Lattice cryptography: isomorphisms & dual attacks

Pulles, L.N.

Part I
Introduction & Preliminaries

Chapter 1

Introduction

> *In a few decades [...] transfer of information will probably be much faster and much cheaper by "electronic mail" than by conventional postal systems. [...]*
>
> *It is hardly surprising that in recent years a number of mathematicians have asked themselves: is it possible to devise a cipher that can be rapidly encoded and decoded by computer, can be used repeatedly without changing the key and is unbreakable by sophisticated cryptanalysis? The surprising answer is yes.*
>
> — Martin Gardner [Gar77]

Today's society is unimaginable without cryptography, and our personal digital presence is inextricably linked by cryptographic protocols: messaging, online banking or filing taxes. The world has changed significantly within a century since the development of the first computer, which aided the breaking of the Enigma machine's encryption during the second world war.

For a very long time, if two people wanted to communicate while keeping its content unknown to others, one could *encrypt* messages using a *secret key*, send this *ciphertext* to the other, and the other could *decrypt* these ciphertexts using that same secret key. Worldwide usage of this encryption method was virtually impossible as the common secret key had to be confidentially communicated beforehand, for example by meeting in person. Encryption was impractical!

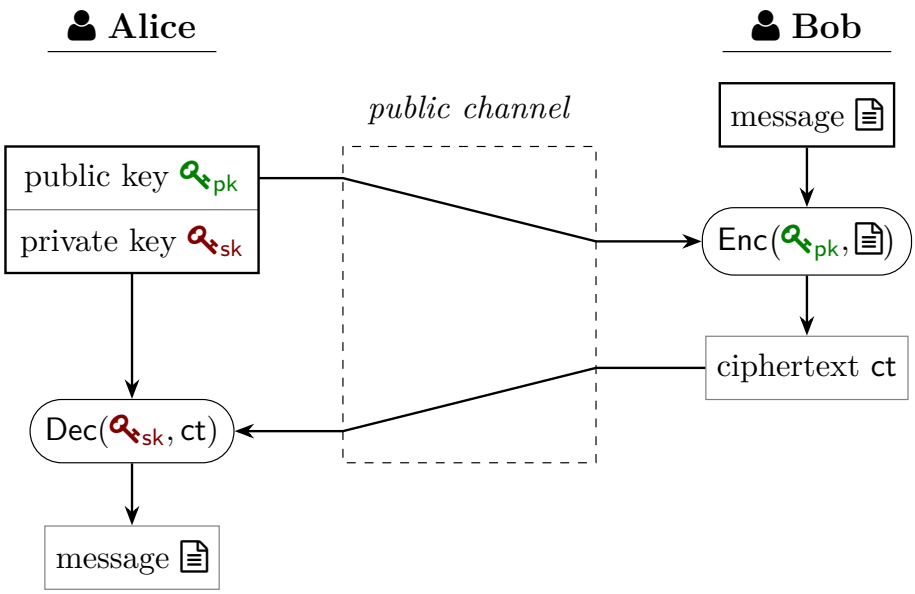


Figure 1.1: Public-key encryption, in which Bob tries to confidentially communicate a message to Alice. Over a public channel, Alice sends her public key to Bob, and Bob sends a ciphertext back.

Rivest–Shamir–Adleman

Fifty years ago, it suddenly became much easier to use cryptography with the discovery of *public-key cryptography*. In an article, Gardner [Gar77] challenges the reader to break the *public-key encryption scheme* RSA, named after its inventors Rivest, Shamir and Adleman [RSA78], who also offered $100 to anyone solving the challenge.

Public-key encryption, visualised in Figure 1.1, is a way of sending a message confidentially that does not require both parties to know a shared secret. The key difference with symmetric cryptography is that encryption now uses a *public key* known to everyone, while decryption requires knowledge of the associated *private key* only known by one individual.

Suppose Alice wants to receive messages confidentially from anyone using RSA encryption. First, Alice randomly generates two large prime numbers p and q, and picks a relatively small integer e. Then, she computes the *RSA modulus* $N = p \cdot q$, and constructs an integer d such that $d \cdot e \equiv 1 \pmod{(p-1)(q-1)}$ holds.

While making the public key $\mathsf{pk} = (N, e)$ known to everyone, she makes sure that the private key $\mathsf{sk} = (p, q, d)$ is kept secret. Now, the encryption of a message $m \in \{0, 1, \ldots, N-1\}$ is simply $c \equiv m^e \pmod{N}$ which can be publicly computed, and passed publicly to Alice. Only Alice, who knows d, can decrypt the ciphertext using $m \equiv c^d \pmod{N}$, which follows from elementary number theory. In order to break encryption, it is sufficient to find a prime divisor of N. Although it is generally not necessary to recover the private key in order to break encryption, factoring N still remains the fastest known method for breaking RSA encryption.

If the number N is sampled uniformly from $\{1, 2, \ldots, n\}$, finding a prime divisor of N is easy in at least 50% probability, that is when N is even, but is considered hard in the worst case [Knu97, 4.5.4]. The number N is generated in RSA specifically as a product of two primes, both of similar size, such that private key recovery requires solving a factorisation instance that is *hard in the average case*. A locksmith should deliver secure locks every time, and similarly, a good cryptosystem should generate a public/private key pair such that the private key is never easily derived from the public key. Hence, RSA generates N as a product of two primes.

