Post-quantum Lattice-based Cryptography

REBECCA STAFFAS
Post-quantum Lattice-based Cryptography

REBECCA STAFFAS

Master’s Thesis in Mathematics (30 ECTS credits)
Master Programme in Mathematics (120 credits)
Royal Institute of Technology year 2016
Supervisor at Ericsson: John Mattsson
Supervisor at KTH: Svante Linusson
Examiner: Svante Linusson

TRITA-MAT-E 2016:23
ISRN-KTH/MAT/E--16/23--SE

Royal Institute of Technology
School of Engineering Sciences

KTH SCI
SE-100 44 Stockholm, Sweden
URL: www.kth.se/sci
Abstract

In a future full of quantum computers, new foundations for asymmetric cryptography are needed. We study the theoretical foundations for lattice-based cryptography as well as the current state on cryptographic attacks against them. We then turn our attention to signature systems and especially the system BLISS from 2013. We give an overview of the BLISS protocol and its security, and analyse its object sizes and resistance to attacks. We find that BLISS does not provide as high security as initially claimed.

We then propose modifications to BLISS in order to allow for freer choices of dimension and modulus. We also propose novel implementation tricks and accommodate for these in the protocol. We call our modified system REBLISS and propose parameter sets. Our performance measurements suggest that this is a good alternative to BLISS.
Sammanfattning


Vi föreslår sedan förändringar i BLISS för att tillåta ett friare val av dimension och primtal. Vi föreslår innovativa trick för en snabbare implementation och gör plats för dessa i algoritmerna. Vår modifierade algoritm får namnet REBLISS och vi förelår nya uppsättningar av systemparametar. Våra prestandamätningar visar att detta är ett bra alternativ till BLISS.
Acknowledgements

I would like to thank Svante for continuous support, and my on-site supervisor John for his enthusiasm and dedication. I would also like to thank Alexander Maximov for the implementation and his performance perspective, and the department at Ericsson, Ericsson Research Security, for their interest in my work.
Contents

1 Background 10
  1.1 Classical Cryptography ................................. 10
  1.2 The Rise of Quantum Computers .......................... 10
  1.3 Post-Quantum Cryptography ............................... 12
  1.4 Report Outline ........................................ 13
  1.5 Notation ............................................... 13

2 Quantum Algorithms 14
  2.1 Qubits and Quantum Gates ............................... 14
  2.2 The Hidden Subgroup Problem ............................. 15

3 Foundation for Lattice Problems 16
  3.1 Notation ............................................... 16
  3.2 Hard Problems ........................................ 17
  3.3 Ring Problems ......................................... 19
  3.4 Heuristics ............................................. 21
  3.5 Relation to the Hidden Subgroup Problem ................. 22

4 Attacks and Security 23
  4.1 Exact Algorithms ....................................... 23
  4.2 Lattice Reduction Algorithms ............................ 24
  4.3 The Hybrid Attack ..................................... 27

5 Lattice-based Signature Systems 28

6 BLISS 29
  6.1 Prerequisites ........................................... 30
  6.2 The BLISS System ....................................... 32
  6.3 BLISS Instantiations .................................... 41
  6.4 Attacks on BLISS ....................................... 42

7 Analysis of BLISS 43
  7.1 Object Sizes ............................................ 43
  7.2 Security ................................................ 45
  7.3 A Comment on Key Generation ............................ 51

8 BLISS Modifications 51
  8.1 Parameter Setting ....................................... 51
  8.2 New BLISS parameters ................................... 52
  8.3 Additional Dimensions ................................... 53
  8.4 Key Generation Modification ............................. 56
  8.5 Implementation Tricks ................................... 58
  8.6 REBLISS ................................................ 60
  8.7 Implementation and Benchmarks .......................... 61
  8.8 Conclusion ............................................. 62
1 Background

1.1 Classical Cryptography

A foremost goal of cryptography is maintaining the secrecy or integrity of some information, as it travels from sender to receiver, over a possibly corrupted medium. It has been used at least since the Romans, when Julius Caesar allegedly came up with the Caesar cipher to encrypt messages. Nowadays, applications of cryptography is partitioned into a few cryptographic primitives.

The Caesar cipher falls into the category of symmetric cryptosystems, where it is accompanied by more modern cousins like AES, different HMAC schemes such as HMAC-SHA-256, and the use of one-time pads. While symmetric ciphers are generally very secure, fast and produce compact ciphertexts, they suffer from one main drawback: Both sender and receiver need to know the same key. This means that before communication with a symmetric cipher can start, the participants need to meet in a secure environment where they can agree on a key (or have access to a trusted third party to perform a key exchange). With the rise of the Internet, this has become completely infeasible.

Luckily, there are so-called asymmetric cryptosystems, in which each agent has two keys. Asymmetric cryptography includes amongst others encryption schemes, in which a message is transformed to be unreadable by anyone other than the intended receiver, and signature schemes, where a message is accompanied by a token which guarantees that it was sent by the correct actor. Each agent may publish one of his keys to the world, to be used by anyone that wishes to communicate with him. In encryption schemes the public key is used to encrypt, in signature schemes the public key is used to verify signatures. The other key is kept secret by the agent, and is used for example to decrypt, or to sign. To asymmetric cryptography we also count most key exchange algorithms, with which two parties can agree on a secret key (often for a symmetric cipher) over an insecure channel without revealing it to anyone else. Such a key is typically used only for the upcoming conversation and is then discarded.

The beauty of asymmetric cryptosystems is twofold: First, secure communication can be established over a more insecure communication medium. The medium needs to be secure against modifications of messages, but eavesdropping is not dangerous. Second, in the symmetric case one key per communication link has to be created and stored, while in the asymmetric case only one key per agent (plus their secret keys) are required. Public keys can be stored commonly on some server and fetched when needed.

The most widely known asymmetric cryptosystem is RSA, which relies on the difficulty of factoring large integers. The asymmetry in the problem comes from that multiplication, on the other hand, is very easy. This can be used for encryption as well as signatures. Among the most known key exchange algorithms are those of the Diffie-Hellman type, which rely on the difficulty of finding discrete logarithms in finite groups, commonly elliptic curve groups (algorithms in that setting are collectively known as ECC, Elliptic Curve Cryptography). Such a setting can also be used for effective signatures. The difficulty of these two number-theoretic problems have given cryptography a sort of “golden age”, with fast and memory-efficient asymmetric cryptographic schemes.

1.2 The Rise of Quantum Computers

The proceeding development of quantum computers changes the setting for cryptography vastly. This is because they differ fundamentally from classical computers. For example, by the use of superposition functions can essentially be evaluated at several inputs at
1 BACKGROUND

Once. The quantum setting comes with a few drawbacks as well, such as a significant probability for faulty measurements and a ban on erasing information erasure, but these problems have generally been relatively easy to handle.

Already in 1996, when quantum computers were a completely theoretical concept, Grover published an article titled *A fast quantum mechanical algorithm for database search* [Gro96]. There he provides an algorithm that, given a function $f : \{0, 1, \ldots, N\} \to \{0, 1\}$ on a quantum computer where there is exactly one $n$ s.t. $f(n) = 1$, finds that $n$ in $O(\sqrt{N})$ iterations (average case) independent of the internal structure of $f$. This is astonishing since this is impossible to do in less than $O(N)$ iterations on a classical computer (in the worst case $N$ function evaluations are needed, in the average case $N/2$).

Grover’s algorithm is relevant for amongst others symmetric ciphers, on which the only feasible attack has previously been the brute force attack – that is, to try all possible keys until the correct one is found. In effect, it halves all key sizes. Today keys of length 128 bits are typically considered secure in the classical setting, but against a quantum computer these keys give only 64 bits of security, which is too little especially for long-term security. However, an upgrade to 256-bit keys solves the problem – definitely, since [BBBV97] shows that no quantum computer algorithm may invert a black-box function in less than $O(\sqrt{N})$ operations.

It is worth noticing that the constant in the asymptotic behaviour of Grover’s algorithm may be large. In [GLRS15, Table 5], 128-bit AES is suggested to provide not 64, but rather 86 bits of security against a quantum computer. The suggested attack implementation needs almost 3000 qubits to run, a staggering number compared to what is available today, but not judged impossible to achieve in the long run.

The scene is more depressing when we turn our eyes to Shor’s paper, first pre-printed the same year as Grover’s [Sho97], in which he provides two quantum algorithms: One which factors large integers, and one which computes discrete logarithms, both based on a quantum version of the Fast Fourier Transform. This means that both modern foundations for public key cryptography, key exchange algorithms and signature algorithms are suddenly void.

The complexity of Shor’s algorithm is $O(n^3)$ where $n$ is the bit-length of the number to be factored. Actual attacks on for example RSA and ECC are also reasonably efficient [PZ03], though exact timings would depend on the physical implementation of the quantum computer.

Incidentally, while Shor is well aware of the impact his paper has on cryptography, he does not consider this a valid reason to try to build a quantum computer: “Discrete logarithms and factoring are not in themselves widely useful problems. [...] If the only uses of quantum computation remain discrete logarithms and factoring, it will likely become a special-purpose technique whose only raison d’être is to thwart public key cryptosystems”, [Sho97]. However, in the years since then the Internet has grown explosively and with it the need for asymmetric cryptography, which means that the impact of Shor’s paper on daily life has grown as well.

It should be mentioned that quantum computers are easier in theory than in practice. Despite the vast energies invested in their construction only very small computational units have been built, and all have been hard-wired to some specific task. However, in recent years the progress has been fast and USA’s Security Agency, NSA, recently announced their intention to start development of a cryptography suite that will be safe against quantum attacks [NSA15]. Moreover, at the Post-Quantum Crypto conference 2016 the American Standardisation Institute, NIST, announced a call for proposals for quantum-safe asymmetric cryptographic algorithms [Moo16], which shall be made formal.
1 BACKGROUND

in the fall of 2016. The current plan is to deliver draft standards somewhere by 2022 or
2024. This call for proposals further highlights the importance of studying post-quantum
crypto.

1.3 Post-Quantum Cryptography

Since all cryptographic schemes based on integer factorisation (such as the RSA) or
discrete logarithms (such as elliptic curve cryptography) will be unusable come quantum
computers, completely new protocols based on entirely different ideas must be designed.
There are a few ideas, but they all suffer from a variety of problems: Some give slower
protocols, others significantly larger keys or messages, yet again others are less well-
studied, so there is less confidence in their security.

1.3.1 Hash-based Cryptography

Hash-based cryptography mainly encompasses signature schemes, and can be based on
a variety of hash functions – that is, functions that are easy to compute but difficult to
invert. The idea of such cryptosystems stems from the 1970’s in for example [Mer89]
and has the advantage that the security is easily estimated, but they have not been
widely used because they suffer from an interesting drawback: Each secret key can only
be used to sign a limited number of messages, then it becomes obsolete and has to be
switched out. These algorithms have now gained popularity again, however the concept
of expiring signature keys is not something that exists within the current cryptographic
infrastructure, which means that a transition to hash-based cryptography would be
difficult for practical reasons.

Very recently, a variant of hash-based signatures that avoids key expiry was designed
[HWOB+14], which brings new attention to these kinds of systems, though the proposed
scheme is still on the slower and more cumbersome side. What is more, standardisation
work on hash-based signatures is being carried out by the Crypto Forum Research Group
(CFRG).

1.3.2 Code-based Cryptography

Code-based cryptography rests on a foundation of error correcting codes. The public key
is a distorted version of the generator matrix for some error correcting code, while the
private key is the check matrix together with a record of which distortions were made.
McEliece introduced this as early as 1978 [McE78], and this specific system has yet to
be broken (though many attempted improvements have been) and is also reasonably
fast. However, it has suffered from horribly large keys, and has therefore never gained
popularity.

Recently development has been made in reduction of the key sizes by using matrices
with special structure, presented at the PQC 2016 conference [vMHG16]. An advantage
of this approach is that error correcting codes are already well studied in other areas of
computer science.

1.3.3 Multivariate Cryptography

Multivariate cryptography uses multivariate polynomials over finite fields. The inversion
of such polynomials is NP-hard which gives high credibility to the cryptographic security,
and the associated algorithms are fast. So far all attempts to create encryption schemes
in this setting have failed, but signature schemes can be constructed. The public key
is then some easily-inverted quadratic map on finite fields, somewhat perturbed so that the resulting map is difficult to invert. The private key consists of the inverse which can be constructed knowing the perturbations. The keys are quite large for fast schemes, while the signatures can be kept small.

1.3.4 Supersingular Elliptic Curve Isogeny

One of the most popular key exchange protocol today is the Diffie Hellman key exchange performed on an elliptic curve. A proposed substitute for this is the idea of supersingular elliptic curves [JDF11] and isogenies (rational maps that preserve the number of points) between them. The supersingularity of the elliptic curves breaks the quantum algorithm used against the ordinary elliptic curve schemes, and so this is claimed to provide quantum security. While still relatively slow, this idea would result in keys and messages of comparable sizes with those of today. Partly for that reason this is one of the more promising post-quantum ideas, but not one that we focus on in this report.

1.3.5 Lattice-based Cryptography

The hardness of lattice problems for cryptographic purposes was first examined in [Ajt96] by Miklós Ajtai. Since then, despite numerous efforts researchers have failed to find any effective algorithms, quantum or classical, to solve these problems. Also many proposed cryptographic systems have been built on this foundation, so that the required space and runtime have steadily been forced down. Therefore, lattice-based cryptography is also one of the more promising post-quantum candidates, and this is the focus of this report.

1.4 Report Outline

Section 2 contains a further discussion on quantum computers and how they differ from classical computers. Sections 3 and 4 provide an extensive overview of the foundation of modern lattice-based cryptography and the corresponding attacks.

We then turn our eyes to signature schemes and give a brief presentation of different lattice-based signatures in Section 5. Section 6 contains a thorough description of BLISS, one of the top contenders for a lattice-based signature system. Our presentation of BLISS contains no new results but is in many aspects clearer than the presentation in the original article [DDLL13].

Sections 7 and 8 contain the new contributions of this thesis. Section 7 analyses the BLISS system and provides formulas for object sizes and security. Here we also discuss improvements to encodings as well as criticise the BLISS security estimates. In Section 8 we present our major suggestions for improvements to BLISS, including a possibility to set system parameters with much higher flexibility.

1.5 Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{Z}, \mathbb{R}, \mathbb{C}$</td>
<td>The integers, the reals and the complex numbers</td>
</tr>
<tr>
<td>$\mathbb{R}^{n \times m}(\mathbb{C}^{n \times m}, \mathbb{Z}^{n \times m})$</td>
<td>The space of $n$ by $m$ matrices with real (complex, integer) entries</td>
</tr>
<tr>
<td>$| \cdot |, | \cdot |_\infty$</td>
<td>The Euclidean norm and the maximum norm</td>
</tr>
<tr>
<td>$D_\sigma^m (D_\sigma^m \mid_B)$</td>
<td>The discrete Gaussian distribution (tail-cut at $\pm B$).</td>
</tr>
</tbody>
</table>
Quantum Algorithms

2.1 Qubits and Quantum Gates

Quantum computers differ from classical computers in a number of ways, the main difference being that while $n$ classical bits together encode one of $2^n$ possible states, $n$ quantum bits (qubits) together encode any superposition of the $2^n$ possible states, with complex coefficients. That is, given two classical bits, the system can be in exactly one of the four states $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$. However, while two qubits can be measured to be in any of the four base states $|00angle$, $|01angle$, $|10angle$, $|11angle$ (using the standard ket-notation for states), as long as they are not measured they can be in any superposition $c_0|00angle + c_1|01angle + c_2|10angle + c_3|11angle$ of them, with $c_i \in \mathbb{C}$ under the restriction that $\sum_i |c_i|^2 = 1$. The state may be represented as a vector $(c_0, c_1, c_2, c_3)^T \in \mathbb{C}^4$, which must then lie on the boundary of the unit ball. Note that it is impossible to treat the two qubits as independent data, like it is possible to do with the two classical bits.

The probability that a measurement of the state $\mathbf{c} = (c_0, c_1, c_2, c_3)^T$ returns, say, the state $|01\rangle$ is $|c_2|^2$, which is why we require that the squares of the amplitudes sum to 1. However, in quantum mechanics there is no such thing as a pure measurement; each measurement inevitably affects the system by condensing it down to the state measured, so that if the measurement returns $|01\rangle$, the system is changed to be in

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Lambda, \mathcal{L}$</td>
<td>A lattice</td>
</tr>
<tr>
<td>$\mathcal{L}(B)$</td>
<td>The lattice with the columns of the matrix $B$ as basis</td>
</tr>
<tr>
<td>$\lambda_i(\mathcal{L})$</td>
<td>The smallest number such that there are $i$ linearly independent vectors in $\mathcal{L}$ with norms at most $\lambda_i$.</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Approximation factor in lattice problems</td>
</tr>
<tr>
<td>$\mathbf{b}_i$</td>
<td>A lattice basis vector</td>
</tr>
<tr>
<td>$\mathbf{b}_i^*$</td>
<td>A vector in the Gram-Schmidt orthogonalisation of a lattice basis</td>
</tr>
<tr>
<td>$\delta$</td>
<td>The root Hermite factor in lattice reduction attacks</td>
</tr>
<tr>
<td>$r, R, \gamma$</td>
<td>Parameters for the hybrid attack</td>
</tr>
<tr>
<td>$\mathbf{x}$</td>
<td>A generic lattice vector</td>
</tr>
<tr>
<td>$\mathcal{R}$</td>
<td>Any ring</td>
</tr>
<tr>
<td>$\mathcal{R}_d$</td>
<td>The quotient ring $\mathcal{R}/d\mathcal{R}$ with $d$ an integer</td>
</tr>
<tr>
<td>$\mathbb{Z}[x]/(f(x))$</td>
<td>The polynomial ring with coefficients in $\mathbb{Z}$ (quoted out by the polynomial $f(x)$)</td>
</tr>
<tr>
<td>$\Phi_N$</td>
<td>The $N$th cyclotomic polynomial</td>
</tr>
<tr>
<td>$p(x)$</td>
<td>A generic polynomial in some polynomial ring</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>The security level of a system (in bits)</td>
</tr>
<tr>
<td>$n$</td>
<td>The degree of a quotient polynomial</td>
</tr>
<tr>
<td>$m$</td>
<td>Lattice dimension</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Norm-limit for solutions to lattice problems</td>
</tr>
<tr>
<td>$q, \zeta$</td>
<td>Prime number $q$, and inverse $\zeta$ of $q - 2$ modulo $2q$</td>
</tr>
<tr>
<td>$p$</td>
<td>BLISS system parameter (not a prime)</td>
</tr>
<tr>
<td>$\mathbf{a}, \mathbf{s}$</td>
<td>A public and a secret key</td>
</tr>
<tr>
<td>$c$</td>
<td>A challenge</td>
</tr>
</tbody>
</table>

See also Table 3 for parameters specific to the BLISS system.
the state \((0, 1, 0, 0)\). Note that the amplitudes themselves cannot be measured other than indirectly, by repeating the same preprocessing and measurement several times and sampling the outcome frequencies – and this only provides the modulus of the amplitudes, not their arguments.

An operation on a quantum computer (a quantum gate) operating on \(n\) qubits consists of a unitary matrix \(U \in \mathbb{C}^{2^n \times 2^n}\), which replaces the state \(c\) by \(Uc\). This means that \(U\) is in a sense applied to all possible states at once, though the states may also affect one another. This is called \textit{quantum parallelisation} and is the key ingredient in Grover’s algorithm and one of the main reasons quantum computers are so powerful compared to their classical counterparts. Another reason is that the Fourier transform can be very easily implemented, simply by letting \(U\) shift one or many amplitudes by some phase \(e^{i\phi}\). This turns out to be a crucial ingredient in many quantum algorithms.

While a vast variety of operations can be encoded in this way quantum gates differ a lot from their classical counterparts, the main difference being that since all gates can be represented with unitary matrices, all gates can be inverted, so a quantum program can never erase information. An effect of this, described in [Sho97], is that while there is a standard way to adapt classical algorithms to a quantum computer this results in algorithms that consume a lot of memory. Luckily, more specialised adaptions have been found for most algorithms.

One way to erase information in a quantum computer is to perform a measurement of one or several of the qubits. The result is random with probabilities given by the state in question, and after the measurement more is known about the state (which has also changed). If the first qubit of the state \((c_1, c_2, c_3, c_4)\) is measured, a 0 will be read with probability \(|c_1|^2 + |c_2|^2\) and a 1 will be read with probability \(|c_3|^2 + |c_4|^2\). If, say, a 0 is read, the probability for a 1 will be set to 0, so that the qubits will be transferred to the state \((c_1, c_2, 0, 0)/(|c_1|^2 + |c_2|^2)\). This method can be used to kill off undesired errors, or produce data with some desired property.

\subsection{The Hidden Subgroup Problem}

The Hidden Subgroup Problem (HSP), described in for example [BBD08], is a generalisation of the problems solved in [Sho97]. Both Shor’s algorithms can be seen as solutions to different specialisations of the HSP. It can be formulated as follows:

\textbf{Problem 1} (The HSP). Given a group \(G\) and a function \(f\) on \(G\) which is constant and distinct on the cosets of some unknown subgroup \(H \leq G\), find a set of generators for \(H\).

The fast implementation of the Fourier transform in quantum computers is the key to solving the HSP when \(G\) is finite and abelian. An overview of the solution is presented in Algorithm 1. Let us take a closer look at the measurement in step 4. By performing a measurement on the second register, it is forced to be in one specified state, chosen at random – in this case uniformly, because that is how the state was initialised. Since the first register is linked to the second, the first register needs to tag along and is forced to be in a state that corresponds to the measurement of the second register. Since we know that \(f\) has the cosets of \(H\) as preimages, the first register is now in a superposition of all elements in some coset \(g_0 + H\) of \(H\), but we do not know which one. However, the Fourier transform can be applied to the group to extract information about the coset, and these measurements together can be used to construct a basis for \(H\).

In the case when \(G\) is finite and abelian, the Fourier transform of this coset representation is shifted from the Fourier transform of the subgroup \(H\) simply by a phase,
Algorithm 1: Solving the finite abelian HSP [BBD08]

Data: Group \( G \), function \( f : G \to S \) for some \( S \).
Result: A basis for \( H \leq G \).

1. for polynomially many times do
   2. Instantiate the qubits in a superposition of all possible group elements:
      \[
      \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g, 0\rangle
      \]
   3. Evaluate \( f \) on the superposition of all elements:
      \[
      \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g, 0\rangle \mapsto \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g, f(g)\rangle
      \]
   4. Measure the second register to condensate the state to some specific coset:
      \[
      \frac{1}{\sqrt{|G|}} \sum_{g \in G} |g, f(g)\rangle \mapsto \frac{1}{\sqrt{|H|}} \sum_{h \in H} |g_0 + h, f(g_0)\rangle
      \]
      for some \( g_0 \in G \).
   5. Compute the Fourier transform and measure.
6. end
7. Use the measurement results to compute \( H \) classically.

which does not affect measurements. Sufficiently many measurements from the Fourier transform \( \hat{H} \) of \( H \) allows a classical computation of \( H \). This is at the heart of both of Shor’s algorithms.

