosen; it is of vital importance to use a group where the discrete logarithm problem is difficult. Multiplicative groups modulo a prime, the standard
setting for Diffie–Hellman, are believed to have hard discrete logarithm problems, as are the unit groups of finite fields with sufficiently large characteristic; the discrete logarithm in \( \mathbb{Z}_n \) under addition can be quickly solved using the Euclidean algorithm, however, rendering \( (\mathbb{Z}_n, +) \) unusable as the group for DH key exchange.

A recently popular type of group for this more general form of DH key exchange are groups on elliptic curves. An elliptic curve over a field \( K \) is the set of solutions to the equation

\[
y^2 = x^3 - ax - b
\]

where the polynomial \( x^3 - ax - b \) has no repeated roots. By including the point at infinity, an abelian group structure can be defined over the points in an elliptic curve, called an elliptic curve group. Elliptic curve groups require much smaller group elements than traditional Diffie–Hellman, greatly improving the scheme efficiency while providing what is believed to be equivalent security.

### 2.2. HMQV

The original DH protocol described above suffers from an important shortcoming. An adversary that can insert himself into the communications channel between Alice and Bob can perform a Man-in-the-Middle attack, performing a Diffie–Hellman exchange with each Alice and Bob, impersonating each one to the other. Then the adversary can manipulate the entire conversation at will, reading and modifying messages as he wishes.

It is preferable to use an authenticated key exchange (AKE) protocol, which guarantees to Alice and Bob that they are in fact establishing a key directly with each other, rather than a Man-in-the-Middle adversary. Such a protocol relies on static keys generated and published by the two parties and authenticated prior to the key exchange itself. Additionally, each run of the protocol between two parties should generate a new key not dependent solely on these

---

1The general form, applicable in fields of arbitrary characteristic, includes additional terms.
long term keys. More specifically, the system should have the forward security property, where an adversary who is able to later learn the static key of either or both parties cannot use this knowledge to retrieve the individual session keys derived from the protocol.

A relatively straightforward technique to provide authentication is to perform a standard key exchange and simply use the static keys for a signature scheme, signing the messages the parties exchange to prove their authenticity. This can, however, be less efficient than a key exchange protocol that directly uses static keys in the key generation process to provide authenticity, and requires further analysis of the signature scheme used to establish the security of the key exchange.

The Diffie–Hellman scheme has been the basis for a number of AKE protocols. One notable AKE protocol based off of DH is the MQV protocol from Menezes, Qu, and Vanstone [Law+03]; this was refined to the Hashed MQV (HMQV) protocol by Krawczyk [Kra05], which we describe in more detail below. An illustration is provided in Figure 2.2.

For HMQV, first we fix public parameters $p$ and $g$ as described for DH. We also fix a key derivation function $H$ and an intermediate hash function $\tilde{H}$. Alice picks a random exponent $a$ as her secret key, and publishes the corresponding public key $A = g^a$. Bob does likewise with $b$ and $B = g^b$. We assume that Alice has a method of verifying that $B$ belongs to Bob and vice versa, such as a certificate authority.

To actually perform key exchange, Alice and Bob begin by performing a standard DH key exchange. Alice generates a secret exponent $x$ and transmits $X = g^x$; Bob generates exponent $y$ and transmits $Y = g^y$. These quantities are referred to as the *ephemeral keys* for the session. Instead of simply computing the DH key $g^{xy}$, however, the two parties next compute the quantities $d = \tilde{H}(X, \hat{B})$ and $e = \tilde{H}(Y, \hat{A})$, where $\hat{A}$ denotes Alice’s identity (some publicly known information specific to Alice; for example, this could be simply her name, or it might include a public-key certificate) and $\hat{B}$ denotes Bob’s. Note that $d$ and $e$ can be derived using publicly transmitted messages, and so are themselves publicly known.
Fix $p, g, H, \tilde{H}$

Alice

Choose a random number

$a \in \{2, ..., p - 1\}$.

Publish $A = g^a$.

Choose a random number

$x \in \{2, ..., p - 1\}$.

Send $X = g^x$

Bob

Choose a random number

$b \in \{2, ..., p - 1\}$.

Publish $B = g^b$.

Choose a random number

$y \in \{2, ..., p - 1\}$.

Send $Y = g^y$

Compute $d = \tilde{H}(X, \hat{B})$.

Compute $e = \tilde{H}(Y, \hat{A})$.

Compute $\sigma_A = (YB^y)^x + da$.

Set $K = H(\sigma_A)$.

Compute $\sigma_B = (XA^d)^y + eb$.

Set $K = H(\sigma_B)$.

Figure 2.2.: HMQV Key Exchange

The key difference between the original MQV protocol and HMQV is the derivation of these two values—the original MQV protocol defines $d = 2^\ell + (X \mod 2^\ell)$ and $e = 2^\ell + (Y \mod 2^\ell)$, where $\ell = \lceil \log_2 q \rceil / 2$. Alice computes the shared secret $\sigma_A = (YB^y)^x + da$, and Bob computes the same shared secret as $\sigma_B = (XA^d)^y + eb$. The final session key is obtained by setting $K = H(\sigma_A) = H(\sigma_B)$.

2.3. Shor’s Algorithm

The original Diffie–Hellman key exchange and various extensions, which include HMQV and the elliptic curve versions of both DH and MQV, are subject to an important theoretical attack.
In 1994, Peter Shor formulated a method for factoring integers quickly using a quantum computer, which he soon modified to provide a way to solve the discrete logarithm problem, as well [Sho97]. Thus, the existence of Shor’s Algorithm implies that a quantum computer could render classical DH schemes impotent. Since, at the time of writing, the construction of a practical, general-purpose quantum computer is an active research field, it is desirable to design replacement cryptographic primitives that are not subject to the same weakness, and so be prepared in the event that a quantum computer capable of defeating DH is created.

Shor’s original factorization algorithm can be split into two main parts. First is a non-quantum algorithm that, given a composite input integer \( n \) and a method to compute the multiplicative order \( r \) of an element \( x \) modulo \( n \), outputs nontrivial factors of \( n \). We give this reduction in algorithm 1. Note that if \( n \) is already prime, or is a prime power, this method fails, but these cases can be checked for ahead of time. The second portion, illustrated in algorithm 2 and described in more detail below, is a subroutine for computing the multiplicative order of \( x \) modulo \( n \). This is the portion of Shor’s algorithm that requires a quantum computer. This quantum portion can also be adapted into a separate algorithm to find discrete logarithms.

**Input**: An integer \( n \) with at least one pair of coprime proper divisors

**Output**: A proper divisor of \( n \)

1. Start

\[ \text{Pick random } x < n; \]

\[ \text{if } \gcd(x, n) \neq 1 \text{ then} \]

\[ \text{return } \gcd(x, n) \]

\[ \text{else} \]

\[ \text{Use algorithm 2 to find the order } r \text{ of } x \text{ modulo } n.; \]

\[ \text{if } r \text{ is odd then } \text{Go back to start;} \]

\[ \text{if } x^{r/2} \equiv -1 \pmod{n} \text{ then } \text{Go back to start;} \]

\[ \text{return } \gcd(x^{r/2} - 1, n) \]

2. **Algorithm 1**: Factorization to Order Reduction

To perform Shor’s order finding quantum algorithm, first we find \( q \) the power \( 2^x \) such that \( n^2 \leq q < 2n^2 \). Next we initialize our quantum computer with two \( x \)-qubit registers, the “input” and “output” registers. We set the input register to a uniform superposition of the integers \( \{0, 1, \ldots, q - 1\} \) and the output register to 0. The resulting state of the computer after this
step is
\[ \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle \otimes \langle 0|. \] (2.3.1)

Next, we apply the function \( f(a) = x^a \mod n \) to the output register, resulting in a state for our quantum computer of
\[ \frac{1}{\sqrt{q}} \sum_{a=0}^{q-1} |a\rangle \otimes |x^a \mod n\rangle. \] (2.3.2)

Next, we apply an algorithm known as the quantum Fourier transform (QFT) to the input register, bringing the quantum state to
\[ \frac{1}{q} \sum_{a=0}^{q-1} \sum_{c=0}^{q-1} \omega^{ac} |c\rangle \otimes |x^c \mod n\rangle \] (2.3.3)

where \( \omega = \exp(2\pi i/q) \) is a primitive \( q \)-th root of unity. Finally, we measure the state of the input register to obtain a value \( y \), with the output register containing the value \( z \). With high probability, \( y \) is a multiple of \( q/r \), where \( r \) is the desired order, so \( y/q = d/r \) for some integer \( d \).

We use continued fraction expansion on \( y/q \) to estimate this \( d/r \), and take the denominator \( r \) that we find as our candidate for the order. We check whether \( x^r \equiv 1 \pmod{n} \). If so, it is the order we seek. Otherwise, we try small multiples of this \( r \) as well as the denominators of other fractions close to \( q/y \).

More generally, Shor’s algorithm can be adapted to find solutions to a general computational problem known as the hidden subgroup problem (HSP) on abelian groups [ME99]. If \( H \) is a subgroup of a group \( G \), we say a function \( f: G \to X \) onto a finite set \( X \) separates cosets of \( H \) if \( f(g_1) = f(g_2) \) precisely when \( g_1H = g_2H \). Equivalently, \( H \) is the stabilizer of \( f \). The hidden subgroup problem asks, given a group \( G \) and oracle/black-box access to a function \( f \) that separates cosets of some unknown subgroup \( H \), to find the hidden subgroup \( H \). A number of number-theoretic problems commonly used as the basis cryptographic primitives, such as integer factorization and discrete logarithm, can be considered special cases of the abelian HSP;
**Input**: Integers \( n \) and \( x < n \)

**Output**: The order of \( x \) modulo \( n \)

Find \( q = 2^k \) such that \( n^2 \leq q < 2n^2 \);
Initialize to \( q^{-1/2} \sum_{a} |a\rangle \otimes |0\rangle \);
Compute \( q^{-1/2} \sum_{a} |a\rangle \otimes |x^a \text{ mod } n\rangle \);
Apply quantum Fourier transform;
Measure input register to obtain \( y \);
Estimate \( y/q \) with \( d/r \);
Test candidate \( r \) and multiples;
if **Any candidates pass** then
    | **return** that \( r \)
else
    | retry from beginning

**Algorithm 2**: Shor’s Order Finder

many of the lattice problems we will describe in section 3.3 are equivalent to HSP over specific non-abelian groups.

We generalize Shor’s procedure to the HSP in algorithm 3. Let \( (G,+) \) be an abelian group, let \( f: G \to X \) be a function that separates cosets of \( H \leq G \), and can be computed on a quantum computer. We generalize the quantum Fourier transform as follows: let \( \{ \chi_\ell \}_{\ell \in G} \) be the set of irreducible characters on \( G \); since there are exactly \( |G| \) of these, we will index them by elements of \( G \). Define

\[
\mathcal{F}|g\rangle = \frac{1}{\sqrt{|G|}} \sum_{h \in G} \chi_\ell(h)|h\rangle.
\tag{2.3.4}
\]

Then the quantum Fourier transform \( \mathcal{F} \) over the group \( G \) is defined linearly as

\[
\mathcal{F} \sum_{g} \alpha_g |g\rangle = \sum_{g} \alpha_g \mathcal{F}|g\rangle.
\tag{2.3.5}
\]

If \( G = \mathbb{Z}_n \), we have the \( n \) characters \( \chi_k(x) = \omega^{kx} \) where \( \omega \) is a primitive \( n \)-th root of unity, so this is consistent with the application of the QFT over \( \mathbb{Z}_q \) in (2.3.3). If we apply \( \mathcal{F} \) to the state \( |G|^{-1/2} \sum_{g \in G} |g\rangle \otimes |f(g)\rangle \) and measure the input register, we obtain an index \( \ell \) at random such that \( \chi_\ell \) is equal to 1 on \( H \). Repeating this procedure a number of times, we can obtain a
number of such characters that we can use to recover $H$ in a manner dependent on the exact nature of the group $G$.

**Input**: Abelian group $G$ and $f: G \to X$ that separates cosets of $H$

**Output**: Hidden subgroup $H$

repeat
  | Initialize to $|G|^{-1/2} \sum_g |g\rangle \otimes |0\rangle$;
  | Compute $|G|^{-1/2} \sum_g |g\rangle \otimes |f(g)\rangle$;
  | Apply quantum Fourier transform;
  | Measure input register to obtain $\ell_k$;
until enough $\ell_k$ obtained;
Use $\{\ell_k\}_k$ to recover $H$;
**return** $H$;

**Algorithm 3**: Generalized Shor’s Algorithm

Shor’s order finding procedure works to find the order $r$ of $x \pmod{n}$ using the function $f(a) = x^a \pmod{n}$, which finds the coset $H = r\mathbb{Z}/N\mathbb{Z}$, where $N$ is some multiple of $r$. Consider the character $\chi_\ell(z) = \omega^{\ell z}$; then $\chi_\ell = 1$ for multiples of $r$ if and only if $\ell r$ is a multiple of $N$. The complication with this approach is finding $N$ to define $G = \mathbb{Z}_N$; this leads to the extra work of testing each candidate order.
In recent years, a great deal of cryptographic focus has been placed on a particular type of mathematical object known as *lattices*, which are objects that exhibit a regular, grid-like structure. Formally, a lattice is a collection of vectors in euclidean space $\mathbb{R}^n$ with two specific properties:

1. The sum or difference of any two vectors in a lattice is again a vector in the lattice—that is, a lattice is a commutative subgroup of $\mathbb{R}^n$ under addition.

2. There is a length $\lambda$ such that no non-zero vector in the lattice has length less than $\lambda$. Topologically, this means that a lattice is a *discrete* subset of $\mathbb{R}^n$.

Much of the vocabulary for discussing vector spaces over $\mathbb{R}$ works for discussing lattices as well. A *basis* for a lattice is a set of linearly independent vectors such that any vector in the lattice can be expressed as a unique $(\mathbb{Z}-)$linear combination of basis vectors. The only difference between this definition and a basis of a real vector space is that a lattice allows only integer coefficients, representing repeated additions, instead of arbitrary real scalars. Likewise, the *rank* of a lattice is the number of vectors in any basis of the lattice. Note that the rank of a lattice cannot exceed
the dimension of the ambient space \(\mathbb{R}^n\); we will largely confine ourselves to full-rank lattices, where the rank is equal to the dimension of the ambient euclidean space.

Given a set of vectors \(B\), we define the lattice generated by those vectors \(\mathcal{L}(B)\) to be the set of all linear combinations of vectors in \(B\) with integer coefficients. Contrast this with \(\text{span}(B)\), the set of all linear combinations with real coefficients. Likewise, if \(B\) is a matrix, the lattice \(\mathcal{L}(B)\) is the lattice generated by the columns of \(B\). Generally, we will work with lattices all of whose points have only rational coordinates. Such a lattice is equivalent to one with integer coefficients, by scaling all the basis vectors by a common factor to cancel all the denominators, so there is no further loss of generality by restricting ourselves to integer lattices.

As an example in two dimensions, the lattice \(\mathcal{L}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\) would be the collection \(\mathbb{Z}^2\) of all points in \(\mathbb{R}^2\) with two integer coefficients, which can be seen in Figure 3.1. The previous example of \(\mathbb{Z}^2\) is a rather special lattice, being generated from an orthonormal basis. If we instead choose as our basis the matrix \(\begin{bmatrix} \frac{2}{3} & \frac{1}{3} \end{bmatrix}\), we obtain the lattice illustrated in Figure 3.2. This lattice is somewhat skewed, and in fact cannot be generated by any orthogonal basis.

Now, consider two \(n \times n\) basis matrices \(B\) and \(B'\) that generate the same lattice \(\Lambda\). Since each column of \(B'\) is a vector in \(\Lambda = \mathcal{L}(B)\), that column can be written as a \(\mathbb{Z}\)-linear combination of the columns of \(B\). This means we can write \(B' = BU\) for some matrix \(U \in \mathbb{Z}^{n \times n}\). In a
similar manner, we have \( B = B'U^{-1} \). That is, any two bases for the same lattice are related by multiplication with an invertible integer matrix. Since \( U \) is an integer matrix with an integer matrix inverse, we have \( \det U = \pm 1 \), and so \( \det B = \pm \det B' \). Thus, every basis for a given lattice \( \Lambda \) has the same determinant, except for a possible difference in sign. We define the determinant of the lattice itself, written \( \det \Lambda \), to be the positive of these values, \( |\det B| \) where \( B \) is any basis for \( \Lambda \).

There is a geometric interpretation of the lattice determinant as well. Consider a basis \( B = [b_1, b_2, \ldots, b_n] \) for a lattice \( \Lambda \). We define the fundamental region of this basis to be the half-open parallelepiped

\[
\mathcal{P}(B) = \left\{ \sum_{i=1}^{n} \alpha_i b_i \mid 0 \leq \alpha_i < 1 \right\}
\]

(3.1.1)

enclosed by the basis vectors. The volume of \( \mathcal{P}(B) \) is equal to \( |\det B| \), and so every fundamental region has volume \( \det \Lambda \). This is shown in Figure 3.3, where the fundamental regions corresponding to the two bases \( \begin{bmatrix} 1 & 3 \\ 2 & 1 \end{bmatrix} \) and \( \begin{bmatrix} 4 & 5 \\ -1 & -3 \end{bmatrix} \) are shown. If we consider \( \mathbb{R}^n \) as an additive group with the lattice \( \Lambda = \mathcal{L}(B) \) as a subgroup, then the fundamental region \( \mathcal{P}(B) \) forms a set of representatives of the quotient group \( \mathbb{R}^n/\Lambda \).
Figure 3.3.: Two Fundamental Regions
Given a lattice $\Lambda$, we may construct the *dual lattice*

$$\Lambda^* = \{ y \in \text{span}(\Lambda) \mid \forall x \in \Lambda, \langle x, y \rangle \in \mathbb{Z} \}$$  \hspace{1cm} (3.1.2)

Another key consideration for a lattice $\Lambda$ is its collection of *successive minima*. For a rank $n$ lattice and $1 \leq i \leq n$, the $i$-th successive minimum, denoted $\lambda_i$, is the radius of the smallest ball centered at $0$ which contains $i$ linearly independent non-zero lattice vectors. In particular, $\lambda_1$ is the length of the shortest non-zero vector in $\Lambda$. This value can be related back to the determinant of the lattice via the Hermitian bound, as seen in (3.1.3):

$$\lambda_1 \leq \sqrt{n} (\det \Lambda)^{1/n}$$  \hspace{1cm} (3.1.3)

The bound itself is a corollary of Minkowski’s convex body theorem:

**Theorem 3.1.1 (Minkowski).** *Let $\Lambda \subset \mathbb{R}^n$ be a full rank lattice, and let $S \subset \mathbb{R}^n$ be a convex set symmetric about the origin with volume greater than $2^n \det \Lambda$. Then $S$ contains a non-zero point of $\Lambda$.***

Lattice-based cryptography tends to focus on a number of specific varieties of lattices. One particularly important class of lattices is the class of *$q$-ary* lattices for a modulus $q$. We say a rank $n$ lattice $\Lambda$ is $q$-ary if $q \cdot \mathbb{Z}^n \subseteq \Lambda \subseteq \mathbb{Z}^n$, so that whether a vector $x$ belongs to $\Lambda$ depends only on $x \mod q$. The other important class of lattices are *ideal lattices*. Consider the ring $\mathbb{R} = \mathbb{Z}[x]/\langle f(x) \rangle$ for a monic polynomial of degree $n$. Then $\mathbb{R} \cong \mathbb{Z}^n$ as an additive group, and can be embedded into $\mathbb{R}^n$ in a natural way by identifying an element $a = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$ with its coefficient vector $a = [a_0, a_1, \ldots, a_{n-1}]$. Then the image of any ideal $I \subseteq \mathbb{R}$ corresponds to a sublattice of $\mathbb{Z}^n$ under this embedding. A lattice isomorphic to such an ideal is called an ideal lattice. Importantly, if the polynomial $f$ is irreducible, every ideal lattice obtained in this way is full-rank.
3.2. Lattice Reduction

In general, the complexity of linear algebra problems will depend on the basis used for computation. The same holds for many lattice problems, with “good” bases typically making computations easier, for some appropriate definition of “good”. Thus, an important computational technique for lattice problems is to convert a given lattice basis into an equivalent basis—one that generates the same lattice as the original—with better computational properties. Typically, this means a basis with shorter vectors that are closer to orthogonal. The general term for this process is reducing the given basis.

