Lattice Signatures and Bimodal Gaussians

Léo Ducas* and Alain Durmus** and Tancrède Lepoint† and Vadim Lyubashevsky‡

{Leo.Ducas, Alain.Durmus, Tancrede.Lepoint, Vadim.Lyubashevsky}@ens.fr

Abstract. Our main result is a construction of a lattice-based digital signature scheme that represents an improvement, both in theory and in practice, over today's most efficient lattice schemes. The novel scheme is obtained as a result of a modification of the rejection sampling algorithm that is at the heart of Lyubashevsky's signature scheme (Eurocrypt, 2012) and several other lattice primitives. Our new rejection sampling algorithm which samples from a *bimodal* Gaussian distribution, combined with a modified scheme instantiation, ends up reducing the standard deviation of the resulting signatures by a factor that is asymptotically square root in the security parameter. The implementations of our signature scheme for security levels of 128, 160, and 192 bits compare very favorably to existing schemes such as RSA and ECDSA in terms of efficiency. In addition, the new scheme has shorter signature and public key sizes than all previously proposed lattice signature schemes.

As part of our implementation, we also designed several novel algorithms which could be of independent interest. Of particular note, is a new algorithm for efficiently generating discrete Gaussian samples over $\mathbb{Z}^n$. Current algorithms either require many high-precision floating point exponentiations or the storage of very large pre-computed tables, which makes them completely inappropriate for usage in constrained devices. Our sampling algorithm reduces the hard-coded table sizes from linear to logarithmic as compared to the time-optimal implementations, at the cost of being only a small factor slower.

1 Introduction

Lattice cryptography is arguably the most promising replacement for standard cryptography after the eventual coming of quantum computers. The most ubiquitous public-key cryptographic primitives, encryption schemes [HPS98,LPR10] and digital signatures [Lyu12,GLP12], already have somewhat practical lattice-based instantiations. In addition, researchers are rapidly discovering new lattice-based primitives, such as fully-homomorphic encryption [Gen09], multi-linear maps [GGH13], and attribute-based encryption [GVW13], that had no previous constructions based on classical number-theoretic techniques. Even though the above primitives are quite varied in their functionalities, many of them share the same basic building blocks. Thus an improvement in one of these fundamental building blocks, usually results in the simultaneous improvement throughout lattice cryptography. For example, the recent work on the lattice trapdoor generation algorithm [MP12] resulted in immediate efficiency improvements in lattice-based hash-and-sign signatures, identity-based encryption schemes, group signatures, and functional encryption schemes.

In this work, we propose an improvement of another such building block – the *rejection sampling* procedure that is present in the most efficient constructions of lattice-based digital signatures [Lyu12,GLP12], authentication schemes [Lyu09], blind signatures [Rüc10], and zero-knowledge proofs used in multi-party computation [DPSZ12]. As a concrete application, we show that with our new algorithm, lattice-based digital signatures become completely practical. We construct and implement a family of digital signature schemes, named BLISS (Bimodal Lattice Signature Scheme)

* ENS Paris, France.
** ENPC and ENS Cachan, France. This work was done while the author was at ENS Paris, France.
† CryptoExperts and ENS Paris, France.
‡ INRIA and ENS Paris, France.