Many problems, amongst them cryptographical problems, can be reduced to the HSP, though often not for finite abelian groups. The HSP has been solved for many other instances, such as when \( G \) is \( \mathbb{R} \) or \( \mathbb{R}^n \) for a fixed dimension \( n \), while the HSP for more complicated groups like the symmetric group (which could be used to solve graph isomorphism) remains unsolved.

3 Foundation for Lattice Problems

In this section we introduce the lattice setting together with a selection of the supposedly hard problems that lattice cryptosystems are based on. More extensive descriptions can be found in for example [BBD08] and [Pei15].

3.1 Notation

A lattice \( \mathcal{L} \subset \mathbb{R}^m \) is the set \( \{ \sum_i x_i b_i : x_i \in \mathbb{Z} \} \) for some basis \( \{ b_i \}_{i=1}^n \subset \mathbb{R}^m \). Letting \( B \) be the matrix which has the basis vectors as columns, we see that \( \mathcal{L} \) is the image of \( \mathbb{Z}^n \) under \( B \) seen as a linear map \( \mathbb{R}^n \to \mathbb{R}^m \). It is also worth noticing that the basis of a lattice is not unique – rather, for any unitary \( U \in \mathbb{Z}^{n \times n} \), \( BU \) is also a basis for the same lattice (actually all bases \( B' \) for \( \mathcal{L} \) can be written as \( BU \) for some unitary matrix \( U \)).

This simple property is important for the cryptographic implementations, since some
bases are easier to handle than others. An example of a lattice with \( n = m = 2 \) can be found in Figure 1, along with a convenient and an inconvenient basis.

Each lattice \( \mathcal{L} \) has a dual lattice, denoted \( \mathcal{L}^* = \{ \mathbf{w} \in \mathbb{R}^m \mid \langle \mathbf{w}, \mathbf{x} \rangle \in \mathbb{Z} \forall \mathbf{x} \in \mathcal{L} \} \) (we leave it as an exercise to the reader to verify that this is indeed a lattice).

Most modern work is done in so called \( q \)-ary lattices, with \( q \) usually a prime. These are lattices such that \( q\mathbb{Z}^m \subseteq \mathcal{L} \subseteq \mathbb{Z}^m \) where \( q\mathbb{Z}^m \) is the scaled lattice \( \{ qx \mid x \in \mathbb{Z}^m \} \) (so the dimension of \( \mathcal{L} \) equals that of the ambient space, which is the usual setting). In this case, for any \( x \in \mathbb{Z}^m \) its membership of \( \mathcal{L} \) is determined solely by its remainder modulo \( q \). We shall write \( x \mod q \) not only to denote the equivalence class of \( x \) modulo \( q \), but also to denote one specific member of this class, usually the absolute remainder.

We also define the following measures on \( \mathcal{L} \): Given a basis matrix \( \mathbf{B} \) for an \( m \)-dimensional lattice in \( \mathbb{R}^m \), \( \det \mathcal{L} \equiv \det \mathbf{B} \). This can be interpreted as the reciprocal density of lattice points in \( \mathbb{R}^m \). Also, for \( i = 1, \ldots, m \), we define \( \lambda_i(\mathcal{L}) \) as the smallest \( \lambda \) such that \( \mathcal{L} \) contains at least \( i \) independent vectors of Euclidean norm at most \( \lambda \). Specifically \( \lambda_1(\mathcal{L}) \) is the length of the shortest non-zero vector in \( \mathcal{L} \).

In the setting of \( q \)-ary lattices another way to specify a lattice is common: Given a matrix \( \mathbf{A} \in \mathbb{Z}_q^{n \times m} \), the set \( \Lambda^\perp(\mathbf{A}) = \{ \mathbf{y} \in \mathbb{Z}^m \mid \mathbf{A}\mathbf{y} = 0 \mod q \} \) is a \( q \)-ary lattice in \( \mathbb{Z}^m \). This is a setting that appears in many cryptographic systems. If \( m > n \) (often \( m = 2n \)) and \( \mathbf{A} \mod q \) is not extremely ill-conditioned, it will have full rank which means that the kernel of \( \mathbf{A} \), which is \( \Lambda^\perp(\mathbf{A}) \), will have \( q^{m-n} \) lattice points modulo \( q \). Since there are \( q^m \) residue classes modulo \( q \), the reciprocal density of lattice points will be \( q^m/q^{m-n} = q^n \), so \( \det(\Lambda^\perp(\mathbf{A})) = q^n \). If on the other hand \( m \leq n \), for most \( \mathbf{A} \) its null space is the zero space. This means that \( \Lambda^\perp(\mathbf{A}) = q\mathbb{Z}^m \), so \( \det(\Lambda^\perp(\mathbf{A})) = q^m \). This result will be important for security estimates later.

### 3.2 Hard Problems

In [Ajt96] the first hint of a structure of provably hard problems in the lattice setting was given. (Interestingly, the first lattice-based cryptosystem NTRU was then already under development, which we shall talk more about in Section 5.) In this paper Ajtai presents a relation between some average case problems and other worst case problems, a feature which is part of lattices’ popularity: He showed that for a specific class of lattice problems (the SIS problem, of which we discuss a more modern version in Section 3.3 and which may be used as the foundation for cryptographic algorithms), any algorithm that breaks this problem on average is also able to solve some of the underlying, assumedly hard lattice problems in the worst case. Such a structure is powerful because if the SIS instance is randomly instantiated it may still rest on the worst case hardness of the general problem.

Most of the underlying, assumedly hard lattice problems come in an approximational flavour as well as a standard one. We here present only the approximation variants since
those are the most commonly used. Throughout, \( \gamma(m) \) will denote an approximation factor dependent on the dimension \( m \) of the lattice. With \( \gamma(m) \equiv 1 \) the approximative problem becomes the standard variant. We shall often omit the \( \gamma \) from the notation even though we talk about the approximate problems since these are prevalent in the discussion, and we shall write explicitly when we mean the exact versions.

**Problem 2** (Shortest Vector Problem, SVP\(_{\gamma} \)). Given an arbitrary basis \( B \) for a lattice \( L \) of dimension \( m \), find a nonzero \( x \in L \) with \( \|x\| \leq \gamma(m) \lambda_1(L) \).

A variant of this problem is the Unique Shortest Vector Problem:

**Problem 3** (Unique Shortest Vector Problem, uSVP\(_{\gamma} \)). Given an arbitrary basis \( B \) of the lattice \( L \) of dimension \( m \), along with the promise that \( \lambda_1(L) \gamma(m) \leq \lambda_2(L) \), find a shortest non-zero vector in \( L \).

While these problems are easy to understand, there are more security proofs ([Pei15]) based on the following two problems:

**Problem 4** (Shortest Independent Vectors Problem, SIVP\(_{\gamma} \)). Given a basis \( B \) of an \( m \)-dimensional lattice \( L \subset \mathbb{R}^m \), compute a set \( \{s_i\}_{i=1}^m \subset L \) of linearly independent vectors with \( \|s_i\| \leq \gamma(m) \lambda_m(L) \).

**Problem 5** (Decisional SVP, GapSVP\(_{\gamma} \)). Given an arbitrary basis \( B \) of a lattice \( L \), with either \( \lambda_1(L) \leq 1 \) or \( \lambda_1(L) > \gamma(m) \), determine which is the case.

Clearly an efficient algorithm to solve SVP\(_{\gamma} \) can be used to create an effective algorithm to solve the GapSVP\(_{\gamma} \): Use the algorithm that solves the SVP to compute a short non-zero vector \( x \in L \). If \( \lambda_1(L) \leq 1 \) we will have \( \|x\| \leq \gamma(n) \), while if \( \lambda_1(L) > \gamma(n) \) then of course \( \|x\| > \gamma(n) \). This way it is always possible to tell which is the case, so the SVP is a more difficult problem than the GapSVP.

[Pei15] also defines the following problem, which is a variant of the more commonly described Closest Vector Problem but with a formulation that corresponds to how it is often used:

**Problem 6** (Bounded Distance Decoding Problem, BDD\(_{\gamma} \)). Given a basis \( B \) of an \( m \)-dimensional lattice \( L \), and a point \( y \in \mathbb{R}^m \) which is guaranteed to be within distance less than \( d = \lambda_1(L)/2 \gamma(m) \) of some (unknown) lattice point, find the unique \( x \in L \) such that \( \|x - y\| < d \).

The question is of course, how hard are Problems 2-6? As is typically the case, hardness cannot be mathematically proven (except when derived from the supposed hardness of some other problem), but to this date no effective algorithms have been found to solve these for polynomial approximation factors \( \gamma \). Several algorithms exist for when \( \gamma \) is exponential (a few years ago, the record that could be solved in practice for SVP was \( \gamma(m) \approx 1.012^m \), [BBD08]). Algorithms for polynomial \( \gamma \) all use exponential time and often also exponential space. We shall talk more about such attacks in Section 4, but for now it suffices to note that despite the mathematical value of lattices not only in this application, no efficient algorithm to solve these problems has emerged. The same is true in the quantum case: Despite the fact that lattice duality which is tightly related to the Fourier transform should be easily accessible on a quantum computer, all attempts to solve these problems have so far met with failure. The best that has been achieved is a mild improvement of factors in complexities over the best classical algorithms [Pei15]. (Coincidentally, in his famous paper Shor [Sho97] suggests that the
next target for quantum algorithms ought to be lattice problems, since they should be easily accessible. He turned out to be wrong in this.)

This leads to the following conjecture:

**Conjecture 3.1.** The SVP\(\gamma\), uSVP\(\gamma\), GapSVP\(\gamma\), SIVP\(\gamma\) and BDD\(\gamma\) for polynomial approximation factors \(\gamma(m)\) cannot be solved in polynomial time, neither on classical nor on quantum computers.

In [LM09] it is proven that there is a classical reduction that makes the uSVP, GapSVP and BDD problems equivalent, up to small polynomial factors in \(\gamma\). The SVP and SIVP problems, however, are still distinct.

### 3.3 Ring Problems

There are two main problem formulations that are used in modern lattice-based cryptosystems, namely the Short Integer Solution-problem (SIS), presented already in [Ajt96], and the Learning With Errors-problem (LWE), presented by Regev in [Reg05]. While SIS-based systems are generally more efficient, SIS cannot be used for encryption, which LWE can.

The main power of these constructions is that both SIS and LWE benefit from a connection between average hardness and worst case hardness, in the sense that any algorithm that solves SIS or LWE on the average can also be used to solve any instance of some selection of the hard problems in Section 3.2. This is a very strong security property and the proof of this for SIS is the main content of Ajtai’s paper [Ajt96].

However, for performance reasons most recent cryptosystems are based not on these problems, but on variants of them that live in rings. We shall only talk about these in this report, but first we need to go over the ring setting.

Rings are denoted \(\mathcal{R}\) and are usually a polynomial ring \(\mathbb{Z}[x]/f(x)\) for some polynomial \(f\) with \(n = \deg f\). For some (often prime) modulus \(q\), we have \(\mathcal{R}_q \triangleq \mathcal{R}/q\mathcal{R} = \mathbb{Z}_q[x]/f(x)\). In order to talk about “short” objects \(\mathcal{R}\) and \(\mathcal{R}_q\) need norms, which they can be endowed with in different ways. A naïve way is to embed \(a \in \mathcal{R}\) into \(\mathbb{C}^n\) using the coefficient embedding, that is, putting the coefficients of \(a\) into a vector in \(\mathbb{Z}^n\), and let \(\|a\|\) be the norm of that vector. However, while this embedding is additive it behaves very badly on products. Instead some cryptosystems use the canonical embedding \(\sigma:\mathcal{R}\to \mathbb{C}^n\), defined by

\[
\sigma(a)_i = a(\alpha_i)
\]

where \(\alpha_i, i = 1, \ldots, n\) are the complex zeroes of \(f\) counted with multiplicity. This has the nice property that it is both additive and multiplicative for elements in \(\mathcal{R}\), and also canonical in the sense that two different representatives of \(a \in \mathcal{R}\) yield the same embedding. Then \(\|a\|\) is taken to be \(\|\sigma(a)\|\). Using the canonical embedding in security analysis provides tighter bounds and thus better performance, but is computationally more complex and is not used in any system we study in this report.

A vector \(a \in \mathcal{R}^m\) has norm defined by \(\|a\|^2 \triangleq \sum_i \|a_i\|^2\), thus depending on the norm that was chosen for \(\mathcal{R}\).

With a fixed choice of embedding it is time to define an **ideal lattice**:

**Definition 3.1** (Ideal lattice). An ideal lattice is a lattice corresponding to an ideal in \(\mathcal{R}\), under some choice of embedding.

The ideal lattices often have extra structure compared to general lattices, though what this structure is depends on \(\mathcal{R}\). For example if \(f(x) = x^n - 1\) and the chosen
embedding is the coefficient embedding, all ideal lattices are cyclic lattices (i.e. closed under cyclic rotations of the coordinates) since all ideals are closed under multiplication by $x$ which corresponds to a rotation of the coefficients in a polynomial. For the two problems to be presented, the security reductions do not go all the way to the conjectured hard problems defined in Section 3.2, but rather to the same problems but only on ideal lattices. Since the security depends on worst case hardness each reduction of the set of applicable lattices can only make the problem easier, though how easy it gets depends on the choice of $\mathcal{R}$ and embedding.

We will also need to talk about cyclotomic polynomials:

**Definition 3.2** (Cyclotomic Polynomials). For each positive integer $N$, the $N$th cyclotomic polynomial $\Phi_N$ is the unique irreducible polynomial in $\mathbb{Z}[x]$ which divides $x^N - 1$ but not $x^k - 1$ for any $k < N$.

Cyclotomic polynomials are irreducible not only over $\mathbb{Z}$ but also over $\mathbb{Q}$. It can also easily be proven that

$$x^N - 1 = \prod_{d|N} \Phi_N(x)$$

We shall talk more about cyclotomic polynomials shortly.

Now let us introduce the $\mathcal{R}$-SIS problem, defined first in [Mic07] (a preliminary version published in 2002).

**Problem 7** ($\mathcal{R}$-SIS$_{q,\beta,m}$). Given $m$ elements $a_i \in \mathcal{R}_q$ chosen uniformly at random, find a nonzero $z \in \mathcal{R}^m$ with $\|x\| \leq \beta$ such that

$$f_a(x) \triangleq \sum_i a_ix_i = 0 \in \mathcal{R}_q$$

If the function $f_a$ defined above is regarded as a hash function we see that $\mathcal{R}$-SIS being hard corresponds to $f_a$ being collision-resistant, that is, it is hard to find two different inputs $x, x'$ such that $f_a(x) = f_a(x')$. This is typically not true if $f(x)$ is not irreducible, since then $\mathcal{R}$ is not an integral domain and zero divisors can be exploited to create collisions. However, for smart choices of $\mathcal{R}$ the situation is promising:

**Fact 3.2** ([PR06]). If $\mathcal{R} = \mathbb{Z}[x]/f(x)$ with $f$ a cyclotomic polynomial, assuming that SVP$_\gamma$ for $\gamma = \beta\text{poly}(nm)$ is hard in the worst case for ideal lattices in $\mathcal{R}$, $f_a$ (as defined in Problem 7) is collision resistant (that is, $\mathcal{R}$-SIS$_{q,\beta,m}$ is hard).

Historically $f(x) = x^n - 1$ has been used, but this is most certainly not irreducible and was proven in [PR06] to be an insecure choice. The most popular choice of cyclotomic polynomial has since then generally been $f(x) = x^n + 1$ with $n$ a power of two, since this allows for fast computations (though in [LPR13a] operations were suggested that would make other choices of cyclotomics feasible as well). However, note that while cyclotomic polynomials are irreducible over $\mathbb{Z}$ and even over $\mathbb{Q}$, they are typically not irreducible over $\mathbb{Z}_q$ for $q$ prime, which means that $\mathcal{R}$ chosen this way is typically not an integral domain and there are sometimes plenty of zero divisors. These do not pose a threat, though, since the zero divisors are not small in norm and can therefore not be used to find small solutions to the SIS problem. Indeed, often $q$ is chosen so that the cyclotomic in question factors down completely into linear terms, because this means that there is a primitive root of unity which can be used to speed up calculations.

The ring variant of LWE was introduced in [LPR13b]. To describe this we first need a definition:
Definition 3.3 (The \(\mathcal{R}\)-LWE distribution). Given some \(s \in \mathcal{R}_q\) and some “error” distribution \(\chi\) over \(\mathcal{R}_q\), the \(\mathcal{R}\)-LWE distribution \(A_{s,\chi}\) is sampled by taking \(a \in \mathcal{R}_q\) uniformly random, sampling \(e\) from \(\chi\) and returning \((a, sa + e \mod q)\).

(On a side note, this formulation is actually somewhat simplified, since both the secret and the product \(as + e\) actually do not live in \(\mathcal{R}\) but rather in a fractional ideal of \(\mathcal{R}\), that is, an \(\mathcal{R}\)-module in \(\mathcal{R}\)’s field of fractions which can be scaled to lie inside \(\mathcal{R}\). Moreover this ideal is required to be dual to \(\mathcal{R}\) for the security proofs to go through. However, since fractional ideals can be scaled back into the ring, it is possible to return to the formulation above and we do not need to care about this here.)

The distribution \(\chi\) is usually said to have a “width” \(\alpha q\), or a relative error rate \(\alpha < 1\), though what this really means depends on the choice of \(\chi\). This choice turns out to be pretty delicate in order to get the security proofs through, but generally a discretized Gaussian of some sort is used. The resulting distribution can then be used to formulate the following problem:

Problem 8. Decision-\(\mathcal{R}\)-LWE\(_{q,\chi,m}\) Given \(m\) elements \((a_i, b_i) \in \mathcal{R}_q \times \mathcal{R}_q\) sampled independently either from \(A_{s,\chi}\) with \(s\) uniformly random (and fixed for all samples), or from the uniform distribution, distinguish which is the case.

This is the decision version of the problem. It’s sister problem, the search version, consists of recovering the secret \(s\) given \(m\) samples from \(A_{s,\chi}\). The search version is of course at least as hard as the decision version in the following sense: Any algorithm that efficiently solves the search-\(\mathcal{R}\)-LWE problem can be applied to the data given in the decision problem. If the algorithm outputs a sensible answer the data probably comes from the \(A_{s,\chi}\)-distribution, otherwise it is probably uniform. In [LPR13b] a reduction from the search problem to the decision problem is provided, so the two problems are essentially equivalent. That result formed part of their proof of the following theorem:

Fact 3.3 ([LPR13b]). For any \(m = \text{poly}(n)\), any ring \(\mathcal{R} = \mathbb{Z}[x]/\Phi_N(x)\) of degree \(n\), and appropriate choices of modulus \(q\) and distribution \(\chi\) with relative error \(\alpha < 1\), solving the \(\mathcal{R}\)-LWE\(_{q,\chi,m}\) problem is at least as hard as quantumly solving the SVP\(_{\gamma}\) on arbitrary ideal lattices in \(\mathcal{R}\), for some \(\gamma = \text{poly}(n)/\alpha\).

Two things are worth noticing here: First, that the LWE also has this strong average case to worst case connection, and second the occurrence of the word quantum in the statement. This is because the reduction from the SVP to the \(\mathcal{R}\)-LWE is a quantum algorithm, so any algorithm for solving the \(\mathcal{R}\)-LWE, classical or quantum, gives a quantum algorithm for solving the corresponding SVP instance, but not a classical one. This does not constitute a real problem since the SVP is conjectured to be hard also for quantum computers, but the question of finding a classical reduction is still open.

It shall also be mentioned that the \(\mathcal{R}\)-LWE problem comes in another variant called the normal form, where the secret is sampled form \(\chi\) rather than uniformly. This version of the problem is at least as hard as the uniform version, since the uniform version of the problem can be reduced to searching for part of the error vector instead of the secret.

3.4 Heuristics

Two commonly used heuristics are the Gaussian Heuristic and the Geometric Series Assumption.

The Gaussian Heuristic is an estimate of \(\lambda_1(\mathcal{L})\): The density of points is \(1/\det(\mathcal{L})\), so within a centered ball of radius \(r\) we expect there to be \(v_n r^n / \det(\mathcal{L})\) lattice points, where
$v_n$ is the volume of the $n$-dimensional unit ball. In particular this gives an estimate for $\lambda_1$ through $v_n \lambda_1^2 / \det(L) \approx 1$, that is, $\lambda_1 = (\det(L)/v_n)^{1/n}$. Since the volume of the unit ball is

$$v_n \approx \frac{1}{\sqrt{n\pi}} \left( \frac{2\pi e}{n} \right)^{n/2}$$

the Gaussian Heuristic gives

$$\lambda_1 \approx \det(L)^{1/n} \sqrt{\frac{n}{2\pi e}}$$

This is used not only to approximate $\lambda_1$, because we shall see that often $\lambda_1$ is known to be much smaller, but also to approximate the next few minima $\lambda_2$, $\lambda_3$ and so on. For high dimensions, the Gaussian heuristic gives the same value for these.

The Geometric Series Assumption, introduced in [Sch03], is an assumption on the basis for a lattice. If $\{b^*_i\}$ is the un-normalised Gram-Schmidt orthogonalisation of the basis $\{b_i\}$, the Geometric Series Assumption states that there is some $c \in (0, 1]$ such that

$$\|b^*_i\| = c^{i-1}\|b^*_1\|$$

This is not completely uncalled for, since the Gram-Schmidt vectors should be expected to decrease in length because for each step, there are more vectors to reduce by. However, it is generally unclear how well lattice bases adhere to this assumption or how to measure the quality of this adherence. The assumption is, however, very useful and is employed often.

### 3.5 Relation to the Hidden Subgroup Problem

In [Reg04] it is shown that lattice problems share a connection with the HSP, namely the HSP for the dihedral group (the group of symmetries of the regular $2n$-gon, denoted $D_n$). The dihedral group is not abelian which means that it does not fit within the standard framework for the HSP, but it nonetheless has a fairly simple structure and therefore does not give the impression of being unmanageable.

Regev shows that any algorithm that solves the HSP on the dihedral group $D_n$ can be extended to a quantum algorithm to solve the uSVP. As stated earlier in this section, the uSVP is classically equivalent to both the GapSVP and the BDD, but it is not clear how it is related to the SVP: It is harder in the sense that the short vector produced must be optimal, but easier in the sense that the uSVP comes with a promised gap between $\lambda_1$ and $\lambda_2$, something which is used in Regev’s paper to generate quantum superpositions of vectors that can be known to differ exactly by the shortest lattice vector since all other vectors are guaranteed to be too long to fit. This is done by measuring the vectors with low precision, so that they are condensed to a superposition of states that are relatively close to one another. Since the SVP, and not the uSVP, is at the heart of both SIS and LWE it is therefore not clear how a solution to the dihedral HSP would affect lattice cryptography.

Despite the fact that the dihedral group seems very simple, the best algorithms for the dihedral HSP as of yet, like the Kuperberg sieve [Kup05], do not offer any speedup to known attacks that go straight for the lattice instead. However, the question remains if someone might one day unlock some smart algorithm which could then render lattice cryptography if not useless, then at least make the scene a lot less secure.
4 Attacks and Security

In this section we survey the current situation when it comes to attacks on lattice problems. There are several different kinds of attacks, and of course none of them are known to be optimal in any sense, but they still give a good idea on the security of lattice-based cryptosystems.