The question then arises, given a basis \( B = [b_1, b_2, \ldots, b_n] \), how close to optimal is \( B \)? Assume without loss of generality that the vectors of \( B \) are sorted in increasing order of length. If these lengths are exactly the successive minima of the lattice, so \( \|b_j\| = \lambda_j \), then no basis can have shorter vectors than \( B \), so \( B \) is optimal. The converse need not be true, however. It is possible to construct a lattice such that every basis contains at least one vector with length not one of the successive minima. The lattice \( \Lambda \) of integer points all of whose coordinates have the same parity is an example when the rank \( n > 4 \). The vector \( v = [1, 1, \ldots, 1] \) is in \( \Lambda \), having all coordinates odd, with \( \|v\| = \sqrt{n} \). On the other hand, \( 2e_i \) has norm 2, giving a total of \( n \) vectors of length 2, so \( \lambda_1 = \cdots = \lambda_n = 2 \).

We can list a number of different methods for measuring the “goodness” of a basis. One simple measure is the length of the longest vector in the basis, \( \mu(B) = \max_i \|b_i\| \). Then a basis \( B \) is better the smaller \( \mu(B) \) is. A somewhat more common measure is the orthogonality defect, defined as the ratio \( \prod_i \|b_i\|/\det(B) \). Since any two equivalent bases have the same determinant, for a fixed lattice the orthogonality defect of each basis is proportional to the product of the lengths of its vectors. Furthermore, the orthogonality defect is always greater than or equal to 1, with equality occurring precisely for an orthogonal basis. One convenient variation of the orthogonality defect is the normalized orthogonality defect, defined for a basis \( B \) as

\[
\delta(B) = \left( \frac{\prod_i \|b_i\|}{\det B} \right)^{1/n}.
\] (3.2.1)
The advantage of this normalization is that scaling the entire lattice by a constant $c$ also scales $\delta(B)$ linearly by $c$. More generally, if $S = [s_1, ..., s_n]$ is a collection of $n$ linearly independent vectors in a lattice $\Lambda$—which need not be a basis as $S$ need not generate all of $\Lambda$—we can define the normalized orthogonality defect of $S$ as $\delta(\Lambda)(S) = (\prod_i \|s_i\|/\det \Lambda)^{1/n}$. The smallest orthogonality defect $\delta(B)$ among all bases $B$ for $\Lambda$ is denoted $\delta(\Lambda)$, while the smallest defect $\delta(\Lambda)(S)$ among all linearly independent sets is denoted $\delta(\Lambda)(S)$, which is equal to $\left(\prod_i \lambda_i/\det \Lambda\right)^{1/n}$. Just as $\lambda_1 \leq \sqrt{n} \cdot (\det \Lambda)^{1/n}$ as in the Hermite bound (3.1.3), we also have that for any lattice $\Lambda$,

$$1 \leq \delta(\Lambda)(\Lambda) \leq \delta(\Lambda) \leq \sqrt[n]{n}.$$  

### 3.2.1. Gram–Schmidt

We now recall Gram–Schmidt orthogonalization. Let $b_1, b_2, ..., b_n$ be a set of linearly independent vectors in $\mathbb{R}^n$. Then the Gram–Schmidt orthogonalization of these vectors is the collection $b_1^*, b_2^*, ..., b_n^*$ where

$$b_1^* = b_1,$$

$$b_i^* = b_i - \sum_{j=1}^{i-1} \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} b_j^* \quad 2 \leq i \leq n.$$  

Then the collection of $b_i^*$ are mutually orthogonal, and span the same $\mathbb{R}$-linear space as the collection of $b_i$. When moving back to the case of a lattice, however, the orthogonal vectors $b_i^*$ in general do not generate the same lattice as the original basis. Consider the Gram–Schmidt coefficients

$$\mu_{i,j} = \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} \quad 1 \leq j < i \leq n.$$  

If all of these were integers, then the bases $B = [b_1, b_2, ..., b_n]$ and $B^* = [b_1^*, b_2^*, ..., b_n^*]$ would be equivalent. Methods for reducing lattice a basis often work in a way similar to Gram–
Schmidt, while taking steps to ensure that the resulting basis generates the same lattice as the original.

### 3.2.2. Gauss Reduction

In the case of a 2-dimensional lattice, the problem of lattice reduction is easily solved. A modified version of Gaussian reduction, given in algorithm 4, is able to exactly find a reduced basis in time polynomial in the length of the longest input vector. We say a lattice basis \([\mathbf{a}, \mathbf{b}]\) in \(\mathbb{R}^2\) is *reduced* if \(\|\mathbf{a}\|\) and \(\|\mathbf{b}\|\) are both smaller than \(\|\mathbf{a} + \mathbf{b}\|\) and \(\|\mathbf{a} - \mathbf{b}\|\). The key result in two dimensions is that a basis is reduced if and only if its two vectors have lengths \(\lambda_1\) and \(\lambda_2\)—i.e., if \(\mathbf{a}\) and \(\mathbf{b}\) are the shortest vectors in the lattice they generate. Furthermore, we may assume without loss of generality that \(\mathbf{a}\) is shorter than \(\mathbf{b}\) (by exchanging the two if not), and that \(\mathbf{a} - \mathbf{b}\) is shorter than \(\mathbf{a} + \mathbf{b}\) (since \(\mathbf{b}\) can be replaced with \(-\mathbf{b}\) without changing the generated lattice).

Then the basis is reduced, and thus optimal, if \(\|\mathbf{b}\| \leq \|\mathbf{a} - \mathbf{b}\|\).

**Algorithm 4:** Gauss Reduction

```plaintext
Input : Two vectors \(\mathbf{a}, \mathbf{b} \in \mathbb{R}^2\)
Output: A fully reduced basis for \(\mathcal{L}(\mathbf{a}, \mathbf{b})\)

if \(\|\mathbf{a}\| > \|\mathbf{b}\|\) then swap(\(\mathbf{a}, \mathbf{b}\));
if \(\|\mathbf{a} - \mathbf{b}\| > \|\mathbf{a} + \mathbf{b}\|\) then \(\mathbf{b} := -\mathbf{b}\);
if \([\mathbf{a}, \mathbf{b}]\) is reduced then return \([\mathbf{a}, \mathbf{b}]\);
if \(\|\mathbf{a}\| \leq \|\mathbf{a} - \mathbf{b}\|\) then go to loop;
if \(\|\mathbf{a}\| = \|\mathbf{b}\|\) then return \([\mathbf{a}, \mathbf{a} - \mathbf{b}]\);
\([\mathbf{a}, \mathbf{b}] := [\mathbf{b} - \mathbf{a}, \mathbf{a}]\);

loop while \([\mathbf{a}, \mathbf{b}]\) is not reduced do

\[
\mu := \frac{\langle \mathbf{a}, \mathbf{b} \rangle}{\|\mathbf{a}\|^2};
\]
\[
\mathbf{b} := \mathbf{b} - \mu \mathbf{a};
\]
if \(\|\mathbf{a} - \mathbf{b}\| > \|\mathbf{a} + \mathbf{b}\|\) then \(\mathbf{b} := -\mathbf{b}\);
swap(\(\mathbf{a}, \mathbf{b}\));

end

return \([\mathbf{a}, \mathbf{b}]\);
```

Algorithm 4: Gauss Reduction
3.2.3. LLL Reduction

The Gauss reduction algorithm described above provides a basis with lengths exactly equal to the successive minima of the generated lattice, and in polynomial time. Unfortunately for the process of lattice reduction, this only works for rank 2 lattices. For higher ranks, other techniques are required. The LLL lattice reduction algorithm, published by Lenstra, Lenstra, and Lovász in 1982 [LLL82], finds relatively short bases in polynomial time. We detail LLL in algorithm 5.

To start with LLL reduction, fix a parameter \( \delta \in (\frac{1}{4}, 1) \). Larger values provide a better basis, at the expense of more time needed. The value \( \delta = \frac{3}{4} \) is relatively common. Let \( B \) be a basis, and let \( \pi_i \) be the projection function onto \( (\text{span}(b_1, \ldots, b_{i-1}))^\perp \). Then the Gram–Schmidt vectors satisfy \( b_i^* = \pi_i(b_i) \).

**Definition 3.2.1.** We say that a basis \( B \) is LLL-reduced (with parameter \( \delta \)) if

- \( |\mu_{i,j}| \leq \frac{1}{2} \) for all \( i > j \), where \( \mu_{i,j} \) is the Gram–Schmidt coefficient defined in (3.2.3).
- For any consecutive basis vectors \( b_i \) and \( b_{i+1} \),

\[
\delta \|\pi_i(b_i)\|^2 \leq \|\pi_i(b_{i+1})\|^2
\]

(3.2.4)

The general idea of LLL reduction is very similar to that of Gaussian reduction. Basis vectors are taken two at a time, and reduced with respect to each other, exactly as in Gauss reduction. The complication comes in when considering the remaining vectors in the lattice. The process of reducing two vectors relative to each other may reduce the angles between those two and the remaining basis vectors, making the basis less orthogonal overall. In general, LLL is able to obtain short bases, in the sense that the \( k \)-th shortest vector in a basis returned by LLL is at most \( 2^{\frac{n}{2}} \lambda_k \). Furthermore, LLL runs in polynomial time; more precisely, LLL on a rank \( n \) lattice

---

1The LLL reduction algorithm remains well-defined for \( \delta = 1 \), but in this case it is an open problem whether the algorithm still runs in polynomial time.
basis in $\mathbb{R}^m$ runs in time $O(n^5 m \log^3 \beta)$ where $\beta$ is the length of the longest vector in the input basis.

**Input**: A basis $B = [b_1, ... , b_n]$

**Output**: An LLL-reduced basis generating the same lattice as the input $B^* = [b_1^*, ..., b_n^*]$ := Gram–Schmidt basis for $B$;

$\mu_{ij} := \frac{(b_i, b_j)}{||b_j||^2}$;

$k := 2$;

**Algorithm 5: LLL Reduction**

```
while $k \leq n$ do
    for $j = k - 1$ to 1 do
        if $|\mu_{kj}| > \frac{1}{2}$ then
            $b_k := b_k - \lfloor \mu_{kj} \rfloor b_j$;
            Update $B^*$, $\mu_{ij}$ as needed;
        end
    end
    if $||b_k||^2 \geq (\delta - \mu_{kk}^2)||b_{k-1}||^2$ then
        $k := k + 1$;
    else
        swap($b_k, b_{k-1}$);
        Update $B^*$, $\mu_{ij}$;
        $k := \max(k - 1, 1)$;
    end
end
```

3.3. Lattice Problems

Recent cryptographic interest in lattices comes from the ability to define computationally hard problems over them. These problems often have a number of attractive computational properties as well. First, many lattice problems have a search/decision equivalence. Another common feature of lattice problems is an average case/worst case equivalence. Both of these properties have useful consequences in constructing security proofs.

We will distinguish between two primary types of computational problems: search problems and decision problems. Somewhat informally, a search problem is a computational problem
that requires as its solution an object with a designated property or a claim that no such object exists; a decision problem is a computational problem that requires a single YES or NO as its solution. For any search problem there is a related decision problem asking if any solution to the search problem exists. As an example, consider the problem of 3-coloring a graph. The decision problem of 3-coloring a graph asks whether or not a given graph can be colored with three colors. The corresponding search problem asks for such a coloring if one exists.

For any search problem and the corresponding decision problem, it is rather easy to solve an instance the decision problem given an algorithm that solves the search problem: if the search version solver returns any solution, then the answer to the decision problem is YES; if the search solver returns that no solution exists, the answer to the decision problem is NO. The converse is not generally true.

Going back to the example of 3-colorings, knowing that a coloring with the desired properties exists does little to actually help find the coloring. For many lattice problems, however, the ability to efficiently solve a decision problem can be leveraged to effectively solve the corresponding search problem. Thus, lattice based cryptographic primitives can typically rely on the hardness of the decision version of a problem—which is often easier to incorporate into security proofs—without sacrificing the hardness of the search version.

The second useful property exhibited by many computational problems over lattices is an equivalence between worst-case and average-case hardness. Traditionally, hardness results in computer science have focused on worst-case hardness, though work on average-case hardness has grown since Levin defined average-case complete problems [Lev86]. The worst-case complexity of an algorithm is the largest running time of the algorithm taken over all possible inputs of a given size, while the average-case complexity is the average, taken over some probability distribution of possible inputs, of the running time of the algorithm.

For practical matters, such as cryptographic protocols, the average-case complexity is the more relevant measure of complexity, rather than worst-case. Unfortunately, average-case hardness can be harder to work with, partly because it requires knowledge of the input distribution
to compute. The ability to relate the average-case hardness of a lattice problem used to define a cryptographic primitive with the worst-case complexity of a similar, or even the same, lattice problem allows us to prove the hardness of the average-case complexity that we need while using the more robust techniques applicable to worst-case complexity analysis.

### 3.3.1. Shortest Vector Problem

We now turn to specific lattice problems useful for cryptographic constructions. The first of these is the Shortest Vector Problem, (SVP). Recall that the discrete structure of a lattice implies that, given a lattice $\Lambda$, there is a length $\lambda_1$ such that some vector in $\Lambda$ has length equal to $\lambda_1$, while no non-zero vector in $\Lambda$ has length less than $\lambda_1$. We define two versions of SVP here, the decision and search versions.

**Definition 3.3.1 (Decision SVP).** An instance of SVP is a pair $(B, r)$ where $B$ is a lattice basis and $r > 0$ is a real number. We say an algorithm $\mathcal{A}$ solves SVP if, given an SVP instance $(B, r)$, $\mathcal{A}$ outputs

- **YES** if $\|x\| \leq r$ for some non-zero $x \in \mathcal{L}(B),$

- **NO** if $\|x\| > r$ for all non-zero $x \in \mathcal{L}(B)$.

**Definition 3.3.2 (Search SVP).** An instance of Search SVP is a lattice basis $B$. We say an algorithm $\mathcal{A}$ solves Search SVP if, given a Search SVP instance $B$, $\mathcal{A}$ outputs some non-zero $x \in \mathcal{L}(B)$, where $\|x\| \leq \|y\|$ for any non-zero $y \in \mathcal{L}(B)$.

One straightforward generalization of the above definition is to replace the norm $\|\cdot\|$ which we define SVP with respect to. Typically, we assume the standard $\ell^2$ norm inherited from $\mathbb{R}^n$, but any norm on $\Lambda$ can be used. Note that changing the norm may change the complexity of SVP; stronger hardness results are known for SVP with respect to the sup norm $\|\cdot\|_{\infty}$ than the usual norm.

In addition to the exact SVP described above, we can also define an approximate version of SVP, known as GapSVP. GapSVP is what’s known as a promise problem, a generalization of
the class of decision problems where some inputs are allowed to return either a YES or NO answer. Let \( \gamma \) be a fixed function of the rank \( n \), which we call the approximation factor. Then \( \text{GapSVP} \) is similar to the decision version of \( \text{SVP} \), except that either a YES or NO answer is allowed for lattices where the shortest vector is in the “gap” between the target length \( r \) and the approximation \( \gamma \cdot r \); that is, \( r < \lambda_1 < \gamma \cdot r \).

**Definition 3.3.3 (GapSVP).** Fix an approximation factor \( \gamma = \gamma(n) \). An instance of \( \text{GapSVP}_\gamma \) is a pair \((B, r)\) where \( B \) is a lattice basis and \( r > 0 \) is a real number. We say an algorithm \( A \) solves \( \text{GapSVP}_\gamma \) if, given a \( \text{GapSVP}_\gamma \) instance \((B, r)\), \( A \) outputs

- **YES** if \( \|x\| \leq r \) for some non-zero \( x \in \mathcal{L}(B) \),
- **NO** if \( \|x\| > \gamma r \) for all non-zero \( x \in \mathcal{L}(B) \).

Note that if \( \gamma = 1 \), then \( \text{GapSVP}_\gamma \) is exactly \( \text{SVP} \).

In addition to defining \( \text{GapSVP}_\gamma \) as an approximate version of decision \( \text{SVP} \), we can define \( \text{SVP}_\gamma \) as an approximate version of search \( \text{SVP} \).

**Definition 3.3.4 (Approximate Search SVP).** As in \( \text{GapSVP} \), fix an approximation factor \( \gamma = \gamma(n) \). An instance of \( \text{SVP}_\gamma \) is a lattice basis \( B \). We say an algorithm \( A \) solves \( \text{SVP}_\gamma \) if, given an \( \text{SVP}_\gamma \) instance \( B \), \( A \) outputs some non-zero \( x \in \mathcal{L}(B) \), where \( \|x\| \leq \gamma \cdot \|y\| \) for any non-zero \( y \in \mathcal{L}(B) \).

**SVP Hardness**

A number of hardness results exists for the various versions of \( \text{SVP} \). First, if we consider the \( \ell_\infty \) norm, the decision \( \text{SVP} \) is \( \text{NP} \)-complete [MG12]. Unfortunately, this result does not fully generalize to other \( \ell_p \) norms. The best known results to date for these norms is that \( \text{SVP} \), and even \( \text{GapSVP}_\gamma \) with \( \gamma < \sqrt[4]{2} \), is \( \text{NP} \)-hard under random or non-uniform reductions. Since \( \text{NP} \)-completeness requires deterministic reductions, this hardness result is weaker than for the \( \ell_\infty \) norm, but the existence of even random reductions from \( \text{NP} \)-complete problems suggests
the practical hardness of SVP. Note that as the approximation factor $\gamma$ increases, the complexity of approximation decreases. As mentioned in subsection 3.2.3, the LLL reduction algorithm can solve $\text{SVP}_\gamma$ in polynomial time for exponential $\gamma$. It remains an open problem whether $\text{SVP}_\gamma$ is hard for polynomial $\gamma$, though it is widely conjectured that this is the case.

### 3.3.2. Closest Vector Problem

In addition to SVP, we discuss here a second computationally hard problem stemming from the discrete nature of lattices. Just as (non-zero) lattice vectors cannot be arbitrarily short, lattice vectors cannot be arbitrarily close to a given point in $\mathbb{R}^n$. For any $t \in \mathbb{R}^n$ and a lattice $\Lambda \subset \mathbb{R}^n$, we can express this distance $\text{dist}(t, \Lambda) = \min_{v \in \Lambda} \|t - v\|$. The Closest Vector Problem (CVP) asks for such lattice vectors for a given target vector $t$. More formally, we can define both decision and search versions as follows:

**Definition 3.3.5 (Decision CVP).** An instance of decision CVP is a triple $(B, t, r)$ where $B$ is a basis for a lattice in $\mathbb{R}^n$, $t \in \text{span}(B)$, and $r \in \mathbb{R}^+$ is a distance. An instance is a YES instance if $\text{dist}(t, \mathcal{L}(B)) \leq r$ and a NO instance otherwise.

**Definition 3.3.6 (Search CVP).** An instance of search CVP is a pair $(B, t)$ where $B$ is a basis for a lattice in $\mathbb{R}^n$, $t \in \text{span}(B)$. Given an instance $(B, t)$, a solution to search CVP is a vector $v \in \mathcal{L}(B)$ such that $\|v - t\| = \text{dist}(t, \mathcal{L})$. That is, $\|v - t\| \leq \|w - t\|$ for any vector $w \in \mathcal{L}(B)$.

Note that, like SVP, there need not be a unique solution to the search CVP problem. To take a somewhat extreme case, if $B$ generates $\mathbb{Z}^n$ and $t$ is the center of the circumsphere around a fundamental region, each corner is a lattice point that solves CVP. Unlike the shortest vector problem, however it is possible to have a completely unique solution, where the negation of any solution to SVP is also a solution.

Also similar to the shortest vector problem, we can define approximate version of both decision and search CVP.
**Definition 3.3.7 (GapCVP).** Fix an approximation factor \( \gamma = \gamma(n) \) as a function of the rank.

An instance of GapCVP, \( \gamma \) is a triple \((B, t, r)\) where \( B \) is a basis for a lattice in \( \mathbb{R}^n \), \( t \in \text{span}(B) \), and \( r \in \mathbb{R}^+ \) is a distance. An instance is a YES instance if \( \text{dist}(t, \mathcal{L}(B)) \leq r \) and a NO instance if \( \text{dist}(t, \mathcal{L}(B)) > \gamma \cdot r \).

**Definition 3.3.8 (Approximate CVP).** An instance of search CVP is a pair \((B, t)\) where \( B \) is a basis for a lattice in \( \mathbb{R}^n \), \( t \in \text{span}(B) \). Given an instance \((B, t)\), a solution to search CVP is a vector \( v \in \mathcal{L}(B) \) such that \( ||v - t|| \leq \gamma \cdot \text{dist}(t, \mathcal{L}) \). That is, \( ||v - t|| \leq \gamma \cdot ||w - t|| \) for any vector \( w \in \mathcal{L}(B) \).