Implementation	Security	Signature Size	SK Size	PK Size	Sign (ms)	Sign/s	Verify (ms)	Verify/s
BLISS-0	$\leqslant$ 60 bits	3.3 kb	1.5 kb	3.3 kb	0.241	4k	0.017	59k
BLISS-I	128 bits	5.6 kb	2 kb	7 kb	0.124	8k	0.030	33k
BLISS-II	128 bits	5 kb	2 kb	7 kb	0.480	2k	0.030	33k
BLISS-III	160 bits	6 kb	3 kb	7 kb	0.203	5k	0.031	32k
BLISS-IV	192 bits	6.5 kb	3 kb	7 kb	0.375	2.5k	0.032	31k
RSA 1024	72-80 bits	1 kb	1 kb	1 kb	0.167	6k	0.004	91k
RSA 2048	103-112 bits	2 kb	2 kb	2 kb	1.180	0.8k	0.038	27k
RSA 4096	$\geqslant$ 128 bits	4 kb	4 kb	4 kb	8.660	0.1k	0.138	7.5k
ECDSA[1] 160	80 bits	0.32 kb	0.16 kb	0.16 kb	0.058	17k	0.205	5k
ECDSA 256	128 bits	0.5 kb	0.25 kb	0.25 kb	0.106	9.5k	0.384	2.5k
ECDSA 384	192 bits	0.75 kb	0.37 kb	0.37 kb	0.195	5k	0.853	1k

Table 1. Benchmarking on a desktop computer (Intel Core i7 at 3.4Ghz, 32GB RAM) with `openssl` 1.0.1c

for security levels of $128, 160$, and 192 bits. On standard 64-bit processors, our proof-of-concept implementations, available at [DL] under license CeCILL, constitute significant improvements over previous lattice-based signatures and compare very favorably to the `openssl` implementations of RSA and ECDSA signatures schemes (see Table 1).

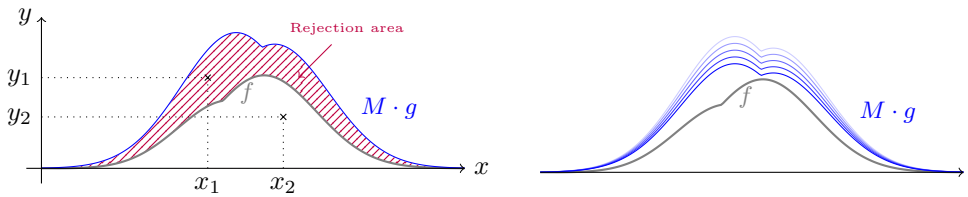
As part of our implementation, we also designed several novel algorithms that could be of independent interest. Chiefly among them is a new procedure that very efficiently samples from the Gaussian distribution over $\mathbb{Z}^m$ without requiring a very large look-up table. The absence of such an algorithm made researchers avoid using the Gaussian distribution when implementing lattice-based schemes on constrained devices, which resulted in these schemes being less compact than they could have been [GLP12].

1.1 Related Work

Rejection Sampling. Rejection sampling in lattice constructions was first used by Lyubashevsky [Lyu08] to construct a three-round identification scheme. A standard identification scheme is a three round sigma protocol that consists of a commit, challenge, and response stages. The main idea underlying their constructions and security proofs from number theoretic assumptions (e.g. Schnorr and GQ schemes [BP02]) is that the value y committed to in the first stage is used to information-theoretically hide the secret key s in the third stage. This is relatively straight-forward to do in number-theoretic schemes because one can just commit to a random y and then add it to (or multiply it by) some challenge-dependent function of s. Since all operations are performed in a finite ring, y being uniformly random hides s. In lattice constructions, however, we need to hide the secret key with a *small* y. The solution is thus to choose y from a narrow distribution and then perform rejection sampling so that s is not leaked when we add y to it (we describe this idea in much greater detail in Section 1.2). The improvements in lattice-based identification schemes (and therefore signature schemes via the Fiat-Shamir transformation) partly came via picking distributions that were more amenable to rejection sampling.

Lattice Signatures. Early lattice-based signature proposals did not have security reductions [GGH97,HPS01,HNHGSW03], and they were all subsequently broken because it turned out that

[1] ECDSA on a prime field $\mathbb{F}_p$: `ecdsap160`, `ecdsap256` and `ecdsap384` in `openssl`.