In Gardner's article, the challenge was, given a ciphertext and public key (N, e), to recover a message, which was encoded as an integer modulo N by mapping the space character to 00 and letters A–Z to 01–26, respectively. The RSA modulus N was said to be a 129-digit, obtained by multiplying a 64-digit prime p and a 65-digit prime q. Rivest estimated for the article that, 'if the best algorithm known and the fastest of today's computers were used,' solving this challenge would take $40 \cdot 10^{12}$ years. According to [AGLL94], the fastest algorithm at that time was the heuristic Pollard's ρ algorithm [Pol75] which outputs a divisor of N using $2\sqrt{p}$ modular operations in practice, although $\mathcal{O}(p/\log\log p)$ is the best known provable upper bound [Hea17].

> *Rivest and his associates have no proof that at some future time no one will discover a fast algorithm for factoring composites as large as the N [red.] they used or will break their cipher by some other scheme they have not thought of. They consider both possibilities extremely remote.*
>
> — Martin Gardner [Gar77]

1

For decades many researchers tried to find a fast algorithm for the factorisation problem, but nobody found one that could *efficiently* factorise N, i.e., there is no factorisation algorithm known with a runtime that grows polynomial in its input size, $\lg N$. Still, better *superpolynomial* algorithms were discovered, such as the quadratic sieve [Pom85], and the number field sieve (NFS) [LL93].

Gardner's challenge was ultimately solved in 1994 by a fast implementation of the quadratic sieve [AGLL94]. In their article, the authors recommend stopping use of RSA with a 512-bit modulus N (155 digits), and move to a 1024-bit modulus. Note this challenge can be solved nowadays within an hour distributed over 8 CPUs [McH15].

In 2009, a 768-bit (232 digits) RSA modulus was factored with the NFS, 2 years after the RSA Laboratories withdrew the $50,000 prize. Subsequently, the recommendation was to use a 2048-bit modulus.

Putting things into perspective, RSA was a big success, and has already seen wide usage for a very long time. Despite progress into integer factorisation, the doublings of the necessary key-sizes have been manageable, so far...

The quantum threat

In 1994, Shor publishes a *quantum algorithm* [Sho94] that could factorise a number N using $\mathsf{poly}(\lg N)$ operations on *qubits*, the quantum analogue of the normal bits inside a classical computer. Although no quantum computer existed at the time, Shor's algorithm threatened the security of RSA. In the same paper, Shor proposes another quantum algorithm that solves the discrete logarithm problem, which threatens other cryptosystems that use elliptic curves.

So far, there does not exist a cryptographically relevant quantum computer (CRQC), i.e., a quantum computer with sufficiently many qubits that are sufficiently fault-tolerant to factorise 2048-bit RSA. Physicists are working hard at lowering quantum gate error rates [SLM+25], and increasing the number of qubits. On the other hand, Shor's algorithm has seen recent improvements [GE21, Reg25, CFS25, Gid25], lowering the required number of qubits further.

With these advances, the time until the first CRQC appears becomes less and less, and with it the need to migrate to quantum-resistant cryptography increases. The current state of quantum computers is summarised in Figure 1.2.

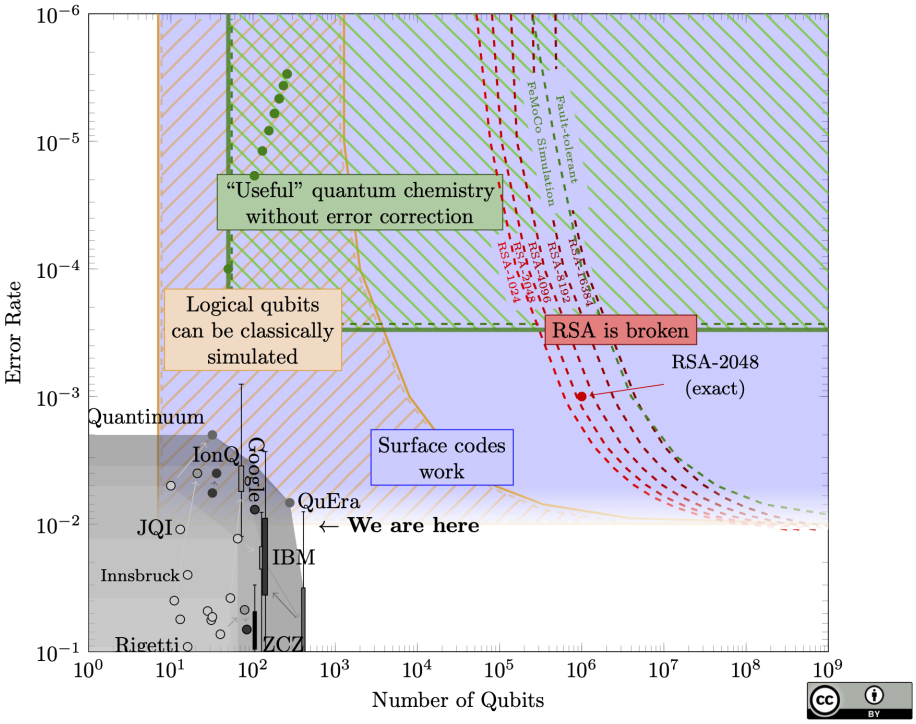


Figure 1.2: Landscape of quantum computing capabilities in terms of fault tolerance and number of qubits in a log scale. Physically realised quantum computers (grey) versus the requirements of Gidney's algorithm [Gid25] to break RSA (red). Created by Sam Jaques [Jaq25].

Note that some products are shipped with immutable cryptography, like vehicles or smart cards. Moreover, an eavesdropper can simply store communication today, and decrypt it when a CRQC is available, which may uncover still relevant sensitive industrial/medical data, or even state secrets. This is the so-called '*store-now/decrypt-later*' threat. Hence, all sensitive data must be migrated to post-quantum cryptography long before a CRQC exists.