**CVP Hardness**

Existing hardness results for CVP are stronger than those for SVP. While SVP can currently only be shown to be NP-hard using randomized or non-uniform reductions, CVP has a deterministic reduction from subset-sum, and is therefore unconditionally NP-complete with respect to any \( \ell_p \) norm. Importantly, this result extends even to approximating CVP.

**Theorem 3.3.9 ([MG12], Corollary 3.11).** Let \( c > 0 \) be constant, and let \( \gamma(n) = \log^c n \). Then for any \( \ell_p \) norm, GapCVP, \( \gamma \) is NP-hard.

Note that the hardness of an individual instance of CVP depends greatly on the specific basis. Consider the following method, due to Babai [Bab86]. Given a basis \( B \) and a target vector \( t \). Since \( t \) is in the linear span of \( B \), we can express it as a linear combination of the basis vectors, \( t = \sum_{i=1}^n t_i b_i \), where \( t_i = \frac{\langle t, b_i \rangle}{||b_i||^2} \). Take the approximation \( \tilde{t} = \sum_{i=1}^n \lfloor t_i \rfloor b_i \) obtained by rounding off the coefficients. This procedure takes linear time in the rank of the lattice. Given a nearly orthogonal basis, this technique can provide sufficiently accurate approximations for practical purposes, such as the GGH cryptosystem described in subsection 3.4.1. A more refined technique from the same paper, Babai’s Nearest Plane algorithm, begins by LLL reducing the lattice, and gives a \( \gamma = 2^{n/2} \) approximation to CVP in polynomial time.
3.3.3. Shortest Independent Vectors Problem

The Shortest Independent Vectors Problem (SIVP) is a straightforward generalization of SVP. Where the problem SVP, asks for a single vector whose length is within $\gamma$ of the shortest length $\lambda_1$, SIVP, asks for a maximal linearly independent set of vectors $s_1, ..., s_n$ (a total of $n$ vectors for a rank $n$ lattice) such that $|s_j| \leq \gamma \lambda_j$. Again, the exact problem SIVP corresponds to the case $\gamma = 1$. The corresponding promise problem (decision problem for the exact case) asks if there exist $n$ linearly independent vectors all of which have length less than $\gamma$ times a given length $r$. In other words, whether $\lambda_n \leq \gamma \cdot r$.

3.4. Lattice Cryptography

In recent years, lattice problems have been a popular basis for the construction of cryptographic primitives. Because of the hardness results given above, it is generally believed that constructions which lattice approximations can be reduced to will remain secure. In particular, since no known quantum algorithms exist that solve these problems in polynomial time, at least for constant approximation factors $\gamma$, lattice-based cryptographic schemes are a promising area of security research as the threat of quantum adversaries grows.

3.4.1. GGH Encryption

In 1997, Goldreich, Goldwasser, and Halevi proposed an asymmetric cryptosystem based off the hardness of solving the closest vector problem [GGH97b]. The general concept of the GGH system is to encode a message as a lattice point, then disguise the point by adding in a small noise vector. Recovery of the plaintext amounts to recovering the lattice vector that encodes it.

**Key Generation**  Alice randomly generates a set of $n$ linearly independent vectors $s_1, ..., s_n$ in $\mathbb{Z}^n$ and repeats until the set is reasonably orthogonal, according to some specific measure. Alice may have to generate a number of candidate sets before finding a sufficiently nice set, but the expected number of attempts is relatively low. A common method of generating these
vectors is to choose the coordinates uniformly randomly from the interval \([-l, \ldots, +l]\) for some small integer \(l\). This set forms a basis \(S\) of the lattice it generates, which will form Alice’s private key. Alice then generates a random \(n \times n\) unitary matrix \(U\), and computes an different basis \(P = SU\) for \(\mathcal{L}(B)\). The basis \(P\) is Alice’s public key. An improvement due to Micciancio [Mic01], applicable to a variety of similar systems, is to use as a public basis the \textit{Hermite Normal Form}, which can be efficiently computed from any other basis.

**Encryption**  
Bob wishes to send Alice a message \(m\), which is a vector in \(\mathbb{Z}^n\) with small coefficients. He takes the public basis \(P\), generates a small, random noise vector \(r\), and computes the ciphertext \(c = Pm + r\).

**Decryption**  
Given a ciphertext \(c\), Alice uses one of Babai’s algorithms to compute the closest vector in \(\mathcal{L}(S)\) to \(c\). Since the secret basis she has is sufficiently close to orthogonal, she is able to recover the exact lattice vector closest to \(c\), which is \(Pm\). She then multiplies this by \(P^{-1}\) to find the message vector \(m\).

The GGH system was cryptanalyzed in detail by Nguyen [Ngu99]. Using lattice reduction, Nguyen was able to attack GGH instances of dimension up to 350 by reducing them to a special type of CVP instances, which are easier to solve than the general case.

### 3.4.2. Ajtai–Dwork

Also in 1997, Ajtai and Dwork published a lattice-based cryptosystem [AD97], which was cryptanalyzed by Nguyen as well [NS98]. The Ajtai–Dwork system remains an important theoretical breakthrough in lattice-based cryptography, as it remains the only known cryptosystem based on the worst-case hardness of the underlying lattice problem. Unfortunately, it requires key sizes and running times on the order of \(n^4\), where \(n\) is the dimension of the underlying lattice, causing it to be too inefficient for practical use.
**Key Generation** The private key is a specific basis designed to easily solve instances of CVP, similar to the GGH system. Fix parameters $M, d$ as large polynomial functions of the rank, with $d \geq n^c M$. Alice picks $n - 1$ random, linearly independent vectors $r_1, r_2, \ldots, r_{n-1}$ and defines the hyperplane $H$ they span. She then picks a final basis vector $r_n$ so that $\text{dist}(r_n, H)$ is approximately $d$. Alice’s secret key is the orthogonalized vector $r_n^*$. In the original proposal of Ajtai and Dwork, the public key is a random basis for the lattice $L(R)$, but the suggestion of Micciancio [Mic01] to use the Hermite normal form is applicable to this system as well.

**Encryption** Bob wishes to send a single bit message $b$ to Alice. If the bit is 0, he picks a random lattice vector and adds a small perturbation to it; if $b = 1$, he instead picks a random point in space, which will be far away from the lattice $L(R)$ with high probability. Specifically, the perturbation added in the case of $b = 0$ is a sum of $O(n)$ uniformly chosen vectors of length less than $n^3 M$.

**Decryption** Note that every lattice point belongs to some hyperplane $H_k = k r_n^* + H$, and the perturbation vector in the encryption process is much smaller than the distance between these hyperplanes. Given a target vector $t$, Alice can compute the distance to the nearest hyperplane using her knowledge of $r_n^*$. Since the lattice is completely contained in the collection of hyperplanes, Alice can use the distance between $t$ and the nearest hyperplane to determine if $t$ is close to the lattice or not.

If the distance from $t$ to the lattice is within a fixed bound, Alice interprets it as encrypting 0; if not, she interprets it as encrypting 1. Note that it is possible for a random vector, an encryption of 1, to wind up close to a hyperplane by random chance. In such a case, it will mistakenly be interpreted as an encryption of 0 instead. This possibility of decryption error is explicitly handled by a method of Goldreich et al [GGH97a].

One very important property of the Ajtai–Dwork system is that any adversary capable of breaking the scheme is also capable of recovering the secret key. The adversary performs a random walk starting from the origin, and uses a decryption oracle to ensure the walk remains
close to the hyperplane $H$. Each step in the walk can be given to the decryption oracle, and if is too far away, the step is rejected. This allows the adversary to find lattice vectors close to $H$, which are then used to recover $r_i$.

3.4.3. NTRU

The NTRU encryption scheme was first proposed by Hoffstein, Pipher, and Silverman in 1998 [HPS98], with a signature scheme based on the same mathematical principles coming a few years later [HPS01]. The scheme is naturally described in terms of convolution polynomial rings, and indeed the original paper included no mention of lattices.

Fix a rank $n$ and two moduli $p$ and $q$. We define three rings used in NTRU. The first is the convolution polynomial ring $R = \mathbb{Z}[x]/(x^n - 1)$, while the other two are obtained from $R$ by reducing the coefficients modulo $p$ or $q$: $R_p = \mathbb{Z}_p[x]/(x^n - 1)$ and $R_q = \mathbb{Z}_q[x]/(n^n - 1)$. We will make plentiful use of the isomorphism between $R$ and $\mathbb{Z}^n$ as abelian groups, identifying the polynomial $F(x) = F_0 + F_1x + \ldots + F_{n-1}x^{n-1}$ with its coefficient vector $F = [F_0, F_1, \ldots, F_{n-1}]$.

Multiplication of an element by $x$ corresponds to a cyclic rotation of its coefficient vector.

Note that we take elements of $\mathbb{Z}_p$ to be between $-p/2$ and $p/2$. We also introduce the notation $T(d_1, d_2)$ to refer to the class of ternary polynomials in $R$. A polynomial in $R$ is contained in $T(d_1, d_2)$ if $d_1$ of its coefficients are equal to 1, $d_2$ of its coefficients are equal to $-1$, and the remaining coefficients are equal to 0.

**Key Generation** Alice fixes public parameters $n, p,$ and $q$ and defines the rings $R, R_p$, and $R_q$ as above. Alice requires that $q$ is coprime to both $n$ and $p$. She also fixes a public parameter $d$. Alice generates random polynomials $f(x) \in T(d + 1, d)$ and $g(x) \in T(d, d)$ to serve as her private key. She then finds multiplicative inverses $f_p = f^{-1} \pmod{p}$ and $f_q = f^{-1} \pmod{q}$, and computes $h = f_q \cdot g \pmod{q}$. Alice’s final private key is the pair $(f, f_p)$, while her public key is the polynomial $h$. 
**Encryption**  Bob wishes to encrypt a message $m(x)$ in the form of a polynomial in $R_p$. He chooses a random polynomial $r \in \mathcal{T}(d,d)$ and computes $c(x) = ph(x) \cdot r(x) + m(x) \mod q$.

**Decryption**  Given a ciphertext $c(x)$, Alice first finds $a(x) = f(x) \cdot c(x) \mod q$. This expands to $a(x) = pf_m(x)g(x)r(x)f(x)+m(x)f(x) \mod q$, and so simplifies to $a(x) = pg(x)r(x)+m(x)f(x) \mod q$. Because of how $g$ and $r$ were chosen, $pg(x)r(x)$ should have small coefficients and so we can lift $a(x)$ to $R$ without changing the coefficients. Alice then finds $a(x)f_p(x) \mod p$, which is equal to $m(x)$.

The connection between NTRU and lattice problems is not immediately apparent, but can be expressed in the following manner. First, consider an NTRU public key $h(x) = h_0 + h_1x + \ldots + h_{n-1}x^{n-1}$. Define a matrix $H$ in terms of these coefficients:

$$H = \begin{bmatrix}
    h_0 & h_{n-1} & \ldots & h_1 \\
    h_1 & h_0 & \ldots & h_2 \\
    \vdots & \vdots & \ddots & \vdots \\
    h_{n-1} & h_{n-2} & \ldots & h_0 
\end{bmatrix}.$$

The matrix $H$ is sometimes called a cyclic matrix as the columns are consecutive cyclic permutations of each other, corresponding to the coefficient vectors of $h(x), x \cdot h(x), \ldots, x^{n-1} \cdot h(x)$. Then we define the $2n \times 2n$ NTRU matrix $M_h$ as the block matrix $\begin{bmatrix} I & 0 \\ H & qI \end{bmatrix}$, and the corresponding NTRU lattice, with rank $2n$, is the lattice $L_h = L(M_h)$.

The key connection to a hard lattice problem is the vector $(f, g)^T = [f_0, f_1, \ldots, f_{n-1}, g_0, g_1, \ldots, g_{n-1}]^T$, where $f(x)$ and $g(x)$ are the secret polynomials chosen during key generation, and $f$ and $g$ are the corresponding coefficient vectors. This vector can be shown without too much difficulty to be in the lattice $L_h$, and is in fact a relatively short vector in that lattice. Thus, the problem of key recovery for the NTRU encryption scheme relies on the hardness of finding short vectors in a lattice.
A similar observation gives that for a ciphertext \( c(x) \) corresponding to a message \( m(x) \), the known ciphertext point \((0, c)\) almost certainly has the lattice vector \((pr, c - m)\) as its closest vector. Thus, recovery of plaintext reduces to the problem of finding lattice vectors close to a target vector.
Chapter 4

Learning with Errors

4.1. The Learning with Errors Problem

The Learning with Errors (LWE) problem was first proposed by Regev in 2005 [Reg05]. The problem deals with slightly perturbed linear equations of the form $\langle \mathbf{a}, \mathbf{s} \rangle + e = b$ over $\mathbb{Z}_p$ for some modulus $p$, where the error $e$ is unknown but taken from a known, “small” distribution. Given a number $m$ of these equations, we can express the LWE system in matrix form as

$$\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}. \quad (4.1.1)$$

We begin to formalize this as follows. Fix a dimension $n$ and modulus $p$. Fix some probability distribution $\chi$ on $\mathbb{Z}_p$ and a vector $\mathbf{s} \in \mathbb{Z}_p^n$. We now define a probability distribution on $\mathbb{Z}_p^n \times \mathbb{Z}_p$. Consider the process of uniformly sampling $\mathbf{a} \leftarrow \mathbb{Z}_p^n$, sampling $e \leftarrow \chi$, and outputting the pair $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e)$. The distribution on $\mathbb{Z}_p^n \times \mathbb{Z}_p$ that results we will denote as $\mathcal{A}_{\mathbf{s}, \chi}$. We contrast this with the uniform distribution $\mathcal{U}$ on the same set. From here we can define the Learning with Errors computational problem.

**Definition 4.1.1 (LWE).** Fix dimension $n$, modulus $p$ and error distribution $\chi$. An instance of the problem $\text{LWE}_{p,\chi}$ is a list of samples from the distribution $\mathcal{A}_{\mathbf{s}, \chi}$ defined above. All samples
are taken from $\mathcal{A}_{s,\lambda}$ for the same vector $s$, which is not provided. An algorithm is said to solve \( \text{LWE}_{p,\chi} \) if, given any such instance, it outputs the vector $s$ with probability exponentially close to 1.

4.1.1. Discrete Gaussian Distributions

We introduce two discrete analogues of the Gaussian distribution useful for working with the learning with errors problem. First is the discretized Gaussian distribution $\Psi_\alpha$ defined on $\mathbb{Z}$ or $\mathbb{Z}_p$. Sample a real number $x$ according to a (continuous) Gaussian distribution, round to the nearest integer $\lfloor x \rfloor$, and, if defining the distribution over $\mathbb{Z}_p$ instead of $\mathbb{Z}$, reduce the result modulo $p$. The distribution of $\lfloor x \rfloor$ or $\lfloor x \rfloor \mod p$ found in this way is the discretized version of the Gaussian distribution. Typically, the distribution $\chi$ the small noise $e$ in an LWE sample $\langle a, s \rangle + e$ is sampled from is a discretized Gaussian.

The second Gaussian-like discrete distribution is more properly referred to as a discrete Gaussian distribution, as it preserves more of the significant properties of the continuous Gaussian; such a distribution can be defined on any lattice. Consider the probability density function for a Gaussian distribution with center $c$ and variance $\sigma^2$ given by

\[
\rho_{\sigma, c}(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-c)^2}{2\sigma^2}}.
\]  
(4.1.2)

We extend this naturally to the multidimensional case by replacing $(x-c)^2$ with $||x-c||^2$ in the preceding equation.

**Definition 4.1.2** (Discrete Gaussian). Let $\Lambda$ be any lattice, and let $c$ be any point in the surrounding space $\mathbb{R}^n$. The discrete Gaussian distribution on $\Lambda$ with parameter $\sigma$ centered at $c$ is the probability distribution

\[
D_{\Lambda,\sigma,c}(x) = \frac{\rho_{\sigma,c}(x)}{\sum_{y \in \Lambda} \rho_{\sigma,c}(y)}.
\]  
(4.1.3)

Typically, we will choose $c = 0$, and omit the subscript for the center in this case. We will also omit the subscript indicating the particular lattice $\Lambda$ when it is clear from context.
This distribution appears in the original hardness proofs of LWE, and is used as well in some variants, such as the Ring LWE variant described in subsection 4.2.3.

4.1.2. Relation to Lattice Problems

The primary hardness result in Regev’s original presentation of LWE [Reg05] is a quantum reduction from GapSVP to LWE, meaning that if any efficient algorithm exists that can solve the LWE problem, then an efficient algorithm exists on a quantum computer that can solve the GapSVP problem. This is proved using an iterative approach between two lattice problems, obtaining tighter approximation bounds with each step. More recent work has provided classical reductions to LWE, though with specific caveats. Peikert provides a classical reduction [Pei09], which was in turn simplified by Lyubashevsky and Micciancio [LM09], showing that LWE with a modulus exponentially large in the dimension is classically as hard as standard lattice problems. More recently, Brakerski et al. give a classical reduction for polynomial modulus [Bra+13].

The first lattice problem used in the original reduction is the Discrete Gaussian Sampling (DGS) problem, which asks, given a lattice $\Lambda$ and a Gaussian parameter $\alpha$, for a vector sampled according to the discrete Gaussian distribution $D_{\Lambda,\alpha}$ described above. For sufficiently large $\alpha$, this is easy. Regev’s first step is an algorithm that, given polynomial-many such samples and an oracle to the LWE problem, solve the approximate closest vector problem in the dual lattice $\Lambda^\ast$.

The second part of the reduction is a quantum algorithm that uses this solution to the approximate CVP to generate samples for an instance of the DGS problem with a smaller parameter. This in turn is used to solve approximate CVP on the dual lattice again, with a smaller approximation factor. Repeating this process gives a (quantum) reduction from polynomial-size approximations of SVP and CVP to LWE.
4.2. Variations

With the popularity of the LWE problem for creation of cryptographic primitives, a number of variations and minor tweaks have been proposed to the original LWE definition. Often these are minor adjustments used in the definition of a cryptographic scheme, which are then related to the hardness of the original LWE problem. Typically, the use of these modified versions simplifies the security reduction of the proposed scheme; occasionally the modification is done for reasons of efficiency, most notably the ring variation described in subsection 4.2.3.

4.2.1. Decision Version

As with many lattice problems, the learning with errors problem has a corresponding decision version which is, under certain parameters, as hard as the search version. Fix a dimension $n$, modulus $q$, noise distribution $\chi$, and secret vector $s$ as above. The search version of LWE, as described above, asks for the secret vector $s$ given a number of samples from the distribution $A_{s,\chi}$ on $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

We phrase the decision version in terms of a game. Fix parameters $n, q, s$, and $\chi$ as above, and randomly pick $b \leftarrow \{0, 1\}$. If $b = 0$, sample some number of samples on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ according to the distribution $A_{s,\chi}$; if $b = 1$, sample from the uniform distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ instead.

The challenge of the decision version is to output the value of $b$, given the samples. That is to say, the decision version of LWE asks an algorithm to distinguish the distribution $A_{s,\chi}$ from the uniform distribution.

Importantly, Regev showed that the decision version of LWE is as hard as the search version for prime modulus $[\text{Reg05}]$; Peikert extended this result to moduli that are the product of small, distinct primes using the Chinese Remainder Theorem $[\text{LP11}]$.

**Theorem 4.2.1.** Fix dimension $n$ and let $q = \text{poly}(n)$ be a prime. Fix the secret $s$ and noise distribution $\chi$. Suppose we are given access to an oracle $\mathcal{O}$ for the decision version of the LWE problem. That is, $\mathcal{O}(\hat{A}, \hat{b})$ is equal to 0 if $(\hat{A}, \hat{b})$ is drawn from $A_{s,\chi}$ or 1 if uniform. Then there is
an algorithm that, given a sample \((A, b)\) drawn from \(A_{s,\chi}\), returns \(s\) using a polynomial number of calls to \(O\).

**Proof.** The procedure recovers \(s\) one coordinate at a time. Pick a guess \(k\) for the first coordinate \(s_1\). Randomly sample \(r \leftarrow \mathbb{Z}_q\). Add \(r \cdot e_1\) to every row of \(A\) to form \(\hat{A}\), and add \(k\) to every entry of \(b\) to form \(\hat{b}\).