4.1 Exact Algorithms

We first review some of the algorithms for finding the actual shortest vector of a lattice, and not just any sufficiently short vector. While these algorithms are expensive, running in either superexponential time or exponential time and memory, they are important as building blocks for the more practical algorithms we shall discuss later.

4.1.1 Sieving

There are different versions of sieving algorithms for finding the shortest lattice vector. The idea is to keep a list $L$ of vectors that are considered as short as they can get this far, and a stack $S$ of vectors that are to be processed. Both are initially empty, and for each round, a vector $v$ is popped from the stack (or randomised, if the stack is empty). Then $v$ is reduced modulo each vector in $L$, and vice versa. In the end, all vectors in $L$ that have been changed are moved to $S$, and if $v$ has been changed it follows suit, else it is added to $L$.

Note that the size of $S$ and $L$ is not limited to the dimension of the lattice. Instead, exponential memory is needed, as well as exponential time. The exponential memory requirement in particular makes sieving algorithms very cumbersome to use in practice.

Among the latest developments in sieving is [BDGL16]. A heuristic assessment of the time and space complexities is $2^{0.202m+o(m)}$ in dimension $n$, and experiments for $n \leq 80$ seem to require $2^{0.387m-15}$ operations and a similar number of vectors to be stored. (A vector has size roughly $m \log_2 q$ bits in a $q$-ary lattice, though short vectors can be more efficiently encoded.) Moreover, there are proposed improvements to be used in the case of ideal lattices [BL15], offering an experimental speedup of roughly $2^4$ in their algorithm. The specialised algorithm is built for the case when the polynomial $f$ in $\mathcal{R} = \mathbb{Z}[x]/f(x)$ is either $x^n - 1$ or $x^n + 1$, the latter case for $n$ a power of two, and uses the fact that multiplication with powers of $x$ in these lattices are just rotations of the coefficients, perhaps with switched signs. It seems possible that similar speed-ups could be achieved for other polynomial rings as well, but the attempt has not been made since no other rings are commonly used.

The improvement of sieving is ongoing, and new tricks or new combinations of old tricks may well improve running times within the near future.

One drawback of sieving algorithms, after their high memory demand, is that their provable running time is significantly worse than the heuristic running time or the time measured in experiments.

4.1.2 Other Exponential Memory Algorithms

A bunch of other algorithms exist for solving the exact SVP, with behaviour similar to that of sieving: Exponential complexity in both time and memory. Some of them have better provable asymptotic behaviour, but none perform better in actual experiments.
Such algorithms include computing the Voronoi cell of the lattice (the set of points that are closer to the origin than to any other lattice point) and using that to find short lattice vectors [MV10]. Both finding the Voronoi cell and the post-processing step have exponential time and space complexity.

Another algorithm is based on the generation of a large number of vectors from the discrete Gaussian distribution on the lattice, and the computation of averages of these. This also runs in exponential time and space, with fairly low provable constants. It was introduced in [ADRSD15] and is still a very young algorithm.

4.1.3 Enumeration

Enumeration is one of the oldest, and still in practice one of the most efficient algorithms for solving the SVP. The algorithm idea is basically just a brute force attack, which has very bad complexity. However, some preprocessing can speed up the search to make it more feasible, for example by using an input basis that is LLL-reduced (see Section 4.2).

A variant of enumeration using stronger preprocessing is due to Kannan [Kan83] and uses $n^{O(n)}$ time and $\text{poly}(n)$ memory. The hidden constants were, however, for many years too high for this to have any practical influence. This was because while the preprocessing asymptotically only performed $n^{O(n)}$ operations, in practice for the dimensions that are relevant it induced a lot of overhead, but in [MW15] an algorithm was provided that only performs linearly many operations in the preprocessing, which makes this idea feasible.

4.2 Lattice Reduction Algorithms

Lattice reduction algorithms aim to take a given basis of a lattice and transform it into another basis for the same lattice but with shorter and more orthogonal vectors. This way a short lattice vector can be recovered. These algorithms are generally not designed to be exact, but rather to extract a short vector to within some approximation factor – in other words, to solve some approximate hard lattice problem.

The overall approach is the following: Given a basis $\{b_i\}$ of $L$, let $\{b_i^*\}$ be its un-normalised Gram-Schmidt orthogonalisation. This set of vectors is not a basis for the same lattice since non-integer scalars are used in the orthogonalisation, but as long as the vectors are not normalised at least the determinant remains unchanged, and in the orthogonal basis the determinant is simply the product of the norms of the vectors. Importantly the first vector of the lattice basis equals the first vector of the Gram-Schmidt basis. This first basis vector will typically be output as a short vector.

Recall the Geometric Series Assumption: That there is some $c \in (0, 1]$ such that $\|b_i^*\| = c^{i-1}\|b_1^*\|$. Since the determinant is unchanged, we know

$$\det(L) = \prod_i \|b_i^*\| = \prod_i c^{i-1}\|b_1^*\| = \|b_1\|^m c^{m(m-1)/2}$$

which means that the output vector will have length $\|b_1\| \approx \det(L)^{1/m} \sqrt{1/c^m}$. Clearly $\sqrt{1/c}$ is what dictates the quality of the output, so we call this the root Hermite factor, denoted $\delta$. The Geometric Series Assumption is then rewritten as

$$\|b_i^*\| = \|b_1^*\|\delta^{2(i-1)}$$

and the smaller $\delta$, the better the quality of the output.
For completeness we also need to relate this to \( \lambda_1(L) \). For this we note that we can reverse the Gram-Schmidt method: There are coefficients \( c_{ij} \) for all \( i < j \) such that for all \( j \), \( b_j = b_j^* + \sum_{i=1}^{j-1} c_{ij} b_i^* \) where all terms are orthogonal. This means that for any \( x \in L \), with \( x = \sum_i a_i b_i \), if \( j \) is the last index such that \( a_j \neq 0 \) then

\[
\|x\|^2 = \sum_{i=1}^j a_i b_i^* + \sum_{k=i+1}^j a_k c_{ki} b_i^* \geq \|a_j b_j^*\|^2 \geq \|b_j\|
\]

Thus specifically \( \lambda_1(L) \geq \min_i \|b_i^*\| \), so for all \( i \),

\[
\lambda_1(L) \geq \|b_i^*\| = \|b_i^*\| \delta^{-2(i-1)} \geq \|b_i\| \delta^{-2(m-1)}
\]

which means that

\[
\|b_i\| \leq \lambda_1(L) \delta^{2(m-1)}
\]

so the SVP_{\delta m} has been solved.

Different lattice reduction algorithms achieve different \( \delta \), generally in polynomial time. Values of \( \delta \) are around 1.01 or slightly higher for algorithms that are possible to run with today’s computation capacity.

### 4.2.1 The LLL Algorithm

The first lattice reduction algorithm is known as LLL (from the authors Lenstra, Lenstra and Lovasz) and stems from before the time of lattice cryptography [LLL82]. Its original purpose was to factor polynomials in \( \mathbb{Q}[x] \), which it does in polynomial time. However, along the way it recovers a lattice vector of length at most \( (2/\sqrt{3})^m \) times that of the shortest vector in some lattice \( L \), that is, it can solve the SVP_{\delta m} in time \( O(n^6 \log_3 B) \) where \( B \) is the length of the longest input basis vector.

The goal of the LLL algorithm is to find a basis \( b_i \) for the lattice which is such that the Gram-Schmidt vectors \( b_i^* \) are “not too decreasing” in size. This is formalised as there being a constant \( k \) such that

\[
\|b_{i+1}^*\|^2 \geq \frac{\|b_i^*\|^2}{k}
\]

for all \( i \). Notice that the equality in the Geometric Series Assumption has been replaced by an inequality. Recall that \( \lambda_1(L) \geq \min_i \|b_i^*\| \), so we have for all \( i \),

\[
\lambda_1(L) \geq \|b_i^*\| \geq \frac{\|b_i^*\|^2}{k^{(i-1)/2}} \geq \frac{\|b_1\|^2}{k^{(m-1)/2}}
\]

which means that \( \|b_1\| \leq k^{(m-1)/2} \lambda_1(L) \) and a short vector has been found within the approximation factor \( O(\sqrt{k}^m) \). Therefore clearly the smaller \( k \) gets the better, though the algorithm performs in polynomial time only for \( k \in (1, 4) \). Usually, \( k = \frac{4}{3} \) is used, which gives the specified approximation factor.

The algorithm works by successively computing a discretized Gram-Schmidt orthogonalisation (simply by rounding all projection coefficients to the nearest integer) and then reordering the vectors if required.

The LLL algorithm has of course been extensively studied and tweaked for improvement. In the next section we present such a modern variant, inspired by the LLL. Many other attacks also use the LLL as a first preprocessing step.
4.2.2 BKZ and its Variants

In [Sch87], the Block Korkin-Zolotarev algorithm for finding short lattice vectors was introduced. This algorithm is parametrised by a block size $b$ and consists of a combination of high-level reduction such as the LLL algorithm and some algorithm for the exact SVP on blocks of $b$ consecutive lattice basis vectors. The runtime is polynomial in the lattice dimension $m$, and linear in the runtime of the chosen exact algorithm on lattices of dimension $b$—but these are generally expensive.

The BKZ algorithm goes through the basis, first performing the LLL-reduction on the entire basis and then running the exact SVP algorithm on $b_1, \ldots, b_b$ together with a reordering that puts a short vector in the place of $b_1$. Then it proceeds to run the SVP algorithm on the new basis vectors $b_2, \ldots, b_{b+1}$ and reordering, and so on. The final $b-1$ blocks consist of fewer than $b$ vectors since the basis runs out. Each round therefore performs one LLL reduction and $m$ SVP algorithms, and the rounds are repeated until no more changes are made (or terminated early when the computed vectors ought to be short enough).

The cost of increasing the block size is rewarded through a better root hermite factor which we subscript $\delta_b$ to indicate its dependence on the block size. Assuming the Gaussian Heuristic and the Geometric Series Assumption from Section 3.4, it can be proven [APS15] that

$$\lim_{m \to \infty} \delta_b \approx \left( \frac{b}{2\pi e} \right)^{1/b} \left( \frac{\pi b}{b-1} \right)^{1/2}$$

and the convergence seems to be fast enough so that for large $m$ (typically those we find in cryptosystems) the limit value can be used for $\delta_b$. Since a larger $b$ means more time spent in the expensive exact SVP algorithm it is desirable to keep $b$ as small as possible given a required output vector length, that is, a required root hermite factor.

While for a given $\delta_b$ it is easy to find the corresponding block size $b$ numerically there is no simple closed-form formula, and this makes it difficult to derive general security estimates. In [APS15] it is suggested to use $\delta_b \approx 2^{1/b}$ as an approximation, which for most smaller $\delta_b$ (at least $\delta_b \leq 1.0113$) works out in the attacker’s favour, that is, it leads to more pessimistic security estimates. Classically $\delta_b \approx b^{1/2b}$ has been used, which gives too optimistic values but a more correct asymptotic behaviour.

The runtime depends on the number of rounds $\rho$ required and also on $m$, which is the number of SVP instances to solve per round, and the cost $t_b$ of solving the exact $b$-dimensional SVP problem. In [APS15] it is suggested to take $\rho = (m^2 \log_2 m)/b^2$. This would give a running time of

$$\rho m t_b = \frac{m^3}{b^2} \log_2(m) t_b$$

Many improvements, small and big, have been introduced over the years. There are also variants in the choice of exact SVP algorithm. Many of these improvements can be found in [CN11] and are referred to as BKZ 2.0. They include terminating the algorithm early, preprocessing of the data and different pruning techniques, amongst others. The early termination decreases $\rho$ significantly, so that the above time estimate is no longer relevant. Instead the authors provide a simulation algorithm that, given the norms of the Gram-Schmidt vectors for the initial basis for the lattice as well as a chosen block size and number of rounds estimates the quality of the output assuming that the input basis behaves randomly. This algorithm can then be used to find the proper number of rounds as well as root hermite factor, and knowing the time for the exact sub-algorithm this gives the running time. The paper claims to reach $b^{O(b)}$ time complexity.
4 ATTACKS AND SECURITY

The paper [CN11] has been updated in a full version, [CN12]. Since the time required to run BKZ 2.0 is difficult to compute theoretically the paper provides a table of upper bounds on the number of nodes required to visit in each enumeration sub-algorithm, depending on the block size. This upper bound has typically been used for performance estimates. Interestingly, the authors did some additional analysis between the publication of the abbreviated paper and the full version which allowed them to decrease this upper bound significantly. This can be seen from a comparison between [CN11, Table 3] and [CN12, Table 4].

The BKZ 2.0 software has not been published, which means that the provided measurements have not been possible to verify independently. It is believed that the exact SVP algorithm runs in $O(b^{O(b)})$ time, and the tabulated data in [CN12] is what must be adhered to. Using sieving for the SVP step could in theory be better, but in practice not possible due to the large memory requirements.

Independently, an improvement to BKZ in which block size is increased between rounds has been introduced under the name Progressive BKZ. A recent paper [AWHT16] claims an experimental speedup by a factor of 50 compared to BKZ 2.0 for lower dimensions (up to 130), and estimate similar behaviour for higher dimensions.

4.3 The Hybrid Attack

The hybrid attack was introduced in [HG07] and the idea is the following: Given a basis $B$ for $L$, first transform it to be in triangular form so that each basis vector $b_i$, $i = 1, \ldots, m$ is non-zero only in the first $i$ coordinates. Then choose an index $R$ and run an exact algorithm on the $R$ last basis vector, looking only at the $R$-dimensional lattice defined by the $R$ last coordinates. The resulting vector $(0, \ldots, 0, x_{m-R+1}, \ldots, x_m)$ exactly equals a shortest vector $x$ in $L$ in the $R$ last coordinates which means that its distance from a point in $L$ is at most $\| (x_1, x_2, \ldots, x_{m-R}, 0, \ldots, 0) \|$. This is then an instance of the BDD-problem (Problem 6) with $d = \| (x_1, x_2, \ldots, x_{m-R}, 0, \ldots, 0) \|$ and some BDD-algorithm can be used to recover the exact shortest vector.

One of the most common algorithms for solving the BDD is Babai’s nearest plane algorithm [Bab]. This algorithm takes as input an LLL-reduced basis $\{b_i\}$ and a target point $y \in \mathbb{R}^m$. Initially the algorithm computes the corresponding Gram-Schmidt orthogonalisation $\{b^*_i\}$, and sets $x = y$. It then iteratively takes the hyperplane $H_i = \text{span}\{b_1, \ldots, b_{m-1}\} = \text{span}\{b^*_1, \ldots, b^*_{m-1}\}$ and translates it by an integral number of $b_n$ in order to get the plane that is closest to $x$ (this can easily be done by computing the length of the projection of $x$ onto $b^*_m$). Then $x$ is projected onto this hyperplane, $b_m$ is taken out of consideration and the iteration step is repeated in one dimension lower. The final value of $x$ is a lattice point, which is the output of the algorithm.

Clearly, during the course of the algorithm $x$ is moved only along the Gram-Schmidt vectors $b^*_i$ and since it is always moved to the nearest hyperplane, the distance it is moved is at most $\frac{1}{2} \|b^*_i\|$. Therefore, Babai’s algorithm outputs $x \in L$ with

$$\|x - y\|^2 \leq \frac{1}{4} \sum_i \|b^*_i\|^2$$

We do not know how well this compares to the actual closest point to $y$ in $L$, but it can be proven that $\|x - y\| \leq 2^{m/2} \|t - y\| \forall t \in L$, though this is of less importance for the hybrid attack. For the hybrid attack we conservatively just demand that Babai’s algorithm could possibly be used to find $x$ from $(0, \ldots, 0, x_{m-R+1}, \ldots, x_m)$. Setting
\( x' = (x_1, x_2, \ldots, x_{m-R}, 0, \ldots, 0) \), this requires that

\[
\begin{vmatrix}
\langle x', b_i^r \rangle \\
\langle b_i^r, b_i^r \rangle
\end{vmatrix} \leq \frac{1}{2} \quad \forall i
\]

Depending on the form of \( x \) this requirement may be easy or hard to fulfil, but in general this is more likely to be true for longer \( b_i^r \). Since the condition needs to be fulfilled for all indices \( i \) it is generally of interest to keep the Gram-Schmidt vectors roughly equal in size, which is a goal we recognise from the lattice reduction algorithms in Section 4.2. The parameter \( R \) should be chosen so that the initial exact solution and the subsequent reduction algorithm take about the same time since that will typically minimise the total runtime.

5 Lattice-based Signature Systems

In this section we give a short overview of different lattice-based signature systems, and what the landscape looks like today.

NTRU-like Systems The NTRU cryptosystem was invented already in 1996 [HPS98] (that is, the same year as [Ajt96] was published) and uses rings as lattices. The solution is patented and owned by Security Innovation, who constantly develop the system. A related signature system was introduced in [HHGP+03].

NTRU has historically operated in the ring \( \mathbb{Z}[x]/(x^n - 1) \) but has in recent years switched to \( x^n + 1 \) for its quotient polynomial since the previous choice was proven to be insecure. It uses two coprime moduli \( p \) and \( q \), where \( q \) is large and \( p \) is typically very small (say \( p = 3 \)). The key is generated by taking two polynomials \( f \) and \( g \), where \( f \) must be invertible modulo both \( p \) and \( q \). The public key is the quotient \( g/f \mod q \) and the private key is \( f \). These define a lattice and the signature process consists of hashing the message to an arbitrary point and, through Babai’s algorithm, finding the closest lattice point to the message. The verifier ascertains that the signature is indeed a lattice point, and that the distance between the message hash and the signature point is small enough. This means that NTRUSign is linked to the BDD problem, which does not have a strong SIS-like average to worst cases reduction to rely on.

For a long time the NTRU family has been regarded as the main contender for a lattice-based cryptosystem, but they have typically not had any provable security. While NTRUEncrypt has held up for a long time despite this, NTRUSign has not been so lucky. Over the years there have been several attacks on NTRUSign, notably when Regev and Nguyen managed to extract the secret key using only 400 signatures [NR09]. Several patches and attacks have followed in the years since, and what variant of NTRUSign is currently used by Secure Innovation is unclear but the trust in the system is low because of its history as well as the lack of provable security.

One key assumption in NTRU is that the distribution produced by taking the quotient of two small polynomials, as done in both the NTRUEncrypt and NTRUSign protocols, is close to uniform. It was proven by Stehlé in [SS11] that if the numerator and denominator are sampled from a discrete Gaussian distribution with high enough \( \sigma \) the quotient distribution is indeed statistically indistinguishable from uniform. While this result is important the lower bound on \( \sigma \) is pretty large, and it is unclear if this has been implemented in any NTRU instance.
LWE-based Systems  Several signature systems are based on \( R\)-LWE. A very recent such system is Ring-TESLA [ABB+16], presented in 2016. This is one of very few systems that claim 128 bit security, and apart from this it has provable security and provably secure instantiation. The secret key is the secret \( s \) as well as the two errors \( e_1, e_2 \) from the \( R\)-LWE distribution. The polynomials \( a_1, a_2 \) are system parameters and the public key is \((t_1, t_2) = (a_1 s + e_1, a_2 s + e_2)\).

The signature process involves generating a lattice point \((a_1 y, a_2 y)\) for some random \( y \). The signature consists of elements that allow the verifier to construct a point close to this lattice point, but perturbed because of the errors \( e_i \). The most significant bits are hashed and it is verified that the hashes of the two points agree. Because the Search \( R\)-LWE is difficult it is impossible for the verifier to extract the secret key from the output of the verification process.

Ring-TESLA sadly has very large key and signature sizes – they claim to be on par with BLISS (see later) but this is because of an erroneous interpretation of the BLISS data (probably bits have been taken for bytes). In reality Ring-TESLA gives public keys that are roughly a factor 8 larger than those of BLISS, with comparable runtimes. However, the secure instantiation is a strong argument for Ring-TESLA.

SIS-based Systems  \( R\)-SIS can, as has been said, not be used for encryption but is good for signatures. One such system was introduced in [Lyu12]. The idea of such systems is to produce a specific equation \( x = az \), where \( a \) is the public key and \( z \) is the signature. The result \( x \) is hashed together with the message. Producing another valid signature \( z' \) is the same thing as solving \( a(z' - z) = 0 \) which would then be a solution to an \( R\)-SIS instance. A similar idea is employed in [HPO+15], which was especially designed for embedded systems. Both these systems have long runtimes and large public key and signature sizes. While [Lyu12] is built around the discrete Gaussian distribution, [HPO+15] uses the uniform distribution over a set of short lattice points.

A significant improvement to the above idea was published under the name BLISS [DDLL13]. As has been said, BLISS is provably secure but does not have provably secure instantiation. Instead it relies on the NTRU-style key generation for its instantiation. More specifically, it generates two short polynomials \( f \) and \( g \), and computes \((2g + 1)/f\) as its secret key. In order for BLISS to have secure instantiation this quotient would have to be indistinguishable from uniform, which it has not been proven to be.

BLISS is the main focus of this report and the following section will describe it in detail.

6 BLISS

BLISS is an abbreviation for “Bimodal Lattice Signature Scheme” and it was constructed by Ducas, Durmus, Lepoint and Lyubashevsky [DDLL13] as an improvement to the previously suggested signature scheme by Lyubashevsky in [Lyu12]. Some improvements have since been suggested, but the only one that really affects the aspects of the system that we discuss here was presented by Ducas in [Duc14], and we shall look at the system together with this improvement.

BLISS is an \( R\)-SIS based signature system which performs very well when it comes to speed and not too bad when it comes to signature and key sizes. The authors propose four parameter sets (as well as a toy variant, maybe a cryptosystem, that they call BLISS-0). As can be seen in Table 1 the system outperforms currently used signature systems (RSA and Elliptic Curve DSA) when it comes to verification speed, and even competes with
the faster of them in signature speed. However, while public key and signature sizes are only slightly higher than those of RSA they are an order of magnitude larger than those of ECDSA. This is a problem partly because communication is expensive energy-wise, especially wireless communication. With the current development of smaller and smaller intelligent devices, saving battery life becomes more important and in that aspect local computations are much cheaper than communication.