Post-quantum cryptography migration

The imminent threat of quantum computers capable of breaking encryption creates the desire to transition towards new cryptographic systems that are secure against *both* quantum and classical computers. Such so-

called *post-quantum cryptography* must run on (classical) computers, and needs to rely on hardness assumptions that are different from RSA and discrete log. This led the National Institute of Standards and Technology (NIST) in the US to invite researchers from all over the world to submit post-quantum cryptography candidates with the purpose of establishing a standard that could be used for free by governments, industries and people worldwide to migrate before a CRQC exists [Nat16].

In 2022, NIST announced ML-KEM will be the new standard for a *key encapsulation mechanism* (KEM) [AAC+22]. A KEM enables two parties to establish a shared secret key over a public communication channel, so they can use (more efficient) symmetric encryption to communicate securely. Hence, a KEM is used to construct a public-key encryption scheme that is efficient regardless of the message size. As of 2026, Cloudflare observes [Wes25] that *at least half* of human web traffic uses the post-quantum secure KEM 'X25519MLKEM768', which is a hybrid of X25519, an elliptic curve-based KEM that is susceptible to Shor's algorithm, and ML-KEM. An attacker needs to break both X25519 *and* ML-KEM in order to obtain the 'X25519MLKEM768' private key. This hybridisation provides security that is best of both worlds at the cost of running and communicating both KEMs.

On the governmental side, US governmental systems need to be quantum-safe by 2035, while the EU high-risk systems need to be quantum-safe by 2030 and the rest by 2035. Australia sets more aggressive goals by demanding migration by 2030 to ML-KEM-1024, which offers more security than ML-KEM-768.

While the urgent migration of public-key encryption is already on the right trajectory, there is an important second migration needed:

> *The second migration, to post-quantum signatures (certificates), is not as urgent: we will need to have this sorted by the time the quantum computer arrives. However, it will be a bigger challenge.*
>
> — Westerbaan and Valenta [WV24]

Migration of digital signatures

Often in internet communication, one needs to know they are talking to the right person, or one needs to assert that some document really originates from someone. This can be achieved using a *digital signature*

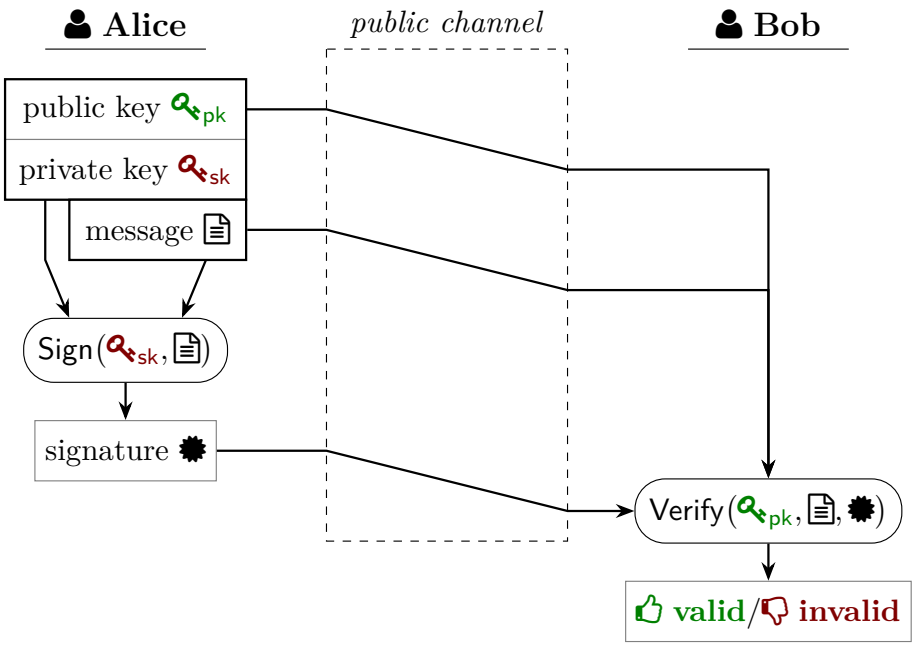


Figure 1.3: Digital signature scheme, in which Bob wants to know whether Alice sent him a particular message.

scheme or *signature scheme*, depicted in Figure 1.3.

Say Alice wants to send a message m to other people. Before the age of computers, Alice would add a wax seal to such a message, convincing people she used the seal die on that message. A digital signature scheme may be considered the digital analogue to the wax seal that offers more security as the seal is now specific to the message. Namely, Alice generates a key pair consisting of a public key pk and private key sk, and then publishes pk. A signature σ is the result of an algorithmic routine that takes as input her private key sk and the message m. Then, anyone acquiring the message and signature, can run a procedure called Verify that takes as input (pk, m, σ) and decides whether the signature is *valid*. Formally, we say a signature scheme is *existentially unforgeable under chosen message attacks* (EUF-CMA secure) if an attacker, without the private key sk, cannot produce within a certain time a valid signature on a message $\mathsf{m}^\star$ of their liking, even if they can persuade Alice to sign a certain number of arbitrary messages $\mathsf{m}_i \neq \mathsf{m}^\star$. Here the words 'certain' depend on the desired level of security.