Adding \(r\) to the first coefficient of each LWE sample \(a\) has the result of adding \(r \cdot s_1\) to the corresponding total. Thus, if \(s_1 = k\), then \(O(\hat{A}, \hat{b}) = 0\). Conversely, if \(s_1 \neq k\), then \(r \cdot k\) is uniform (since \(q\) is prime) and independent of \(s_1\) so adding \(r \cdot k\) to each entry of \(\hat{b}\) makes it uniform and independent of the corresponding change to \(A\). Thus, \(O(\hat{A}, \hat{b}) = 1\) in this case.

Given the ability to check the guess \(k\), we can try every possibility in \(\mathbb{Z}_q\) with only \(q = \text{poly}(n)\) many queries to \(O\), eventually finding the correct value of \(s_1\). We repeat this procedure for every coordinate of \(s\), adding \(r e_j\) to each row of \(A\) to check a guess for \(s_j\). This can be done using at most \(n \cdot q = \text{poly}(n)\) queries to \(O\). \(\square\)

### 4.2.2. Robustness Results

The original definition and hardness results for LWE [Reg05] use rather specific parameters. Both the sample vector \(a\) and the secret vector \(s\) are, by definition, drawn from a uniform distribution. The small noise \(e\) is allowed to be from a specified distribution \(\chi\), but the original hardness theorems apply to specific discretized Gaussian distributions.

Goldwasser et al. provide a number of hardness results for LWE with more general probability distributions [Gol+10]. The secret \(s\) can come from any distribution with sufficient min-entropy, without compromising the hardness of the LWE instance. In particular, the secret can be sampled according to a discrete Gaussian and remain hard. If information about the secret \(s\) is leaked, as long as the remaining entropy is high enough, the problem remains hard.
4.2.3. Ring Learning with Errors

One particularly important variant of the LWE problem is the Ring Learning with Errors (RLWE) problem. Let $n$ be a power of 2, and consider the cyclotomic integer ring $R = \mathbb{Z}[x]/(x^n + 1)$ and its reduction modulo $q$, $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. We can define a variant of the LWE problem on this ring. The advantage of this setting is that the additional multiplicative structure makes the system more efficient by requiring a single ring sample in the place of a linear number of ordinary LWE samples.

Suppose $a$ is uniformly chosen from $R_q$, and $s$ and $e$ are both chosen according to a discrete Gaussian distribution—more precisely, choose vectors $s$ and $e$ in $\mathbb{Z}^n$ according to a discrete Gaussian distribution, and set $s = s_0 + s_1 x + \ldots + s_{n-1} x^{n-1}$. Then it is a computationally hard problem to find $s$ given knowledge of $a$ and $b = a \cdot s + e$. Likewise, given $a$ but not $s$, it is computationally hard to determine whether a ring element $b$ has the form $b = a \cdot s + e$ or is uniformly sampled.

4.3. LWE Cryptography

A wide variety of cryptographic primitives have been proposed based on both the LWE and RLWE problems due to their perceived quantum resistance and efficiency. Regev’s original paper [Reg05] defining LWE includes a proposed public key cryptosystem in which the message to be sent is encoded as part of the noise of a LWE instance. Peikert also proposed a public key system [Pei09] which is provably CCA-secure under the assumption that LWE is hard.

Lattice systems have also been very popular in recent years for designing fully homomorphic encryption schemes. A fully homomorphic scheme is an encryption scheme designed so that a third party is able to perform arbitrary computations over ciphertexts without also having the ability to decrypt the ciphertexts; such an ability allows one to outsource involved numerical and statistical analysis of sensitive data such as financial or medical records to a computationally powerful third party in a secure way. The key breakthrough in fully homomorphic
encryption came when Gentry published his technique called *bootstrapping*, with a concrete system based on ideal lattices [Gen+09].

Follow-up work in this area includes *modulus switching*, an alternative technique to bootstrapping for RLWE-based homomorphic schemes [BGV12]; a homomorphic scheme based on RLWE which has an important property in that area called *circular security* [BV11]; and a general LWE based technique called the *approximate eigenvector* method [GSW13] used to create fully homomorphic schemes that can operate on ciphertexts without needing the associated public keys, leading to the construction of fully homomorphic *identity-based* and *attribute-based* encryption schemes.

Besides encryption schemes, work has been done to base various other cryptographic primitives on the hardness of the LWE problem. One particularly active area in recent years has been using the LWE problem as a basis for key exchange protocols. In chapter 5 we describe a number of key exchange protocols reminiscent of Diffie–Hellman in more detail. Also, a number of signature schemes [GPV08; Cas+12; Lyu12; MP12; Boy10] have been based on the *short integer solution* (SIS) problem, which can be considered a dual problem to the LWE problem itself.
Chapter 5

LWE Diffie–Hellman

Classical Diffie–Hellman key exchange works because modular exponential maps, maps of the form \( f_a(x) = x^a \mod q \) for some fixed modulus \( q \), commute with each other. Thus, the core idea of Diffie–Hellman-like LWE schemes is to use a similar pair of commuting maps to obtain a shared secret. If we consider the original LWE problem in its matrix form, a natural candidate for these operations is construction of a bilinear form by creating LWE samples with a given secret.

For any two secret vectors \( \mathbf{s}_1, \mathbf{s}_2 \) and a matrix \( A \), we have that \( \mathbf{s}_1^\top (A \mathbf{s}_2) = (\mathbf{s}_1^\top A)\mathbf{s}_2 \). In the RLWE setting, the ring multiplication replaces the matrix multiplication. We focus on this setting in what follows in order to simplify the presentation. In this case, \( (a \cdot s_1) \cdot s_2 = (a \cdot s_2) \cdot s_1 \).

In order to use the hardness of the RLWE problem, however, we cannot use \( a \cdot s_1 \) or \( a \cdot s_2 \) directly; instead we use the RLWE samples \( a \cdot s_1 + e_1 \) and \( a \cdot s_2 + e_2 \). In doing so, we lose the exact commutivity of direct multiplication, gaining instead the almost equal multiplications

\[
(a s_1 + e_1) \cdot s_2 = a s_1 s_2 + e_1 s_2 \\
(a s_2 + e_2) \cdot s_1 = a s_1 s_2 + e_2 s_1
\]
which differ by the value $e_2s_2 - e_1s_1$. Fortunately, this difference can be accounted for, leading to the key exchange protocols described below.

5.1. Basic LWE Diffie–Hellman

Based off the prior observations, Ding et al. published a key exchange protocol [DL12] using the structure of classical Diffie–Hellman key exchange, which we sketch in Figure 5.1. Before exchanging keys, Alice and Bob agree to use specific parameters for the dimension $n$, modulus $q$, base point $a \in R_q$, and Gaussian parameter $\alpha$. Each of the two creates a RLWE instance using the agreed-upon ring element $a$. Alice transmits $p_A = as_A + 2e_A$, where $s_A$ and $e_A$ are sampled according to the distribution $\chi_a = D_{R_{q^n}}$ and known only to her; likewise, Bob samples $s_B, e_B \leftarrow \chi_a$ and transmits $p_B = as_B + 2e_B$. Alice combines her secret with Bob’s public transmission to obtain $k_A = p_B \cdot s_A + 2g_A$, where $g_A$ is a small noise element sampled from the same distribution $\chi_a$ as $e_A$. In a similar manner, Bob samples $g_B \leftarrow \chi_a$ and obtains $k_B = p_A \cdot s_B + 2g_B$.

As previously mentioned, $k_A \neq k_B$, the two being separated by a small noise value.

The process of ensuring Alice and Bob obtain an exact shared secret from this approximate shared secret, is called reconciliation, and can be done in a number of different manners. We will use the technique of [DL12] as a basis, but also note that Peikert proposes a protocol using a very similar structure but a different reconciliation method [Pei14]. To perform reconciliation between Alice and Bob, first note that the small noise elements—the $e$’s and $g$’s—are all doubled before being added in. If we look at the difference between the values $k_A$ and $k_B$, we have

$$k_A - k_B = p_Bs_A + 2g_A - p_As_B - 2g_B$$

$$= as_As_B + 2e_Bs_A + 2g_A - 2eAs_B - 2g_B$$

$$= 2(e_Bs_A - e_As_B + g_A - g_B)$$

$$= 2\tilde{g}.$$
That is, \( k_A = k_B + 2\tilde{g} \) for some noise value \( \tilde{g} \). Thus, the core idea behind the reconciliation technique is to take \( k_A \mod 2 \) and \( k_B \mod 2 \), reducing each coefficient modulo 2. Unfortunately, the fact that we are already working modulo \( q \) introduces two complications.

The first complication occurs if \( q \) is odd, which is often the case, since \( q \) is typically chosen to be prime for efficiency reasons. No matter what representatives are chosen for \( \mathbb{Z}_q \), reducing the coefficients modulo \( q \) and then modulo 2 introduces a minor bias into the shared secret bits. A post-processing step, passing the secret through a final key derivation function, can work to mitigate this issue, but a reconciliation method that directly eliminates this bias is preferable.

Secondly, and more importantly, if this small noise \( 2\tilde{g} \) crosses the bounds of the representatives chosen for \( \mathbb{Z}_q \), then the reduction modulo \( q \) of \( k_A \) and \( k_B \) may disagree on the reduction modulo 2, even though the two can be expressed as being a small, even distance apart. For example, consider the case \( q = 31 \), and suppose \( k_A = 15 \) and \( k_B = 19 \). Since the difference between the two is \( 2\tilde{g} = 4 \), if we simply reduce modulo 2, both are equal to 1; if we do the reduction modulo \( q \) first, however, we have \( k_B = -12 \), so that \( k_A \mod q \mod 2 = 1 \) and \( k_B \mod q \mod 2 = 0 \). If this sort of rollover occurs in any of the \( n \) coefficients, then Alice and Bob will disagree on the shared key.

We need a way to avoid or otherwise handle this rollover. The original proposal in [DL12] has Bob send a binary signal \( w \), which is used to adjust \( k_A \) and \( k_B \) to avoid this roll-over. We will use a modified version here, called a robust reconciler, which will additionally handle the bias from an odd \( q \).

### 5.1.1. Robust Reconciler

To begin, we note some useful results regarding the size of the noise \( 2\tilde{g} \). Together with the triangle inequality, they allow us to choose our parameters \( n \) and \( q \) in a way that bounds \( \|2\tilde{g}\|_\infty \leq \|2\tilde{g}\| \leq q/4 \).
Choose elements $s_A, e_A \leftarrow \chi_{a'}$.
Set $p_A = as_A + 2e_A$.

Choose elements $s_B, e_B \leftarrow \chi_{a'}$.
Set $p_B = as_B + 2e_B$.
Choose $g_B \leftarrow \chi_{a'}$.
Set $k_B = p_A s_B + 2g_B$.
Find signal $w = \text{Cha}(k_B)$.

Choose $g_A \leftarrow \chi_{a'}$.
Set $k_A = p_B s_A + 2g_A$.
Compute $\sigma_A = \text{Mod}_2(k_A, w)$.

Choose $g_B \leftarrow \chi_{a'}$.
Set $k_B = p_A s_B + 2g_B$.
Compute $\sigma_B = \text{Mod}_2(k_B, w)$.

Figure 5.1.: Ring Learning with Errors Key Exchange
Lemma 5.1.1 ([GPVo8; MR07]). Fix a dimension \( n \) and Gaussian parameter \( \alpha = \omega(\sqrt{\log n}) \). Sample a vector \( x \) according to the discrete Gaussian distribution \( D_{Z^n,\alpha} \). Then the probability that \( \|x\| > \alpha \sqrt{n} \) is no greater than \( 2^{-n+1} \).

Lemma 5.1.2 ([Zha+15]). Define the ring \( R = \mathbb{Z}[x]/(x^n + 1) \) as above, where \( n \) is a power of 2. Then, for any elements \( s, t \in R \), we have the following bounds on \( s \cdot t \):

\[
\|s \cdot t\| \leq \sqrt{n} \cdot \|s\| \cdot \|t\|,
\]
\[
\|s \cdot t\|_\infty \leq n \cdot \|s\|_\infty \cdot \|t\|_\infty.
\]

For now, we consider a single coefficient of \( k_A, k_B \), and \( \tilde{g} \) at a time; the analysis can be done in the one-dimensional case and applied to each coefficient individually. Suppose \( -\frac{q}{4} \leq k_A < \frac{q}{4} \).

Then, since we have \( |\tilde{g}| < \frac{q}{4} \) and \( k_B = k_A + 2\tilde{g} \), we know that \( -\frac{q}{2} \leq k_B < \frac{q}{2} \). In this case, the reduction modulo \( q \) does not affect the further reduction modulo 2, as \( k_B = k_B \mod q \) already.

If we could guarantee that \( k_A \) were between \( -\frac{q}{4} \) and \( \frac{q}{4} \), then the reduction modulo \( q \) would be no problem.

We define two functions from \( \mathbb{Z}_q \) to \( \{0, 1\} \) that will assist in this goal:

\[
\text{Cha}_0(v) = \begin{cases} 
0 & [-\frac{q}{4}] \leq v \leq \lfloor \frac{q}{4} \rfloor, \\
1 & \text{otherwise}.
\end{cases}
\]

\[
\text{Cha}_1(v) = \begin{cases} 
0 & [-\frac{q}{4}] + 1 \leq v \leq \lfloor \frac{q}{4} \rfloor + 1, \\
1 & \text{otherwise}.
\end{cases}
\]

Now, no matter where in \( \mathbb{Z}_q \) the number \( v \) lies, we have \( v + \text{Cha}_0(v) \cdot \frac{q-1}{2} \) lies in the center of \( \mathbb{Z}_q \) as desired. We also define an augmented modulo function

\[
\text{Mod}_2(v, w) = \left(v + w \cdot \frac{q-1}{2} \mod q\right) \mod 2.
\]
The following lemma summarizes the fact that as long as two values $k$ and $k'$ are close enough, the functions $\text{Cha}_0$ and $\text{Mod}_2$ can be combined in order to obtain the same reduction modulo 2.

**Lemma 5.1.3** ([Zha+15]). Let $k \in \mathbb{Z}_q$, and let $g \in \mathbb{Z}_q$ such that $|g| < \frac{q}{8}$. Then, for $k' = k + 2g$, we have $\text{Mod}_2(k, \text{Cha}_0(k)) = \text{Mod}_2(k', \text{Cha}_0(k))$.

Thus, we have a way to reconcile the shared secrets of Alice and Bob. Bob computes $w = \text{Cha}_0(k_B)$ and sends it to Alice along with $p_B$. Then Alice obtains the shared secret as $\sigma_A = \text{Mod}_2(k_A, w)$ while Bob computes it as $\sigma_B = \text{Mod}_2(k_B, w)$. This procedure solves the larger of the two complications, that of ensuring the two parties have the same shared secret. A similar analysis applies if we replace the $\text{Cha}_0$ function with $\text{Cha}_1$.

The other complication remains, however. The distribution of $\text{Mod}_2(k, \text{Cha}_0(k))$ is close to uniform on $\{0, 1\}$, but a small bias still remains. Note that if $\text{Cha}_0$ is replaced with $\text{Cha}_1$, a bias still exists, but this bias is exactly opposite to the bias for $\text{Cha}_0$. Therefore, we combine these to obtain a random function, $\text{Cha}$. The output of $\text{Cha}(v)$ is equal to $\text{Cha}_b(v)$, where $b$ is uniformly sampled from $\{0, 1\}$. By randomly choosing for each run of the protocol, the overall bias is reduced to zero.

### 5.2. LWE HMQV

The preceding section describes a key exchange protocol based on the RLWE problem and the Diffie–Hellman key exchange’s structure. Like classical Diffie–Hellman, it is an unauthenticated protocol, and therefore vulnerable to Man-in-the-Middle attacks.

There are a number of ways to construct an authenticated key exchange protocol using an unauthenticated protocol as a basis. One simple technique is through the use of a signature scheme. In both the original DH scheme and the LWE DH scheme described in section 5.1, the protocol can be readily adapted to include an explicit signature on the transmitted values.
Peikert’s recent proposal [Pei14] for a key exchange based on LWE, he also gives an authenticated variant using the SIGMA (SIGn-and-MAc) transformation of Krawczyk [Kra03].

There are advantages to avoiding signatures, however. The use of signatures directly imposes additional requirements on a key exchange protocol. The security analysis must take the signature into account, and the cost of implementing the protocol includes the additional cost of the signature—both the computational cost and the communications cost. In particular regarding the security of the signature scheme, many currently used signature schemes, such as RSA or DSA/ECDSA, are based on number-theoretic problems that Shor’s algorithm can easily break; if LWE key exchange is being used for security against quantum attackers, every aspect of the system must also use quantum-secure primitives.

In a similar manner to the construction of HMQV (section 2.2) from the basic Diffie–Hellman problem, we present here an authenticated key exchange protocol based on the LWE Diffie–Hellman exchange above. A graphical overview is provided in Figure 5.2. This initial version is due to Zhang et al. [Zha+15], and follows the structure of HMQV rather closely; in section 5.3 we describe a modified version with improved efficiency and a simpler security analysis.

The basic design of the RLWE-HMQV protocol is much like classical HMQV. Alice and Bob publish long-term keys \( p_A = a s_A + 2e_A \) and \( p_B = a s_B + 2e_B \) for authentication. To perform key exchange, Alice samples \( r_A, f_A \) according to the distribution \( \chi_{\beta} \). For technical reasons, the ephemeral portion of the key exchange uses a discrete Gaussian with parameter \( \beta \gg \alpha \), the Gaussian parameter used for the static keys. This is a technique called noise flooding and serves to make the final distribution of the shared secret material largely independent of the static keys. She then transmits the RLWE sample \( x_A = ar_A + 2f_A \). Bob receives this, generates his own sample \( x_B = ar_B + 2f_B \), and computes the approximate shared secret \( k_B \).

Like when moving from classical Diffie–Hellman to HMQV, the manner of computing \( k_B \) is different from the unauthenticated version. First, Bob obtains the ring values \( c \) and \( d \) using a designated hash function \( \bar{H} \) with output distribution \( \chi_{\beta} \), which we model as a random oracle. Then he samples \( g_B \leftarrow \chi_{\beta} \), calculates \( k_B = (p_A \cdot c + x_A)(s_B \cdot d + r_B) + 2g_B \) and obtains the
Fix $R_q$ and $a$

Alice

Choose elements
$s_A, e_A \leftarrow \chi_a$.

Publish $p_A = as_A + 2e_A$.

Choose elements
$r_A, f_A \leftarrow \chi_\beta$.

Set $x_A = ar_A + 2f_A$.

Bob

Choose elements
$s_B, e_B \leftarrow \chi_a$.

Publish $p_B = as_B + 2e_B$.

Choose elements
$r_B, f_B \leftarrow \chi_\beta$.

Set $x_B = ar_B + 2f_B$.

Choose $g_A \leftarrow \chi_\beta$.

Compute $c = \tilde{H}(\hat{A}, \hat{B}, x_A)$.

Compute $d = \tilde{H}(\hat{B}, \hat{A}, x_B, x_A)$.

Set $k_A = (s_A \cdot c + r_A) \cdot (p_B \cdot d + x_B) + 2g_A$.

Compute $\sigma_A = \text{Mod}_2(k_A, w)$.

Compute $\sigma_B = \text{Mod}_2(k_B, w)$.

Figure 5.2.: Authenticated RLWE Key Exchange
signal $w = \text{Cha}(k_B)$. He transmits $x_B$ and $w$ to Alice, who also finds the values $c$, $d$, and $k_A = (s_A \cdot c + r_A)(p_B \cdot d + x_B) + 2g_A$, where $g_A$ is sampled according to $x_B$. The two parties then respectively apply the reconciliation technique to obtain respective shared secrets $\sigma_A = \text{Mod}_2(k_A, w)$ and $\sigma_B = \text{Mod}_2(k_B, w)$. As in the unauthenticated version, as long as $\beta$ is chosen to be small enough with respect to $q$, these two values will be equal to each other, giving a shared secret for Alice and Bob.

This protocol gives an implicitly authenticated key to the two parties: the two parties are authenticated to each other by obtaining the same shared key at the end of the procedure without performing any explicit checks along the way. Thus, it is vitally important that a party claiming to be Alice cannot derive the shared secret from Bob’s transmitted values without knowing Alice’s secret key. The derivation of the values $c$ and $d$ using a hash is important for this: if they are omitted from the protocol completely (without further modifications), then an attacker can trivially impersonate any party.