Table 1: Self-reported BLISS performance benchmarking: [DDLL13][Table 1] updated with the results in [Duc14][Table 3]. Highlighted systems have comparable security levels.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Security</th>
<th>Sign. size</th>
<th>SK Size</th>
<th>PK Size</th>
<th>Sign (ns)</th>
<th>Sign/s</th>
<th>Verify (ms)</th>
<th>Verify/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLISS-0</td>
<td>≤ 60 bits</td>
<td>3.3 Kib</td>
<td>1.5 Kib</td>
<td>3.3 Kib</td>
<td>0.108</td>
<td>9.3k</td>
<td>0.017</td>
<td>59k</td>
</tr>
<tr>
<td>BLISS-1</td>
<td>128 bits</td>
<td>5.6 Kib</td>
<td>2 Kib</td>
<td>7 Kib</td>
<td>0.128</td>
<td>7.8k</td>
<td>0.030</td>
<td>33k</td>
</tr>
<tr>
<td>BLISS-II</td>
<td>128 bits</td>
<td>5 Kib</td>
<td>2 Kib</td>
<td>7 Kib</td>
<td>0.185</td>
<td>5.4k</td>
<td>0.030</td>
<td>33k</td>
</tr>
<tr>
<td>BLISS-III</td>
<td>160 bits</td>
<td>6 Kib</td>
<td>3 Kib</td>
<td>7 Kib</td>
<td>0.146</td>
<td>6.8k</td>
<td>0.031</td>
<td>32k</td>
</tr>
<tr>
<td>BLISS-IV</td>
<td>192 bits</td>
<td>6.5 Kib</td>
<td>3 Kib</td>
<td>7 Kib</td>
<td>0.167</td>
<td>6.0k</td>
<td>0.032</td>
<td>31k</td>
</tr>
<tr>
<td>RSA 1024</td>
<td>72-80 bits</td>
<td>1 Kib</td>
<td>1 Kib</td>
<td>1 Kib</td>
<td>0.167</td>
<td>6k</td>
<td>0.004</td>
<td>91k</td>
</tr>
<tr>
<td>RSA 2048</td>
<td>101-112 bits</td>
<td>2 Kib</td>
<td>2 Kib</td>
<td>2 Kib</td>
<td>1.180</td>
<td>0.8k</td>
<td>0.038</td>
<td>27k</td>
</tr>
<tr>
<td>RSA 4096</td>
<td>≥ 128 bits</td>
<td>4 Kib</td>
<td>4 Kib</td>
<td>4 Kib</td>
<td>8.660</td>
<td>0.1k</td>
<td>0.138</td>
<td>7.5k</td>
</tr>
<tr>
<td>ECDSA P-160</td>
<td>80 bits</td>
<td>0.32 Kib</td>
<td>0.16 Kib</td>
<td>0.16 Kib</td>
<td>0.058</td>
<td>17k</td>
<td>0.203</td>
<td>5k</td>
</tr>
<tr>
<td>ECDSA P-256</td>
<td>128 bits</td>
<td>0.5 Kib</td>
<td>0.25 Kib</td>
<td>0.25 Kib</td>
<td>0.106</td>
<td>9.3k</td>
<td>0.384</td>
<td>23k</td>
</tr>
<tr>
<td>ECDSA P-384</td>
<td>192 bits</td>
<td>0.75 Kib</td>
<td>0.37 Kib</td>
<td>0.37 Kib</td>
<td>0.195</td>
<td>5k</td>
<td>0.854</td>
<td>1k</td>
</tr>
</tbody>
</table>

BLISS is the currently most efficient and practically usable lattice-based signature scheme, and in this section we shall describe the system thoroughly. While everything is based on the contents of [DDLL13], quite extensive reverse engineering has gone into reconstructing formulas and techniques, especially regarding the security analysis. A few paper details have been clarified in communication with Léo Ducas, one of the authors. The even more interested reader can of course look at the original papers. First, however, we shall need to go over some prerequisites.

6.1 Prerequisites

6.1.1 Rejection Sampling

Rejection sampling is a commonly used technique that can produce output distributed according to some target probability distribution, given a method that samples according to some other source distribution. This can be used when the goal is to generate the target distribution, but more importantly here it can be used to hide the source distribution.

Let us formulate rejection sampling in the following lemma:

**Lemma 6.1.** Let $C$ be any set, and let $h$ be a probability density function on $C$. Moreover, let $f$ be a probability density function on $\mathbb{Z}^n$ and $g_c$ be a family of probability density functions on $\mathbb{Z}^n$ indexed by $c \in C$. If there exists a constant $M$ s.t.

$$Mg_c(z) \geq f(z) \quad \forall c \in C, \forall z \in \mathbb{Z}^n$$

the following two algorithms have identical output distributions:

1. Sample $c$ from $h$, sample $z$ from $g_c$, and output $(z, c)$ with probability $f(z)/(Mg_c(z))$.
2. Sample $c$ from $h$, sample $z$ from $f$, and output $(z, c)$ with probability $1/M$.

**Proof.** The probability density function of the first distribution is:

$$p(c, z) = h(c)g_c(z) \frac{f(z)}{Mg_c(z)} = \frac{1}{M}h(c)f(z)$$
which is the same as the probability density function of the second distribution. Thus they are equal.

Distribution 1 is the source distribution, while distribution 2 is the target one. Typically the distributions \( g_c \) contains secret information about for example the secret key. Therefore the output from \( g_c \) cannot be used directly. However, if \( f \) is chosen to be something innocent like, for example, a Gaussian distribution, the output contains no statistical information about the \( g_c \) (except that \( M \) is a commonly known constant).

The process can be understood as follows: Sampling from \( g_c \) can be seen as choosing a point \((z, y) \in \mathbb{Z}_m \times \mathbb{R}\) (where \( y \) is a coordinate in the direction of the graph) uniformly in the area under the graph of \( M g_c \), as can be seen in Figure 2. The point is accepted if it also happens to lie under the graph of \( f \) (that is, \( y < f(z) \)), otherwise it is rejected and we sample again. Since \( M g_c \geq f \) at all points this gives the same distribution as sampling directly from \( f \). Since the total area under the graph of \( f \) is 1 while the total area under the graph of \( M g_c \) is \( M \), the sampling will produce an output with probability \( 1/M \) overall.

![Figure 2](image.png)

(a) Uncentered Gaussian  
(b) Bimodal Gaussian  
(c) Bimodal Gaussian, larger \( \sigma \)

**Figure 2:** Rejection sampling from the upper (gray) distribution to produce the lower (black), which is a centered Gaussian. Rejection constants \( M \) are 20, 4 and 2.2 respectively. In 2a it is impossible to make the constant \( M \) large enough without employing a tail-cut.

Since the algorithm has to be re-run on average \( M \) times in order to produce an output keeping \( M \) small is of capital interest, but the relationship between \( f \) and the \( g_c \)-family limits how low \( M \) can go: The more similar the two distributions are, the smaller \( M \) can be taken to be. The major improvement in BLISS compared to earlier schemes is a choice of distribution family \( g_c \) that allows for a significantly lower \( M \).

### 6.1.2 Polynomials in Matrix Form

The correspondence between the ring \( \mathcal{R} = \mathbb{Z}[x]/f(x) \) with \( \deg f = n \) and \( \mathbb{Z}^n \) is obvious from the coefficients of \( p \in \mathcal{R} \): If \( p = p_0 + p_1 x + \cdots + p_{n-1} x^{n-1} \), we have the bijection

\[
\mathcal{R} \ni p \leftrightarrow (p_0, p_1, \ldots, p_{n-1})^T \in \mathbb{Z}^n.
\]

While this representation works nice with addition, it does not respect multiplication at all. For this purpose polynomials are sometimes represented as matrices instead, in the following way: Let \((p)_i\) denote the coefficient for \( x^i \) in \( p \in \mathcal{R} \). Then the matrix representation of \( p \) is

\[
\begin{pmatrix}
(p)_0 & (xp)_0 & \cdots & (x^{n-1}p)_0 \\
(p)_1 & (xp)_1 & \cdots & (x^{n-1}p)_1 \\
\vdots & \ddots & \ddots & \vdots \\
(p)_{n-1} & (xp)_{n-1} & \cdots & (x^{n-1}p)_{n-1}
\end{pmatrix}
\]
The product $ph$ can then be understood as multiplying the matrix representation of $p$ with the vector representation of $h$, rendering the vector representation of the product. This is not an efficient way to implement the multiplication but the matrix representation can be good for understanding for example the coefficient norm of the product.

### 6.2 The BLISS System

#### 6.2.1 Overview

BLISS is based on $\mathcal{R}$-SIS$_{q,\beta,2}$ and works, schematically, as follows: $\mathcal{R}$ is taken to be $\mathbb{Z}[x]/x^n + 1$ with $n$ a power of 2 (which makes $x^n + 1$ the $N$th cyclotomic polynomial with $N = 2n$). A prime $q$ is chosen so that $q = 1 \mod 2^n$, because that means that $x^n + 1$ has a root in $\mathbb{Z}_q$ and this root can be used to speed up calculations with the Number Theoretic Transform.

The secret key $s$ is a $2 \times 1$-matrix of short polynomials in $\mathcal{R}$, and the public key $a$ is a $1 \times 2$-matrix such that $as = q \mod 2q$. The system uses a hash function $H: \mathcal{R} \times M \to \mathcal{R}$, where $M$ is the message space, such that $H$ produces polynomials with exactly $\kappa$ coefficients set to 1 and all others set to 0 (for some system parameter $\kappa$).

A message $\mu$ is signed in the following way: $y \in \mathcal{R}^2$ is chosen according to a central discrete Gaussian distribution with width parameter $\sigma$. Then a challenge $c$ is produced as $c = H(ay, \mu)$. A specific algorithm produces $v$ which is such that $v = sc'$ for some $c' = c \mod 2$ but chosen so that $v$ is fairly small. A bit $b \in \{-1, 1\}$ is chosen uniformly at random and $z = y + bv$ is computed, but rejection sampling is performed to make this look like a central Gaussian with width parameter $\sigma$. If the rejection sampling fails the signature process restarts. Else $(z, c)$ is output as signature. Figure 3 illustrates where each object in the BLISS algorithm lives.

The verifier does the following: First, $z$ is verified to be no larger than should be expected from a discrete Gaussian variable with width parameter $\sigma$, and also the max-norm of $z$ is verified to be small for technical reasons. Then the signature is accepted if $H(az + qc \mod 2q, \mu) = c$. This will hold for a proper signature because

$$az + qc = a(y + bsc') + qc = ay + bqc' + qc \mod 2q = ay \mod 2q$$

This is because $q \mid (bqc' + qc)$ and since $b \equiv_2 1$, we have $bqc' + qc \equiv_2 q(c' + c) \equiv_2 0$ because $c' = c \mod 2$. With $q$ odd, this means that $2q \mid (bqc' + qc)$.

The complete implementation of BLISS has some extra features, such as a compression of bits in the signature. There are also many parameters involved that can be tuned for speed, signature size or security, which we shall describe in later sections. First, however, a security argument should be presented.
6 BLISS

6.2.2 Security

We refer to [DDLL13, Appendix C] for the complete security reduction. Here we shall merely describe the idea behind why BLISS is to be considered secure.

Say that a forger has managed to find $z, \mu, c$ such that $H(az + qc \mod 2q, \mu) = c$. As long as $H$ is chosen to be collision resistant, with overwhelming probability the attacker knows only one $x$ such that $H(x, \mu) = c$, which means that $az = x - qc$. But recall (from Fact 3.2) that with a properly chosen $a$, the operation $az$ is collision resistant in $z$ under the restriction that $\|z\|_2$ is small. This means that such a preimage is infeasible to find, which means that an attack is infeasible.

The real security proof is of course much more involved and makes use of the fact that the produced signatures are statistically indistinguishable from simple Gaussians, that is, that they leak no statistical information about the secret key. Therefore a proper rejection sampling procedure is of major importance to the integrity of the signature scheme.

6.2.3 BLISS and Rejection Sampling

The distribution of the output $y + bv$ with $b = \pm 1$ is a bimodal discrete Gaussian on $R^2 \cong Z^{2n}$, with one mode at $v$ and one at $-v$ and width parameter $\sigma$. The probability density function of this distribution is

$$p_1(x) = \frac{C}{2} \left( \exp \left( -\frac{\|x - v\|^2}{2\sigma^2} \right) + \exp \left( -\frac{\|x + v\|^2}{2\sigma^2} \right) \right)$$

where $C$ is the normalisation constant. We want to perform rejection sampling to imitate a central discrete Gaussian, which has probability density function

$$p_2(x) = C \exp \left( -\frac{\|x\|^2}{2\sigma^2} \right)$$

for the same $C$. We therefore need to pick $M$ so that $p_1 M \geq p_2$, that is,

$$M \geq \frac{p_2(x)}{p_1(x)} = \frac{\exp \left( -\frac{\|x\|^2}{2\sigma^2} \right)}{\frac{1}{2} \left( \exp \left( -\frac{\|x - v\|^2}{2\sigma^2} \right) + \exp \left( -\frac{\|x + v\|^2}{2\sigma^2} \right) \right)} = \frac{\exp \left( -\frac{\|x\|^2}{2\sigma^2} \right)}{\frac{1}{2} \left( \exp \left( -\frac{\|x\|^2 + 2\langle x, v \rangle - \|v\|^2}{2\sigma^2} \right) + \exp \left( -\frac{\|x\|^2 - 2\langle x, v \rangle - \|v\|^2}{2\sigma^2} \right) \right)} = \frac{\exp \left( \frac{\|v\|^2}{2\sigma^2} \right)}{\cosh \left( \frac{\langle x, v \rangle}{\sigma^2} \right)} \leq \exp \left( \frac{\|v\|^2}{2\sigma^2} \right)$$

since $\cosh(x) \geq 1$ always. (Note that this bound is also sharp by looking at $x \perp z$.) But this means that as long as we choose $M = \max_v \exp \left( \frac{\|v\|^2}{2\sigma^2} \right) = \exp \left( \max \frac{\|\cdot\|^2}{2\sigma^2} \right)$ rejection sampling works, and there is no reason to choose $M$ higher since that only makes the process slower. The situation is shown in Figures 2b and 2c. As can be seen, a larger $\sigma$ allows for a smaller $M$ since the larger $\sigma$ allows the two modes to “bleed into” one another, approximating the target distribution better.

The main difference between BLISS and its predecessors was the introduction of the bit $b$, that is, the switch to a bimodal Gaussian distribution. Previous proposals would only use $b = 1$ and this distribution does not fit very well over the central discrete Gaussian curve, making the rejection factor $M$ very large.
In BLISS, $M$ depends only on the quotient $\max \|v\|/\sigma$, which should be kept as low as possible. However, as we shall see later a larger $\sigma$ gives both worse security and larger signatures, so it is also of interest to keep $\max \|v\|$ small. Since $v = sc'$ with $c' = c \mod 2$, $\|v\|$ depends on the secret key $s$ as well as the challenge $c$ produced as output from $H$. The challenge is for this reason chosen to have all coefficients zero except $\kappa$ of them set to 1. The weight $\kappa$ should be chosen high enough to give enough entropy, but can otherwise be kept low. Note that what matters here is the worst case scenario for $v$, so that $\max \|v\|$ must be kept low while the exact distribution of $\|v\|$ is of little or no interest.

### 6.2.4 Key Generation

The secret key $s$ and public key $a$ must satisfy $as = q \mod 2q$. Moreover we have the condition that $v$ should be possible to make small which puts a constraint on $s$, and it should be infeasible to recover the secret key from the public key.

In BLISS the key generation is as follows: Two densities $\delta_1$ and $\delta_2$ are chosen. Typically $\delta_1 \approx 0.3$ and $\delta_2$ is very small, or zero. Then two polynomials $f, g \in R$ are generated uniformly with the constraint that they each have $d_1 = \lceil \delta_1 n \rceil$ coefficients in $\pm 1$, $d_2 = \lceil \delta_2 n \rceil$ coefficients in $\pm 2$ and all other coefficients zero. This is redone until $f$ is invertible in $R_q$. Then $s$ is set to $(f, 2g + 1)^T \in R^{2 \times 1}$ and $a = (2((2g + 1)/f \mod q), q - 2) \in R^{1 \times 2}$, because then

\[
as \mod 2q = 2 \left(\frac{2g + 1}{f} \mod q\right) f + (q - 2)(2g + 1) \mod 2q = 4g + 2 + 2qg - 4g + q - 2 \mod 2q = q \mod 2q\]

The key generation is in the spirit of NTRU and experience has shown that even when $f$ and $g$ are generated with small coefficient norm the quotient $(2g + 1)/f \mod q$ seems statistically uniform and it is therefore deemed difficult to extract the secret key from this.

Let us then move on to the last requirement: The possibility of a small $v$. First let us study the matrix representations $F$ and $G'$ of $f, 2g + 1$ respectively. Note that since for any $p \in R$, where $p = p_0 + \cdots + p_{n-1}x^{n-1}$ we have $xp = -p_{n-1} + p_0x + \cdots + p_{n-2}x^{n-1}$, which means that the columns of $F, G$ are just rotations of the first column with some signs switched. In particular all columns have the same norm. In $F$ that norm will be exactly $\sqrt{d_1 + 4d_2}$, while in $G'$ it would have been $\sqrt{4d_1 + 16d_2}$ if it were not for the added 1 at the end. What effect this has on the norm depends on the constant term in $g$, but the resulting vector has norm at most $\sqrt{4d_1 + 16d_2 + 9}$ (when the constant term is 2). In many parameter choices $\delta_2 = 0$, that is, there are no 2’s. In that case, $\|2g + 1\| \leq \sqrt{4d_1 + 5}$.

The matrix $S$ is then defined as the block matrix with $F$ as upper block and $G'$ as lower block. Then letting $c$ be the vector representation of $c$ the vector representation of their product is exactly $Sc$, which we can use to understand the norm of $v$. Note that because of the structure of $c$, $Sc$ is just a sum of some choice of $\kappa$ columns of $S$, so that

\[
\|Sc\|^2 = \|s_{i_1} + \cdots + s_{i_\kappa}\|^2 = \sum_{j=1}^\kappa \|s_{i_j}\|^2 + \sum_{1 \leq r < j \leq \kappa} 2\langle s_{i_r}, s_{i_j}\rangle = \sum_{j=1}^\kappa \|s_{i_j}\|^2 + 2 \left(\sum_{r=1}^{j-1} \langle s_{i_r}, s_{i_j}\rangle\right)
\]

and we want to choose $s$ so that this expression is limited, no matter the choice of $c$. This was done in a cumbersome and fairly expensive way in [DDLL13] but was improved
in [Duc14] using the following idea: If we are allowed to switch some of $c$’s coefficients from 1 to $-1$, the resulting signature will still be valid but denoting the coefficients in this new $c'$ by $c'_i$, we get

$$\|S_c\|^2 = \|c'_{i_1} s_{i_1} + \cdots + c'_{i_\kappa} s_{i_\kappa}\|^2 = \sum_{j=1}^\kappa \left( \|s_{i_j}\|^2 + 2c'_{i_j} \left( \sum_{r=1}^{j-1} c'_{i_r} s_{i_r} \right) \right)$$

This means that it is possible to choose $c'$ in such a way that the inner product part of each term is negative, and it is even possible to do so greedily by first setting $c'_{i_1}$ (the first coefficient may WLOG stay as 1) and then proceeding. This then generates $v = sc'$ with the guarantee that $\|v\|^2 \leq \sum_{j=1}^\kappa \|S_{i_j}\|^2 \leq \kappa \max_i \|s_i\|^2 \leq \kappa (\|f\|^2 + \|2g + 1\|^2)$.

Moreover, since $c'$ is never revealed and the rejection sampling is performed exactly to hide $\|v\|$, this does not leak any information about the secret key even though $c'$ depends on it heavily.

This guaranteed bound on $\|v\|$ is used to provide a low repetition factor $M$ in the rejection sampling in the signature algorithm.

### 6.2.5 Gaussians and Exponentials

The BLISS scheme generates a fair number of Gaussian variables in the signature algorithm, as well as evaluates the exponential function (both directly and as a part of the cosine hyperbolicus) in the rejection step. This can be done fast using large precomputed tables with values of the exponential function, but this approach is not feasible for small devices with memory constraints so on these more time-consuming processes have been used. One of the major contributions in [DDLL13] is a suggestion of a new way of sampling discrete Gaussians using small precomputed tables and small computation time.

Since the publication of [DDLL13] other papers have suggested further speedups in the Gaussian sampler. The impact of these improvements affect only the performance of BLISS, and the effect is highly independent from performance variations due to system parameters or similar. Therefore we do not discuss this here.

### 6.2.6 Compression

BLISS performs two compressions on the $z = (z_1, z_2)$-part of the signature. However, it should be noted that the authors’ own BLISS implementation does not fully implement these compressions which means that the benchmarking in Table 1 is without accounting for this runtime (but the reported signature sizes are with the compression taken into account).

The first compression is due to simple Huffman encoding. Since $z$ is non-uniform, some values are more common than others and this can be used to create a Huffman encoding of each coefficient of the $z_i$. Each coefficient is discrete Gaussian with width parameter $\sigma$, which means that the entropy is at most $\log_2(\sqrt{2\pi e\sigma}) \approx \log_2(4.1\sigma)$. Huffman coding provides near-optimal encoding, so that the expected required size is no more than one bit over the entropy. Moreover, since most of the entropy loss is in the
high-order bits BLISS chooses to only Huffman-encode these, while the near-uniform low-order bits are kept uncompressed in order to decrease overhead. This alone brings the size of \( x \) down from the initial \( 2^n \log_2 q \) to \( 2^n \log_2(\sqrt{2\pi e\sigma}) \) (plus the challenge, which can be encoded very efficiently almost down to the security level).

The second compression is an elimination of the low-order bits in \( z_2 \), replacing it with \( z_2^\dagger \) which is essentially \( z_2 \) rounded to the nearest multiple of \( 2^d \). This operation requires some changes to the signature and verification protocols: First, letting \( \zeta \) be such that \( \zeta(q - 2) = 1 \mod 2q \), the public key may equivalently be chosen as \( \zeta a = (\zeta a_1, 1) \) where we recall that \( a_1 = 2 \left( \frac{2q+1}{f} \mod q \right) \). Then we denote by \( \lfloor x \rfloor_d \) the number \( x \) rounded to the nearest multiple of \( 2^d \), that is,

\[
x = \lfloor x \rfloor_d 2^d + (x \mod 2^d)
\]

where we use the absolute remainder. Finally, we take \( p = [2q]_d \) and we require that \( q = 1 \mod 2^{d-1} \).

Now the modification is as follows: The input to the hash function, formerly \( ay \mod 2q \), is replaced by \( \lfloor \zeta ay \rfloor_d \mod p = [\zeta a_1 y_1 + y_2]_d \mod p \). Then, while \( z_1 \) is chosen in the same way, \( z_2 \) is replaced by

\[
\begin{align*}
z_2^\dagger &\triangleq [\zeta ay \mod 2q]_d - [\zeta a_1 z_1 + \zeta qc] \mod 2q]_d \mod p 
\end{align*}
\]

which can be described as an element in \([0, p-1]^n\). The signature now consists of \((z_1, z_2^\dagger, c)\), and the verification is changed to checking that the concatenated vector \((z_1 \parallel 2^d z_2^\dagger)\) is short both in Euclidean norm and in max-norm, and verifying

\[
H(\zeta a_1 z_1 + \zeta qc \mod 2q]_d \mod p, \mu) = c
\]

This will hold for valid signatures because of how \( z_2^\dagger \) is defined.

This leads to roughly a compression of \( d \) bits in each coefficient of \( z_2 \), but this comes at a high security cost. In the original protocol, the corresponding \( R \)-SIS instance required finding a solution of length at most \( 2B_2 \), where \( B_2 \) is the maximum Euclidean norm of signatures that the verifier allows. With this modification the \( R \)-SIS problem only requires a length of at most \( 2B_2 + (2^d + 1)\sqrt{n} \), that is, after only a few compressed bits the allowed size starts growing rapidly. We shall see, though, that there is often some space left over to perform at least a few bits of compression.