1

Rivest–Shamir–Adleman [RSA78] proposed a signature scheme that is surprisingly similar to RSA encryption (p. 4). Namely, using the same public/private keys, a message m is signed using $\sigma \equiv m^d \pmod N$, and a signature σ is valid when $m \equiv \sigma^e \pmod N$ holds. This is essentially the idea of RSA signatures, although it is not EUF-CMA secure: if an attacker learns a valid signature σ for message m, then they can forge a signature 2σ for the artificial message $m' \equiv 2^e m \pmod N$. By slightly modifying the scheme, e.g., hashing the message [Cor00], one can make it EUF-CMA secure. Moreover, it is not secure to reuse the key pair from RSA encryption for RSA signatures.

In contrast to encryption, which needs to be quantum-safe as soon as possible due to store-now/decrypt-later, digital signatures cannot be forged until a CRQC exists. Still, these signatures need to be implemented in critical parts of the internet, especially because it takes multiple years before X.509 certificates, used in secure web-browsing, are replaced by new (post-quantum) signatures.

In 2022, NIST chose [AAC+22] to standardise three signature schemes: ML-DSA, FN-DSA and SLH-DSA. These three have different characteristics in terms of hardness assumption, runtimes, communication sizes, and implementation complexity. It therefore depends on the application which of these three is the most favourable. An SLH-DSA signature requires more than 7 kB, which is too large for some internet applications. The first two have smaller signature sizes of 2420 B and 666 B, respectively.

Based on signature sizes, one would guess that NIST would recommend using FN-DSA instead of ML-DSA, but the opposite is true: ML-DSA is recommended in general! FN-DSA signatures need floating-point numbers to generate signatures, but some CPUs do not support these. Namely, CPUs of some small chips, such as smart cards, do not support double-precision floating-point numbers. Although one can carefully emulate floating-points with 64-bit integers, this severely slows down signing. The implementation difficulties of FN-DSA are also evident from the fact that there is no FN-DSA standard yet, as opposed to ML-KEM [Nat24a], ML-DSA [Nat24b] and SLH-DSA [Nat24c].

When the third round of NIST's PQC standardisation process finished, there were no more signature schemes under consideration [AAC+22]. Among the standardised signature schemes, only SLH-DSA is not based on structured lattices, but this one has relatively large signature sizes.

As such, NIST opened a call for additional digital signature proposals to be considered in the PQC standardisation process [Nat22]. While NIST is primarily interested in diversifying the post-quantum signature standards by schemes that do not rely on structured lattices, NIST is also interested in lattice-based proposals that significantly outperform ML-DSA and FN-DSA. This standardisation process, also known as the PQC signature 'on ramp', considers nine candidates in its ongoing third round [ABC+26].

1.1 Motivation

Lattices are the central object of study in this thesis, as three of the four schemes standardised by NIST are based on lattices: ML-KEM, ML-DSA and FN-DSA.

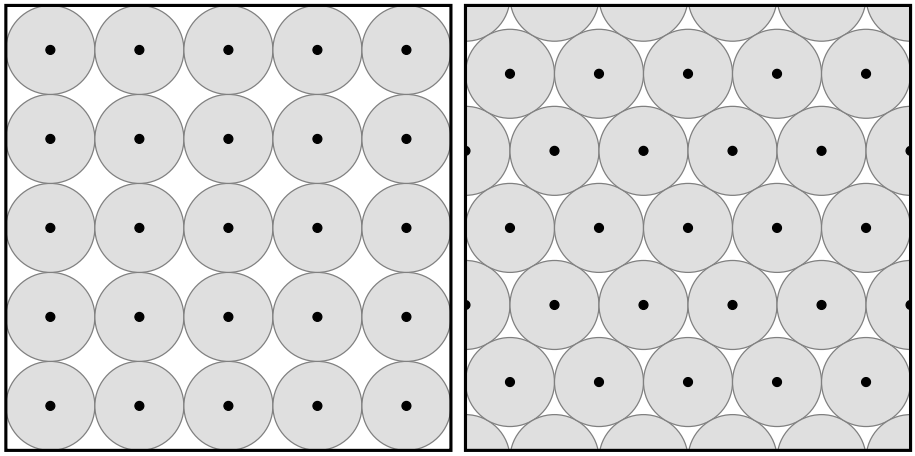
We will perform *cryptanalysis* on lattices. Cryptanalysis is the process of designing, analysing and optimising algorithms that can break a scheme, and determining the most efficient one. The schemes are supposed to offer security, so it will take too long to run the best attack in an experiment. Instead, we can only estimate the runtime of the best attack by understanding it theoretically, and by extrapolating runtimes required to break weaker instances. For accurate runtime estimates of lattice-based cryptosystems, it is essential to have a thorough understanding of lattices.

Lattices

A lattice is a discrete, additive subgroup of n-dimensional Euclidean space, $\mathbb{R}^n$. Figure 1.4 contains two examples of a lattice. Here, *discrete* means that there is a minimal distance between any two lattice points, i.e., we can draw a ball of some positive radius around every lattice point such that none of these balls intersect. Moreover, *additive subgroup* means that the collection of points is closed under addition and negation, i.e., if $\mathbf{x}$ and $\mathbf{y}$ are both lattice points, then so are $-\mathbf{x}$ and $\mathbf{x}+\mathbf{y}$. Note that the zero vector, $\mathbf{0}$, must also lie in a lattice.

For any computational task with a lattice, we use a basis, which consists of n lattice points such that any lattice point can be expressed as an integral combination of the basis vectors.