Suppose that we omit $c$ and $d$ completely from the protocol, so that $k_A = (s_A + r_A)(p_B + x_B) + 2g_A$ and $k_B = (p_A + x_A)(s_B + r_B) + 2g_B$ and consider an attacker attempting to impersonate Alice. To obtain $k_A$, the attacker needs to know the sum of $s_A$ and $r_A$, but need not know the individual terms themselves. The attacker can generate a target RLWE sample $\hat{x}_A = a\hat{r}_A + 2\hat{f}_A$ and reverse-engineer the value $x_A = \hat{x}_A - p_A$ to transmit. When Bob goes to compute $k_B$, he winds up with $\hat{x}_A(s_B + r_B) + 2g_B$, which corresponds to $k_A = \hat{r}_A(p_B + x_B) + 2g_A$; since the adversary knows $\hat{r}_A$, she obtains the same shared secret as Bob after the reconciliation step, and so passes the implicit authentication.

Even if we include $c$ and $d$ as constants, the attacker can simply compute $x_A = \hat{x}_A - c \cdot p_A$. In order to thwart this attack, we require the attacker to commit to a value of $x_A$ before knowing what $c$ will be, by making $c$ depend on $x_A$. In particular, $c$ is derived from $x_A$ using a hash function to prevent the adversary from finding a value of $c$ as a solution to some simple equation involving $\hat{x}_A$ and $p_A$. 

51
5.3. **Improved Ring LWE HMQV**

Despite the argument above about how the presence of \( c \) and \( d \) in the protocol defends against impersonation attacks, it would be desirable to avoid them if possible. Their use contributes to the need for noise flooding, requiring larger Gaussian parameters which in turn require a larger modulus \( q \). Furthermore, the fact that \( c \) and \( d \) are multiplied by other ring elements during the protocol increases the size of the noise, again resulting in the modulus needing to be increased for the protocol to work correctly. Thirdly, since \( c \) and \( d \) are derived using a hash function, the security analysis in [Zha+15] is done in the *random oracle* (RO) model.

The use of the RO model in the analysis in [Zha+15] is less than desirable. While random oracles are relatively common in classical security proofs, their use in quantum security proofs is somewhat more suspect. It is not apparent that a quantum adversary with access to a random oracle should be constrained to interact with the oracle in the same way as a non-quantum adversary.

Boneh et al. propose the *quantum random oracle* model [Bon+11] as a replacement that better formalizes the abilities of a quantum adversary. Where a classical adversary (or honest party) is able to submit a single value \( x \) to an oracle \( \mathcal{O} \) at a time and receive back a value \( \mathcal{O}(x) \), a quantum adversary—but not an honest party—in the quantum RO model can query \( \mathcal{O} \) in superposition: the adversary can submit a quantum state \( \sum_{x} \alpha_{x} |x\rangle \) and obtain back the resultant quantum state \( \sum_{x} \alpha_{x} |\mathcal{O}(x)\rangle \). This new model has gained some traction, including a provably secure encryption scheme from Zhandry [Zha15], but existing security analysis in the classical RO model typically does not translate directly into the quantum RO model. We avoid the issue completely by reworking the protocol of section 5.2 to avoid the use of random oracles altogether. In doing so, our security proof against a quantum adversary can be directly based on the quantum hardness of the RLWE problem.

Even without this particular benefit, it can be useful to avoid random oracles even in the classical setting, as a random oracle is an incredibly powerful theoretical tool. No actual computable hash function can provide the full security guarantees of the random oracle. In fact, there exist
cryptographic primitives which are provably secure in the RO model, but are insecure when the random oracle used is replaced with any particular function implementation [CGH04]. Thus, while the RO model is a powerful tool and a security proof using it is certainly suggestive of the security for a concrete implementation, a security proof that makes no recourse to a random oracle is an important improvement even in the pre-quantum world.

The streamlined variation of the protocol, which is illustrated in Figure 5.3, starts off much the same way as described in section 5.2. Alice and Bob publish public keys $p_A$ and $p_B$ with the same form as before. The first change is when the two go to generate their ephemeral keys. In this version of the protocol, we eschew the noise flooding technique, and use a single noise distribution $\chi_a$ throughout. With that small difference in place, Alice generates and transmits her ephemeral $x_A = ar_A + 2f_A$ as in the prior version; Bob generates his $x_B$ as well.

The second and biggest difference in this version is that Bob no longer computes the elements $c$ and $d$. Since the previous definition of $k_B$ relied on these values, we must change how $k_B$ is computed. We set $k_B = (p_A + x_A)r_B + x_As_B + 2g_B$ where $g_B$ is sampled from $\chi_a$. Equivalently, we have $k_B = p_Ar_B + x_Ar_B + x_As_B + 2g_B$ and $k_B = p_Ar_B + x_A(r_B + s_B) + 2g_B$. We will examine the fully expanded version at points in the following discussion, but the somewhat factored versions require fewer multiplications, and should be used for implementations. From this newly redefined $k_B$, Bob finds the signal $w$, and sends both $x_B$ and $w$ to Alice. Alice sets $k_A = (p_B + x_B)r_A + x_As_A + 2g_A$; this may be expressed in equivalent forms just as $k_B$ can. Finally, Alice and Bob each obtain the shared secret $\sigma$ using their respective $k$ and the signal $w$ with the augmented modulus function $\text{Mod}_{2^t}$, just as in the prior version.

Consider that we have a total of four key pairs: Alice and Bob each has both a static key pair and an ephemeral key pair. We can pair one of Alice’s public keys with one of Bob’s private keys in a total of four ways: $p_As_B$, $p_Ar_B$, $x_As_B$, and $x_Ar_B$, each of which has a corresponding pairing of one of Alice’s private keys with one of Bob’s public keys. Within certain limits, we can take the approximate shared secret $k_B$ to be any linear combination of these values, plus...
Fix \( R_q \) and \( a \)

Choose elements
\[ s_A, e_A \leftarrow \chi_{\alpha}. \]
Publish \( p_A = as_A + 2e_A \).

Choose elements
\[ r_A, f_A \leftarrow \chi_{\alpha}. \]
Set \( x_A = ar_A + 2f_A \).

Send \( x_A \)

Choose elements
\[ r_B, f_B \leftarrow \chi_{\alpha}. \]
Set \( x_B = ar_B + 2f_B \).
Choose \( g_B \leftarrow \chi_{\alpha}. \)
Set \( k_B = (p_A + x_A)r_B \)
\[ + x_A s_B + 2g_B. \]
Find signal \( w = \text{Cha}(k_B) \).

Send \( x_B, w \)

Choose \( g_A \leftarrow \chi_{\alpha}. \)
Set \( k_A = (p_B + x_B)r_A \)
\[ + x_A s_B + 2g_B. \]
Compute \( \sigma_A = \text{Mod}_2(k_A, w) \).

Compute \( \sigma_B = \text{Mod}_2(k_B, w) \).

Figure 5.3.: Authenticated RLWE Key Exchange II
the small noise $2g_B$, which we will temporarily ignore for clarity. The most general form is then

$$
k_B = K p_A s_B + L p_A r_B + M x_A s_B + N x_A r_B \tag{5.3.1}
$$

$$
k_A = K p_B s_A + L p_B r_A + M p_B r_A + N x_B r_A
$$

for some coefficients $K$, $L$, $M$, and $N$. The relationships between these values then determine various properties of the resulting key exchange protocol. For example, if $K = L = M = 0$ and $N = 1$, the entire scheme reduces to the unauthenticated protocol described in section 5.1. The authenticated version of section 5.2 can be thought of as the case where $K = c \cdot d$, $L = c$, $M = d$, and $N = 1$, with $c$ and $d$ obtained via hash functions as previously described. The protocol diagram in Figure 5.3 uses $K = 0$ and $L = M = N = 1$. In what follows, we will assume that all four are constant, integer values. Equations (5.3.1) give well-defined $k_A$ and $k_B$ for values in $R_q$, but doing so increases the number of ring multiplications, increasing the noise size and reducing performance somewhat.

We can express this somewhat more compactly as a quadratic form

$$
k_B = \begin{pmatrix} p_A & x_A \end{pmatrix} \begin{pmatrix} K & L \\ M & N \end{pmatrix} \begin{pmatrix} s_B \\ r_B \end{pmatrix}, \tag{5.3.2}
$$

Consider the rank of $\forall$. We will ignore the case rank $\forall = 0$, as $k_B$ is trivially 0. Then we have the following result relating this rank to the form of $k_B$.

**Theorem 5.3.1.** Let $k_B$ be non-zero and have the form defined in (5.3.2). Then $k_B$ can be factored into the form $(\mu p_A + \nu x_A) \cdot (\kappa s_B + \lambda r_B)$ if and only if rank $\forall = 1$.

**Proof.** Note that rank $\forall \neq 0$ since we exclude the case $k_B = 0$. Suppose that $k_B$ can be factored as $k_B = (\mu p_A + \nu x_A) \cdot (\kappa s_B + \lambda r_B)$. Expanding this we have $k_B = (\kappa \mu) p_A s_B + (\lambda \mu) p_A r_B + (\nu \lambda) x_A s_B + (\nu \lambda) x_A r_B$. Then the matrix $\forall = \begin{pmatrix} \kappa \mu & \lambda \mu \\ \nu \lambda & \lambda \nu \end{pmatrix}$, which is easily seen to be singular. Since $\forall$ cannot have full rank, and cannot have rank 0 as $k_B \neq 0$, we must have rank $\forall = 1$. 

55
Conversely, suppose that $\mathbb{A} = 1$. Then the rows linearly dependent and thus both multiples of some non-zero row vector $(\kappa, \lambda)$. Say the first row is $\mu$ times this vector, and the second row is $\nu$ times. That is, $\mathbb{A} = \begin{pmatrix} \kappa \mu & \lambda \mu \\ \kappa \nu & \lambda \nu \end{pmatrix}$. Expanding the quadratic form for $k_B$ gives

$$k_B = (\kappa \mu)p_A s_B + (\lambda \mu)p_A r_B + (\kappa \nu)x_A s_B + (\lambda \nu)x_A r_B$$

$$= \mu p_A (\kappa s_B + \lambda r_B) + \nu x_A (\kappa s_B + \lambda r_B)$$

$$= (\mu p_A + \nu x_A)(\kappa s_B + \lambda r_B).$$

Now, recall the attack on authentication described in section 5.1 if $c$ and $d$ are omitted from the key derivation. That attack works because the key material depends only on the sum $\hat{r}_A = s_A + r_A$, without requiring knowledge of the individual terms. The mitigation technique in that version of the protocol is to require the attacker to commit to $x_A$ before knowing how to back-form it from $\hat{x}_A$.

Alternatively, we note that the attack also fails if the attacker requires both $s_A$ and $r_A$ individually rather than simply the sum. We do this by avoiding factorable $k_A$ and $k_B$. The attacker can generate $\hat{r}_A$ and form $\hat{x}_A = \hat{a} \hat{r}_A + 2\hat{f}_A$ and $x_A = \hat{x}_A - p_A$ as before, but requires $r_A$ in addition to $\hat{r}_A$, which is equivalent to compromising Alice’s static key $s_A$. For this reason, we choose coefficients in $\mathbb{A}$ so that $k_B$ does not factor. Also, note that each coefficient in $\mathbb{A}$ winds up being multiplied by a noise vector. In order to keep the overall noise down, we will pick the coefficients to be small—in particular in the set $\{-1, 0, +1\}$.

To authenticate Alice to Bob, the key should depend on Alice’s static key in a way that binds each session of the key exchange to Alice in this way. Thus, we require that $L \neq 0$. To similarly authenticate Bob to Alice, we require that $M \neq 0$. Finally, we take $N \neq 0$ to have a portion of the key dependent wholly on the ephemeral keys to support forward secrecy even in the event both private static keys are later leaked. Somewhat less importantly, we would like the roles of Alice and Bob to be symmetric, so $L = M$, making $\mathbb{A}$ symmetric.
5.4. Variations

In addition to the full scheme described above, we give here a number on minor changes that can be implemented on top of our protocol.

5.4.1. Matrix Variant

The learning with errors problem was originally proposed as a computational problem over \( \mathbb{Z}_q \), and then extended to the case of cyclotomic rings \( \mathbb{Z}_q[x]/(x^n + 1) \) for efficiency reasons. The unauthenticated protocol in section 5.1 was originally proposed based on the LWE problem rather than the RLWE problem. We take the opposite approach here: we give the above protocols using the ring form as the default as they are simpler and more efficient, and present the following matrix form for completeness. An illustration of the protocol is given in Figure 5.4.

Recall as mentioned above that for vectors \( s_1, s_2 \in \mathbb{Z}_q^n \) and a matrix \( M \in \mathbb{Z}_q^{n \times n} \), we have \( s_1^\top (Ms_2) = (s_1^\top M) s_2 \); compare this to the identity \( (as_1) s_2 = (as_2) s_1 \) in the ring form of the LWE problem. We convert the scheme of section 5.3 to a matrix form by making the following changes and replacing ring multiplication with matrix/vector multiplication as appropriate:

- The public parameter \( a \in R_q \) is replaced with a matrix \( M \in \mathbb{Z}_q^{n \times n} \), increasing its size by a factor of \( n \). Alice derives her public keys using \( M^\top \) rather than \( M \).

- The secret ring elements \( s_A, s_B, r_A, \) and \( r_B \); noise elements \( e_A, e_B, f_A, f_B \); and public elements \( p_A, p_B, x_A, \) and \( x_B \) are replaced with vectors \( s_A, s_B, \ldots, x_A, x_B \). Since the ring elements could already be identified with their coefficient vectors, this has no direct effect on memory requirements or transmission sizes.

- The noise distribution \( \chi_a \) is replaced with an appropriate small distribution on \( \mathbb{Z}_q^n \) rather than \( R_q \). We set \( \chi_a^n \) to be a one-dimensional discrete Gaussian, and use \( \chi_a^n \) to sample the secret vectors mentioned above with coordinates chosen according to \( \chi_a^n \).
• The approximate secret $k_B$ is now a single element of $\mathbb{Z}_q$ rather than $R_q$ computed in a similar manner as before: $k_B = \langle p_A, r_B \rangle + \langle x_A, r_B + s_B \rangle + 2g_B$. The extra noise value $g_B$ is changed from an element of $R_q$ to an element of $\mathbb{Z}_q$ to match $k_B$. The same change is made to $g_A$, $k_A$, and $w$.

• The resulting shared secret $\sigma_A = \sigma_B$ is now a single bit, greatly reducing the efficiency of the scheme compared to the ring form. In order to obtain an equal amount of key material, the exchange must involve a total of $n$ vectors each time. (Hypothetically, each party could use $\sqrt{n}$ vectors instead of $n$, giving a total of $n$ pairs, but this leads to undesirable dependencies between the key bits while only reducing the problem of greater communication requirements instead of avoiding it completely, so we avoid it).

5.4.2. Noise Scaling

Consider in detail how our authenticated key exchange protocol obtains a shared key. The basic premise is the associativity and commutivity of multiplication in the ring $R_q$. Thus, we combine $s_A$ and $r_A$ with $as_B$ and $ar_B$ on one hand—$as_A$ and $ar_A$ with $s_B$ and $r_B$ on the other—to obtain $as_Ar_B, ar_As_B$, and $ar_Ar_B$ at both parties. We need to add small noise values to make this secure, but in a way that the noise doesn’t affect the two sides obtaining the same result. Thus, we make the noise a multiple of 2, so that the two sides obtain the same result modulo 2 (after accounting for the possibility of wrap-around from the noise).
**Fix \( n, q \) and \( M \)**

---

**Alice**

Choose elements
\[
s_A, e_A \leftarrow \lambda^n_a.
\]
Publish \( p_A = Ms_A + 2e_A \).

Choose elements
\[
r_A, f_A \leftarrow \lambda^n_a.
\]
Set \( x_A = Mr_A + 2f_A \).

Send \( x_A \)

Choose \( s_B, e_B \leftarrow \lambda^n_a \).

Publish \( p_B = Ms_B + 2e_B \).

**Bob**

Choose elements
\[
s_B, e_B \leftarrow \lambda^n_a.
\]

 Publish \( p_B = Ms_B + 2e_B \).

---

Compute \( \sigma_A = \text{Mod}_2(k_A, w) \).

Compute \( \sigma_B = \text{Mod}_2(k_B, w) \).

---

Figure 5.4.: Matrix Form
As a generalization, consider if we were to choose a scaling factor for the noise other than 2.

Fix a integer $\delta$, and re-define the following quantities in terms of $\delta$:

\begin{align*}
 p_A &= a_A + \delta e_A \\
 x_A &= a_A + \delta e_A \\
 k_A &= p_A r_A + x_A (s_A + r_A) + \delta g_A \\
 \sigma_A &= \text{Mod}_\delta(k_A, w) \\
 &= k_A + w \cdot \frac{q-1}{2} \mod q \mod \delta
\end{align*}

\begin{align*}
 p_B &= a_B + \delta e_B \\
 x_B &= a_B + \delta e_B \\
 k_B &= p_B r_B + x_B (s_B + r_B) + \delta g_B \\
 \sigma_B &= \text{Mod}_\delta(k_B, w) \\
 &= k_B + w \cdot \frac{q-1}{2} \mod q \mod \delta
\end{align*}

The basic behavior of this generalized scheme is somewhat like the standard case with $\delta = 2$. We find that $k_A = k_B + \delta \hat{g}$ for a combined noise value $\hat{g}$. If $\delta \hat{g}$ is small enough to avoid wrap-around—for example, if we force $k_B$ into $\{-[q/4], \ldots, \lfloor q/4 \rfloor\}$ using Cha and Mod$_\delta$ and bound $\|\delta \hat{g}\|_\infty \leq q/4$—then $k_A \mod \delta = k_B \mod \delta$. Thus, we obtain a shared secret $\sigma_A = \sigma_B$ consisting of $n$ values in $\mathbb{Z}_\delta$ rather than $n$ bits. In using a larger scaling factor for the noise, we obtain a higher bandwidth for our key exchange. We also raise the possibility of using some of this additional bandwidth in the key exchange process itself as an explicit authentication step.

If we pick $\delta = 4$, then we can split $\sigma_A$ into its most significant and least significant bits, giving two bit strings of length $n$. The low order bits can be used as the shared secret as before; we can have Alice send back the high order bits to Bob in a third pass to confirm she obtains the correct secret $\sigma_A$.

This increased bandwidth is not free, however. Most importantly, the size of the noise scales linearly in $\delta$. As we increase $\delta$, we require tighter bounds on $\|\hat{g}\|_\infty$ relative to $q$ in order for the shared secrets $\sigma_A$ and $\sigma_B$ to actually be equal. For large enough $\delta$, the scheme simply does not work. Furthermore, recall that we had to define Cha in a probabilistic way in order to avoid a bias in the final secret $\sigma_A$. If we increase $\delta$, then this definition must also be adapted to prevent bias, with possible consequences for efficiency and secrecy.
5.4.3. Single Pass

The original MQV protocol [MQV95] was published with a one-pass variant, where the initiator is the only party to generate an ephemeral key; the final secret key is based on both the initiator’s static and ephemeral keys, but only the static key of the responder, as there is no ephemeral key on the responder’s end. A corresponding construction is given for HMQV [Kra05] and the first LRWE-HMQV [Zha+15] as described in section 5.2. We apply the same construction to our simplified HMQV protocol here.

The protocol starts off with Alice generating her ephemeral key $x_A$ as in the two-pass version. However, instead of simply sending $x_A$ to Bob, Alice computes $k_A = p_B(s_A + r_A) + 2g_A$ on her side first, using her two secret keys and Bob’s static key; she then computes the signal $w$ on her end and sends over $x_A$ and $w$. When Bob receives this, he computes $k_B = (p_A + x_A)s_B + 2g_B$, and both parties obtain $\sigma_j = \text{Mod}_2(k_j, w)$.