(a) (x_i, y_i) is sampled uniformly in the area under $M \cdot g$, and accepted when $y_i \leqslant f(x_i)$

(b) M can be reduced when g is better adapted to f

Fig. 1. Rejection sampling from the distribution of g to get the distribution of f

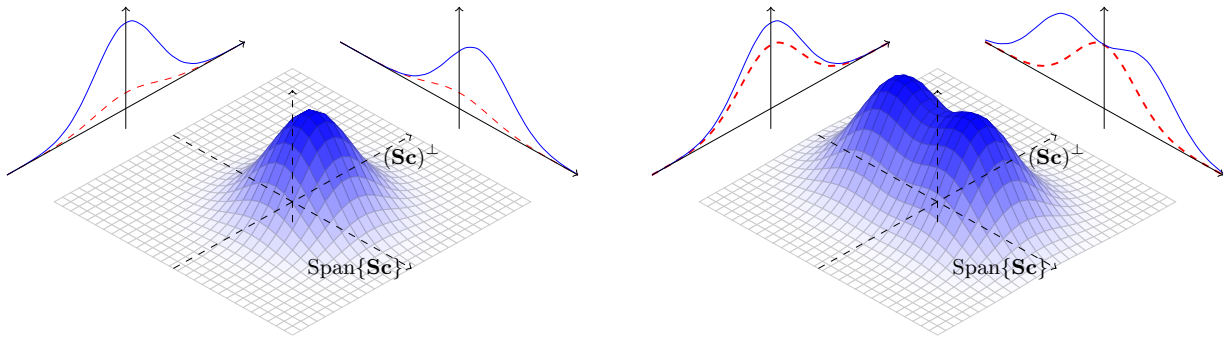
every signature leaked a part of the secret key [GS02,NR09,DN12b]. Among known provably-secure signature schemes, [GPV08,Lyu09], [Lyu12,MP12], the most efficient seems to be that of [Lyu12] whose most efficient instantiation has both signature and key size of the order of 9kb [GLP12] for approximately 80 bits of security.[2]

1.2 Our Results and Techniques

Rejection Sampling and Signature Construction. To understand our improvement of the rejection sampling procedure, we believe that it is useful to first give an overview of rejection sampling and the most efficient way in which it is currently used in constructing lattice-based signatures [Lyu12]. Rejection sampling is a well-known method introduced by von Neumann [vN51] to sample from an arbitrary target probability distribution f, given a source bound to a different probability distribution g. Conceptually, the method works as follows. A sample x is drawn from g and is accepted with probability $f(x)/(M \cdot g(x))$, where M is some positive real. If it is not accepted, then the process is restarted. It is not hard to prove that if $f(x) \leqslant M \cdot g(x)$ for all x, then the rejection sampling procedure produces exactly the distribution of f. Furthermore, because the expected number of times the procedure will need to be restarted is M, it is crucial to keep M as small as possible, possibly by tailoring the function g so that it resembles the target function f as much as possible. In particular, since rejection sampling can be interpreted as sampling a random point (x_i, y_i) in the area under the distribution $M \cdot g$ (see Figure 1) and accepting if and only if $y_i \leqslant f(x_i)$, reducing the area between the two curves will reduce M.

The digital signature from [Lyu12] works as follows (for the sake of this discussion, we will present the simplest version based on SIS): the secret key is an $m \times n$ matrix $\mathbf{S}$ with small coefficients, and the public key consists of a random $n \times m$ matrix $\mathbf{A}$ whose entries are uniform in $\mathbb{Z}_q$ and $\mathbf{T} = \mathbf{AS} \bmod q$. There is also a cryptographic hash function H, modeled as a random oracle, which outputs elements in $\mathbb{Z}^n$ with small norms. To sign a message digest μ, the signing algorithm first picks a vector $\mathbf{y}$ according to the distribution D_σ^m, where D_σ^m is the discrete Gaussian distribution over $\mathbb{Z}^m$ with standard deviation σ. The signer then computes $\mathbf{c} = H(\mathbf{Ay} \bmod q, \mu)$ and produces a potential signature $(\mathbf{z}, \mathbf{c})$ where $\mathbf{z} = \mathbf{Sc} + \mathbf{y}$. Notice that the distribution of $\mathbf{z}$ depends on the distribution of $\mathbf{Sc}$, and thus on the distribution of $\mathbf{S}$ – in fact, the distribution of $\mathbf{z}$ is exactly D_σ^m *shifted* by the vector $\mathbf{Sc}$.