### 6.2.7 Algorithm and Security Statement

Here we present the final version of the BLISS protocol:

**Algorithm 2: BLISS Key Generation.**

**Result:** Keypair \( s = (s_1, s_2)^T \in \mathcal{R}_q^{2 \times 1} \) and \( a = (a_1, q - 2) \in \mathcal{R}_{2q}^{2 \times 1} \) such that

\[ as = q \in \mathcal{R}_{2q}. \]

1. Sample \( f, g \in \mathcal{R}_q \) uniformly with exactly \( d_1 \) coefficients in \( \{\pm 1\} \) and \( d_2 \) coefficients in \( \{\pm 2\} \), all others set to 0;
2. while \( f \) not invertible do Resample \( f \);
3. \( s \leftarrow (f, 2g + 1)^T \in \mathcal{R}_q^{2 \times 1} \);
4. \( a_1' \leftarrow (2g + 1)/f \in \mathcal{R}_q \);
5. \( a_1 \leftarrow 2a_1' \in \mathcal{R}_{2q} \);

The security statement is as follows:
can be multiplied with the inverse of one of them without changing the problem. Having one element set to 1 is no easier than the more general setting, since all elements from Problem 7 with the elements a_i drawn according to the distribution $\mathcal{K}$ rather than uniformly. The a_i’s here are a_1 and q - 2 from the public key (or $\zeta a_1$ and 1). Having one element set to 1 is no easier than the more general setting, since all elements can be multiplied with the inverse of one of them without changing the problem. The distribution $\mathcal{K}$ is double the result of the operation $(2g + 1)/f$ where g and f are drawn.

Algorithm 3: GreedySC(S, c), Choose Challenge Signs.

**Data:** A matrix $S = [s_1, \ldots, s_n] \in \mathbb{Z}^{m \times n}$, a vector $c \in \mathbb{Z}_2^m$ with exactly $\kappa$ non-zero entries.

**Result:** $v = Sc'$ such that $c' = c \mod 2$ and $\|v\|^2 \leq \kappa \max_i \|s_i\|^2$.

1. $v \leftarrow 0 \in \mathbb{Z}^m$.
2. for $i : c_i \neq 0$ do
3. \hspace{1em} $\zeta \leftarrow \text{sgn}(\langle v, s_i \rangle)$ \hspace{1em} // We let \text{sgn}(0) = 1, arbitrary choice
4. \hspace{1em} $v \leftarrow v - \zeta s_i$
5. end

**Output:** v

Algorithm 4: BLISS Signature Algorithm

**Data:** Message $\mu$, public key $a = (a_1, q - 2) \in \mathcal{R}^{1 \times 2}_{2q}$ and $\zeta = 1/(q - 2) \in \mathbb{Z}_{2q}$, secret key $s = (s_1, s_2)^T \in \mathcal{R}^{2 \times 1}_{2q}$

**Result:** A signature $(z_1, z_2, c)$ of $\mu$

1. Sample $y_1, y_2 \in \mathcal{R}$ from $D_{\mathbb{Z}^n, \sigma}$
2. $u \leftarrow \zeta a_1 y_1 + y_2 \mod 2q$
3. $c \leftarrow H(|u|_d \mod p, \mu)$
4. $v = \text{GreedySC}(S, c)$ \hspace{1em} // $S$ is the matrix representation of $s$
5. Sample $b \in \{\pm 1\}$ uniformly
6. $(z_1, z_2) \leftarrow (y_1, y_2) + bv$
7. With probability $1 - 1/(M \exp(-\|v\|^2/2\sigma^2) \cosh(\Theta\|v\|/\sigma^2))$ go to 1
8. $z_2^\dagger = ([u]_d - [u - z_2]_d) \mod p$

**Output:** $(z_1, z_2^\dagger, c)$

Theorem 6.2 ([DDLL13, Theorem 4.4]). Consider the BLISS signature scheme with $d \geq 3, q = 1 \mod 2^{d-1}$, and $2B_\infty + (2^d + 1) < q/2$. Suppose there is a polynomial-time algorithm $\mathcal{F}$ which succeeds in forging signatures with non-negligible probability. Then there exists a polynomial-time algorithm which can solve $\mathcal{R}$-SIS$_q^{K^d/2}$ for $\beta = 2B_2 + (2^d + 1)\sqrt{n}$.

**Remark.** The $\mathcal{R}$-SIS$_q^{K^d/2}$ problem referenced in Theorem 6.2 is the $\mathcal{R}$-SIS$_q^{\beta, 2}$ problem from Problem 7 with the elements $a_i$ drawn according to the distribution $\mathcal{K}$ rather than uniformly. The $a_i$’s here are $a_1$ and $q - 2$ from the public key (or $\zeta a_1$ and 1). Having one element set to 1 is no easier than the more general setting, since all elements can be multiplied with the inverse of one of them without changing the problem. The distribution $\mathcal{K}$ is double the result of the operation $(2g + 1)/f$ where g and f are drawn.

Algorithm 5: BLISS Signature Verification

**Data:** Message $\mu$, public key $a = (a_1, q - 2) \in \mathcal{R}^{1 \times 2}_{2q}$, signature $(z_1, z_2^\dagger, c)$

**Result:** Accept or Reject signature

1. if $\|(z_1, z_2^\dagger)\| > B_2$ then reject
2. if $\|(z_1, z_2^\dagger)\|_\infty > B_\infty$ then reject
3. if $c = H(|\zeta a_1 z_1 + \zeta q c|_d + z_2^\dagger \mod p, \mu)$ then accept else reject
according to the distribution in Algorithm 2.

### 6.2.8 Security Estimates

In [DDLL13] the authors provide security estimates based on resistance to forgery through the BKZ 2.0 attack, the Hybrid attack for key recovery, as well as key recovery using either primal or dual lattice reduction.

<table>
<thead>
<tr>
<th>Attack</th>
<th>Security estimates for $\lambda$ bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>BKZ 2.0</td>
<td>$((2B_2 + (2^d + 1)\sqrt{m})/\sqrt{q})^{1/m} \leq \delta(\lambda)$ for target root hermite factor $\delta(\lambda)$.</td>
</tr>
<tr>
<td>Primal lattice reduction</td>
<td>$(\sqrt{q/2\pi\delta(\lambda)} + 4\delta(\lambda))^{1/2n} \leq \delta(\lambda)$ for target root hermite factor $\delta(\lambda)$.</td>
</tr>
<tr>
<td>Dual lattice reduction</td>
<td>$T(\delta) \exp(2\pi(\delta_1 + 4\delta_2)\delta^{2m}/q) \geq 2^\lambda$ for all $\delta$, where $T$ is the time for the BKZ reduction.</td>
</tr>
<tr>
<td>Hybrid attack</td>
<td>No clear formula</td>
</tr>
</tbody>
</table>

#### BKZ Forgery

The tightest security bounds are provided by the resistance to BKZ 2.0 together with the resistance to the Hybrid attack. The goal is to forge a signature, which requires finding a vector of length at most $\beta$ in the lattice $\mathcal{L} = \Lambda^\perp(A)$, where $A$ is the $n \times m$-matrix representation of the public key. Since the determinant of this lattice is $q^n$ (see Section 3.1), this requires a root hermite factor such that

$$\delta^m(q^n)^{1/m} \leq \beta \iff \delta \leq \left(\frac{\beta}{\sqrt{q}}\right)^{1/m}$$

where the dimension of the lattice is $m = 2n$. Both $\beta$ and $q$ are known, so $\delta$ can be found. Using either the tables in [CN11] or the formula from Section 4.2.2 the appropriate block size is found, and the corresponding runtime is interpolated from tabulated values in [CN11]. The authors choose to give the security level as the number of nodes each enumeration subroutine in BKZ needs to visit, taken from the tabulated values in the abbreviated version, [CN11]. That each such visit takes around 100 clock cycles and that each round requires $2n$ enumerations (so a minimum of $2n$ enumerations must be made) is disregarded and instead taken as a margin in the security estimate, which amounts to around 17 bits.

#### Brute Force Key Recovery

Brute force key recovery consists of guessing $g$ according to the key generation distribution, and computing the corresponding $f$ from the public key. If $f$ seems to have been generated using the key generation distribution the chance is good that the secret key has actually been recovered. The complexity of this attack is determined by the entropy in $g$, which is

$$e_g = 2^d_1 + d_2 \begin{pmatrix} n \\ d_1 \\ d_2 \end{pmatrix} \begin{pmatrix} n - d_1 \\ d_2 \end{pmatrix}$$

For all reasonable choices of parameters this is well above the desired security level. The factor of $2^d_1 + d_2$ alone provides a fair amount of entropy. The total entropy is over 600 for all suggested parameter choices (except BLISS-0).
There is also a Meet-in-the-Middle variant of this attack, using a tradeoff between memory and time. The article here chooses to model each entry of the secret key as an independent variable with entropy
\[ e = -\left( \delta_0 \log_2 \delta_0 + \delta_1 \log_2 \frac{\delta_1}{2} + \delta_2 \log_2 \frac{\delta_2}{2} \right) \]

where \( \delta_0 \) is the density of zeros. This leads to a slight overestimation of the entropy since the entries are not actually independent from one another, but this overestimation is very small: For the parameter set BLISS I, the true entropy \( e_g \) is 601, while the estimate \( n \cdot e \) gives \( e_g \approx 605 \).

The MiM-attack stores all possible values of the first half of the coefficients in \( g \) and then searches through the second half of them, combining these with the stored data to find a good candidate for \( g \). The complexity is split between time and space so that both complexities are about \( 2^{3\lambda/2} \). This attack is still not feasible, since the security against it is around 300 bits. It is, however, used as a subroutine for the Hybrid attack.

The Hybrid Attack Key Recovery

The Hybrid attack can be used to recover the secret key. However, instead of recovering \((f \mid 2g + 1)^T\) as a solution to \((a_1', -1)s = 0 \mod q\), we can recover \((f \mid g)^T\) as a solution to \((a_1', -2)s = 1 \mod q\). Of course this is not a lattice but rather a translation of a lattice, which can be translated back by adding some solution to \((a_1', -2)s = -1 \mod q\) to each lattice point. Shortness is not preserved, but closeness is and since the hybrid attack relies on the BDD problem this is what matters. The advantage of changing lattices is that \((f \mid g)\) is significantly shorter, which makes it easier to find.

The cutoff index \( R \) must be chosen, but also the parameter \( r \) which is introduced, where the BKZ reduction is performed not on \( b_1, \ldots, b_R \) but instead on \( b_{R+1}, \ldots, b_R \).

The advantage of this extra degree of freedom is that typically the first few basis vectors will already be fairly large and therefore not need any reduction. Some large vectors should still be included in the reduction in order to bring up the determinant of the sub-lattice and therefore the length of the last BKZ-vectors. Leaving \( r \) out is equivalent to setting \( r = 0 \), so this is not a restriction of the attack.

\( R \) is chosen so that the \( 2n - R \) last indices can be extracted using \( 2^\lambda \) time and memory via the MiM attack, where \( \lambda \) is the desired security level. The memory consumption is about \( 2^{(2n-R)\epsilon/2}e(2n-R) \) and the time is about \( 2^{(2n-R)\epsilon/2}n^2 \), the authors set \( R \) so that
\[ (2n - R)e = 2\lambda - \log_2(e(2n - R)) - \log_2(n^2) \]

Now the problem is to get a basis for the preceding \( R \) coordinates which satisfies
\[ \left| \langle s', b_i^* \rangle \right| / \left( \left| \langle b_i^* , b_i^* \rangle \right| \right) \leq \frac{1}{2} \quad \forall i \]

where \( s' = (s_1, \ldots, s_R, 0, \ldots, 0) \). The authors assume that the lattice reduction algorithm provides a basis with random direction, in which case the above quantity can be modelled as a centered Gaussian of standard deviation \( ||s'||/(\sqrt{R}||b_i^*||) \). Since \( ||s'|| = \sqrt{R(\delta_1 + 4\delta_2)} \) the standard deviation is \( \sqrt{\delta_1 + 4\delta_2}/||b_i^*|| \).

The first \( r \) basis vectors are left as they are, the remaining are reduced using BKZ 2.0. Under the Geometric Series Assumption the reduced lattice is such that \( ||b_i^*|| = \gamma \delta^{2(R-i)} \) for \( \gamma = ||b_R^*|| \). The paper chooses to require \( \gamma \geq 2.5||s'||/\sqrt{R} = 2.5\sqrt{\delta_1 + 4\delta_2} \) and claims that this gives a probability of 0.01 for the algorithm to succeed, though the derivation of this number is not explained in detail.
The authors then run the BKZ simulation algorithm on the lattice in order to find which block size and \( r \) are required to get a large enough \( \gamma \), and read off the cost of this reduction. It is then verified that the cost is no smaller than the desired security level. The runtime of Babai’s algorithm is not taken into account.

**Primal Lattice Reduction Key Recovery** The authors also consider recovering \((f \mid g)^T\) using pure lattice reduction. Since this vector is unusually short, much shorter than expected according to the Gaussian heuristic, it can typically be found using lattice reduction. Previous experiments with lattices with just one abnormally short vector have suggested that the required root hermite factor is determined by \( \lambda_2/\lambda_1 \), but the authors run experiments that suggest that the root hermite factor is really governed by \( \lambda_2'/\lambda_1 \), where \( \lambda_2' \) is the expected value of \( \lambda_2 \) according to the Gaussian heuristic, that is, \( \det(L)^{1/2n}/\sqrt{2n/2\pi e} = \sqrt{qn/\pi e} \). Recalling \( \lambda_1 = \| (f \mid g) \| \), the root hermite factor is

\[
\delta^{2n} = \frac{\lambda_2'}{\lambda_1} = \frac{q\sqrt{\pi e}}{\sqrt{2n(\delta_1 + 4\delta_2)}}
\]

This typically gives very small \( \delta \), so this attack does not pose any real threat.

**Dual Lattice Reduction Key Recovery** It is also possible to recover the secret key using dual lattice reduction. This is the idea that a short vector in the dual lattice can act as a distinguisher for a very short vector in the original lattice. The method is described in [BBD08].

The short dual vector is not actually taken as a vector in the dual lattice \( L^* \), but rather as a vector in the scaled dual lattice \( qL^* \). Given a vector \( x \) in this lattice, with random direction, it is possible to distinguish the existence of an unusually short vector \( s \) with probability

\[
\varepsilon = e^{-\pi \left( \frac{\| s \| q^{\sqrt{2n}}} {2n(\delta_1 + 4\delta_2)} \right)^2}
\]

where again we have \( s = (f \mid g) \) as our short vector, so \( \| s \| = \sqrt{2n(\delta_1 + 4\delta_2)} \).

An algorithm was provided in [MM11] which, using the distinguisher as an oracle, can recover one entry of the private key with \( 1/\varepsilon^2 \) oracle calls. Therefore the overall time is dictated by \( T/\varepsilon^2 \) where \( T \) is the runtime of BKZ 2.0 to produce the short dual vector \( x \). The authors try several different block sizes in order to find which one gives the fastest key recovery algorithm for the different BLISS parameter sets. Since the scaled dual lattice \( qL^* \) has determinant \( q^{2n} \det(L^*) = q^{2n}/\det L = q^n \), a BKZ run providing root hermite factor \( \delta \) produces a \( x \) with

\[
\| x \| \approx \delta^{2n} \det(qL^*)^{1/2n} = \delta^{2n} \sqrt{q}
\]

so both \( T \) and \( \varepsilon \) can be computed knowing \( \delta \).

**6.2.9 Parameter Overview**

A summary of all parameters associated with BLISS can be found in Table 3, together with some restrictions for their values. They can all be set individually except \( N, n, q \) and \( m \) which are linked, \( M \) and \( \sigma \) which are linked, and the two verification bounds which can be computed from other parameters.

The verification bound \( B_\infty \) is found in the BLISS security statement. The other bound, \( B_2 \), must be set high enough so that the risk that valid signatures become too large can be overlooked. For this, we have the following fact:
Table 3: The parameters for BLISS

<table>
<thead>
<tr>
<th>Notation</th>
<th>Significance</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Index of cyclotomic polynomial</td>
<td>$\varphi(N)$ (Euler’s totient function)</td>
</tr>
<tr>
<td>$n$</td>
<td>Ring dimension</td>
<td>$m = 2n$</td>
</tr>
<tr>
<td>$m$</td>
<td>Lattice dimension</td>
<td>$q = 1 \mod N$ for efficiency</td>
</tr>
<tr>
<td>$q$</td>
<td>Prime modulus</td>
<td></td>
</tr>
<tr>
<td>$\delta_1, \delta_2$</td>
<td>Secret key densities</td>
<td>Large enough for Hybrid attack</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>Standard deviation</td>
<td></td>
</tr>
<tr>
<td>$\kappa$</td>
<td>Hamming weight of challenge $c$</td>
<td>$\log_2 \binom{n}{\kappa} \geq \lambda$</td>
</tr>
<tr>
<td>$M$</td>
<td>Rejection repetition rate</td>
<td>$M = \exp\left(\frac{\kappa \max |s|^2}{2^{d-1}}\right)$</td>
</tr>
<tr>
<td>$d$</td>
<td>Dropped bits in $z_2$</td>
<td>$q = 1 \mod 2^{d-1}$</td>
</tr>
<tr>
<td>$p$</td>
<td>Additional modulus</td>
<td>$p = \lfloor q/2^{d-1} \rfloor$</td>
</tr>
<tr>
<td>$B_2$</td>
<td>Verification threshold Euclidean norm</td>
<td>$B_2 \propto \sigma \sqrt{m}$ (see Fact 6.3)</td>
</tr>
<tr>
<td>$B_\infty$</td>
<td>Verification threshold max norm</td>
<td>$2B_\infty \leq q/2 - (2^d + 1)$</td>
</tr>
</tbody>
</table>

Fact 6.3 ([Lyu12, Lemma 4.4]). Let $D^m_\sigma$ be the $m$-dimensional discrete Gaussian distribution. For any $k > 0$, it holds that

$$Pr[\|z\| > k\sigma \sqrt{m} \mid z \leftarrow D^m_\sigma] < k^m \exp\left(\frac{m}{2}(1-k^2)\right)$$

Knowing $m$, $k$ can be computed so that the desired probability is lower than, say, $2^{-\lambda}$. The associated $k$ is usually between 1.3 and 1.4 for the relevant parameters. A low $B_2$ allows for a low $\beta$ in the associated $R$-SIS instance, which means higher security, so otherwise keeping $B_2$ low is preferable. Notice that a failed signature does not leak any information, so a lower $B_2$ only affects performance and can be traded for other performance gains.

Table 2 lists the security reductions according to the four attacks that the BLISS article considers, except for the hybrid attack, for which the reduction is too complicated to fit in a table – see, however, the analysis in Section 7.2. Note that the hybrid attack is the only attack that takes direct advantage of low entropy in the secret key. Note that as long as a formula for $T(\delta)$, the runtime of BKZ 2.0 given a $\delta$, is known the parameter relations given a desired security level can be easily deduced for all but the hybrid attack.

6.3 BLISS Instantiations

The original paper proposes four sets of parameters for BLISS along with a fifth variant intended as a bait for cryptanalysts. The parameters are as in Table 4. Here at all times $N = 2n$ since the cyclotomic polynomial in question is $x^n + 1$ with $n$ a power of two, so we omit this. Also the security reduction results are listed – however, recall that these claim to have some security margin. Notably a security margin of at least 10 bits in terms of clock cycles is claimed in the BKZ security estimates.

The follow-up paper [Duc14] provided a reduction of the repetition parameter $M$ compared to the original sets; Table 4 has incorporated this change. The table also incorporates a correction of the values of $\lambda_1(\mathcal{L}) \approx \sqrt{m(\delta_1 + 4\delta_2)}$, that is, a correction to the primal lattice reduction resistance, compared to the original paper, which has been detected in correspondence with one of the authors.

BLISS-I was optimised for speed at the 128 bit security level, while BLISS-II was optimised for size. These optimisations were performed before [Duc14], and that paper
Table 4: The BLISS parameter sets

<table>
<thead>
<tr>
<th></th>
<th>BLISS-0</th>
<th>BLISS-I</th>
<th>BLISS-II</th>
<th>BLISS-III</th>
<th>BLISS-IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security</td>
<td>≤ 60 bits</td>
<td>128 bits</td>
<td>128 bits</td>
<td>160 bits</td>
<td>192 bits</td>
</tr>
<tr>
<td>( n )</td>
<td>256</td>
<td>512</td>
<td>512</td>
<td>512</td>
<td>512</td>
</tr>
<tr>
<td>( q )</td>
<td>7681</td>
<td>12289</td>
<td>12289</td>
<td>12289</td>
<td>12289</td>
</tr>
<tr>
<td>( \delta_1, \delta_2 )</td>
<td>0.55, 0.15</td>
<td>0.3, 0</td>
<td>0.3, 0</td>
<td>0.42, 0.03</td>
<td>0.45, 0.06</td>
</tr>
<tr>
<td>( \sigma )</td>
<td>100</td>
<td>215</td>
<td>107</td>
<td>250</td>
<td>271</td>
</tr>
<tr>
<td>( \kappa )</td>
<td>12</td>
<td>23</td>
<td>23</td>
<td>30</td>
<td>39</td>
</tr>
<tr>
<td>( M )</td>
<td>2.44</td>
<td>1.21</td>
<td>2.18</td>
<td>1.40</td>
<td>1.61</td>
</tr>
<tr>
<td>( d )</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>( B_2 )</td>
<td>2492</td>
<td>12872</td>
<td>11074</td>
<td>10206</td>
<td>9901</td>
</tr>
<tr>
<td>( B_\infty )</td>
<td>530</td>
<td>2100</td>
<td>1563</td>
<td>1760</td>
<td>1613</td>
</tr>
<tr>
<td>Sign. size</td>
<td>3.3 Kib</td>
<td>5.6 Kib</td>
<td>5.0 Kib</td>
<td>6.0 Kib</td>
<td>6.5 Kib</td>
</tr>
<tr>
<td>SK size</td>
<td>1.5 Kib</td>
<td>2.0 Kib</td>
<td>2.0 Kib</td>
<td>3.0 Kib</td>
<td>3.0 Kib</td>
</tr>
<tr>
<td>PK size</td>
<td>3.3 Kib</td>
<td>7.0 Kib</td>
<td>7.0 Kib</td>
<td>7.0 Kib</td>
<td>7.0 Kib</td>
</tr>
<tr>
<td>BKZ 2.0 sec.</td>
<td>53</td>
<td>130</td>
<td>136</td>
<td>168</td>
<td>188</td>
</tr>
<tr>
<td>Primal red. sec.</td>
<td>&gt; 130</td>
<td>&gt; 240</td>
<td>&gt; 240</td>
<td>&gt; 240</td>
<td>&gt; 240</td>
</tr>
<tr>
<td>Dual red. sec.</td>
<td>56</td>
<td>177</td>
<td>177</td>
<td>201</td>
<td>211</td>
</tr>
<tr>
<td>Hybrid attack sec.</td>
<td>≥ 60</td>
<td>≥ 128</td>
<td>≥ 128</td>
<td>≥ 160</td>
<td>≥ 192</td>
</tr>
</tbody>
</table>

chose not to change any parameters other than \( M \) in order to get backwards compatibility. Therefore additional parameter tweaks could offer more improvement.