We note that the security properties of this variation are significantly lower than the two-pass version. Since Bob sends no message and provides only a static key, if Bob’s static key is ever exposed, any session key he establishes as the receiver in the one-pass version is compromised. In a similar manner, a one-pass protocol is susceptible to replay attacks; if Alice sends a message $(x_A, w)$ in a one-pass run of the protocol and Eve somehow obtains the session key $\sigma_A$, then Eve can establish a session with Bob, claiming to be Alice, by sending the same message $(x_A, w)$ and using the leaked session key $\sigma_A$. Bob can avoid this, but doing so requires either recording all of his sessions to check for repeats, not a particularly efficient solution, or force every session to be different by generating an ephemeral key, which requires a second pass at least.
Fix $R_q$ and $a$

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose elements $s_A, e_A \leftarrow X_a$.</td>
<td></td>
</tr>
<tr>
<td>Publish $p_A = as_A + 2e_A$.</td>
<td></td>
</tr>
<tr>
<td>Choose elements $r_A, f_A \leftarrow X_a$.</td>
<td></td>
</tr>
<tr>
<td>Set $x_A = ar_A + 2f_A$.</td>
<td></td>
</tr>
<tr>
<td>Choose $g_A \leftarrow X_a$.</td>
<td></td>
</tr>
<tr>
<td>Set $k_A = p_B(s_A + r_A) + 2g_A$.</td>
<td></td>
</tr>
<tr>
<td>Find signal $w = \text{Cha}(k_A)$.</td>
<td></td>
</tr>
<tr>
<td>Send $x_A, w$</td>
<td></td>
</tr>
<tr>
<td>Set $k_B = (p_A + x_A)s_B + 2g_B$.</td>
<td></td>
</tr>
<tr>
<td>Compute $\sigma_A = \text{Mod}_2(k_A, w)$.</td>
<td></td>
</tr>
<tr>
<td>Compute $\sigma_B = \text{Mod}_2(k_B, w)$.</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.5.: One-Pass Protocol
Chapter 6

Security Analysis

In what follows, we follow typical usage and say that a function $f$ is negligible if $f(n)$ is asymptotically less than $n^{-c}$ for any constant exponent $c$. We denote an arbitrary negligible function in formulas by $\text{negl}(n)$. A sum of polynomially many negligible functions is still negligible, as is any polynomial function times a negligible function.

6.1. Proof Outline

We organize the security analysis of the key exchange scheme from section 5.3 as a sequence of games using the method of Shoup [Sho04]. We define two worlds, the real world and ideal world, in which we run our protocol. The real world version of the protocol behaves as described in section 5.3; this represents actual sessions between actual persons in real life. The ideal world is a hypothetical construction that modifies the working of the protocol. In the ideal world, after Alice and Bob perform key exchange, they obtain a shared secret key uniformly chosen independently of the actual transmitted values. An adversary in the ideal world is unable to learn anything about the shared key, because the key exchange literally contains no information about the key.
The objective of the security proof is to show that an adversary cannot differentiate between the real and ideal worlds. Since the ideal world is secure by its very definition, if an adversary in the real world could break the scheme, he could use this to determine which of the two worlds he is in. Our procedure is to create a number of games the adversary plays with a simulator. The initial game corresponds to the real world setting, the final game corresponds to the ideal world setting, and a number of games are inserted in between. We show that each game is computationally indistinguishable from the next under the assumption that the RLWE problem is hard. Since any two consecutive games are indistinguishable, we have that the initial and final games are also indistinguishable, and so the scheme is secure.

6.2. Security Model

In order to show that our AKE protocol is secure, we need to have a definition of “secure” in this context. The first definition of security for an AKE protocol is due to Bellare and Rogaway [BR94], and is defined in terms of an indistinguishability game the adversary plays. Let Π be an AKE protocol, and let $A$ be an adversary. We model the adversary $A$ as a probabilistic polynomial time (PPT) Turing machine that can interact with sessions of Π by querying a specified collection of oracles. We hold off on giving a detailed listing of these oracles for now; the precise capabilities of the adversary as modeled by the oracles it has access to are what differentiate the various security models based on the original Bellare–Rogaway model.

The adversary has access to one specific oracle, Test, which is used in the actual definition of security. The adversary queries Test with the session id of some session Σ of the protocol Π, subject to a few restrictions, in order to play the indistinguishability game. When $A$ does so, the oracle uniformly and randomly samples a bit $b \sim \{0, 1\}$. If $b = 0$, the oracle returns the session key associated with the session Σ; if $b = 1$ the oracle instead samples a random value $r$ according to the distribution of session keys in the protocol Π and returns that. The game ends when $A$ returns a guess $b'$ as to the value of $b$. We say that $A$ wins the game if $b' = b$ and
define the advantage an adversary has as

\[ \text{Adv}_\Pi \mathcal{A} := \Pr[b' = b] - \frac{1}{2}. \]  

(6.2.1)

An adversary that is always correct has an advantage of 1/2, while an adversary that does not interact with the protocol and instead flips a coin for \( b' \) has an advantage of 0.

Note that \( \mathcal{A} \) may interact with the protocol between receiving as response from the Test oracle and submitting its guess. In the original BR model, this was not the case, and \( \mathcal{A} \) could only submit a query to Test after making all its various other queries. The ability of the adversary to make such queries after submitting a Test query is vital for the model to capture the notion of perfect forward secrecy, described in subsection 6.2.1. The CK model of Canetti and Krawczyk [CK01] improves the BR model by including this, as well as other adversarial capabilities present in [BCK98]. We use a slightly modified version of the CK model due to how our protocol interacts with forward secrecy.

**Definition 6.2.1** ([CK01]). A key exchange protocol \( \Pi \) is secure if the following two properties hold:

1. If two honest parties complete matching sessions of \( \Pi \), they obtain the same shared key
2. For any adversary \( \mathcal{A} \), \( \text{Adv}_\Pi \mathcal{A} \) is negligible in the security parameter.

The first of these conditions is often called the correctness of the protocol \( \Pi \), and follows from Lemma 5.1.3 for all three protocols described in chapter 5. This condition says nothing about the keys the two parties obtain if an active adversary changes their messages, causing them to have non-matching sessions. The second condition formalizes the idea that no adversary should be able to gain any information about the session key.

**6.2.1. Forward Secrecy**

Broadly speaking, perfect forward secrecy (PFS) is the property that session keys should remain unrecoverable by an adversary, even in the event of future leakage of static keys as in
a Heartbleed-type attack [CVE13]. In contrast, consider a simple key encapsulation method of key exchange, where Alice simply generates and signs a session key to share, and sends it to Bob encrypted with Bob’s public key. Any adversary that can record the encrypted key and later compromise Bob’s secret key can decrypt and recover the session key. Formally, the protocol $\Pi$ has the PFS property if no adversary has non-negligible advantage at distinguishing a session key from random, even given the ability to later obtain the static keys of the parties to that session.

The notion of perfect forward secrecy is implicitly captured in the CK model by allowing the adversary to further interact with protocol sessions after obtaining the response from the Test query. Since the adversary is given this capability, we simply have to prove our scheme is secure according to the security model, and PFS comes along for free. Unfortunately, we cannot prove that our scheme is secure when the adversary fully has the ability to arbitrarily forge and modify messages between the two parties; no two-pass protocol that uses public keys with no shared state between sessions can. This includes the original MQV and HMQV protocols, as well as both RLWE versions presented here.

Krawczyk gives a generic attack against two-pass protocols [Kra05] which we describe using the notation of our protocol: suppose that $\mathcal{A}$ establishes a session with Bob, attempting to impersonate Alice without knowing her secret key $s_A$. Even without being able to authenticate as Alice to Bob, if $\mathcal{A}$ later obtains Alice’s private key, he can reconstruct the session key, violating the PFS property. Assuming the protocol in question handles authentication appropriately, $\mathcal{A}$ won’t have this key at the time of key exchange, and so Bob will abort the session prematurely; we cannot rely on this at the key exchange level, though, and it does not work at all for an unauthenticated protocol.

Instead of the full-strength PFS property, two-pass protocols can have a weaker variant on perfect forward secrecy, appropriately known as weak perfect forward secrecy (wPFS). A protocol $\Pi$ is said to have the wPFS property if an adversary cannot distinguish a session key from random, even in the event of future leakage of static keys, for sessions in which the adversary
**does not actively interfere.** We formalize this notion by requiring the session that $A$ issues a Test query for to have a property we call *freshness*, to be defined in Definition 6.2.2. Unfortunately, this restriction on $A$ means that our security proof does not guarantee full security against an active adversary. A refinement of our protocol that could be shown to be secure in the full CK model, even at the expense of requiring an extra pass, would be an interesting theoretical result.

### 6.2.2. Adversarial Capabilities

We model the interaction between an adversary $A$ and our protocol using a simulator $S$. First, $S$ begins by initializing a maximum number $N$ of parties, a maximum number $m$ of sessions each individual party may participate in, and static keys $s_i$ and $p_i$ for each party $i \in \{1, \ldots, N\}$. The simulator also randomly picks parties $i^*$ and $j^*$ from the list $\{1, \ldots, N\}$ as well as session numbers $\ell_{i^*}$ and $\ell_{j^*}$ from $\{1, \ldots, m\}$. The simulator is predicting that the session that $A$ issues a Test query on will have party $i^*$ as the initiator, party $j^*$ as the responder, and be the $\ell_{i^*}$-th and $\ell_{j^*}$-th sessions, respectively, that these parties are involved in.

**Simulator Queries** The adversary interacts with the simulator by making the following queries to the simulator and receiving the simulator’s response, representing the various abilities an attacker might employ in order to break the protocol in the real world. We give here a high level overview of what the simulator’s responses are in relation to the protocol; in subsection 6.3.1 we present a sequence of specific sets of rules for the simulator to follow, which we refer to as games, determining exactly what response the simulator gives to each query.

$Send_0(\Pi, I, i, j)$: The simulator activates a new session with initiator $i$ and responder $j$; $S$ returns a message $x_i$ intended for party $j$. 

67
Send\(_1(Π, R, j, i, x_i)\): The adversary relays a message purportedly from party \(i\) as initiator for party \(j\) as responder. The simulator proceeds by computing the session key \(σ_j\) and return a message \((x_j, w)\) to \(A\) intended to be party \(j\)’s response to the message \(x_i\).

Send\(_2(Π, I, i, j, x_i, x_j, w)\): The adversary relays a message purportedly for party \(i\) as initiator from party \(j\) as responder to complete a session initiated at party \(i\) with message \(x_i\) returned from a Send\(_0\) query. If no such query returned \(x_i\), the simulator does nothing; otherwise the simulator computes the session key \(σ_j\).

The three Send queries together correspond to an adversary’s ability to monitor and modify messages between honest parties, divided into multiple queries to represent the passes of the protocol. Send\(_0\) queries handle the start of a session at party \(i\) and the message \(i\) sends in the first pass. Send\(_1\) queries represent party \(j\) receiving this first pass and responding with the message for the second pass. Finally, Send\(_2\) queries represent the response of \(j\) being received by \(i\) and the protocol finishing.

Because \(S\) cannot directly pass messages between simulated parties, all protocol messages must pass through \(A\), formalizing the ability of \(A\) to eavesdrop. An active adversary is also able to manipulate the massages in the channel as well, whether by changing them, dropping them completely, or simply inserting new messages. This ability is formalized in the sense that no restrictions are imposed on \(A\) when making Send queries: he may inject whatever messages he wishes into protocol sessions by making Send queries in any order with any payload regardless of whether it was output by prior Send query or not. The only special case is passing a value of \(x_i\) to a Send\(_2\) query without it being output by a Send\(_0\) query at the same initiator; this represents the adversary attempting to finish a session that was never started at an initiator that knows the session was never started.

SessionKeyReveal(sid): If the session referred to by the session id \(\text{sid}\) has been completed, then \(S\) returns the session key associated with that session.
A session id has one of three forms. Either \( \text{sID} = (\Pi, I, i, j, x_i) \), \( \text{sID} = (\Pi, R, j, i, x_i, x_j, w) \), or \( \text{sID} = (\Pi, I, i, j, x_i, x_j, w) \). The third coordinate of a session id—the party \( i \) for the first or third form and the party \( j \) for the second—is called the owner of that session; the other party is called the peer. A session is said to be completed if the owner has computed a session key for that session. If \( \text{sID} = (\Pi, I, i, j, x_i, x_j, w) \), we define the matching session to be \( \tilde{\text{sID}} = (\Pi, R, j, i, x_i, x_j, w) \). If \( \text{sID} = (\Pi, R, j, i, x_i, x_j, w) \), then \( \tilde{\text{sID}} = (\Pi, I, i, j, x_i, x_j, w) \).

Corrupt\((i)\): The simulator returns the static key \( s_i \) of party \( i \). A party whose secret key has been revealed to \( A \) in this manner is called dishonest; a party whose secret key has not been so revealed is called honest.

Test\((\text{sID}')\): Suppose that \( \text{sID}' \) identifies a session with \( i \) as the initiator and \( j \) as the responder. If if \( S \) did not correctly guess which session \( A \) would test then \( S \) aborts. Otherwise, \( S \) samples a single bit \( b \leftarrow \{0,1\} \). If \( b = 0 \), \( S \) returns the session key \( \sigma_i \) for the session identified by \( \text{sID}' \) belonging to party \( i \) (\( S \) always returns the key of the initiator, regardless of which form \( \text{sID}' \) has). If \( b = 1 \), \( S \) instead randomly samples and returns \( \sigma \leftarrow \{0,1\}^n \).

The Test query is the key query for the indistinguishability game. In order to fit into the structure of the security definition, we impose some restrictions on Test that are not present for the other queries. The adversary is only allowed to make a single Test query, and only on a fresh session’s session id. Further, we do not allow \( A \) to submit a session id \( \text{sID}' \) to a SessionKeyReveal query after submitting \( \text{sID}' \) to a Test query.

**Definition 6.2.2** (Freshness). Let \( \text{sID}' = (\Pi, I, i', j', x_i', x_j', w) \) or \( (\Pi, I, i', j', x_i', x_j', x_{i'}, w) \) be a completed session with initiator \( i' \) and responder \( j' \). Let \( \tilde{\text{sID}'} \) be the matching session if it exists. We say that \( \text{sID}' \) is fresh if the following conditions all hold:

- The adversary has not made a SessionKeyReveal query on \( \text{sID}' \).
- The adversary has not made a SessionKeyReveal query on \( \tilde{\text{sID}'} \) if it exists.

69
• If \( \widetilde{\text{sid}}^\ast \) does not exist, then both parties \( i^\ast \) and \( j^\ast \) are honest—A has not made a Corrupt query on either of them.

The first two points are a basic measure to make the security game that A plays non-trivial. If the adversary is allowed to submit a single session id \( \text{sid}^\ast \) to both SessionKeyReveal and Test queries, then A can trivially win the security game by comparing the purported key returned from Test with the actual key returned from SessionKeyReveal.

The final point in the definition of freshness is the property that pertains to weak perfect forward secrecy. We restrict the adversary from corrupting the parties in the middle of them completing a session, but do allow A to corrupt the parties immediately afterwards. Thus we still model an adversary’s ability to recover static keys afterwards and attempt to use that information to distinguish session keys, but by requiring that \( \text{sid}^\ast \) has a matching session \( \widetilde{\text{sid}}^\ast \) in this case, we ensure that A has acted in a passive manner, accurately transmitting \( x_i^\ast \), \( x_j^\ast \), and \( w \) between honest parties \( i^\ast \) and \( j^\ast \).

6.3. Security of RLWE HMQV

Recall the decisional Diffie–Hellman (DDH) problem.

**Definition 6.3.1** (Decisional Diffie–Hellman). Fix a cyclic group \( G = \langle g \rangle \) of order \( q \). A decisional Diffie–Hellman challenge is a triple \((g^x, g^y, g^z)\) of elements in \( G \) where \( x \) and \( y \) are independently and uniformly chosen from \( \mathbb{Z}_q \). A Yes instance (or DDH triple) is a triple where \( z = xy \); a No instance is a triple where \( z \) is uniformly chosen from \( \mathbb{Z}_q \) independently of \( x \) and \( y \).

The DDH assumption states that the probability distributions of the two instances are computationally indistinguishable, or, equivalently, that given \( g^x, g^y, \) and \( Z \) chosen with probability \( 1/2 \) each to be either \( g^{xy} \) or uniformly sampled from \( G \), no adversary can tell which choice was made for \( Z \) with non-negligible advantage. The DDH assumption is precisely the assumption that Diffie–Hellman key exchange over the group \( G \) is secure.
We base the security of our protocol on a novel variation of the RLWE problem we call RLWE-DH, based off the DDH problem. Intuitively, we consider two RLWE samples $X = as_X + 2e_X$ and $Y = as_Y + 2e_Y$. Then the direct analogue of the DDH problem is asking, given $Z \in \mathbb{R}_q$, whether $Z = as_Xs_Y + 2g$ is obtained in the manner of our protocol—by taking $X \cdot s_Y + 2g_Y$ or $Y \cdot s_X + 2g_X$, or $Z \leftarrow \mathbb{R}_q$ is uniformly chosen. We make the formal definition slightly different, however.

**Definition 6.3.2 (RLWE-DH).** Fix the ring $\mathbb{R}_q$ and base point $a \in \mathbb{R}_q$. A RLWE-DH challenge is a triple $(X, Y, K)$ of elements of $\mathbb{R}_q$. We say the challenge is a *Yes* instance if there is some $s \leftarrow x_a$ such that $K = X \cdot s + 2g_X$ and $Y = a \cdot s + 2e_Y$ are RLWE samples with the same secret $s$; the challenge is a *No* instance if $Y$ and $K$ are uniformly sampled. The RLWE-DH problem is to tell, given a triple $(X, Y, K)$, whether the triple is a *Yes* or *No* instance.

Note that we allow that $X$ is either uniformly sampled from $\mathbb{R}_q$ or else itself a RLWE instance $X = as_X + 2e_X$; reversing the roles of $X$ and $Y$ gives an equivalent computational problem.

**Theorem 6.3.3.** Given the RLWE assumption, no adversary can distinguish whether a given RLWE-DH challenge triple is a *Yes* instance or a *No* instance. Furthermore, this is true even if given a sequence of polynomially many RLWE-DH challenge triples $\{(X_k, Y_k, K_k)\}_{k}$ that are either all *Yes* instances or all *No* instances instead of a single triple.

*Proof.* The way we have formalized the RLWE-DH problem, a RLWE-DH distinguisher is a distinguisher that can determine whether two purported RLWE samples with a common secret are actual RLWE samples or uniform. Since the RLWE assumption states that this is hard, even given polynomially many samples, the RLWE-DH problem reduces directly from the standard RLWE problem. \[\square\]
6.3.1. Sequence of Games

As alluded to above, we proceed to prove the security of our protocol via a series of games. Each of these games is between a PPT adversary \( A \) and a simulator \( S \) that simulates a possibly modified version of our protocol by responding to queries from \( A \). During the course of the game, \( A \) makes a Test query to \( S \) as detailed in subsection 6.2.2. We say that \( A \) wins a game if the bit \( b' \) guessed is equal to the bit \( b \) chosen during the Test query in a run of that game. As in the general protocol case, we define an adversary’s advantage in game \( G_k \) to be \( \text{Adv}_{G_k} A = \Pr[A \text{ wins game } G_k] - 1/2 \).

We make a few observations about the sequence of games before we give them. First, the key point of the sequence of games is that the protocol we have given corresponds directly to the game \( G_0 \), and so for any adversary \( A \), the true advantage \( \text{Adv}_{\Pi} A \) is equal to the adversary’s advantage \( \text{Adv}_{G_0} A \) in \( G_0 \). Second, as we will show later, the final game has, by its very definition, zero advantage for any adversary \( A \). Lastly, as we move from game to game, the advantage that \( A \) has does not change by a non-negligible amount.

**Theorem 6.3.4** (Protocol security). Let \( A \) be any PPT adversary for game \( G_0 \). The advantage \( \text{Adv}_{G_0} A \) is negligible in the dimension of the lattice. That is, no adversary has non-negligible advantage in game \( G_0 \).

**Lemma 6.3.5.** Suppose that games \( G_k \) and \( G_k' \) are computationally indistinguishable—that is, any distinguisher \( D \) between \( G_k \) and \( G_k' \) has negligible advantage. Then, for any adversary \( A \), \( |\text{Adv}_{G_k} A - \text{Adv}_{G_k'} A| = \text{negl}(n) \).

**Proof.** We present the proof for the games \( G_0 \) and \( G_1 \); the proof holds *mutatis mutandis* for any pair of games in the sequence. Let \( \varepsilon_0 = \text{Adv}_{G_0} A \) and \( \varepsilon_1 = \text{Adv}_{G_1} A \), and assume without loss of generality that \( \varepsilon_0 > \varepsilon_1 \). By definition of Adv and the fact that \( G_0 \) and \( G_1 \) are chosen
uniformly, we have the following basic probabilities:

\[
\begin{align*}
\Pr[A \text{ wins } | G_0] &= \frac{1}{2} + \epsilon_0 \\
\Pr[G_0] &= \frac{1}{2} \\
\Pr[A \text{ wins } | G_1] &= \frac{1}{2} + \epsilon_1 \\
\Pr[G_1] &= \frac{1}{2}
\end{align*}
\]

which lead to the following compound probabilities:

\[
\begin{align*}
\Pr[A \text{ wins } \cap G_0] &= \frac{1}{4} + \frac{\epsilon_0}{2} \\
\Pr[A \text{ loses } \cap G_0] &= \frac{1}{4} - \frac{\epsilon_0}{2} \\
\Pr[A \text{ wins } \cap G_1] &= \frac{1}{4} + \frac{\epsilon_1}{2} \\
\Pr[A \text{ loses } \cap G_1] &= \frac{1}{4} - \frac{\epsilon_1}{2}
\end{align*}
\]

We build a distinguisher $D$ for games $G_0$ and $G_1$. The distinguisher takes the simulator $S$ and runs an adversary $A$ on it. If $A$ wins the security game, then $D$ outputs $G_0$; if $A$ loses, then $D$ outputs $G_1$. Then $D$ wins the distinguishing game in the event $[A \text{ wins } \cap G_0] \cup [A \text{ loses } \cap G_1]$.