Example 1.1. Let $n \in \mathbb{Z}_{\geq 1}$. The elementary unit vectors $\mathbf{e}_1, \ldots, \mathbf{e}_n$, where



(a) $\mathbb{Z}^2$ (b) Hexagonal lattice

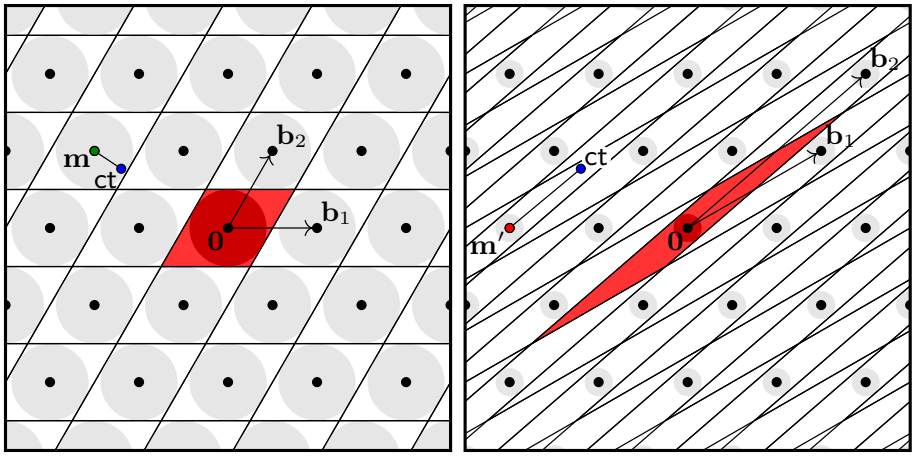
Figure 1.4: Two examples of a lattice, together with nonoverlapping balls centred at each lattice point.

$\mathbf{e}_i \in \mathbb{Z}^n$ has 1 as its ith coefficient and zeros elsewhere, form a basis for the integer lattice $\mathbb{Z}^n$.

While a 1-dimensional lattice has only two bases, any higher dimensional lattice has an infinite number of choices for a basis. For the remainder of this chapter, let us assume the dimension is at least 2. Informally, we say a basis is 'good' if it consists only of short vectors, and say a basis is 'bad' if it does not. For any lattice, there exist arbitrarily 'bad' bases.

Every basis gives a tiling of the space as can be seen in Figure 1.5. The radius of the largest ball that fits in a tile is larger for a good basis than for a bad basis. We may use lattices for public-key cryptography using a good basis as private key, and a bad basis as public key. Namely, we can associate the message space with lattice points, expressed in terms of the public, bad basis. Then, given a lattice point $\mathbf{m}$, we can generate noise $\mathbf{e}$ of a length r such that $r_a > r > r_b$, where r_a, r_b are the radii of the grey circles in (a) and (b) of Figure 1.5, respectively, after which the ciphertext is $\mathsf{ct} = \mathbf{m} + \mathbf{e}$. This way, anyone cannot decode the ciphertext to the correct message using the tiling, unless they have the good basis.

In Figure 1.6, it is shown how a lattice-based signature scheme can be built. While a KEM is usually used once by two parties to establish a



(a) Good basis (b) Bad basis

Figure 1.5: Tiling of space based on a good, and a bad basis for the hexagonal lattice, respectively. The encryption $\mathsf{ct} = \mathbf{m} + \mathbf{e}$ of message $\mathbf{m}$, obtained by adding a short error $\mathbf{e}$, can only be successfully decrypted using the tiling of a good basis.

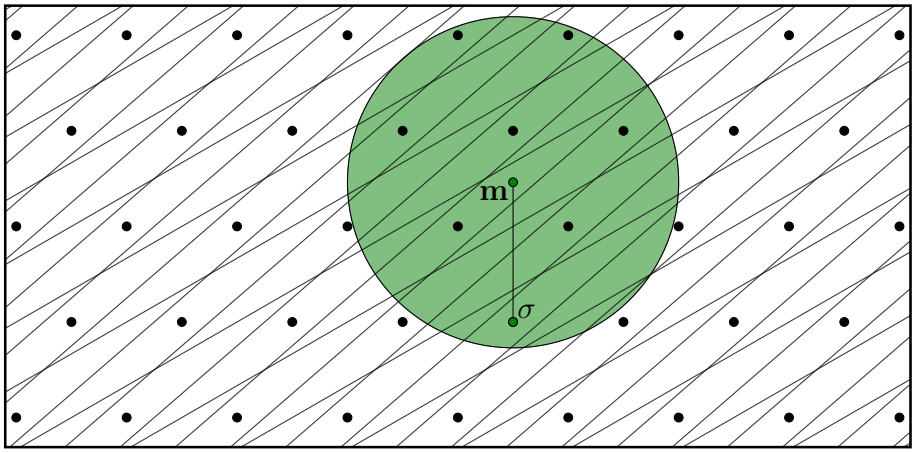


Figure 1.6: High-level idea of a lattice-based signature scheme. A message is mapped into space (a non-lattice point), and a signature is a lattice point in the vicinity of the message.

shared key for (symmetric) encryption, a signature scheme could be used
1 millions of times with the same key. As a consequence, signatures should not leak too much information about the private key. The early lattice-based signature schemes GGH and NTRU, for example, were insecure because the private key could be recovered if many valid signatures were known [NR09]. This makes it more of a delicate matter to build a signature scheme [GPV08].