### 6.4 Attacks on BLISS

A few side-channel attacks have been mounted on BLISS.

**Timing Attack** In [BHLY16] an attack is described that using timing measurements on the Gaussian sampler can extract information about \( y \), since the sampler has different runtimes depending on the output. With a perfect side-channel they manage to extract the secret key with high probability from only 450 signatures, though it should be kept in mind that this is an idealised situation. This weakness can be mitigated by making the Gaussian generation constant-time which is a possibly heavy runtime penalty.

**Fault Attack** A collection of early loop termination attacks was presented in [EFGT16]. They demonstrate that if it is possible to trigger faults in the BLISS signature algorithm that result in early termination of some specific loops, the resulting signature leaks a lot of information about the secret key, enough so that one single faulty signature could be enough to recover the key. One version of the attack centers around triggering a low-degree \( y \) by terminating its generation early.

Other fault attacks are presented in [BBK16], where randomisation, skipped operations and zeroing are considered.

Both these papers consider other lattice-based signature schemes as well as BLISS, and the vulnerability is similar between them.
7 Analysis of BLISS

Here we present our analysis of the BLISS system. More specifically, we present formulas for the sizes of keys and signatures as well as the resistance against the studied attacks. We will also go over some controversy concerning the original paper’s security estimates and reevaluate them.

7.1 Object Sizes

Public Key The public key can be represented by only the polynomial \( a'_1 = (2g+1)/f \in \mathcal{R}_q \), from which \( a \in \mathcal{R}_{2q}^{1 \times 2} \) can be reconstructed. The idea is that this should be infeasible to distinguish from something uniformly distributed, in order to rely on the security of \( \mathcal{R} \)-SIS. Therefore any more efficient way to encode this would be likely to open up the way for an attack.

BLISS uses the trivial encoding which requires \( n \lceil \log_2 q \rceil \) bits. A more efficient and optimal encoding is achieved by encoding the polynomial as the integer \( a_0 + a_1 q + a_2 q^2 + \cdots + a_{n-1} q^{n-1} \in [-(q^n - 1)/2, (q^n - 1)/2] \) which requires

\[
\lceil n \log_2 q \rceil \text{ bits (3)}
\]

This is non-problematic because the interpretation of \( a'_1 \) as an integer can be done efficiently in \( O(n) \) time using Horner’s method.

Secret Key The secret key can be encoded in many ways. The BLISS paper does not specify how their encoding is done, but the following algorithm agrees with their data: If \( \delta_2 = 0 \) there are 3 possible different values that can go in each entry, if \( \delta_2 \neq 0 \) there are five. Therefore the number of bits required to encode the secret key is

\[
\begin{align*}
2n \lfloor \log_2 3 \rfloor &= 4n, & \delta_2 &= 0 \\
2n \lfloor \log_2 5 \rfloor &= 6n, & \delta_2 &\neq 0
\end{align*}
\]

There are however several more effective ways to store the secret key. As long as \( \delta_2 = 0 \) and \( \delta_1 \leq 2/3 \) a Huffman encoding can be used by assigning 0 to 0, 10 to 1 and 11 to \(-1\) giving a representation length of \( 2n(1+\delta_1) \) bits.

As long as \( \delta_2 \) is very small a similar idea can be used, namely to encode the signs in the above way, and afterwards indicate which indices are supposed to carry \( \pm 2 \)'s instead of \( \pm 1 \)'s. This can be done by printing out the \( n\delta_2 \) indices using \( \lceil \log_2 n \rceil \) bits for each one, but since we know that the positions containing zeros are off-limits anyways we can count only the \( n(\delta_1 + \delta_2) \) non-zero positions in \( f \) and \( g \) respectively, resulting in an encoding using

\[
2n((1 + \delta_1 + \delta_2) + \delta_2 \lceil \log_2(n(\delta_1 + \delta_2)) \rceil) \text{ bits}
\]

This is already significantly smaller than the method used in [DDLL13], as can be seen in Table 5. However, we stress that efficient compression of the secret key is of interest only if it needs to be stored away for some time. While in use, it is instead preferable to precompute many objects related to the secret key, such as its matrix representation.

An optimal way to store the secret key between rounds is to specify the PRNG used to generate secret keys, and simply store the random seed used for the key. This would reduce the space needed to store the secret key to \( 2\lambda \) bits, in order to resist Grover’s attack. The drawback is that the key has to be reconstructed using the exact same random number generator as when it was first created.
Table 5: Secret key sizes according to the BLISS paper compared to our Huffman-related method.

<table>
<thead>
<tr>
<th>BLISS instance</th>
<th>SK size original (bits)</th>
<th>SK size our method (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLISS-0</td>
<td>1536</td>
<td>1562</td>
</tr>
<tr>
<td>BLISS-I/II</td>
<td>2048</td>
<td>1331</td>
</tr>
<tr>
<td>BLISS-III</td>
<td>3072</td>
<td>1731</td>
</tr>
<tr>
<td>BLISS-IV</td>
<td>3072</td>
<td>2099</td>
</tr>
</tbody>
</table>

Signature

The signature consists of two entities: The challenge $c$ and the vector $(z_1, z_2^\dagger) \in \mathbb{R}^2 \cong \mathbb{Z}^m$. Here we need to take into the discussion a recent paper [Saa16], which discusses the encodings of these two objects.

Starting with the challenge, in [DDLL13] no actual encoding is specified, and only the entropy of $c$ is specified. In [Saa16] it is suggested to replace the hash function $H(ay \mod 2q, \mu)$ with the composition $H_0(H_i(ay \mod 2q, \mu))$. $H_i$ would hash the vector and message down to a bit string of the desired entropy. The [Saa16] version of BLISS, called BLZZRD, uses for $H_i$ a function from the SHA3-family with appropriate entropy. Then this value is used as a seed to produce the indices in $c$, using $H_0(ay \mod 2q, \mu)$ can be sent over to the verifier, which reduces the size of the $c$ representation down to exactly the entropy. Any further reduction is then impossible.

As for $z_1$ and $z_2^\dagger$, the original BLISS paper uses Huffman encoding to represent these. The specific encoder is not specified and also not implemented in their proof of concept code, but the strongSwan BLISS implementation employs some sort of Huffman compression algorithm. In theory such an encoding can bring sizes down to the entropy level plus at most one bit of overhead, and it is suggested to pack the entries of $z_1, z_2^\dagger$ in groups of $k$ in order to bring the average overhead down to $1/k$ bits. Exactly which parameters are chosen is, however, unclear.

In BLZZRD another encoding is suggested, namely a binary encoder (an arithmetic encoder with two bits per character) which achieves moderately lower signature sizes compared to those reported in the BLISS paper. We shall take this to mean that it is possible, with any one of these two techniques, to encode this part of the signature using very close to as few bits as the entropy and therefore the entropy of $(z_1, z_2^\dagger)$ is of primary importance.

The entropy of $z_1$ is easily assessed: Because of the rejection sampling, $z_1$ is a discrete Gaussian in $n$ dimensions with width parameter $\sigma$. The entries are independent so the total entropy is just the sum of the individual entropies. Since $\sigma$ is fairly large, around 100 or larger, the entropy can be approximated with an integral:

$$H(z_1) = -\sum_{x \in \mathbb{Z}} C \exp \left( -\frac{||x||^2}{2\sigma^2} \right) \log_2 \left( C \exp \left( -\frac{||x||^2}{2\sigma^2} \right) \right) \approx$$

$$\approx -\int_{\mathbb{R}} C \exp \left( -\frac{||x||^2}{2\sigma^2} \right) \log_2 \left( C \exp \left( -\frac{||x||^2}{2\sigma^2} \right) \right) \, dx$$

Since for large $\sigma$, $C \approx 1/\sqrt{(2\pi\sigma^2)^n}$, this can be computed to be $\log_2(\sigma\sqrt{2\pi\sigma})$. The total entropy is then $n \log_2(\sigma\sqrt{2\pi\sigma})$ for $z_1$.

Let us now focus on $z_2^\dagger$: Recall that

$$z_2^\dagger \triangleq [\zeta ay \mod 2q]_d - [\zeta a z_1 + \zeta qc \mod 2q]_d \mod p =$$
7 ANALYSIS OF BLISS

\[
\begin{align*}
|\zeta a y \mod 2q|_d - |\zeta a z - z_2 + \zeta q c \mod 2q|_d \mod p &= \\
= |\zeta a y \mod 2q|_d - |\zeta a y - z_2 \mod 2q|_d \mod p
\end{align*}
\]

We see that \( z_2^2 \) is then not exactly equal to \( |z_2|_d \), but we can prove that for each coefficient \( i \) it holds that \(|z_2^2|_i - (|z_2|_d)_i| \leq 1 \): Given any \( a, b \in \mathbb{Z} \), we have that

\[
(|a|_d + |b|_d - |a+b|_d)2^d = a - (a \mod 2^d) + b - (b \mod 2^d) - (a+b - (a+b \mod 2^d)) =
\]

\[
= (a + b \mod 2^d) - (a \mod 2^d) - (b \mod 2^d) =
\]

\[
= ((a \mod 2^d) + (b \mod 2^d)) \mod 2^d - ((a \mod 2^d) + (b \mod 2^d))\]

The expression \((x \mod 2^d) - x\) with \( x = (a \mod 2^d) + (b \mod 2^d) \in [-2^d, 2^d-1] \) may take the values

\[
\begin{align*}
-2^d & \geq x \geq 2^d-1 \\
0 & -2^d \leq x < 2^d-1 \\
2^d & x < -2^d-1
\end{align*}
\]

so clearly, \(||a|_d + |b|_d - |a+b|_d| \leq 1 \). Since we may take \( a = (\zeta a y)_i \) and \( b = (z_2)_i - (\zeta a y)_i \), it follows that \(||z_2^2|_i - (|z_2|_d)_i| \leq 1 \). As long as the coefficients of \(|z_2|_d \) are not too constrained, that is, as long as \( d \) is not too large (\( 2^d \) significantly smaller than \( \sigma \)) this variation is relatively small and \( z_2^2 \) follows approximately the same distribution as \( |z_2|_d \).

Again, for \( d \) fairly small in the same sense as above, \(|z_2|_d \) can be approximated with a discrete Gaussian distribution with width parameter \( \sigma/2^d \). Therefore for \( d \) fairly small, \( z_2^2 \) should have entropy close to \( \log_2(\sigma \sqrt{2\pi e}/2^d) = \log_2(\sigma \sqrt{2\pi e}) - d \) per coefficient.

This is not true for large \( d \) in BLISS I/II this expression even gives a negative entropy for \( z_2^2 \). For high \( d \) the entropy of \(|z_2|_d \) can be computed numerically and quickly drops to zero when \( \sigma/2^d < 1 \). However, the effect of the difference \((z_2^2)_i - (|z_2|_d)_i\) becomes significant and this is difficult to predict. Where the exact sizes presented in the BLISS article come from is unclear, perhaps they are experimental, but for small \( d \) we can compute the signature entropy as

\[
\lambda + 2n \log_2(\sigma \sqrt{2\pi e}) - nd
\]

where \( \lambda \) is the security level and thus the entropy of the challenge. For large \( d \) this is an understatement.

7.2 Security

We shall discuss the security of the BLISS system from many perspectives, and specifically address problems with the claimed security levels.

The Hybrid Attack We shall start by studying the hybrid attack further, and provide a more theoretical methodology to ascertain a specific security level.

First we shall study the value \( \gamma \), which is set to no less than \( 2.5 ||s'||/\sqrt{R} \) in the original paper. This is claimed to give a success probability of 0.01 for Babai’s algorithm.

Let us recall that the requirement was to have \( ||s', b^*_i||/||b^*_i||^2 \leq 1/2 \), and this quantity was modelled as a central Gaussian variable with standard deviation \( \sigma_i = ||s'||/(\sqrt{R}||b^*_i||) \), where we have that \( ||b^*_i|| = \gamma \delta^{2(R-i)} \) if \( i > r \) (recall that we do not perform lattice reduction on the first few vectors). Therefore we have the standard deviation

\[
\sigma_i = \frac{||s'||}{\sqrt{R} \gamma \delta^{2(R-i)}} = \frac{1}{\gamma \delta^{2(R-i)}}
\]
where we denote by $\hat{\gamma}$ the ratio $\frac{\gamma}{\|s'\|/\sqrt{R}}$. Note that these are modelled as independent. The probability that a Gaussian variable with mean 0 and standard deviation $\sigma$ is within $[-1/2, 1/2]$ is

$$\frac{1}{2} \left( \text{erf} \left( \frac{1}{2\sigma\sqrt{2}} \right) - \text{erf} \left( -\frac{1}{2\sigma\sqrt{2}} \right) \right) = \text{erf} \left( \frac{1}{2\sigma\sqrt{2}} \right)$$

where erf is the error function. Since the distributions are independent, the probability that all fall within this span can be found simply by taking the product:

$$\prod_{i=1}^{R} \text{erf} \left( \frac{1}{2\sigma_i\sqrt{2}} \right) = \prod_{i=1}^{R} \text{erf} \left( \frac{\hat{\gamma}\delta^{2i(R-i)}}{2\sqrt{2}} \right)$$  \hspace{1cm} (5)

One problem here is that we might not know $R$. The other is that Equation 5 is really for $r = 0$, while for larger $r$ the first few standard deviations will likely be even larger. However, we find that this has very little bearing on the result. For illustration we set $\delta = 1.006$ and $\hat{\gamma} = 2.5$ as suggested in the paper. In this last representation of the probability change in $R$ and change in $r$ both affect only high-valued $i$, so we experiment with moving the upper limit around. The results can be found in Table 6. Note that we expect $R$ to be around 800, which means that for our applications the exact value of $R$ is highly insignificant for this computation.

<table>
<thead>
<tr>
<th>Upper limit</th>
<th>800</th>
<th>600</th>
<th>400</th>
<th>200</th>
<th>100</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probability</td>
<td>0.00312</td>
<td>0.00312</td>
<td>0.00312</td>
<td>0.00312</td>
<td>0.00312</td>
<td>0.00400</td>
</tr>
</tbody>
</table>

This result means that the probability computation above can, given a security level and thus a $\delta$ for the BKZ-reduction, be used to find the appropriate $\hat{\gamma}$ numerically.

Secondly, we study $r$. An optimal choice of $r$ should be such that as large a $\gamma$ as possible is achieved. Let $L'$ be the sub-lattice spanned by $b_{r+1}, \ldots, b_R$, that is, the basis vectors that shall be BKZ-reduced. Meanwhile let $L''$ be the lattice spanned by $b_1, \ldots, b_R$. Recall that the basis is triangulated. This means that $L''$ is exactly the intersection of $L$ with the subspace spanned by the $R$ first coordinate vectors. It can then be seen as the lattice generated not by $A$ (the matrix representation of the public key) but by $A'' \in \mathbb{Z}^{n \times R}$ consisting of the first $R$ columns in $A$, so the determinant of $L''$ is again $q^n$ (since $R > n$), that is, $L''$ consists of $q^{R-n}$ points modulo $q$.

Next, define the map $P : L'' \rightarrow L'$ by

$$P \left( \sum_{i=1}^{R} a_i b_i \right) = \sum_{i=r+1}^{R} a_i b_i$$

This map is clearly surjective. Moreover, let us study its kernel

$$\ker P = \left\{ \sum_{i=1}^{r} a_i b_i \in L'' \right\}$$

But with $r < n$ such an element in $L''$ would imply a combination of the first $r$ columns in $A$ that is zero modulo $q$. In all but very exceptional cases this can only happen when $a_1 = \ldots = a_r = 0 \mod q$ which in turn means that $P$ as a map between $q$-ary
7 ANALYSIS OF BLISS

lattices is bijective. Thus \( L' \) also consists of \( q^{R-n} \) points modulo \( q \). Since the lattice is \( R-r \)-dimensional, the determinant is

\[
\det(L') = q^{R-r} / q^{R-n} = q^{n-r}
\]

This will allow us to derive a relationship between \( \delta, \gamma, r \) and \( R \):

\[
q^{n-r} = \det(L') = \prod_{i=r+1}^{R} \|b_i^*\| \approx \gamma^{R-r} \delta^{R} \implies \gamma = \frac{q^{n-r}}{\delta^{R-r}}
\]

From this we can determine the optimal choice of \( r \): Given \( \delta \) and \( R \), we want \( \gamma \) to be maximised, so we take the derivative of \( \gamma \) with respect to \( r \) and find that this is zero when

\[
R - r = \sqrt{\log_2 q / \log_2 \delta} (R - n)
\]

Let for the moment \( x = 2\sqrt{\log_2 \delta} \log_2 q \). Then the expression for \( \gamma \) can be rewritten as

\[
\gamma = x^{n-r} = \frac{x^{n-R-(R-r)}}{x^{R-n}} = x^{R-n} = x^{R-n} \implies \gamma = \frac{q^{n-r}}{\delta^{R-r}}
\]

That is, given a desired security level \( \delta \), it is possible to deduce a required \( \gamma \) (as a function of \( \delta_1, \delta_2 \)) and from that, the above expression gives the maximum possible \( R \) for which the BKZ reduction can still get an appropriate \( \gamma \). We have that \( \gamma = \hat{\gamma} \|s'\| / \sqrt{R} = \hat{\gamma} \sqrt{\delta_1 + 4\delta_2} \), so let us finally substitute this in:

\[
R_{BKZ} = n + \frac{1}{4} \left( \log_2 q - \log_2 \left( \hat{\gamma} \sqrt{\delta_1 + 4\delta_2} \right) \right)^2
\]

If we start from the other end, recall that \( R \) was chosen so that

\[
(2n - R)e = 2\lambda - \log_2(e(2n - R)) - \log_2(n^2)
\]

with \( \lambda \) the desired security level and \( e \) the entropy per secret key entry. This can be solved numerically to give a value \( w \) for \( (2n - R)e \), which translates to \( R = 2n - \frac{w}{e} \). Therefore knowing \( \delta_1 \) and \( \delta_2 \), \( R \) can be found also from here. Recall the expression \( e = -(\delta_0 \log_2 \delta_0 + \frac{1}{2} \delta_1 \log_2 \delta_1 + \frac{1}{2} \delta_2 \log_2 \delta_2) \) where \( \delta_0 = 1 - \delta_1 - \delta_2 \), this gives

\[
R_{MiM} = 2n + \frac{w}{\delta_0 \log_2 \delta_0 + \delta_1 \log_2 \delta_1 + \delta_2 \log_2 \delta_2} (7)
\]

which is the lowest possible value that \( R \) can have in order for the Meet-in-the-Middle reduction to work. In order to claim a specific security level, Equation 6 must give a smaller value for \( R \) than Equation 7.

This presentation of the problem also opens up a new approach to the hybrid attack, namely that it can be used to choose parameters. With a desired security level, \( n \) and
Table 7: Examination of hybrid resistance.

<table>
<thead>
<tr>
<th></th>
<th>BLISS-0</th>
<th>BLISS-I/II</th>
<th>BLISS-III</th>
<th>BLISS-IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>δ</td>
<td>1.0083</td>
<td>1.006</td>
<td>1.0055</td>
<td>1.0053</td>
</tr>
<tr>
<td>$\hat{\gamma}$</td>
<td>2.44</td>
<td>2.67</td>
<td>2.72</td>
<td>2.74</td>
</tr>
<tr>
<td>$R_{\text{BKZ}}$ (Eq. 6)</td>
<td>486</td>
<td>874</td>
<td>880</td>
<td>883</td>
</tr>
<tr>
<td>$R_{\text{MiM}}$ (Eq. 7)</td>
<td>466</td>
<td>829</td>
<td>840</td>
<td>822</td>
</tr>
</tbody>
</table>

$q$, only $\delta_1, \delta_2$ affect the resistance. Overall, it is desirable to keep the expression $\delta_1 + 4\delta_2$ as low as possible, because this gives a small $\|s\|$ and therefore smaller signatures and better security against BKZ reduction. On the other hand, the entropy must be kept up in order to have resistance against the hybrid attack. Optimal parameters can thus be found by running some numerical optimisation algorithm on the following problem:

Minimise $\delta_1 + 4\delta_2$ under the condition

$$2n + \frac{w}{\delta_0 \log_2 \delta_0 + \delta_1 \log_2 \frac{q}{2} + \delta_2 \log_2 \frac{q}{2}} \geq n + \frac{1}{4} \left( \log_2 q - \log_2 (\hat{\gamma} \sqrt{\delta_1 + 4\delta_2})^2 \right)$$

(8)

where $w$ is found as described above and $\hat{\gamma}$ is derived from setting Equation 5 equal to 0.01. In theory this failure probability could be used to provide about 6 additional bits of security, but we choose to leave this out for two reasons: First, this probability is not known to be the optimal choice. Second, the BKZ 2.0 reduction is deterministic which means that if the attack fails, it will fail every time for that set of keys, while if it works it works. Therefore this does not add uniformly to the security of each instance.

We finally reexamine the resistance of BLISS to the hybrid attack. Let us take $\delta$ as the root Hermite factors demonstrated in the resistance against the pure BKZ reduction, but treat BLISSI/II together since they claim the same security level and are equal in all relevant parameters. We then find $\hat{\gamma}$ and compare $R$ from Equations 6 and 7. The results are in Table 7. We find that despite the fact that we can apparently take $\hat{\gamma}$ larger than the 2.5 from the original paper, which should give better security, we do not at all get the security levels we want. The results from Equation 7 are consistent with those in the original paper which is to be expected since the computations are the same. What differs is the assessment of the BKZ reduction cost. In the original paper this was done using the BKZ 2.0 simulation algorithm and some trial-and-error testing. The choice of $r$ is not explained and the value is not presented.

While the simulation algorithm is allegedly more exact than a more general approach, owing to that it takes into account that bases differ in how difficult the reduction is, the approach in [DDLL13] is difficult to verify externally. Moreover, the trial-and-error methodology means that some optimal attack parameters could easily be missed. Moreover results may depend highly on the exact choice of keys with which the experiment was run (it does not say how many different keys were tried).

**BKZ Performance** In [DDLL13] all BKZ performances are deduced from tabulated values in the abbreviated version of [CN11]. The full version of the same paper [CN12], however, contains updated values that show that the BKZ 2.0 attack is more powerful than the abbreviated version suggested. Therefore the security estimates in [DDLL13] with the claimed margins are too generous. For example, in [DDLL13] it is claimed that a block size of 215 (corresponding to $\delta = 1.006$) would require $2^{130}$ visits in the enumeration tree of the exact SVP subroutine of BKZ. In the full version this block size
is said to require only about $2^{108}$ visits in each enumeration tree. Since the security estimates in [DDLL13] are based on this value alone, letting all additional costs act as security margins, this means that the claimed 128 bit security with a significant margin for BLISS-I is incorrect.

Since the BKZ 2.0 software is not (to our knowledge) made publicly available, the only resource for estimating reduction cost is Table 4 in the full BKZ 2.0 paper. This table lists an upper bound for the number of nodes that need to be visited in the enumeration subroutine for 16 different block sizes, ranging from 100 to 250. Looking at that table it is clear that the value for block size 190 is too large, so it is probably a typo. We shall omit that data point.