[2] In [GLP12], a 100-bit security level was claimed, but the cryptanalysis we use in this paper, which combines lattice-reduction attacks with combinatorial meet-in-the-middle techniques [HG07], estimates the actual security to be around 75-80 bits.



(a) In the original scheme of [Lyu12]

(b) In our scheme

Fig. 2. Improvement of Rejection Sampling with Bimodal Gaussian Distributions. In blue is the distribution of $\mathbf{z}$, for fixed $\mathbf{Sc}$ and over the space of all $\mathbf{y}$ in Figure (a) and all $(b, \mathbf{y})$ in Figure (b), before the rejection step and its decomposition as a Cartesian product over Span$\{\mathbf{Sc}\}$ and $(\mathbf{Sc})^\perp$. In dashed red is the target distribution scaled by $1/M$.

To remove the dependence of the signature on $\mathbf{S}$, rejection sampling is used. The target distribution that we want for signatures is D^m_σ, whereas we obtain samples from the distribution D^m_σ shifted by $\mathbf{Sc}$ (call this distribution $D^m_{\mathbf{Sc},\sigma}$). To use rejection sampling, we need to find a positive real M such that for all (or all but a negligible fraction) $\mathbf{x}$ distributed according to D^m_σ we have $D^m_\sigma(\mathbf{x}) \leqslant M \cdot D^m_{\mathbf{Sc},\sigma}(\mathbf{x})$. A simple calculation (see [Lyu12, Lemma 4.5]) shows that

$$D^m_\sigma(\mathbf{x})/D^m_{\mathbf{Sc},\sigma}(\mathbf{x}) = \exp\left(\frac{-2\langle \mathbf{x}, \mathbf{Sc}\rangle + \|\mathbf{Sc}\|^2}{2\sigma^2}\right). \tag{1}$$

The value of $\langle \mathbf{x}, \mathbf{Sc}\rangle$ behaves in many ways as a one-dimensional discrete Gaussian, and it can be thus shown that $|\langle \mathbf{x}, \mathbf{Sc}\rangle| < \tau\sigma\|\mathbf{Sc}\|$ with probability $1 - \exp(-\Omega(\tau^2))$. Asymptotically, the value of τ is proportional to the square root of the security parameter. Concretely, if we would like to have, for example, $1-2^{-100}$ certainty that $|\langle \mathbf{x}, \mathbf{Sc}\rangle| < \tau\sigma\|\mathbf{Sc}\|$, we would set $\tau = 12$. Thus with probability $1 - \exp(-\Omega(\tau^2))$, we have $\exp\left(\frac{-2\langle \mathbf{x},\mathbf{Sc}\rangle + \|\mathbf{Sc}\|^2}{2\sigma^2}\right) \leqslant \exp\left(\frac{2\tau\sigma\|\mathbf{Sc}\| + \|\mathbf{Sc}\|^2}{2\sigma^2}\right)$. So if $\sigma = \tau\|\mathbf{Sc}\|$, we will have $D^m_\sigma(\mathbf{x})/D^m_{\mathbf{Sc},\sigma}(\mathbf{x}) \leqslant \exp\left(1 + \frac{1}{2\tau^2}\right)$. Therefore if we set $M = \exp\left(1 + \frac{1}{2\tau^2}\right)$, rejection sampling outputs signatures that are distributed according to D^m_σ where $\sigma = \tau\|\mathbf{Sc}\|$ and the expected number of repetitions is $M \approx \exp(1)$.[3]