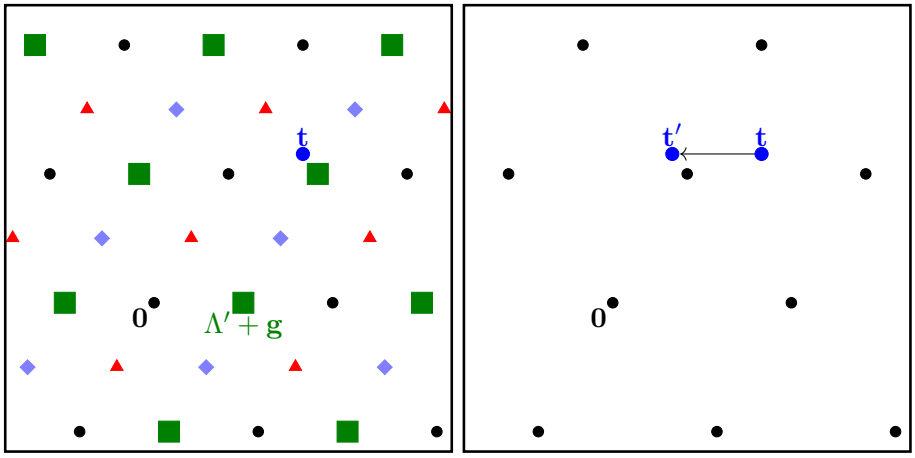
Lattice basis reduction

The process of improving a basis with shorter basis vectors is called *lattice basis reduction*. The *shortest vector problem* (SVP) asks, given an arbitrary basis, to find a shortest lattice point. It was shown [Ajt98] that SVP is NP-hard for randomised reductions, i.e., there is a probabilistic Turing machine that, in polynomial time, reduces any NP-problem to an SVP instance. Note that SVP is trivial when the input basis contains the shortest vector. Yet, even for the average case there is no polynomial-time algorithm known solving SVP, where we omit the average case input distribution as it is technically too involved for a gentle introduction. In contrast to factorisation of a random integer (p. 5), which was trivial with 50% probability, SVP is hard on average with high probability.

On a more algorithmic side, there exists a polynomial-time algorithm [LLL82] that finds a lattice point of length at most 2^n times larger than the smallest norm for a lattice of dimension n. This so-called LLL algorithm improves a basis and prevents its basis vectors from becoming arbitrarily long. In small dimensions, LLL performs well, and the approximation factor is rather small, making LLL useful in many applications [NV10].

Ultimately, secure lattice-based cryptosystems can only be broken by stronger basis reduction, in particular, using the so-called *block Korkine–Zolotarev* (BKZ) reduction [Sch87]. This algorithm requires exponential time in a parameter β, called *block size*, which controls the dimension of projected sublattices. BKZ improves the basis by repeatedly solving SVP locally in these projected sublattices.

To determine security of a KEM, we would like to know, given a ciphertext, how easily one finds the closest lattice point, which is called the *bounded distance decoding problem* (BDD). Similarly for a signature scheme, we would like to know, given a (specific) message, how easily one finds a nearby lattice point, which is called the *(approximate) closest*



(a) Finding nearest coset (b) After sparsification

Figure 1.7: Solving BDD by finding the nearest coset $\Lambda' + \mathbf{g}$ to the target $\mathbf{t}$. In (a), out of the four given cosets of Λ', the green coset is nearest to $\mathbf{t}$. After applying a shift of $-\mathbf{g}$ to the coset, we get (b), which is another BDD instance with lattice Λ' and target $\mathbf{t}' = \mathbf{t} - \mathbf{g}$.

vector problem (CVP). There are two ways of solving BDD: the *primal attack* and the *dual attack*.

Primal attack

The primal attack constructs a lattice such that the BDD solution can be easily computed from the shortest vector of this lattice. Thus, by running BKZ, one can solve BDD. In these lattice constructions, the shortest vector is smaller than that of a random lattice of same dimension. The ratio between these two quantities can then be used to determine the required block size [ADPS16].

Dual attack

The dual attack, visualised in Figure 1.7, partitions the lattice Λ into multiple *cosets* of a sparser sublattice Λ', and then determines which coset the input is closest to. By repeatedly doing this, the lattice becomes sparser during each step, and at a certain point, the lattice is so sparse that the nearest lattice point is easy to recover, e.g., using LLL.

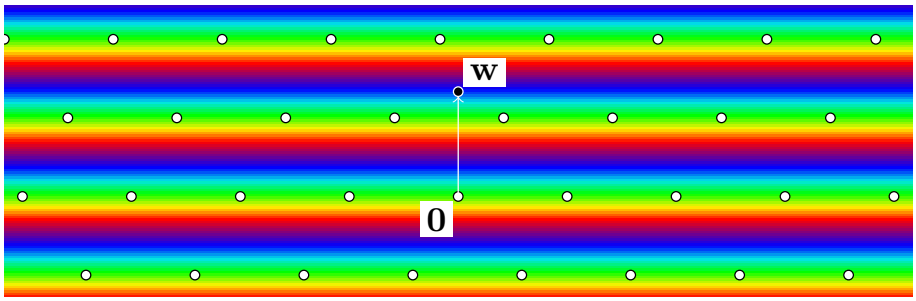


Figure 1.8: Visualisation of a dual vector, using the correspondence between dual vectors and characters. The green colour indicates the top of the wave function, which is at every lattice point.