Let us denote by $T(b)$ the number of nodes to visit in the enumeration tree depending on block size. We would like to interpolate an expression for $T(b)$ from the given data points. Recall that the complexity is likely to be $b^{O(b)}$, so we use the model $\log_2 T(b) = c_1 b \log_2 b + c_2 \log_2 b + c_3 b + c_4$. Simple least-squares fitting gives

$$\log_2 T(b) = 0.17970 b \log_2 b - 6.89068 \log_2 b - 0.89741 b + 54.94996$$

(9)

The data points are plotted together with this formula in Figure 4. We see that the agreement is good. Moreover the block sizes we will be interested in are within the range of the data points, so we do not need to worry about changed behaviour for other ranges of values.

We see that we would need a block size of about 241 in order to claim 128 bits of security. This translates to a $\delta$ of 1.0056 through Equation 1, which is significantly lower than what is offered by BLISS I/II.

On the other hand, we may not be interested in having quite as large margins. Each enumeration visit takes around 200 clock cycles [CN12] which adds 7 bits of security. Moreover each BKZ round consists of $m$ enumerations, adding $\log_2 m$ bits of security. The number of rounds required is more difficult to assess and can therefore be left out, but for BLISS we may still add $7 + 10 = 17$ bits to the BKZ 2.0 security level (16 bits for BLISS-0). This would for example bring BLISS-I back almost to 128 bit security, as can be seen in Table 8. Here we also see that the loss in security is higher for higher security levels, so that with the decreased margin BLISS-I and II are essentially saved, while BLISS-III and IV are nowhere close to the security level that they claim to have.
Table 8: BLISS resistance to BKZ 2.0 recalculated, where the new security assessment has less margins than the old one. The old resistance was taken to be equal to the old assessment of \(\log_2 T(\delta)\).

<table>
<thead>
<tr>
<th>Version</th>
<th>BLISS-0</th>
<th>BLISS-I</th>
<th>BLISS-II</th>
<th>BLISS-III</th>
<th>BLISS-IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old (\log_2 T(\delta))</td>
<td>53</td>
<td>130</td>
<td>136</td>
<td>168</td>
<td>188</td>
</tr>
<tr>
<td>New (\log_2 T(\delta))</td>
<td>51</td>
<td>108</td>
<td>111</td>
<td>130</td>
<td>141</td>
</tr>
<tr>
<td>New resistance (bits, no margin)</td>
<td>67</td>
<td>125</td>
<td>128</td>
<td>148</td>
<td>159</td>
</tr>
</tbody>
</table>

Table 9: Resistance to dual lattice key recovery.

<table>
<thead>
<tr>
<th></th>
<th>BLISS-0</th>
<th>BLISS-I/II</th>
<th>BLISS-III</th>
<th>BLISS-IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal block size</td>
<td>109</td>
<td>236</td>
<td>250</td>
<td>256</td>
</tr>
<tr>
<td>Resistance (margin) (bits)</td>
<td>54</td>
<td>146</td>
<td>160</td>
<td>187</td>
</tr>
<tr>
<td>Resistance (no margin) (bits)</td>
<td>70</td>
<td>163</td>
<td>177</td>
<td>204</td>
</tr>
</tbody>
</table>

This higher performance of BKZ 2.0 of course affects the other attacks as well. Resistance against primal lattice reduction is clearly still sufficient.

Let us revisit the dual lattice reduction attack in light of this new data. We use Equation 9 for \(T(\delta)\) to find the cost according to the formula described in Table 2, and add the 17 (or 16) bit margin. We find that the security is decreased, though the situation is not as bad as with the pure BKZ 2.0 attack, as seen in Table 9. Moreover, recall that this attack recovers only one entry of the secret key, though it is unclear if some of the data generated in this process can be reused to find additional entries.

Quantum Resistance Although the authors of [DDLL13] never claim quantum security of their system, quantum resistance is one of the absolute top reasons why lattice cryptography is being so thoroughly examined today. In the recent BLZZRD paper [Saa16], it is pointed out that the BLISS scheme is vulnerable to Grover’s attack through the hash function \(H\). For example, given any \(z\) and \(c\) a preimage-search can be conducted to find a message \(\mu\) of which this will be a valid signature. Also, given a challenge \(c\) and a message \(\mu\) it is possible to use Grover’s algorithm to search for a sufficiently short \(z_1\) such that \(H(\lceil z_1 + gc \rceil_d \mod p, \mu) = c\) (where \(z_1^* = 0\) is always a valid choice). The size of the search space is around \(2^{H(c)} = 2^\lambda\) by design, which means that using Grover’s attack such a solution can be found in \(O(2^{\lambda/2})\) time. This means that the security of the BLISS systems is suddenly halved.

The remedy suggested in [Saa16] is simple: Increase \(\kappa\) so that \(\binom{n}{\kappa} = 2\lambda\). This gives roughly a doubled \(\kappa\) which impacts \(\|v\|\), increasing it by a factor of roughly \(\sqrt{2}\). With a maintained \(\sigma\), this leads to an increase in \(M\) which is underestimated in [Saa16]. No new values for \(M\) are listed in the paper, but in the text it says that the increase is no more than 30%. As can be seen in Table 10, the increase is generally much higher than that. Since [Saa16] does not contain any actual values for \(M\) it is hard to understand where the misunderstanding comes from, but probably the old repetition rate from [DDLL13] rather than the new one from [Duc14] has been used.

The increase in \(M\) is not the only change that makes BLZZRD run slower than BLISS. The increase in \(\kappa\) also means that the GreedySC-algorithm will run slower since it performs \(\kappa\) iterations of the loop, and this is one of the heaviest computational steps in the signature algorithm. Thus BLZZRD looses more in performance compared to BLISS than is suggested in the paper, though no actual benchmarks are provided.

Since BLISS turns out to be neither quantumly secure nor classically as secure as...
Table 10: Repetition rate and $\kappa$ before and after the change proposed in [Saa16].

<table>
<thead>
<tr>
<th>BLISS-I</th>
<th>BLISS-II</th>
<th>BLISS-III</th>
<th>BLISS-IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old $\kappa$</td>
<td>23</td>
<td>23</td>
<td>30</td>
</tr>
<tr>
<td>Old $M$</td>
<td>1.21</td>
<td>2.17</td>
<td>1.40</td>
</tr>
<tr>
<td>New $\kappa$</td>
<td>58</td>
<td>58</td>
<td>82</td>
</tr>
<tr>
<td>New $M$</td>
<td>1.62</td>
<td>7.09</td>
<td>2.52</td>
</tr>
</tbody>
</table>

it claims to be, we shall propose a new set of parameters that give better security properties.

7.3 A Comment on Key Generation

Recall that BLISS is based on an $R$-SIS scheme where the $a_i$ are generated at random from the distribution of public keys. The hardness result on $R$-SIS, however, concerns the problem where the $a_i$ are generated uniformly at random. Therefore BLISS cannot really rely on this result for its own hardness, which is why BLISS does not have secure instantiation.

The key generation is of the NTRU kind, where the public key is taken as the quotient of two small polynomials. Noone has yet succeeded in exploiting the difference between this distribution and the uniform one for an attack. Recall that in [SS11] it was shown that for sufficiently generous definitions of “small” in the generation of numerator and denominator ($f$ and $g$ in our case), the quotient is statistically indistinguishable from uniform. On the down side, this requires significantly larger $f$ and $g$ than we currently have. Their paper considers $f$ and $g$ distributed according to a discrete Gaussian with $\sigma \geq 2n\sqrt{\ln(8mq)/q}$ ([SS11, Theorem 3]) which in the case for BLISS I-IV would mean $\sigma \geq q$. This is of course a ridiculous number since if $\sigma > q/2$ the distribution has lost much of its normal shape. Therefore a switch to such a key generation would be impossible.

Note that the security proof in [SS11] requires that $x^n + 1$ splits into linear factors modulo $q$. This is equivalent to $q = 1 \mod N$, which is also the case in BLISS because such a choice allows for use of the NTT in polynomial multiplication. The choice $q = 1 \mod N$ does however not seem to have any additional purpose, and since BLISS does not make use of [SS11] anyways it should be possible to choose another $q$ if it would give better performance.

8 BLISS Modifications

In this section we suggest a few modifications to BLISS. First we present a method for setting BLISS/BLZZRD parameters in order to reach a certain security level, and propose new parameter sets in light of the discussion in the previous section.

Second, we examine the possibility of using non-power-of-two dimensions in order to increase the flexibility in the parameter choices. We also show that it is possible to allow a freer choice of $q$. We call our modified scheme REBLISS, since it has reinforced security and relaxed dimension and modulus requirements compared to BLISS.

8.1 Parameter Setting

The following method can be used to choose parameters for a BLISS scheme:
1. Choose \( N \) (and \( n, m \)), \( q \) and security parameter \( \lambda \).

2. Use \( \lambda \) and \( n \) to find the root Hermite factor \( \delta \) required for BKZ 2.0 resistance.

3. Consider the hybrid attack:
   
   (a) Find \( \hat{\gamma} \) using the expression in Equation 5.
   
   (b) Determine \( \delta_1, \delta_2 \) by solving the optimisation problem 8.

4. Choose \( \kappa \) so that \( \log_2 \left( \frac{n}{\kappa} \right) \geq 2\lambda \). Compute \( \max \|v\| \) and deduce an order of magnitude for \( \sigma \).

5. Use \( \delta^m \sqrt{q} = 2B_2 + (2d + 1)\sqrt{n} \) to determine \( B_2 \) for a few different values of \( d \). Recall that only \( d = 0 \) or \( d \geq 3 \) are permitted.

6. From \( B_2 \), determine (an approximate) \( \sigma \) using Fact 6.3.

7. From \( \sigma, \kappa, \delta_1, \delta_2 \) compute \( M \). Choose a reasonable \( M \) and the corresponding \( \sigma, B_2, d \).

8. Compute \( p, B_\infty \).

9. Verify that resistance against primal and dual lattice reduction is sufficient.

This process gives a well-founded choice of \( \delta_1 \) and \( \delta_2 \). Very few ad-hoc decisions are needed and as many parameters as possible are set before the weigh-off between speed and object sizes has to be considered.

Note that \( \delta_i \) is not equal to \( d_i/n \), since \( d_i \equiv \lceil n\delta_i \rceil \) for \( i = 1, 2 \). Instead \( \delta_i \) is marginally lower than the true density of \( \pm i \) in \( f, g \). We must take this into consideration when we construct the key and when we later introduce the limit \( B_v \), since the estimate \( d_i = n\delta_i \) leads to an underestimate of \( \|s\| \). However, in security estimates for the hybrid attack as well as for primal and dual lattice reduction according to Table 2, an understatement of the key densities gives marginally more modest security estimates. The resistance to BKZ 2.0 depends on \( \sigma \) rather than the \( \delta_i \). Therefore we do not need to worry that the rounding will destroy anything.

**Hash Modifications** We adopt the change proposed in [Saa16] where \( H \) is replaced by \( H_i \) to produce an intermediate bit string and \( H_o \) to map such a string to the challenge vector \( c \). \( H_o \) can also be used in the key generation.

We also choose to hash the message \( \mu \) to a digest through another hash function, \( H_M \), at the beginning of the signature loop. This saves time in the signature process since in the case of a very long message, the penalty of hashing it again for every rejection may be severe.

**8.2 New BLISS parameters**

We shall use the method from Section 8.1 to set parameters for a BLISS scheme with 128 bit security level. Let us stick to \( n = 512 \) and \( q = 12289 \), which is the smallest prime such that \( q = 1 \mod N \).

1. \( N = m = 1024, n = 512, q = 12289, \lambda = 128 \).
8 BLISS MODIFICATIONS

2. With 7 bits from each enumeration node visit and 10 bits from the repetition of enumerations in each round, we require \( \log_2 T(b) = 111 \). This gives a block size of 220 and \( \delta = 1.0059 \).

3. For the hybrid attack:
   
   (a) We find that \( \hat{\gamma} = 2.68 \) gives a success probability of 0.01.
   
   (b) The optimisation problem gives \( \delta_1 = 0.39 \) and \( \delta_2 = 0.01 \).

4. We need \( \kappa \geq 58 \) and therefore take \( \kappa = 58 \). This gives
   
   \[
   \max \|v\| = \sqrt{\kappa(n(5\delta_1 + 20\delta_2) + 9)} = 254
   \]

If we demand \( M \leq 10 \) we must have \( \sigma \geq 118 \).

5 - 7. \( \delta^m \sqrt{q} = 45802 \). We try different values for \( d \) and find that the largest value that gives a possible \( \sigma \) is \( d = 10 \), which gives \( B_2 = 11304 \), \( \sigma = 260 \) and \( M = 1.61 \). However, this gives a \( B_\infty \) so low that, since \( z_{2}^1 \) may fluctuate by \( \pm 2^d \) a large proportion of signatures is rejected. Instead we choose \( d = 8 \) which gives \( B_2 = 19993 \), with the same \( \sigma \) and \( M \). This gives a public key of 6956 bits and a signature entropy of 6741 bits.

8. We get \( p = 96 \) and \( B_\infty = 2943 \).

9. For primal lattice reduction we find that \( (\sqrt{q/2\pi e(\delta_1 + 4\delta_2)})^{1/2m} = 1.0018 \) which is clearly less than \( \delta = 1.0059 \). Moreover the optimal block size for dual lattice key recovery is around 245, which also provides sufficient security.

This system is called 512a in Table 12. Also BLISS-III can be saved, resulting in system 512c in Table 12, but it turns out to be impossible to achieve 192 bit security in dimension \( n = 512 \) without letting \( M \) be astronomically high. It is not surprising that the recovery works better for lower security levels, since the updated BKZ 2.0 data had more severe effects on higher security levels.

The dimension after \( n = 512 \) is \( n = 1024 \). Here we find that it is very easy to achieve even 256 bit security, but because of the large \( n \) the object sizes are cumbersome. The problem is that powers of two are very sparse, which we shall address in the following sections.

8.3 Additional Dimensions

In this section we suggest some modifications to BLISS in order to allow non-power-of-two dimensions. While BLISS in its current form is very fast, both signatures and public keys are relatively large, so these modifications aim at lowering these sizes at the price of some speed.

Looking at Equations 3 and 4, we see that the signature can be made smaller by decreasing \( \sigma \) at the price of runtime or (if permitted) increasing \( d \) which, for high \( d \), induces a severe security penalty. The public key, however, can only be made smaller by decreasing \( n \) or \( q \), and \( n \) also has significant bearing on the signature size. In [DDLL13] attempts are made to create a somewhat secure system for \( n = 256 \), but to no avail.

Since \( n \) is restricted to powers of 2, the next possible value is \( n = 512 \) which we have seen is sufficient for 128 bit security but not much more. From there the step is long
to the next level of \( n = 1024 \), where reaching 256 bit security was very easy. With this setup, the control over the public key and signature sizes is limited.

We shall show that it is possible to use other dimensions than powers of two. Recall from Fact 3.2 that as long as the quotient polynomial is a cyclotomic, the associated \( \mathcal{R} \)-SIS problem is on average as hard as worst-case SVP on ideal lattices in \( \mathcal{R} \).

Switching to another cyclotomic polynomial has two main drawbacks. First, it makes polynomial multiplication slower since the Number Theoretic Transform experiences a speedup in power-of-two dimensions. The second drawback of switching to another polynomial multiplication slower since the Number Theoretic Transform experiences a speedup in power-of-two dimensions. The second drawback of switching to another cyclotomic polynomial is that as long as \( n \) is not a power of 2, \( x^n + 1 \) is not a cyclotomic (and \( x^n - 1 \) is never a cyclotomic except for \( n = 1 \)). Instead a cyclotomic polynomial of degree \( n \) needs to have more than two non-zero coefficients. Since we use the coefficient norm for ring elements, this means we no longer have \( \parallel x^k f \parallel = \parallel f \parallel \) as in the power-of-two case. Instead the tendency will be \( \parallel x^k f \parallel > \parallel f \parallel \) (though this is not always true), which means that the challenge product \( \mathbf{s} \mathbf{c}' \) with \( c' = c \mod 2 \) will typically be larger than in the original BLISS schemes. We shall deal with these issues.

### 8.3.1 Choice of Dimension

Since 256 was a too small dimension and 512 comfortably large enough for 128 bit security, we search for suitable cyclotomics with intermediate degrees.

Moreover, in order to keep the product \( \mathbf{s} \mathbf{c}' \) small, we want to keep the \((x^k f, x^k(2g+1))\) small. Let us first study this with \( k = 1 \). Take \( \mathcal{R} = \mathbb{Z}[x]/\Phi_N(x) \) where \( \Phi_N(x) = x^n - \sum_i \phi_i x^i \). Then

\[
\|x(p_0 + p_1 x + \cdots + p_{n-1} x^{n-1})\| = \left\| (0 + p_0 x + \cdots + p_{n-2} x^{n-2}) + p_{n-1} \sum_i \phi_i x^i \right\| = \\
= \| (0, p_0, \ldots, p_{n-2}) + p_{n-1}(\phi_0, \phi_1, \ldots, \phi_{n-1}) \| = \sqrt{ (\phi_0 p_{n-1})^2 + \sum_i (p_{i-1} + p_{n-1} \phi_i)^2 }
\]

In a cyclotomic polynomial we always have \( \phi_0 = \pm 1 \) since \( \Phi_N \) is a factor in \( x^N - 1 \). As for the other \( \phi_i \), ideally we would like \( p_{i-1} \) and \( p_{n-1} \phi_i \) to have opposite signs to reduce the norm, but we can of course not tailor the \( \phi_i \) to specific \( p \)'s. To keep the expected value as small as possible we should wish to have as many \( \phi_i = 0 \) as possible, and therefore search for cyclotomic polynomials with at most 3 non-zero coefficients (that is, one \( \phi_i \neq 0 \) with \( i \geq 1 \)).

The following result is of interest. It is previously well-known but we include the proof for completeness.

**Proposition 8.1.** \( \Phi_{3^l2^k}(x) = x^{2^l3^k-1} - x^{2^l-1}3^k-1 + 1 \) if \( l, k \geq 1 \), while \( \Phi_{3^l}(x) = x^{2^l3^k-1} + x^{3^k-1} + 1 \) for \( k \geq 1 \).

**Proof.** Recall that

\[
x^N - 1 = \prod_{d | N} \Phi_d(x)
\]

First set \( N = 3^k \). Then

\[
x^{3^k} - 1 = \prod_{i=0}^k \Phi_{3^i}(x) = \Phi_{3^k}(x)(x^{3^k-1} - 1) \Leftrightarrow \Phi_{3^k} = x^{2^l3^k-1} + x^{3^k-1} + 1
\]
Second, we prove that for $u > 1$ odd, $\Phi_{2u}(x) = \Phi_u(-x)$. Recall that $\Phi_1(x) = x - 1$ and $\Phi_2(x) = x + 1$. We shall do this by induction: Suppose that for all odd $k$ with $1 < k < u$, $\Phi_{2k}(x) = \Phi_k(-x)$. Then

$$x^{2u} - 1 = \prod_{d|2u} \Phi_d(x) = \prod_{d|u} \Phi_d(x) \prod_{v|2u} \Phi_{2d}(x) = (x^u - 1)\Phi_{2u}(x)\Phi_2(x) \prod_{d|u, d\neq 1,u} \Phi_{2d}(x)$$

By the induction hypothesis we get

$$x^u + 1 = \frac{\Phi_{2u}(x)\Phi_2(x)}{\Phi_u(-x)} \prod_{d|u} \Phi_d(-x) = \frac{\Phi_{2u}(x)(x + 1)}{\Phi_u(-x)(-x + 1)}((-x)^u - 1) = \frac{\Phi_{2u}(x)}{\Phi_u(-x)}(x^u + 1)$$

which means that $\Phi_{2u}(x) = \Phi_u(-x)$ and the induction step is done. We do not need a specific induction basis, since for $u = 3$ the set $1 < k < u$ for odd $k$ is empty and thus trivially fulfills the induction hypothesis. This completes the proof for when $l = 1$.

Finally, we prove that for any even $e$, $\Phi_{2e}(x) = \Phi_e(x^2)$, which will complete the proof of the statement. Again we do this by induction: Suppose the statement holds for all even numbers up to $e$. Let $e = 2^e u$ with $u$ odd. Then

$$x^{2e} - 1 = \prod_{d|2e} \Phi_d(x) = \prod_{d|e} \Phi_d(x) \prod_{d|u} \Phi_{2e+1d}(x) = (x^e - 1)\prod_{d|u} \Phi_{2e+1d}(x)$$

Using the induction hypothesis and the previous result we get

$$x^e + 1 = \frac{\Phi_{2e}(x)\Phi_{2e+1}(x)}{\Phi_u(-x^2)} \prod_{d|u} \Phi_d(-x^2) = \frac{\Phi_{2e}(x)(x^{2e} + 1)}{\Phi_e(x^2)(-x^{2e} - 1)}(-x^{2e}u - 1) = \frac{\Phi_{2e}(x)}{\Phi_e(x^2)}(x^e + 1)$$

we have that $\Phi_{2e}(x) = \Phi_e(x^2)$. Again no base case is needed since the induction hypothesis is trivially true for $e = 2$. This completes the proof.

This allows us to find the following cyclotomic polynomials with three non-zero coefficients and a degree between 256 and 512:

$$\Phi_{729}(x) = x^{486} + x^{243} + 1$$

$$\Phi_{964}(x) = x^{288} - x^{144} + 1$$

$$\Phi_{972}(x) = x^{324} - x^{162} + 1$$

$$\Phi_{1152}(x) = x^{384} - x^{192} + 1$$

$$\Phi_{1296}(x) = x^{432} - x^{216} + 1$$

$$\Phi_{1458}(x) = \Phi_{729}(-x)$$

These are also the only such polynomials for cyclotomics of index at most 3000. Out of these we may immediately discard $n = 486$ since it is too close to $n = 512$ to hope for any significant improvement. The same goes for $n = 288$ which is very close to $n = 256$.

We are also interested in designing a 192 bit system and for this purpose search for dimensions between 512 and 1024, and find the following:

$$\Phi_{1728}(x) = x^{576} - x^{288} + 1$$

$$\Phi_{1944}(x) = x^{648} - x^{324} + 1$$

$$\Phi_{2304}(x) = x^{768} - x^{384} + 1$$

$$\Phi_{2592}(x) = x^{864} - x^{432} + 1$$

$$\Phi_{2916}(x) = x^{972} - x^{486} + 1$$

For similar reasons as before, we immediately exclude $n = 576$ and $n = 972$ from the discussion.

A choice of prime $q$ must also be made. In order for the NTT to be possible we should have $q = 1 \pmod{N}$, but with $n$ not a power of two the NTT is not particularly fast anyways and in our experiments the NTT is not the preferred multiplication algorithm for polynomials of these degrees. Therefore we release this requirement on $q$. Since $d$ must be chosen so that $q = 1 \pmod{2^d-1}$, we restrict ourselves to $q = 1 \pmod{64}$ which means that $d$ up to 7 will always be allowed. This is a significant improvement since the primes with $q = 1 \pmod{N}$ are quite limited for some of our suggested moduli, see Table 11.