Prior to explaining our technique to improve the scheme, we need to state how the verification algorithm in [Lyu12] works. Upon receiving the signature $(\mathbf{z}, \mathbf{c})$ of μ, the verifier checks that $\|\mathbf{z}\|$ is "small" (roughly $\sigma\sqrt{m}$) and also that $\mathbf{c} = H(\mathbf{Az} - \mathbf{Tc} \bmod q, \mu)$. It is easy to check that the outputs of the signing procedure satisfy the two requirements. In this work, we show how to remove the factor τ (in fact even more) from the required standard deviation. Above, we described how to perform rejection sampling when we were sampling potential signatures as $\mathbf{z} = \mathbf{Sc} + \mathbf{y}$. Consider now, an alternative procedure, where we first uniformly sample a bit $b \in \{-1, 1\}$ and then choose the potential signature to be $\mathbf{z} = b\mathbf{Sc} + \mathbf{y}$. In particular $\mathbf{z}$ is now sampled from the distribution $\frac{1}{2}D^m_{\mathbf{Sc},\sigma} + \frac{1}{2}D^m_{-\mathbf{Sc},\sigma}$. If our target distribution is still D^m_σ, then, as above, we need to have $D^m_\sigma(\mathbf{x})/\left(\frac{1}{2}D^m_{\mathbf{Sc},\sigma}(\mathbf{x}) + \frac{1}{2}D^m_{-\mathbf{Sc},\sigma}(\mathbf{x})\right) \leqslant M$. By using Equation (1) and some algebraic manipulations

[3] More precisely $\sigma = \tau \max_{\mathbf{S},\mathbf{c}} \|\mathbf{Sc}\|$, since $\mathbf{Sc}$ is not known in advance.

(see Section 3.2), we obtain that

$$D_\sigma^m(\mathbf{x}) / \left(\frac{1}{2}D_{\mathbf{Sc},\sigma}^m(\mathbf{x}) + \frac{1}{2}D_{-\mathbf{Sc},\sigma}^m(\mathbf{x})\right) = \exp\left(\frac{\|\mathbf{Sc}\|^2}{2\sigma^2}\right) / \cosh\left(\frac{\langle \mathbf{x}, \mathbf{Sc}\rangle}{\sigma^2}\right) \leqslant \exp\left(\frac{\|\mathbf{Sc}\|^2}{2\sigma^2}\right), \quad (2)$$

where the last inequality follows from the fact that $\cosh(y) \geqslant 1$ for all y. Thus for rejection sampling to work with $M = \exp(1)$, as in the previous example, we only require that $\sigma = \|\mathbf{Sc}\|/\sqrt{2}$ rather than $\tau\|\mathbf{Sc}\|$.

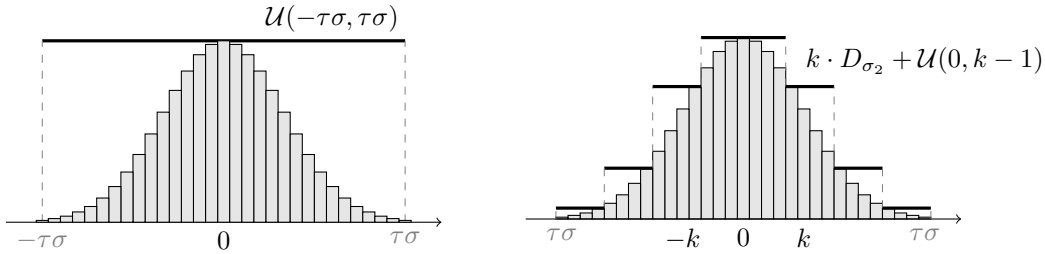
Our improvement is depicted in Figure 2. Part 2(a) shows the rejection sampling as done in [Lyu12]. There, the distribution D_σ^m (the dashed red line) must be scaled by a somewhat large factor so that all but a negligible fraction of it fits under $D_{\mathbf{Sc},\sigma}^m$. In 2(b), which represents our improved sampling algorithm, the distribution from which we are sampling is bimodal having its two centers at $\mathbf{Sc}$ and $-\mathbf{Sc}$. As can be seen from the figure, the distribution D_σ^m fits much "better" (*i.e.* needs to be scaled by a much smaller factor) underneath the bimodal distribution and therefore there is a much smaller rejection area between the two curves. As a side note, whereas in (a), a negligible fraction of the scaled D_σ^m is still above $D_{\mathbf{Sc},\sigma}^m$, in (b), all of D_σ^m is underneath the bimodal distribution $\frac{1}{2}D_{\mathbf{Sc},\sigma}^m + \frac{1}{2}D_{-\mathbf{Sc},\sigma}^m$.