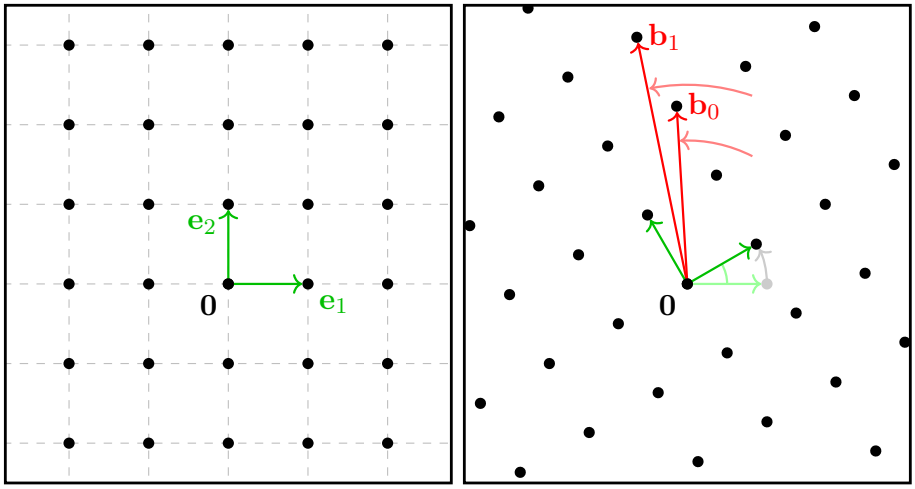
To determine the closest coset to a target, we use many *dual vectors*, i.e., vectors from the so-called *dual lattice*. A dual vector corresponds to something called a *character*, which can be thought of as a travelling wave in n-dimensions, such that the wave is at its top in every lattice point, visualised in Figure 1.8. More formally, a character is a function (with certain properties) from n-dimensional Euclidean space to the unit circle that maps every lattice point to $(1, 0)$. A character needs to be continuous, and therefore a point that lies near the lattice must evaluate to something close to $(1, 0)$. The converse is, however, not necessarily true. By combining many characters, one gains more confidence whether a point is really near the lattice. Namely, we can assign a *score* to a target $\mathbf{t}$, by considering the sum of the evaluations of characters at $\mathbf{t}$. Since the y-coordinate cancels out with that of a dual vector's negation, we may consider the score as a *real* value. Now, given a list of targets, in which only one is close to the lattice, very naturally, the target with the highest score is most likely the closest one to the lattice.

Lattice Isomorphism Problem

The integer lattice, $\mathbb{Z}^n$, cannot be used directly as a cryptosystem because it is publicly known that the decoding function,

$$\lceil \cdot \rfloor \colon \mathbb{R}^n \to \mathbb{Z}^n, \quad (x_1, \ldots, x_n) \mapsto (\lceil x_1 \rfloor, \ldots, \lceil x_n \rfloor),$$

efficiently solves CVP and BDD by rounding each coefficient to its nearest integer. What if we would be able to use such a simple lattice in a cryptosystem?



(a) Good basis for Λ (b) Rotated bad basis for $\mathbf{O}\Lambda$

Figure 1.9: Lattice Isomorphism Problem

Such a cryptosystem would be fast and simple because of this efficient, easily implementable decoding function. Moreover, generating signatures is easy because each coefficient can be sampled independently, and distributions can be precomputed. By building a signature scheme with $\mathbb{Z}^n$, one can generate signature without using floating-points, contrary to the intricate signing in Falcon which requires floating-points because it uses NTRU lattices [HPS98]. How could we use this simple lattice?

The answer lies in the Lattice Isomorphism Problem. Lattices Λ_1 and Λ_2 are said to be *isomorphic* if there is an orthogonal transformation, i.e., a length-preserving map that maps Λ_1 to Λ_2 one-to-one, in which case we call the transformation the *isomorphism*. Such a map can be a combination of a rotation and a reflection through the origin. The *Lattice Isomorphism Problem* (LIP) asks, given two isomorphic lattices, to find such an isomorphism. An illustration is found in Figure 1.9. All best known ways to solve LIP require finding short vectors in each lattice using lattice basis reduction [PS97, HR14, DHVvW20]. Hence, we believe LIP is, on average, as least as hard as solving SVP.

Example 1.2. In the remarkable case of $\mathbb{Z}^n$, of which the shortest vectors form an orthonormal basis, solving $\mathbb{Z}$LIP, LIP for the lattice $\mathbb{Z}^n$, is computationally as hard as finding all its shortest vectors. It is sufficient to find all shortest vectors of a rotation of $\mathbb{Z}^n$ to recover the orthogonal

transformation in LIP, and this is also necessary, as all shortest vectors
1 are encoded in the transformation matrix.

Remark 1.3. To create an instance of LIP, the basis $\mathbf{B}_2$ of Λ_2 must be generated by *both* applying the orthogonal transformation to $\mathbf{B}_1$ *and* applying a random basis transformation to it. Without the basis transformation, one can easily compute the orthogonal transformation as $\mathbf{B}_2\mathbf{B}_1^{-1}$.

In the current formulation, a LIP-based signature scheme certainly must use floating-points to rotate a basis, which would make a robust implementation more tedious than what FN-DSA offers. However, one can avoid all these difficulties by working with bases modulo orthogonal transformations, and things become simpler. Given a basis $\mathbf{B}$, the *Gram matrix* $\mathbf{G} = \mathbf{B}^\mathsf{T}\mathbf{B}$ encodes all the pairwise inner products between basis vectors. This Gram matrix is invariant under orthogonal transformations as these transformations are length-preserving. Moreover, if $\mathbf{B}$ has integer entries, $\mathbf{G}$ is also integral. Hence, we prefer to describe an instance of LIP, not using bases of two lattices but using the Gram matrices of these bases. A Gram matrix is an instance of what is formally called a (positive-definite) *quadratic form*.

Remark 1.4. Quadratic forms were already studied, e.g., in Disquisitiones arithmeticæ (1801) by Gauß, long before lattices came into fashion.

Given a quadratic form $\mathbf{Q}$, one can find a particular lattice basis with Gram matrix $\mathbf{Q}$. However, most lattice algorithms have a quadratic form equivalent, so an actual basis is often not needed when manipulating a quadratic form.