Thus, we have $\Pr[D \text{ wins}] = 1/2 + (\epsilon_0 - \epsilon_1)/2$. Then $\text{Adv } D = (\epsilon_0 - \epsilon_1)/2$, so that $\text{Adv } G_0 A = 2 \text{Adv } D + \text{Adv } G_1 A$. Therefore, if $\text{Adv } D = \text{negl}(n)$, we also have $\text{Adv } G_0 A = \text{Adv } G_1 A + \text{negl}(n)$, or equivalently, $\text{Adv } G_0 A - \text{Adv } G_1 A$ is negligible.

**GAME** $G_0$: The initial game, $G_0$, corresponds to how the protocol works in the real world. The simulator responds to the various Send queries by generating actual ephemeral keys and returning those. We formally define the simulator’s responses as follows:

Send$_0(\Pi, I, i, j)$: The simulator activates a new session with initiator $i$ and responder $j$; $S$ then samples $r_i, f_i \leftarrow \chi_n$, and returns the message $x_i = ar_i + 2f_i$ intended for party $j$.  

73
Send$_1$(Π,𝑅,𝑗,𝑖,𝑥$^i$): The adversary relays a message purportedly from party $i$ as initiator for party $j$ as responder. The simulator proceeds by sampling $r_j, f_j, g_j \leftarrow \chi_\alpha$. Then $S$ computes $x_j = ar_j + 2f_j, k_j = p_i r_j + x_j(s_j + r_j) + 2g_j, w = \text{Cha}(k_j)$, and $\sigma_j = \text{Mod}_2(k_j, w)$. Lastly, $S$ returns the message $(x_j, w)$ to $A$.

Send$_2$(Π,𝐼,𝑖,𝑗,𝑥$^i$, 𝑥$^j$, 𝑤): The adversary relays a message purportedly for party $i$ as initiator from party $j$ as responder to complete a session initiated at party $i$ with message $x_i$ returned from a Send$_0$ query. If no such query returned $x_i$, the simulator does nothing; otherwise the simulator proceeds by sampling $g_j \leftarrow \chi_\alpha$. Then $S$ computes $k_i = p_j r_i + x_j(s_i + r_i) + 2g_i$ and $\sigma_i = \text{Mod}_2(k_i, w)$.

SessionKeyReveal(sid): If the session referred to by the session id sid has been completed, then $S$ returns the session key associated with that session.

Corrupt(i): The simulator returns the static key $s_i$ of party $i$. A party whose secret key has been revealed to $A$ in this manner is called dishonest; a party whose secret key has not been so revealed is called honest.

Test(sid$^*$): Suppose that sid$^*$ = (Π,𝐼,𝑖,𝑗,𝑥$^i$, 𝑥$^j$, 𝑤) or sid$^*$ = (Π,𝑅,𝑗,𝑖,𝑥$^i$, 𝑥$^j$, 𝑤) so the session has $i$ as the initiator and $j$ as the responder. If $(i, j) \neq (i^*, j^*)$, or if $x_i$ was not output by the $\ell$-th session of $i^*$ or if $x_j$ was not output by the $\ell$-th session of $j^*$—that is, if $S$ did not correctly guess which session $A$ would test—then $S$ aborts. Otherwise, $S$ samples a single bit $b \leftarrow \{0, 1\}$. If $b = 0$, $S$ returns the session key $\sigma_i$ for the session identified by sid$^*$ belonging to party $i$ ($S$ always returns the key of the initiator, regardless of which form sid$^*$ has). If $b = 1$, $S$ instead randomly samples and returns $\sigma \leftarrow \{0, 1\}^n$.

Game $G_1$: This game provides simply a conceptual change in how the secret key $\sigma_i$ is generated for the test instance, without actually changing the value.
Send\(_0(Π, I, i, j)\): The simulator activates a new session with initiator \(i\) and responder \(j\); \(S\) then samples \(r_μ, f_i \leftarrow \chi_α\), and returns the message \(x_i = ar_i + 2f_i\) intended for party \(j\).

Send\(_1(Π, R, j, i, x_i)\): The adversary relays a message purportedly from party \(i\) as initiator for party \(j\) as responder. The simulator proceeds by sampling \(r_μ, f_i, g_j \leftarrow \chi_α\). Then \(S\) computes \(x_j = ar_j + 2f_j, k_j = p_j r_j + x_i(s_i + r_i) + 2g_j, w = \text{Cha}(k_j)\), and \(σ_j = \text{Mod}_2(k_j, w)\). Lastly, \(S\) returns the message \((x_j, w)\) to \(A\).

Send\(_2(Π, I, i, j, x_i, x_j, w)\): If \((i, j) \neq (i^∗, j^∗)\) or this is the \(ℓ_i\)-th session of party \(i\), then \(S\) proceeds as in \(G_0\): If no Send\(_0\), query returned \(x_i\), the simulator does nothing; otherwise the simulator proceeds by sampling \(g_j \leftarrow \chi_α\). Then \(S\) computes \(k_i = p_j r_i + x_j(s_i + r_i) + 2g_i\) and \(σ_i = \text{Mod}_2(k_j, w)\). If, \((i, j) = (i^∗, j^∗)\) and this is the \(ℓ_{i^∗}\)-th session of party \(i^∗\), then \(S\) sets \(σ_i = σ_j\) if \((x_j, w)\) was output by the \(ℓ_{j^∗}\)-th session of party \(j^∗\), and computes \(k_i = p_j r_i + x_j(s_i + r_i) + 2g_i\) and \(σ_i = \text{Mod}_2(k_j, w)\) normally if it was not.

SessionKeyReveal(sid): If the session referred to by the session id \(\text{sid}\) has been completed, then \(S\) returns the session key associated with that session.

Corrupt(i): The simulator returns the static key \(s_i\) of party \(i\). A party whose secret key has been revealed to \(A\) in this manner is called dishonest; a party whose secret key has not been so revealed is called honest.

Test(sid\(^∗\)): Suppose that \(\text{sid}\(^∗\) = (Π, I, i, j, x_i, x_j, w)\) or \(\text{sid}\(^∗\) = (Π, R, j, i, x_i, x_j, w)\) so the session has \(i\) as the initiator and \(j\) as the responder. If \((i, j) \neq (i^∗, j^∗)\), or if \(x_i\) was not output by the \(ℓ_i\)-th session of \(i^∗\) or if \(x_j\) was not output by the \(ℓ_{j^∗}\)-th session of \(j^∗\)—that is, if \(S\) did not correctly guess which session \(A\) would test—then \(S\) aborts. Otherwise, \(S\) samples a single bit \(b \leftarrow \{0, 1\}\). If \(b = 0\), \(S\) returns the session key \(σ_i\) for the session identified by \(\text{sid}\(^∗\)\) belonging to party \(i\) (\(S\) always returns the key of the initiator, regardless of which form \(\text{sid}\(^∗\)\) has). If \(b = 1\), \(S\) instead randomly samples and returns \(σ \leftarrow \{0, 1\}\)
Lemma 6.3.6. For any adversary $A$, $\text{Adv}_{G_0} A$ is not greater than $\text{Adv}_{G_1} A$ by a non-negligible amount.

Proof. The only difference between games $G_0$ and $G_1$ is that in $G_1$ when completing the (honestly run) Test session at the initiator, the initiator’s secret key is defined to be the same as that of the responder. By the correctness of our protocol, this fails to happen in $G_0$ with negligible probability. Thus, we have that $G_0$ and $G_1$ cannot be distinguished with non-negligible probability. \hfill \Box

**Game $G_2$:** In this game, we replace the initiator’s ephemeral key with random noise. Since the initiator has no secret ephemeral key with which to calculate the shared secret, the approximate secret $k_i$ is also randomized.

Send$_0(\Pi, I, i, j)$: The simulator activates a new session with initiator $i$ and responder $j$; If $(i, j) = (i^*, j^*)$ and this is the $\ell_j$-th session of $i^*$, $S$ samples $\hat{x}_i \leftarrow R_{q''}$, and returns $x_i = \hat{x}_i - p_i$. Otherwise, $S$ samples $r_i, f_i \leftarrow \chi_{\alpha_i}$, and returns the message $x_i = ar_i + 2f_i$ intended for party $j$.

Send$_1(\Pi, R, j, i, x_i)$: The adversary relays a message purportedly from party $i$ as initiator for party $j$ as responder. The simulator proceeds by sampling $r_j, f_j, g_j \leftarrow \chi_{\alpha_j}$. Then $S$ computes $x_j = ar_j + 2f_j, k_j = p_jr_j + x_j(s_j + r_j) + 2g_j, w = \text{Cha}(k_j)$, and $\sigma_j = \text{Mod}_2(k_j, w)$. Lastly, $S$ returns the message $(x_j, w)$ to $A$.

Send$_2(\Pi, I, i, j, x_i, x_j, w)$: If $(i, j) \neq (i^*, j^*)$ or this is the $\ell_i$-th session of party $i$, then $S$ proceeds as in $G_0$: If no Send$_0$, query returned $x_j$, the simulator does nothing; otherwise the simulator proceeds by sampling $g_j \leftarrow \chi_{\alpha_j}$. Then $S$ computes $k_i = p_j r_i + x_i(s_i + r_i) + 2g_i$, and $\sigma_i = \text{Mod}_2(k_i, w)$. If $(i, j) = (i^*, j^*)$ and this is the $\ell_i$-th session of party $i^*$, then $S$ sets $\sigma_i = \sigma_j$ if $(x_j, w)$ was output by the $\ell_i$-th session of party $j^*$, and samples $k_i \leftarrow R_{q'}$ and sets $\sigma_i = \text{Mod}_2(k_i, w)$ if it was not.

SessionKeyReveal(sid): If the session referred to by the session id sid has been completed, then $S$ returns the session key associated with that session.
Corrupt(i): The simulator returns the static key \( s_i \) of party \( i \). A party whose secret key has been revealed to \( A \) in this manner is called dishonest; a party whose secret key has not been so revealed is called honest.

Test(sid\(^*\)): Suppose that \( \text{sid}^* = (\Pi, I, i, j, x_i, x_j, w) \) or \( \text{sid}^* = (\Pi, R, i, x_i, x_j, w) \) so the session has \( i \) as the initiator and \( j \) as the responder. If \((i, j) \neq (i^*, j^*)\), or if \( x_j \) was not output by the \( \ell_{j^*} \)-th session of \( j^* \)—that is, if \( S \) did not correctly guess which session \( A \) would test—then \( S \) aborts. Otherwise, \( S \) samples a single bit \( b \leftarrow \{0, 1\} \).

If \( b = 0 \), \( S \) returns the session key \( \sigma_i \) for the session identified by \( \text{sid}^* \) belonging to party \( i \) (\( S \) always returns the key of the initiator, regardless of which form \( \text{sid}^* \) has). If \( b = 1 \), \( S \) instead randomly samples and returns \( \sigma \leftarrow \{0, 1\} \). 

**Lemma 6.3.7.** For any adversary \( A \), \( \text{Adv}_{G_1} A \) is not greater than \( \text{Adv}_{G_2} A \) by a non-negligible amount.

**Proof.** Suppose that an adversary \( B \) can distinguish between \( G_1 \) and \( G_2 \), we build a distinguisher \( D \) for RLWE-DH as follows. The distinguisher \( D \) takes a RLWE-DH challenge instance \( \{(X_k, Y_k, K_k)\}_{k=1}^N \) and performs as the simulator for the adversary, responding as described below. In each simulation of the game, the triple \( (X, Y, K) \) is chosen to be the next unused triple from the challenge.

**Send\( _0(\Pi, I, i, j) \):** The simulator activates a new session with initiator \( i \) and responder \( j \); If \((i, j) = (i^*, j^*)\) and this is the \( \ell_{i^*} \)-th session of \( i^* \), \( D \) sets \( \hat{x}_i = X \), and returns \( x_i = \hat{x}_i - p_i \). Otherwise, \( D \) samples \( r_i, f_i \leftarrow \chi_{\alpha} \), and returns the message \( x_i = ar_i + 2f_i \) intended for party \( j \).

**Send\( _1(\Pi, R, j, i, x_i) \):** The adversary relays a message purportedly from party \( i \) as initiator for party \( j \) as responder. If \((i, j) = (i^*, j^*)\) and this is the \( \ell_{j^*} \)-th session of \( j^* \), \( D \) sets \( \hat{x}_j = Y \), \( x_i = \hat{x}_j - p_j \), and \( k_j = K - as_j \). Otherwise the distinguisher proceeds by sampling \( r_j, f_j, g_j \leftarrow \chi_{\alpha} \). Then \( D \) computes \( x_j = ar_j + 2f_j \) and \( k_j = p_j r_j + x_j(s_j + r_j) + 2g_j \). In either case, \( D \) computes \( w = \text{Cha}(k_j) \), and \( \sigma_j = \text{Mod}_2(k_j, w) \) and returns the message \((x_j, w)\) to \( B \).
Send\(_2(\Pi, I, i, j, x_\nu, x_\rho, w)\): If \((i, j) \neq (i^*, j^*)\) or this is the \(\ell_{i^*}\)-th session of party \(i\), then \(S\) proceeds as in \(G_0\): If no Send\(_0\), query returned \(x_i\), the simulator does nothing; otherwise the simulator proceeds by sampling \(g_j \leftarrow \chi\). Then \(S\) computes \(k_i = p_j r_i + x_j (s_i + r_i) + 2g_j\) and \(\sigma_i = \text{Mod}_2(k_i, w)\). If \((i, j) = (i^*, j^*)\) and this is the \(\ell_{i^*}\)-th session of party \(i^*\), then \(S\) sets \(\sigma_i = \sigma_{i^*}\) if \((x_j, w)\) was output by the \(\ell_{j^*}\)-th session of party \(j^*\), and samples \(k_i \leftarrow R_\alpha\) and sets \(\sigma_i = \text{Mod}_2(k_i, w)\) if it was not.

SessionKeyReveal\(\text{(sid)}\): If the session referred to by the session id \(\text{sid}\) has been completed, then \(S\) returns the session key associated with that session.

Corrupt\((i)\): The simulator returns the static key \(s_i\) of party \(i\). A party whose secret key has been revealed to \(B\) in this manner is called dishonest; a party whose secret key has not been so revealed is called honest.

Test\(\text{(sid\')}: Suppose that \text{sid'} = (\Pi, I, i, j, x_\nu, x_\rho, w)\) or \(\text{sid'} = (\Pi, R, i, x_\rho, w)\) so the session has \(i\) as the initiator and \(j\) as the responder. If \((i, j) \neq (i^*, j^*)\), or if \(x_j\) was not output by the \(\ell_{j^*}\)-th session of \(i^*\) or if \(x_j\) was not output by the \(\ell_{i^*}\)-th session of \(j^*\)—that is, if \(S\) did not correctly guess which session \(B\) would test—then \(S\) aborts. Otherwise, \(S\) samples a single bit \(b \leftarrow \{0, 1\}\). If \(b = 0\), \(S\) returns the session key \(\sigma_i\) for the session identified by \(\text{sid'}\) belonging to party \(i\) (\(S\) always returns the key of the initiator, regardless of which form \(\text{sid'}\) has). If \(b = 1\), \(S\) instead randomly samples and returns \(\sigma \leftarrow \{0, 1\}^n\).

If \((X_k, Y_k, K_k)\) is a Yes instance then \(D\) behaves as the simulator in \(G_1\) and if a No instance, \(D\) behaves as the simulator in \(G_2\). Then \(D\) returns Yes if \(B\) returns \(G_1\) and No if \(B\) returns \(G_2\), resulting in the same advantage against RLWE-DH as \(B\) has between \(G_1\) and \(G_2\). Assuming that RLWE-DH samples cannot be distinguished from uniform with non-negligible advantage—in particular, \(D\) cannot have non-negligible advantage—\(B\) also cannot have non-negligible advantage distinguishing the two games. As stated above, if the games cannot be distinguished, no adversary \(A\) can have a non-negligible difference between \(\text{Adv}_{G_1} A\) and \(\text{Adv}_{G_2} A\). \(\Box\)
Game $G_3$: In this game, we replace the responder's ephemeral key with random noise. As

$\text{Send}_0(\Pi, I, i, j)$: The simulator activates a new session with initiator $i$ and responder $j$; If $(i, j) = (i^*, j^*)$ and this is the $\ell_i$-th session of $i^*$, $S$ samples $\hat{x}_i \leftarrow R_q$, and returns $x_i = \hat{x}_i - p_i$. Otherwise, $S$ samples $r_i, f_i \leftarrow \chi_\alpha$, and returns the message $x_i = ar_i + 2f_i$ intended for party $j$.

$\text{Send}_1(\Pi, R, j, i, x_i)$: The adversary relays a message purportedly from party $i$ as initiator for party $j$ as responder. If $(i, j) = (i^*, j^*)$ and this is the $\ell_j$-th session of $j^*$, $S$ samples $\hat{x}_j \leftarrow R_q$, sets $x_j = \hat{x}_j - p_j$, and samples $k_j \leftarrow R_q$. Otherwise the distinguisher proceeds by sampling $r_j, f_j, g_j \leftarrow \chi_\alpha$. Then $S$ computes $x_j = ar_j + 2f_j$ and $k_j = p_j r_j + x_i(s_j + r_j) + 2g_j$. In either case, $S$ computes $w = \text{Cha}(k_j)$, and $\sigma_j = \text{Mod}_2(k_j, w)$ and returns the message $(x_j, w)$ to $A$.

$\text{Send}_2(\Pi, I, i, j, x_i, x_j, w)$: If $(i, j) \neq (i^*, j^*)$ or this is the $\ell_i$-th session of party $i$, then $S$ proceeds as in $G_0$: If no $\text{Send}_0$, query returned $x_i$, the simulator does nothing; otherwise the simulator proceeds by sampling $g_j \leftarrow \chi_\alpha$. Then $S$ computes $x_j = ar_j + 2f_j$ and $k_j = p_j r_j + x_i(s_j + r_j) + 2g_j$. In either case, $S$ computes $w = \text{Cha}(k_j)$, and $\sigma_j = \text{Mod}_2(k_j, w)$. If $(i, j) = (i^*, j^*)$ and this is the $\ell_i$-th session of party $i^*$, then $S$ sets $\sigma_j = \sigma_i$ if $(x_j, w)$ was output by the $\ell_i$-th session of party $j^*$, and samples $k_i \leftarrow R_q$ and sets $\sigma_i = \text{Mod}_2(k_i, w)$ if it was not.

$\text{SessionKeyReveal}(\text{sid})$: If the session referred to by the session id $\text{sid}$ has been completed, then $S$ returns the session key associated with that session.

$\text{Corrupt}(i)$: The simulator returns the static key $s_i$ of party $i$. A party whose secret key has been revealed to $A$ in this manner is called dishonest; a party whose secret key has not been so revealed is called honest.

$\text{Test}(\text{sid}')$: Suppose that $\text{sid}' = (\Pi, I, i, j, x_i, x_j, w)$ or $\text{sid}' = (\Pi, R, j, i, x_i, x_j, w)$ so the session has $i$ as the initiator and $j$ as the responder. If $(i, j) \neq (i^*, j^*)$, or if $x_i$ was not output by the $\ell_i$-th session of $i^*$ or if $x_j$ was not output by the $\ell_j$-th session of $j^*$—that is, if $S$ did not correctly guess which session $A$ would test—then $S$ aborts. Otherwise, $S$ samples a single bit $b \leftarrow \{0, 1\}$.
If $b = 0$, $S$ returns the session key $\sigma_i$ for the session identified by $\sin^*$ belonging to party $i$ ($S$ always returns the key of the initiator, regardless of which form $\sin^*$ has). If $b = 1$, $S$ instead randomly samples and returns $\sigma \leftarrow \{0, 1\}^n$.

**Lemma 6.3.8.** For any adversary $A$, $\text{Adv}_{G_2} A$ is not greater than $\text{Adv}_{G_3} A$ by a non-negligible amount.