While the above sampling procedure potentially produces much shorter signatures since the Gaussian "tail-cut" factor τ is never used, it does not give an improved signature scheme by itself because the verification procedure is no longer guaranteed to work. The verification checks that $\mathbf{c} = H(\mathbf{Az} - \mathbf{Tc} \bmod q, \mu)$ and so will verify correctly if and only if $\mathbf{Ay} = \mathbf{Az} - \mathbf{Tc} = \mathbf{A}(b\mathbf{Sc} + \mathbf{y}) - \mathbf{Tc} = \mathbf{Ay} + b\mathbf{Tc} - \mathbf{Tc}$, which will only happen if $b\mathbf{Tc} = \mathbf{Tc} \bmod q$ for $b \in \{-1, 1\}$. In other words, we will need $\mathbf{Tc} = -\mathbf{Tc} \bmod q$, which will never happen if q is prime unless $\mathbf{T} = \mathbf{0}$. [4] Our solution, therefore, is to work modulo $2q$ and to set $\mathbf{T} = q\mathbf{I}$ where $\mathbf{I}$ is the $n \times n$ identity matrix. In this case $\mathbf{Tc} = -\mathbf{Tc} \bmod 2q$, and so the verification procedure will always work.

Changing the modulus from q to $2q$ and forcing the matrix $\mathbf{T}$ to always be $q\mathbf{I}$ creates several potential problems. In particular, it is no longer clear how to perform key generation, and also the outline for the security proof from [Lyu12] no longer holds. But we show that these problems can be overcome. We will now sketch the key generation and the security proof based on the hardness of the SIS problem in which one is given a uniformly random matrix $\mathbf{B} \in \mathbb{Z}_q^{n\times m}$, and is asked to find a short vector $\mathbf{w}$ such that $\mathbf{Bw} = \mathbf{0} \pmod q$. To generate the public and secret keys, we first pick a uniformly random matrix $\mathbf{A}' \in \mathbb{Z}_q^{n\times(m-n)}$ and a random $(m-n)\times n$ matrix $\mathbf{S}'$ consisting of short coefficients. We then compute $\mathbf{A}'' = \mathbf{A}'\mathbf{S}' \bmod q$ and output $\mathbf{A} = [2\mathbf{A}'|2\mathbf{A}'' + q\mathbf{I}]$ as the public key. The secret key is $\mathbf{S} = [\mathbf{S}'|-\mathbf{I}]^T$. Notice that by construction we have $\mathbf{AS} = q\mathbf{I} \pmod{2q}$ and $\mathbf{S}$ consists of small entries. The dimensions m and n are picked so that the distribution of $[\mathbf{A}'|\mathbf{A}'\mathbf{S}' \bmod q]$ can be shown to be uniformly random in $\mathbb{Z}_q^{n\times m}$ by the leftover hash lemma.