Additional structure

Lattice-based cryptosystems offer security when using a dimension of at least a thousand. A basis for such a lattice involves a million entries, so the communication cost is on the order of megabytes. To reduce the communication and computation costs, one can enrich the lattice with extra algebraic structure.

Ring of integers of a number field are versatile objects that come naturally with a lattice structure by their canonical embeddings into the complex numbers. The additional structure allows us to reduce communication costs to the order of kilobytes. Moreover, we often use

cyclotomic number fields as the so-called *fast Fourier transform* (FFT) reduces computation costs to a complexity that is almost linear. However, on the other hand, the question is:

Question 1.5. Does the additional structure lead to improved attacks?

Initially, researchers considered enriching the lattice constructions from *Learning with Errors* (LWE) [Reg09] with a ring structure, leading to Ring-LWE [SSTX09, LPR13], in which security is now based on the hardness of *ideal lattice* problems, instead of unstructured lattice problems. A quantum algorithm was discovered for finding relatively short vectors in a principal ideal lattice [CGS14], and was generalised to general ideal lattices [CDW17]. This algorithm did not directly impact Ring-LWE as:

(1) Ring-LWE reduces quantumly to an ideal lattice problem [SSTX09] that in the worst case asymptotically has a *smaller* approximation factor than above algorithm, and
(2) Ring-LWE could be computationally harder since there is no reduction known going the other way.

Nevertheless, if one could significantly lower the approximation factors in the algorithm [CGS14], it could leave security of Ring-LWE unsupported.

> *Despite these two serious obstacles to attack Ring-LWE based schemes by the algebraic approach developed in [CGS14, BS16, CDPR16] and in this paper, it seems a reasonable precaution to start considering weaker structured lattice assumptions, such as Module-LWE [LS15] (i.e., an "unusually-Short Vector Problem" in a module of larger rank over a smaller ring), which provides an intermediate problem between Ring-LWE and general LWE.*
>
> — Cramer *et al.* [CDW17]

Since then, module structures have received a lot of cryptanalytic attention and have shown to be fruitful, as ML-KEM and ML-DSA use module structures. Similarly, FN-DSA also uses a module structure that comes from the NTRU lattice construction [HPS98] instead of LWE. In Part III, we will investigate how to incorporate a module structure into LIP.

1.2 Overview

This thesis is divided into three parts. Part I consists of this introductory chapter and Chapter 2, which covers preliminaries that are necessary for the remaining two parts of the thesis. Part II is cryptanalytic work on the dual attack. Part III constructs and analyses a signature scheme based on a module variant of the Lattice Isomorphism Problem.

Part II — Dual-Sieve Attack

Part II is related to the *Dual-Sieve attack*, which is a dual attack that uses lattice sieving to acquire many dual vectors. It is difficult to determine the runtime of the Dual-Sieve attack, because the lattice sieve is a probabilistic algorithm that receives a randomly generated lattice. Therefore, one usually requires a heuristic assumption on the output distribution of a lattice sieve, to analyse the Dual-Sieve attack.

Chapter 3 investigates the so-called independent score heuristic. Using this heuristic, prior works [GJ21, MAT22] claimed the Dual-Sieve attack can break Kyber512 (ML-KEM-512) in an estimated runtime that is lower than NIST's security requirement. In this chapter, we show that this heuristic leads to three contradictions in certain scenarios. One of these contradictions, depicted in Figure 3.3, is concrete and is observed in extensive experiments. This work was initially published in the proceedings of CRYPTO 2023 [DP23c].

Chapter 4 introduces an alternative heuristic, which can be used to simplify the analysis of the Dual-Sieve attack. This new heuristic is verified in practice using multiple experiments. Showing accurate predictions, this work contributes to a better attack cost prediction for state-of-the-art Dual-Sieve attacks. It is left as future work to determine the time of breaking Kyber512 using the Dual-Sieve attack. This work was originally uploaded to the cryptology ePrint archive [DP23b].

The two works, on which Chapter 3 and Chapter 4 are based, were combined into one paper that is published in the Journal of Cryptology [DP26].

Part III — Lattice Isomorphism Problem

Part III is related to the signature scheme Hawk. This scheme was submitted to NIST's standardisation process of additional signature

schemes [BBD+23], and is the only lattice-based candidate that advanced to the third round [BBD+26]. Security of Hawk is based on the hardness of the Lattice Isomorphism Problem for the integer lattice ($\mathbb{Z}$LIP) having a module structure (smLIP).

Chapter 5 introduces the signature scheme Hawk. First, a theoretically simple version of Hawk is explained. Then, Hawk is explained in detail by applying some optimisations to the simplified variant. This scheme is then cryptanalysed in theory, and we experiment with smaller instances of the scheme. Based on this cryptanalysis, parameter sets are given in Table 5.3, which should provide two levels of security. The novel design of Hawk avoids floating-points, and offers small signature and public key sizes, similar to Falcon. Moreover, Table 5.1 shows great performance on high-end and low-end devices. This chapter is based on the initial work published in the proceedings of ASIACRYPT 2022 [DPPvW22a], and is based on the specification document submitted to NIST [BBD+26].

Chapter 6 looks into the so-called one more shortest vector problem (omSVP), which is the hardness problem on which Hawk is based. In this chapter, we want to characterise the symmetries in the definition of omSVP such that there is no trivial algorithm solving omSVP. Specifically, assuming the private key in Hawk cannot be recovered easily, it is shown that the formulation of omSVP covers all symmetries that are easily computed. This work is published in Communications in Cryptology [vGP25].