55
Table 11: Primes with $q = 1 \mod N$ and $q < 20000.$

<table>
<thead>
<tr>
<th>$N$</th>
<th>$n$</th>
<th>$q$</th>
<th>Maximal $d$</th>
<th>$N$</th>
<th>$n$</th>
<th>$q$</th>
<th>Maximal $d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>972</td>
<td>324</td>
<td>2917</td>
<td>3</td>
<td>1296</td>
<td>432</td>
<td>1297</td>
<td>5</td>
</tr>
<tr>
<td>3889</td>
<td>5</td>
<td></td>
<td></td>
<td>2593</td>
<td>432</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>4861</td>
<td>3</td>
<td></td>
<td></td>
<td>3889</td>
<td>4</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>9721</td>
<td>4</td>
<td></td>
<td></td>
<td>6481</td>
<td>5</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>12637</td>
<td>3</td>
<td></td>
<td></td>
<td>10369</td>
<td>8</td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>17497</td>
<td>4</td>
<td></td>
<td></td>
<td>19441</td>
<td>5</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>19441</td>
<td>5</td>
<td></td>
<td></td>
<td>19441</td>
<td>5</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>1152</td>
<td>384</td>
<td>1153</td>
<td>8</td>
<td>9721</td>
<td>4</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>3457</td>
<td>8</td>
<td></td>
<td></td>
<td>17497</td>
<td>4</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>10369</td>
<td>8</td>
<td></td>
<td></td>
<td>19441</td>
<td>5</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>18433</td>
<td>12</td>
<td></td>
<td></td>
<td>2592</td>
<td>864</td>
<td>2593</td>
<td>6</td>
</tr>
<tr>
<td>2304</td>
<td>768</td>
<td>18433</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8.3.2 Security Considerations

The security reduction from [DDLL13] does not use the form of the quotient polynomial, so with this modification Theorem 6.2 still stands. Moreover, since the quotient polynomial is still cyclotomic, the connection to SVP is as strong as before. The only real change is that the ideal lattices in these rings may differ significantly from ideal lattices in the original BLISS rings. However, there is no real reason to believe that this should be the case, and we have not found any new structure that could be exploited. Therefore there is no reason to believe that this should be any less secure than the original BLISS.

8.4 Key Generation Modification

Let us again, for some $p \in \mathcal{R}$, consider $\|xp(x)\|$: 

$$
\|xp(x)\|^2 = \|(-p_{n-1}, p_0, \ldots, p_{\frac{n}{2}} \pm p_{n-1}, \ldots, p_{n-2})\|^2 = 
\|p(x)\|^2 + p_{n-1}^2 \pm 2p_{\frac{n}{2}}p_{n-1}
$$

We are concerned with this rate of growth in relation to the signature: We know that $\|sc'\|^2 \leq \kappa \max_i \| (x'f \mid x'\cdot(2g + 1)) \|^2 = \kappa \max_i (\|x'f\|^2 + \|x'\cdot(2g + 1)) \|^2).$ In the original BLISS schemes these norms were all the same, but here they could potentially grow quite a lot.

We solve this by introducing a modification to the key generation algorithm: For all $k < \frac{n}{2}$, we choose to disqualify keys where $f_k$ and $f_{\frac{n}{2} + k}$ ($g_k$ and $g_{\frac{n}{2} + k}$ resp.) are such that the last term in the expression above is positive. In other words, if the quotient polynomial is $x^n - x^{n/2} + 1$ we do not allow these to have the same sign, while if the quotient polynomial is $x^n + x^{n/2} + 1$ we do not allow them to have opposite signs.

Clearly this keeps down the norms of $x^k f$ and $x^k (2g + 1)$ for $k < \frac{n}{2}$. But what about larger $k$? Then each pair of coefficients “interact again”, but they have changed: That which was once $f_{\frac{n}{2} + k}$ has changed its sign, and that which was once $f_k$ now has the value $f_k \pm f_{\frac{n}{2} + k}$. The new pair of coefficients becomes $- (f_k \pm f_{\frac{n}{2} + k})$ and $- f_{\frac{n}{2} + k} \pm (f_k \pm f_{\frac{n}{2} + k}) = \pm f_k$ (since the choice of sign is the same in the two $\pm$’s). Therefore clearly the choice of signs that was made earlier still keeps the norm down.

Practically this is easily implemented by iteratively choosing the indices for the $\pm 1$ and $\pm 2$ coefficients. If there is already another coefficient set that forces the sign of this coefficient then just follow that, otherwise, choose the sign randomly.
How does this choice affect the entropy? Remember that the old key generation gave
\[
2^{d_1 + d_2 \left( \binom{n}{d_1} \binom{n-d_1}{d_2} \right)} = 2^{d_1 + d_2 \left( \binom{n}{d_1 + d_2} \binom{d_1 + d_2}{d_1} \right)}
\]
possible values for \( f \) and \( g \) respectively. The new key generation can be seen as follows:
The situation where both \( f_k \) and \( f_{2+k} \) (or \( g_k \) and \( g_{2+k} \)) are non-zero, we call a coincidence. We first choose the number \( c \) of coincidences, and then choose the \( d_1 + d_2 - c \) indices \( k \) in \([0, \frac{n}{2}]\) such that at least one of \( f_k \) and \( f_{2+k} \) is non-zero. Then out of these \( d_1 + d_2 - c \) indices, choose which \( c \) represent coincidences, and for the remaining \( d_1 + d_2 - 2c \), choose whether it is \( f_k \) or \( f_{2+k} \) that is non-zero. Then make \( d_1 + d_2 - c \) choices of signs, since we lose one choice per coincidence, and finally choose which \( d_1 \) of the \( d_1 + d_2 \) non-zero indices shall hold \( \pm 1 \).

The total number of possible \( f \) (and \( g \) resp.) is then
\[
\sum_{c=0}^{(d_1+d_2)/2} \binom{n/2}{d_1 + d_2 - c} \binom{d_1 + d_2}{c} 2^{d_1 + d_2 - 2c} \binom{d_1 + d_2 - c}{d_1 + d_2 - c} = \binom{n/2}{d_1 + d_2} 2^{(d_1+d_2)/2} \sum_{c=0}^{(d_1+d_2)/2} \binom{n/2}{d_1 + d_2 - c} \binom{d_1 + d_2}{c} 2^{-3c}
\]
Comparing this to the original value, the factor \( \binom{n}{d_1+d_2} \) is exchanged for the more complicated expression
\[
2^{d_1 + d_2} \sum_{c=0}^{(d_1+d_2)/2} \binom{n/2}{d_1 + d_2 - c} \binom{d_1 + d_2 - c}{c} 2^{-3c}
\]
Clearly the decrease in entropy depends only on the sum \( d_1 + d_2 \) of non-zero entries. We can plot this for the considered dimensions, which is done in Figure 5. Clearly the entropy loss is quite low up to \( d_1 + d_2 = n/3 \) but then increases quickly, and for \( d_1 + d_2 > n/2 \) it is quite large.

Loss in entropy can be exploited by the MiM attack, and therefore by the hybrid attack. The question is whether this specific entropy loss is of relevance to the MiM attack. As it has been explained in [DDLL13], the MiM attack consists of indexing all combinations of some consecutive coefficients, and then searching for a match among the remaining coefficients. If the entropy loss would be in the relationship between coefficients close to one another, this could save time or space since less combinations of coefficients need be searched. But in this case the coefficients that are related to one another are far apart, so it can not be used to decrease the number of coefficient combinations to be stored or searched. For this reason we shall keep the old per-coefficient entropy expression \( e = -(\delta_0 \log_2 \delta_0 + \delta_1 \log_2 \frac{\delta_1}{\delta_0} + \delta_2 \log_2 \frac{\delta_2}{\delta_1}) \) in the hybrid attack computations despite this change.

However, despite these precautions there will be no practical common bound on \( \|x's\| \), which is needed to find the repetition factor \( M \). For each key the maximum can easily be computed, but we will need to add a rejection step in the key generation in order to omit very unfortunate keys. Therefore, we propose the new key generation algorithm in Algorithm 6. The new parameter \( B_\nu \) will be the maximum value of \( \nu \). We shall determine this value experimentally so that 1/2 of the keys are rejected, which gives an entropy loss of at most one bit.
Note also that we retain a limit on $\|v_1\|_\infty$ from BLISS, namely $\|v_1\|_\infty \leq 2\kappa$. In the power-of-two case, each coefficient in $v_1$ is a sum of $\kappa$ terms in $\{-2, -1, 0, 1, 2\}$, which gives the desired result. With our modified key generation this property is preserved even in the suggested new dimensions, as per the previous discussion. Since $\|z_1\| \leq B_\infty$ and $y_1 = \pm(z_1 - v_1)$ we know that $\|y_1\|_\infty \leq B_\infty + 2\kappa$ – or rather, any other $y_1$ can never give rise to a valid signature. (If $\delta_2 = 0$, $2\kappa$ is replaced by $\kappa$.)

8.5 Implementation Tricks

8.5.1 GMP Packing for Multiplication

We use the following trick to compute polynomial products $p(x)h(x)$ modulo $f(x)$, where $f(x)$ has low Hamming weight, so the modulo operation is $O(n)$. We employ the GMP library that implements all the best known multiplication algorithms with assembly optimisations. We will show how to perform a polynomial multiplication modulo $q$ using GMP’s multiplication of big numbers, and propose some improvements as well.

The GMP type for large numbers is $\texttt{mpz} \_\texttt{t}$, and though its internal structure is hidden from the user, it is represented as 64-bit limbs, accessed as $\texttt{P[0]}.\texttt{mp}_\texttt{d}[i]$. The sign and number of limbs is stored in $\texttt{P[0]}.\texttt{mp}_\texttt{size}$. Thus, if we can guarantee that the coefficients of the product $p(x)h(x)$ are less than $2^{t-1}$ in absolute value, reserving one bit for the sign, we can pack polynomials into GMP numbers as $P = \sum_{i=0}^{n-1} p_i \cdot 2^i$.

The packing algorithm can be realised efficiently where each limb of a GMP number is constructed directly, and this operation is linear. The 64-bit limbs of $P$ are then viewed as one continuous bit string which is split into “slots” of $t$ bits each. Slot $i$ is filled with the signed value of $p_i$, and the sign propagates to all the higher slots. (In reality one rather keeps a variable with the sign value that “switches” from one slot to another, accumulating all previous sign propagations.) In our implementation we use packings with $t = 32$ and $t = 36$.

In (RE)BLISS we know that $y_1$ and $z_1$ have small signed coefficients. Then the coefficients of the product $\zeta a_1 y_1$ (and $\zeta a_1 z_1$) are generally small. Since $\|a_1\|_\infty \leq q$ (the
where from the definition of $e$ properly chosen the packing can be guaranteed to work. The price is a potentially lower add a rejection step for the max-norm in the signature algorithm.

Indeed, if the rejection sampling in Figure 6a is taken to be done in two steps where first the tails of the modes of the bimodal distribution are cut off and then the rest, this second step is exactly what is depicted in Fig 6b while the first step could just as well be performed together with the generation of $y$ – which means that with parameters properly chosen the packing can be guaranteed to work. The price is a potentially lower $B_{\infty,1}$ which means that the rejection rate might go up since fewer signatures are accepted. This effect will have to be weighed against the advantage of faster multiplication. We add a rejection step for the max-norm in the signature algorithm.
We introduce $D_{\sigma}^{x_{\infty}}_{|B}$ as the discrete probability distribution on $\mathbb{Z}^m$ with p.d.f

$$p(x) = \begin{cases} C \exp \left( -\frac{\|x\|^2}{2\sigma^2} \right) & \|x\|_{\infty} \leq B \\ 0 & \|x\|_{\infty} > B \end{cases}$$

where $C$ is a normalisation constant. Then we take $y_1 \leftarrow D_{\sigma}^{x_{\infty}}_{|B} + 2\kappa$ but keep $y_2 \leftarrow D_{\sigma}$ and rejection sample so that $z_1$ mimics $D_{\sigma}^{x_{\infty}}_{|B,1}$, which is in practice already done in BLISS except with a common $B_{\infty}$.

### 8.5.2 Faster GreedySC

After the improved multiplication, the by far heaviest operation is the computation of $v$ because of the large number of scalar products and the involved multiplications.

However, as we have already noticed the coefficients of $x^i f$ are between $-2$ and $2$, and the coefficients of $x^i (2g + 1)$ are, for the same reason, between $-4$ and $5$. Since the number of possible values is so small, it is possible to first sum up all equal coefficients that are included in the scalar product and then perform the multiplication. This leads to a significant speedup which has also been implemented.

Also note that when $\delta_2 = 0$ we get a similar gain as in the GMP packing: Now suddenly the coefficients of $x^i f$ are between $-1$ and $1$ while the coefficients of $x^i (2g + 1)$ are between $-2$ and $3$. This can be exploited to significantly lower the

### 8.6 REBLISS

#### 8.6.1 Parameters for REBLISS

The parameter selection method for REBLISS differs from that in Section 8.1 in two ways: Step 4 we experimentally determine $B_v$ so that half of the keys are rejected in the key generation. Secondly, in the same step we can compute a maximum $B_{\infty,1}$ for the desired GMP packing. When $\sigma$ is determined the extra rejection probability a low $B_{\infty,1}$ gives can be computed.

We manage to obtain 128 bit security with $n = 432$, with parameters as shown in Table 12. We also examine $n = 384$, but despite trying several different primes for $q$ it is impossible to get a 128 bit secure system, without having a ridiculously high $M$ (well over 100).

We note a tendency that smaller $q$ admit smaller $\delta_1, \delta_2$. The effect is significant, so for fairness of comparison $n = 512$ should be revisited with a freer choice of $q$. While the rejection step in the secret key generation does nothing in the power-of-two case, the
Table 12: REBLISS instances\(^1\)

<table>
<thead>
<tr>
<th>Set</th>
<th>512a</th>
<th>512b</th>
<th>432</th>
<th>512c</th>
<th>648</th>
<th>768</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\lambda)</td>
<td>128</td>
<td>128</td>
<td>128</td>
<td>160</td>
<td>192</td>
<td>192</td>
</tr>
<tr>
<td>(N)</td>
<td>1024</td>
<td>1024</td>
<td>1296</td>
<td>1024</td>
<td>1944</td>
<td>2304</td>
</tr>
<tr>
<td>(n)</td>
<td>512</td>
<td>512</td>
<td>432</td>
<td>512</td>
<td>648</td>
<td>768</td>
</tr>
<tr>
<td>(q)</td>
<td>12289</td>
<td>3329</td>
<td>5953</td>
<td>12289</td>
<td>3329</td>
<td>3329</td>
</tr>
<tr>
<td>(d_1, d_2)</td>
<td>200, 6</td>
<td>149, 0</td>
<td>204, 26</td>
<td>251, 67</td>
<td>214, 0</td>
<td>162, 0</td>
</tr>
<tr>
<td>(\kappa)</td>
<td>58</td>
<td>58</td>
<td>64</td>
<td>82</td>
<td>95</td>
<td>86</td>
</tr>
<tr>
<td>(B_\nu)</td>
<td>254</td>
<td>208</td>
<td>321</td>
<td>461</td>
<td>349</td>
<td>305</td>
</tr>
<tr>
<td>(\sigma)</td>
<td>260</td>
<td>185</td>
<td>148</td>
<td>217</td>
<td>210</td>
<td>194</td>
</tr>
<tr>
<td>(d)</td>
<td>8</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>(B_2)</td>
<td>19993</td>
<td>11184</td>
<td>5876</td>
<td>9404</td>
<td>13658</td>
<td>44921</td>
</tr>
<tr>
<td>(B_{\infty,1}, B_{\infty,2})</td>
<td>2943, 2943</td>
<td>832, 799</td>
<td>707, 1471</td>
<td>3039, 3039</td>
<td>832, 799</td>
<td>753, 799</td>
</tr>
<tr>
<td>GMP packing</td>
<td>- (36)</td>
<td>32</td>
<td>32</td>
<td>- (36)</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>(M)</td>
<td>1.61</td>
<td>1.88</td>
<td>10.51</td>
<td>9.55</td>
<td>3.98</td>
<td>3.44</td>
</tr>
<tr>
<td>Total rej.</td>
<td>1.63</td>
<td>1.93</td>
<td>10.56</td>
<td>9.55</td>
<td>4.93</td>
<td>3.93</td>
</tr>
<tr>
<td>PK size (bits)</td>
<td>6956</td>
<td>5991</td>
<td>5417</td>
<td>6956</td>
<td>7582</td>
<td>8986</td>
</tr>
<tr>
<td>Sign entropy (bits)</td>
<td>6471</td>
<td>6992</td>
<td>6094</td>
<td>9147</td>
<td>10594</td>
<td>9826</td>
</tr>
<tr>
<td>Sign (ms)</td>
<td>0.16</td>
<td>0.16</td>
<td>0.87</td>
<td>1.12</td>
<td>0.65</td>
<td>0.53</td>
</tr>
<tr>
<td>Verification (ms)</td>
<td>0.058</td>
<td>0.051</td>
<td>0.044</td>
<td>0.057</td>
<td>0.070</td>
<td>0.084</td>
</tr>
</tbody>
</table>

GMP packing may well compensate for the loss of the NTT. This results in the system 512b in Table 12.

It is also of interest to produce a system with 192 bit security, since this security level is often required for long-term signature support. Even with more freely chosen \(q\) this is out of reach with \(n = 512\). We therefore make a try with \(N = 1944\) and \(n = 648\), which is successful. We also examine \(n = 768\) for \(\lambda = 192\), which results in yet another feasible scheme.

In the process of choosing parameters we have prioritised keeping signature and public key sizes down in the different dimensions, while not letting the runtime get out of hand. We believe that REBLISS is fast enough to be able to support smaller signatures and keys, and that this is important since there are many applications where very short messages have to be signed, so that a signature of several Kib may increase the message size dramatically. Such applications will probably become more and more common with the growing Internet of Things. Because of the big gain that comes from setting \(\delta_2 = 0\) we have done so when possible, which has usually required quite a low \(q\), though lower \(q\)'s can be problematic since they force both \(B_2\) and \(B_{\infty,i}\) down.

Exact choice of parameters involves a weigh-off between speed and size that is quite free. Therefore systems could easily be tailored to different environments, and we do not claim that our system parameters are generally optimal. Therefore it is important to understand the process for parameter choices, so that this can be redone in order to fit the application.

### 8.7 Implementation and Benchmarks

REBLISS has been implemented using C++ with the GMP library. Random numbers are generated by the Ziggurat RNG. Runtimes are in Table 12. The total rejection rate (\(M\)-rejection and \(B_{\infty}\)-rejection) is experimentally measured because of the unpredictable

---

\(^1\)Benchmarks done on an Intel Core i7-4800MQ CPU, 2.70GHz, single-threaded.
behaviour of $z^2$. Systems 512a and 512c, which were not designed for the GMP packing, work fine with packing into 36 bits.

We find that the GMP multiplication, especially with some packing, is very efficient. Though this is one of the theoretically heaviest steps in the signature algorithm, it takes only about 0.2 ms/signature. The accelerated computation of $v$ also has a big impact on the runtime. All in all, if we take into account the lower CPU frequency and the increased $M$, $\kappa$ our implementation is even better than the original BLISS implementation in Table 1.

8.8 Conclusion

We find that while BLISS lacks in security compared to what is presented in [DDLL13], it is possible to save this by altering parameters. We also find that it is clearly possible, and maybe advantageous, to use non-power-of-two dimensions. Such systems are slower because of the worse geometry, but thanks to our new multiplication algorithm the loss of the NTT is not a problem.

We see also that a lower dimension seems to correspond quite exactly with a lower verification time. This may be of interest in a usecase where most of the signing is done by a powerful device while most of the verification is done on a constrained device.

Ultimately the choice of parameters, including dimension and modulus, will have to come down to prioritisation based on the intended usage. We have shown both how runtimes can be kept down (systems 512a and 512b) and how object sizes can be decreased at the cost of runtime (system 432). The desired weigh-off could well be somewhere in between.

However, attacks against lattice-based cryptosystems are still developing rapidly and for this reason we do not find it recommendable to use REBLISS (or any of its BLISS cousins) yet. Once the development has calmed down, new parameters may be selected using the method in this paper. Because of the increased flexibility in $n$ and $q$ it is very likely that appropriate parameters can be found.

Complete descriptions of our modified algorithms can be found in Algorithms 6-8.

References


**Algorithm 7: REBLISS Signature Algorithm**

**Data:** Message $\mu$, public key $a = (a_1, q - 2) \in \mathbb{R}_{2q}^{1 \times 2}$, $\zeta = 1/(q - 2) \in \mathbb{Z}_{2q}$, secret key $s = (s_1, s_2)^T \in \mathbb{R}_{2q}^{2 \times 1}$

**Result:** A signature $(z_1, z_2^\dagger, c)$ of $\mu$

1. $\hat{\mu} \leftarrow H_M(\mu)$
2. Sample $y_1, y_2 \in \mathcal{R}$, $y_1 \leftarrow D_{\sigma}^n|_{B_{\infty,1} + j_2}$ (with $j = 2$ unless $\delta_2 = 0$, then $j = 1$), $y_2 \leftarrow D_{\sigma}^n$
3. $u \leftarrow \zeta a_1 y_1 + y_2 \mod 2q$
4. $\hat{c} \leftarrow H_i(\lfloor u \rfloor_d \mod p, \hat{\mu})$
5. $c \leftarrow H_o(\hat{c})$
6. $v = \text{GreedySC}(S, c)$ // $S$ is the matrix representation of $s$
7. Sample $b \in \{\pm 1\}$ uniformly
8. $(z_1, z_2) \leftarrow (y_1, y_2) + bv$
9. with probability $1 - 1/M \exp\left(-\frac{\|v\|_2^2}{2\sigma^2}\right) \cosh\left(\frac{\langle z, v \rangle}{\sigma^2}\right)$ go to 2
10. if $\|z_1\|_\infty > B_{\infty,1}$ go to 2
11. $z_2^\dagger = (\lfloor u \rfloor_d - \lfloor u - z_2 \rfloor_d) \mod p$
12. if $2^d\|z_2^\dagger\|_\infty > B_{\infty,2}$ go to 2

**Output:** $(z_1, z_2^\dagger, \hat{c})$

---

**Algorithm 8: REBLISS Signature Verification**

**Data:** Message $\mu$, public key $a = (a_1, q - 2) \in \mathbb{R}_{2q}^{1 \times 2}$, signature $(z_1, z_2^\dagger, \hat{c})$

**Result:** Accept or Reject signature

1. if $\|(z_1 | 2^d z_2^\dagger)\| > B_2$ then reject
2. if $\|z_1\|_\infty > B_{\infty,1}$ or $2^d\|z_2^\dagger\|_\infty > B_{\infty,2}$ then reject
3. if $\hat{c} = H_i(\lfloor \zeta a_1 z_1 + \zeta q H_o(\hat{c}) \rfloor_d + z_2^\dagger \mod p, H_M(\mu))$ then accept, else reject
REFERENCES


64
REFERENCES


REFERENCES


REFERENCES