In the security proof, we are given a random matrix $\mathbf{B} = [\mathbf{A}'|\mathbf{A}''] \in \mathbb{Z}_q^{n\times m}$ by the challenger and use the adversary that forges a signature to find a short vector $\mathbf{w}$ such that $\mathbf{Bw} = \mathbf{0} \pmod q$. We create the public key $\mathbf{A} = [2\mathbf{A}'|2\mathbf{A}'' + q\mathbf{I}]$ and give it to the adversary. Even though we do not know a secret key $\mathbf{S}$ such that $\mathbf{AS} = q\mathbf{I} \pmod{2q}$, we can still create valid signatures for any messages of the adversary's choosing by picking the $(\mathbf{z}, \mathbf{c})$ according to the correct distributions and then programming the random oracle as is done in [Lyu12]. When the adversary forges, we use the forking lemma to create two equations $\mathbf{Az} = q\mathbf{c} \pmod{2q}$ and $\mathbf{Az}' = q\mathbf{c}' \pmod{2q}$. Combining

[4] One may think that a possible solution could be to output the bit b as part of the signature, but this is not secure. Depending on the sign of $\langle \mathbf{z}, \mathbf{Sc}\rangle$, one of the two values of b is more likely to be output than the other. Therefore outputting the bit b leaks information about $\mathbf{S}$.



(a) from uniform distribution (repetition rate ≈ 10)

(b) from our adapted distribution (repetition rate ≈ 1.47)

Fig. 3. Rejection Sampling

them together, we obtain $\mathbf{A}(\mathbf{z} - \mathbf{z}') = q(\mathbf{c} - \mathbf{c}') \pmod{2q}$. Under some very simple requirements for $\mathbf{z}, \mathbf{z}', \mathbf{c}$, and $\mathbf{c}'$, the previous equation implies that $\mathbf{A}(\mathbf{z} - \mathbf{z}') = \mathbf{0} \pmod{q}$ and $\mathbf{z} \neq \mathbf{z}'$. This then implies that $2\mathbf{B}(\mathbf{z} - \mathbf{z}') = \mathbf{0} \pmod{q}$ and since 2 is invertible modulo q, we have found a $\mathbf{w} = (\mathbf{z} - \mathbf{z}')$ such that $\mathbf{B}\mathbf{w} = \mathbf{0} \pmod{q}$.

The above scheme construction and proof work for SIS and equally well for Ring-SIS, when instantiated with polynomials. As in [Lyu12], we can also construct much more efficient schemes based on LWE and Ring-LWE by creating the matrix $\mathbf{A}'' = \mathbf{A}'\mathbf{S}'$ such that $(\mathbf{A}', \mathbf{A}'')$ is not uniformly random, but only computationally. For optimal efficiency, though, we can create the key in yet a different manner related to the way NTRU keys are generated. The formal construction is described in Section 4, and we just give the intuition here. We could create two small polynomials $\mathbf{s}_1, \mathbf{s}_2 \in \mathbb{Z}[x]/(x^n + 1)$ and output the public key as $\mathbf{a} = \frac{\mathbf{q} - \mathbf{s}_2}{\mathbf{s}_1} \pmod{2q}$. Notice that this implies that $\mathbf{a}\mathbf{s}_1 + \mathbf{s}_2 = \mathbf{q} \pmod{2q}$, and so we can think of the public key as $\mathbf{A} = [\mathbf{a}, \mathbf{1}]$ and the secret key as $\mathbf{S} = [\mathbf{s}_1, \mathbf{s}_2]^T$. Assuming that it is a hard problem to find small vectors $\mathbf{w}$ such that $\mathbf{A}\mathbf{w} = \mathbf{0} \pmod{2q}$, the signature scheme instantiated in the above manner will be secure. To those readers familiar with the key generation in the NTRU encryption scheme, the above key generation should look very familiar, except that the modulus is $2q$ rather than q. Since we are not sure what happens when the modulus is $2q$, we show in Section 4 how to instantiate our scheme so that it is based on NTRU over modulus q. We then explain how for certain instantiations, this is as hard a problem as Ring-SIS (using the results of Stehlé, Steinfeld [SS11]) and how for more efficient instantiations, it is a *weaker* assumption than the ones underlying the classic NTRU encryption scheme and the recent construction of fully-homomorphic encryption [LATV12].