**Proof.** Suppose that an adversary $A$ can distinguish between $G_2$ and $G_3$, we build a distinguisher $D$ for RLWE-DH as follows. The distinguisher $D$ takes a RLWE-DH challenge instance $\{(X_k, Y_k, K_k)\}_{k=1}^N$ and performs as the simulator for the adversary, responding as described below. In each simulation of the game, the triple $(X, Y, K)$ is chosen to be the next unused triple from the challenge.

Send$_0(\Pi, I, i, j)$: The simulator activates a new session with initiator $i$ and responder $j$; If $(i, j) = (i^*, j^*)$ and this is the $\ell_i$-th session of $i^*$, $D$ sets $\hat{x}_i = X$, and returns $x_i = \hat{x}_i - p_i$. Otherwise, $D$ samples $r_i, f_i \leftarrow \chi_a$, and returns the message $x_i = ar_i + 2f_i$ intended for party $j$.

Send$_1(\Pi, R, j, i, x_i)$: The adversary relays a message purportedly from party $i$ as initiator for party $j$ as responder. If $(i, j) = (i^*, j^*)$ and this is the $\ell_j$-th session of $j^*$, $D$ sets $\hat{x}_j = Y$, $x_j = \hat{x}_j - p_j$. Otherwise the distinguisher proceeds by sampling $r_j, f_j, g_j \leftarrow \chi_a$. Then $D$ computes $x_j = ar_j + 2f_j$ and $k_j = p_j r_j + x_j (s_j + r_j) + 2g_j$. In either case, $D$ computes $w = \text{Cha}(k_j)$, and $\sigma_j = \text{Mod}_2(k_j, w)$ and returns the message $(x_j, w)$ to $A$.

Send$_2(\Pi, L, i, j, x_i, x_j, w)$: If $(i, j) \neq (i^*, j^*)$ or this is the $\ell_i$-th session of party $i$, then $S$ proceeds as in $G_0$: If no Send$_0$, query returned $x_i$, the simulator does nothing; otherwise the simulator proceeds by sampling $g_j \leftarrow \chi_a$. Then $S$ computes $k_i = p_j r_i + x_i (s_i + r_i) + 2g_i$ and $\sigma_i = \text{Mod}_2(k_i, w)$. If $(i, j) = (i^*, j^*)$ and this is the $\ell_j$-th session of party $j^*$, then $S$ sets $\sigma_j = \sigma_i$ if $(x_j, w)$ was output by the $\ell_i$-th session of party $i^*$, and samples $k_i \leftarrow R_q$ and sets $\sigma_i = \text{Mod}_2(k_i, w)$ if it was not.
SessionKeyReveal(sid): If the session referred to by the session id sid has been completed, then S returns the session key associated with that session.

Corrupt(i): The simulator returns the static key $s_i$ of party $i$. A party whose secret key has been revealed to $A$ in this manner is called dishonest; a party whose secret key has not been so revealed is called honest.

Test(sid\*): Suppose that $\text{sid}^* = (\Pi, I, i, j, x_i, x_j, w)$ or $\text{sid}^* = (\Pi, R, j, i, x_j, x_i, w)$ so the session has $i$ as the initiator and $j$ as the responder. If $(i, j) \neq (i^*, j^*)$, or if $x_i$ was not output by the $\ell_i$-th session of $i^*$ or if $x_j$ was not output by the $\ell_j$-th session of $j^*$—that is, if $S$ did not correctly guess which session $A$ would test—then $S$ aborts. Otherwise, $S$ samples a single bit $b \leftarrow \{0, 1\}$. If $b = 0$, $S$ returns the session key $\sigma_i$ for the session identified by $\text{sid}^*$ belonging to party $i$ ($S$ always returns the key of the initiator, regardless of which form $\text{sid}^*$ has). If $b = 1$, $S$ instead randomly samples and returns $\sigma \leftarrow \{0, 1\}^n$.

If $(X_k, Y_k, K_k)$ is a Yes instance then $D$ behaves as the simulator in $G_2$ and if a No instance, $D$ behaves as the simulator in $G_3$. Then $D$ returns Yes if $B$ returns $G_2$ and No if $B$ returns $G_3$, resulting in the same advantage against RLWE-DH as $B$ has between $G_2$ and $G_3$. Assuming that RLWE-DH samples cannot be distinguished from uniform with non-negligible advantage—in particular, $D$ cannot have non-negligible advantage—$B$ also cannot have non-negligible advantage distinguishing the two games. As stated above, if the games cannot be distinguished, no adversary $A$ can have a non-negligible difference between $\text{Adv}_{G_2} A$ and $\text{Adv}_{G_3} A$. 

Note that in $G_3$, the Tested session has a session key that is chosen independently of the messages exchanged. Therefore, no adversary can have non-negligible advantage against $G_3$, as there is no information about the key contained in the messages exchanged.
6.3.2. Key Reuse and Verification

The most computationally expensive of many key exchange protocols is the generation of ephemeral keys. For performance reasons, a server may wish to generate a single key at start-up and use that in a static manner throughout the server’s uptime rather than regenerating it anew for every connection. In a classical Diffie–Hellman scenario (including Elliptic Curve Diffie–Hellman), this would be the case where Bob uses the same $g^b$ for every incoming connection; in the protocols described above, it would be the case where Bob uses the same $p_B$ and $x_B$ for every connection. (Recall that in the authenticated version, $p_B$ is the same while $x_B$ differs; in the unauthenticated version $p_B$ differs and $x_B$ does not exist).

Note that version 1.2 of the TLS protocol explicitly allows the server to use a static DH key in this manner [Die08]. The current draft (draft 13) of version 1.3 allows a client to take further advantage of this in the so-called Zero-RTT mode [Res], specifying to the server that it is using the server’s static key it already knows from a previous session, allowing the client to start using the shared key to encrypt application data to be sent along with the initial handshake. This has the goal of improving latency and security by reducing the time spent and data transferred before the secure channel is established. The reuse of static Diffie–Hellman keys introduces separate security concerns, mainly related to forward secrecy. The attacks described in the following paragraphs rely on this reuse of keys to gain information about the static key over a number of sessions.

**Small Subgroup Attack** Since Bob can derive the Diffie–Hellman secret $g^{ab}$ solely from Alice’s public $g^a$ and his own private $b$, if Mallory can learn the secret key $b$, she can also retrieve any of Bob’s previous shared secrets generated with $b$. If Mallory engages in a key exchange session with Bob, she can send him a dishonestly formed ephemeral key $A$ in an attempt to gain information about $b$. A typical example against both original and elliptic curve DH is the small subgroup attack. In ordinary DH key exchange, consider an ephemeral key $M = g^m$ that has small order $k \ll |g|$ in $\mathbb{Z}_p^\times = \langle g \rangle$. If Mallory can intentionally pick $m$ so $M$ has sufficiently
small order, she can find \( K = M^b = (g^b)^m \), and then brute-force the discrete log of \( K \) in \( \langle M \rangle \) to find \( b \mod k \). If Bob reuses \( b \) across many sessions, Mallory could potentially find \( b \mod k_j \) for a number of factors \( k_j \) of \( |g| = p - 1 \) and find \( b = b \mod p - 1 \) using the Chinese remainder theorem.

A common mitigation for this type of attack is to pick \( p \) so that \( p - 1 \) has a large prime factor \( q \), and choose \( g \) to generate a subgroup of order \( q \) rather than all of \( \mathbb{Z}_p^\times \). Then Mallory cannot generate \( M \) with order other than \( q - 1 \), so this attack cannot work as-is. As Lim and Lee [LL97] showed, however, if Mallory picks a malformed ephemeral key \( M \) that does not actually have order \( q \), so that \( M \) is not in the subgroup \( \langle g \rangle \), a similar attack can still recover information about \( b \) if Bob does not verify that \( M^q \mod p = 1 \).

**Invalid Curve Attack** A distinct, but similar, attack called an invalid curve attack can be used to attack elliptic curve-based Diffie–Hellman protocols [BMMoo; Ant+03]. Recall that a (non-identity) element of an elliptic curve group over a finite field \( \mathbb{F}_q \) is a pair of coordinates \((x,y) \in \mathbb{F}_q^2\) that satisfies a specific equation of the form \( y^2 = x^3 + ax + b \) (the general form is slightly different, but most curves used in practice are of this form); the group addition on the elliptic curve can be defined purely in terms of the coefficients of the points \( P = (x_P, y_P) \) and \( Q = (x_Q, y_Q) \) as \((x_P, y_P) + (x_Q, y_Q) = (x_R, y_R)\) where

\[
\begin{align*}
x_R &= \lambda^2 - x_P - x_Q, \\
y_R &= \lambda(x_P - x_R) - y_P, \\
\lambda &= \begin{cases} 
\frac{y_Q - y_P}{x_Q - x_P} & P \neq Q, \\
\frac{3x_P^2 + a}{2y_P} & P = Q.
\end{cases}
\end{align*}
\]

As a special case, if \( Q = \bar{P} \), so that \( x_Q = x_P \) and \( y_Q = -y_P \), we set \( P + Q = \infty \). Note that these formulas can be used for any points \( P \) and \( Q \) in \( \mathbb{F}_q^2 \), again taking \( P + Q = \infty \) if \( x_P = x_Q \).
and \( y_p \neq y_Q \), even if \( P \) and \( Q \) are not on the given curve. Scalar multiplication by integers is defined as repeated addition in the usual way.

Consider the elliptic curve \( E(\mathbb{F}_q) \) given by \( y^2 = x^3 + ax + b \), and a base point \( P \in E \) for ECDH key exchange. A slightly different curve \( E'(\mathbb{F}_q) \) given by \( y^2 = x^3 + ax + b' \) that differs only in its constant coefficient is called an invalid curve relative to \( E \). The curves \( E \) and \( E' \) have the same group addition formulas, which depend on \( a \) but not \( b \). In an honest run of ECDH, Alice picks an integer \( w_A \) and generates a point \( Q = w_A P \) to send to Bob. Bob uses the secret integer \( w_B \) that he generates to find the shared secret \( w_B Q = (w_A w_B) P \).

Suppose, on the other hand, that Mallory takes an invalid curve \( E' \) with a point \( Q \) of small order \( \ell \), and sends that to Bob instead of an honestly generated \( Q \). Since the group operation uses the same formulas over both curves, when Bob goes to compute \( w_B Q \) over \( E \), he instead computes it over \( E' \). Mallory repeats this process, guessing each possible value for \( w_B Q \), and finding \( w_B \mod \ell \) when she finds the correct shared secret and Bob continues communications. From there, the attack proceeds similar to the small subgroup attack, finding \( w_B \mod \ell_j \) for various factors \( \ell_j \) of the order \( n \) of \( P \), then recombining them to get the actual value of \( w_B = w_B \mod n \) using the Chinese remainder theorem. Invalid curve attacks have been demonstrated against popular cryptographic software as recently as 2015 [JSS15].

**RLWE Verification**  The above attacks rely on the fact that the server reuses the same static key across multiple sessions. The simple mitigation is to always generate new ephemeral keys, but as key generation is the most expensive part of our key exchange protocol and static keys are popular for performance reasons, we would like to have additional measures for servers that do not generate new keys every session.

Unfortunately, the basic technique of the previous paragraphs to verify that the message from the client is properly formed before using it. The security of our scheme relies on the fact that the message \( p_A \) that Alice sends, if honestly generated, is computationally indistinguishable from uniformly random without knowledge of \( s_A \). Thus, for Alice to prove to Bob that \( p_A \) was
honestly generated seems to be to somehow send Bob the value of $s_A$. This has to be confirmed before attempting to use the shared key, however, because even whether the adversary can successfully connect to Bob can potentially leak information about the shared key. Thus, we cannot simply have Alice send the secret $s_A$ as the first message encrypted with the derived key.

We instead apply an idea proposed by Lackey [Lac] for RLWE-based Diffie–Hellman type protocols called indirect key verification. The idea is to adapt the Fujisaki-Okamoto transform [FO99] to provide a way to transfer knowledge of $s_A$ over to Bob in the key exchange itself in a manner somewhat reminiscent of key encapsulation. We will assume for the sake of the description that we are working with the original, unauthenticated version of the RLWE-DH protocol described in section 5.1—the technique of proving the key $p_A$ of the unauthenticated version is honestly generated applies in much the same way as proving that $x_A$ is in the authenticated version. Unfortunately, this same technique is somewhat less than ideal to verify that a static key such as $p_A, p_B$ in the authenticated version is properly generated; since demonstrating that a key is honestly generated compromises the key, any party that the honesty of the key is verified to can impersonate the party whose key it is.

Instead of directly generating a random $s_A \in \mathbb{R}_q$, Alice first generates a secret string $\gamma$ of bits, which she uses to generate her secret $s_A$ using some key derivation function that outputs according to $\chi_\alpha$. She computes $p_A$ from $s_A$ as usual. Using Bob’s static key $p_B$ that she already knows, she computes $k_A$ as usual, computes $w = \text{Cha}(k_A)$ and $\sigma_A = \text{Mod}_2(k_A, w)$. Since Alice already knows Bob’s key, she can compute the signal $w$ on her end before communicating with Bob. Alice splits $\sigma_A$ into two pieces, a secret key $sk$ and a verification key $vk$, and the final shared secret key will be $sk$. (Note that the precise derivation of $sk$ and $vk$ from $k_A$ and $w$ can differ).

Next, Alice applies a variation of the Fujisaki–Okamoto transformation. She sends $p_A$ and the signal $w$ to Bob combined with a symmetric encryption of the seed $\gamma$ using the key $vk$. The precise details of this encryption step are omitted here. Bob then uses Alice’s key $p_A$ and the
signal $w$ together with his static secret $s_B$ to compute $k_B$ and $\sigma_B = \text{Mod}_2(k_B, w)$. He uses $\sigma_B$ to obtain the shared secret $sk$ and the verification key $vk$, then uses $vk$ to decrypt $\gamma$. With $\gamma$, Bob can reconstruct $s_A$, and verify that $p_A$ has the form $as_A + 2e_A$ for some small noise $e_A$. If it does not, then Alice has sent a malformed key, and Bob refuses to communicate with her using the session key $sk$.

This behavior for a one-pass variant is slightly different from how we presented the protocol before, having Alice compute and send the signal $w$ instead of Bob, but as the protocol is symmetric up to that point, which party computes and sends the signal $w$ is largely irrelevant. We have Bob do so in the description above because $w$ can only be computed by a party who has received the public key $p_j$ of the other, so if Alice initiates the key exchange, having her send the signal would require a third pass; in this case having Bob send the signal would require it to be a second pass after Bob receives $p_A$, while Alice can send it in her first pass as described here. There is a minor security consideration as well. If Bob uses a static key $p_B$ and sends the signal $w$ himself, there is a theoretical possibility that a series of specifically constructed $p_A$ from Alice could have $w$ leak information about $e_B$ or, equivalently, about $s_B$, based on watching for $w$ to change as $p_A$ is. By moving the signal computation to the party without the static key, it cannot reveal information about that key.
Conclusions and Future Work

7.1. Conclusions

We have presented here an authenticated key exchange scheme, based on the ring learning with errors problem, and building upon some previous RLWE key exchange protocols. The scheme’s primary improvement over the prior RLWE-based authenticated key exchange scheme is the removal of random oracles from the protocol description. This allows us to base the quantum security of our scheme directly on the quantum hardness of the RLWE problem.

Beyond the fact that we avoid the random oracle model, the modifications we present for the authenticated key exchange protocol bring performance improvements as well. By keeping the overall noise down, we can reduce the size of the modulus $q$, resulting in both faster computations and smaller transmission sizes.

Overall, we conclude that the ring learning with errors problem can be used to create efficient, secure cryptographic tools for use in a post-quantum world. We do not have every issue resolved, but we have shown that the basic protocol of Ding et al. [DL12] works as a basis for building more intricate protocols.
7.2. Future Work

A number of further directions to take this research suggest themselves. A further refined version with stronger security guarantees against active adversaries would be a significant result. Even without an improved security guarantee, a better understanding of the security implications of key reuse in RLWE key exchange schemes like ours is important for real-world use.

As mentioned above in subsection 6.2.1, no two-pass protocol with implicit authentication and no pre-established secret from prior communications can attain perfect forward secrecy, only the weakened version that ours has. If we allow for a third message to be transmitted, which could be used as an explicit authentication mechanism rather than the implicit authentication of our current system, that result does not apply anymore. A promising line of research would be extending our scheme to use a third pass in order to attain stronger security guarantees against active adversaries, including perfect forward secrecy.

Another significant direction of research is to better understand the security implications of key reuse. As mentioned in subsection 6.3.2, it is common practice to use static keys in place of ephemeral keys in order to avoid having to generate separate ephemeral keys for every session. A more in-depth analysis of how much information an attacker can gain from sending malformed messages would give a better understanding of exactly how this reuse affects the scheme’s security. The small-subgroup and invalid curve attacks against static keys in classical Diffie–Hellman protocols have no direct application to RLWE schemes, but that fact does not preclude similar attacks from improperly generated keys. Finding better ways for the initiator to prove the honest generation of his/her key to a server using a static key remains an open problem for now. A zero-knowledge method of proof for this would be preferable, but such a zero-knowledge proof technique will be very difficult to find due to the indistinguishability of the RLWE distribution from uniform; indeed, it may be the case that no such technique can exist.
Bibliography


Appendix A

Performance Comparison

We give a brief comparison of the performance of our streamlined RLWE authenticated key exchange to various lattice-based key exchange proposals. The primary comparison in Table A.1 to be made is with the previous authenticated key exchange protocol of [Zha+15] that we base our protocol on; we also provide data from some recent published implementations of Peikert’s unauthenticated protocol [Pei14] to establish a rough baseline.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Initiator (ms)</th>
<th>Responder (ms)</th>
<th>Total (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLWE Authenticated Key Exchange</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLWE-HMQV</td>
<td>7.254</td>
<td>7.261</td>
<td>14.515</td>
</tr>
<tr>
<td>Improved RLWE-AKE</td>
<td>3.281</td>
<td>3.298</td>
<td>6.579</td>
</tr>
<tr>
<td>RLWE Key Exchange in TLS [Bos+14]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLWE in TLS</td>
<td>1.0</td>
<td>1.7</td>
<td>2.7</td>
</tr>
<tr>
<td>Constant-time RLWE in TLS</td>
<td>1.4</td>
<td>2.1</td>
<td>3.5</td>
</tr>
</tbody>
</table>

Table A.1.: Performance Comparison

The primary comparison for our purposes is between the RLWE-based HMQV protocol of [Zha+15], as described in section 5.2 and our modified version as described in section 5.3. The implementation uses common dimension $n = 1024$, modulus $q = 2^{32} - 1$, and Gaussian parameter $\alpha = 8/\sqrt{2\pi} \approx 3.2$ for more meaningful comparison. Both versions were implemented in C++ using the NTL and GMP libraries; polynomial arithmetic was done using the
Fast Fourier transform for additional performance. The time taken is broken into the Initiator, which includes time taken both to generate the first pass message $x_A$ (initiation step) and compute the shared secret $\sigma_A$ after receiving the response $(x_B, w)$ (finish step), and the Responder, which includes the time taken to generate the response $(x_B', w')$ and shared secret $\sigma_B$ (response step).

As mentioned above, we have three primary areas with performance advantages over the hash-based version. The hash-based version includes three multiplications to compute the shared secret; the new version can be computed in two multiplications using a partly-factored form for $k_A$ and $k_B$. The use of only a single parameter $\alpha$ for the discrete Gaussian sampling instead of a small $\alpha$ and larger $\beta$ means less time is taken to sample those elements that were taken from $\chi_\beta$ in the previous version. Finally, the time taken to compute $c$ and $d$ is no longer needed, as $c$ and $d$ no longer exist.

For further comparison, we give the timings from [Bos+14], which implemented Peikert’s unauthenticated scheme within the larger context of the TLS protocol. Their work consists of a number of modifications to the OpenSSL software suite to provide support for the RLWE key exchange as part of a custom ciphersuite. We report the timings only for the key exchange portion rather than the entire TLS handshake. The lattice parameters of $n$, $q$, and $\alpha$ match those for the authenticated implementations discussed above; in fact, the parameters for the authenticated implementations were chosen to provide this direct comparison with the implementation of [Bos+14].

Our authenticated key exchange system takes significantly less time than the hash-based version, but lags somewhat behind the custom TLS implementation of [Bos+14]. Some of this may be attributable to the fact that they implemented an unauthenticated system and use the usual technique for TLS of RSA or ECDSA signatures to provide authentication.