Gaussian Sampling. There are two generic methods for sampling according to a discrete Gaussian distribution. The first one uses basic rejection sampling as follows: choose a uniform integer $x \in \{-\tau\sigma, \ldots, \tau\sigma\}$ (where $\tau \approx 12$, as in the preceding discussion) and accept it with probability proportional to $\exp(-x^2/2\sigma^2)$ (and restart otherwise). This involves the computation of the exp function to high precision and requires an average of $2\tau/\sqrt{2\pi} \approx 10$ trials, thus wasting a lot of random bits. The second one involves storing large pre-computed data, namely the cumulative distribution table of the discrete Gaussian from $-\tau\sigma$ to $\tau\sigma$. While the second method is very efficient when given enough memory, neither of the two approaches is appropriate for use in constrained devices.

We solve this issue by modifying the first approach to exploit the properties of discrete Gaussians. We recall that a Bernoulli distribution $\mathcal{B}_c$ assigns 1 (True) with probability $c \in [0, 1]$ and 0

	Floating Point exp	Precomputation Storage	: BLISS-II	BLISS-IV	Table Lookups	Entropy Consumption
Naïve Reject.	10	0			0	$45 + 10\log_2\sigma$
C.D. Table Alg.	0	$\lambda\tau\sigma$	: 170kb	630kb	$\log_2(\tau\sigma)$	$2.1 + \log_2\sigma$
Knuth-Yao Alg. [GD12]	0	$1/2\lambda\tau\sigma$	: 85kb	315kb	$\log_2(\sqrt{2\pi e}\sigma)$	$2.1 + \log_2\sigma$
Our Algorithm	0	$\lambda\log_2(2.4\tau\sigma^2)$	: 2.3kb	4kb	$\approx \log_2\sigma$	$\approx 6 + 3\log_2\sigma$

Table 2. Comparison of various sampling techniques and pre-computation storage sizes for our scheme.

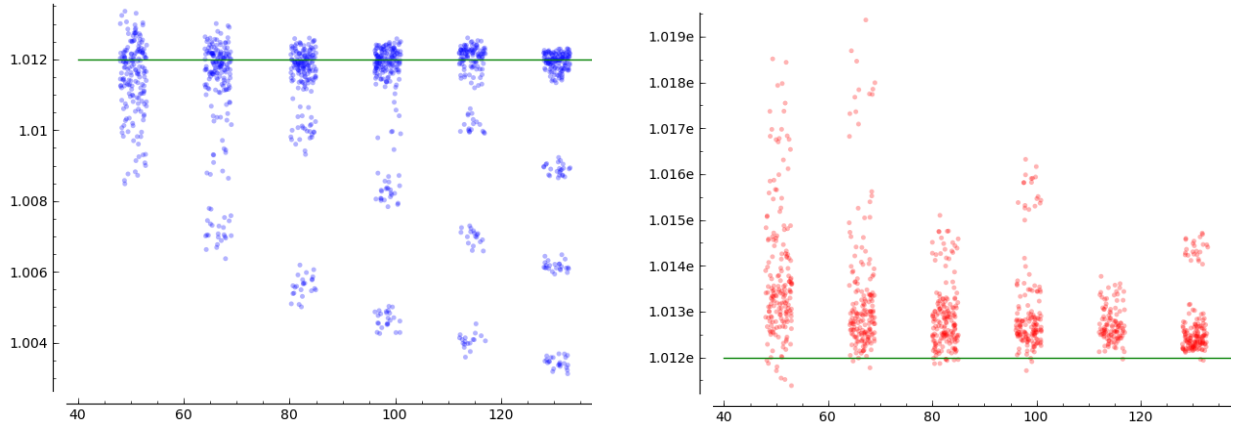
(False) with probability $1-c$. Overloading the notation for the sake of clarity, we will denote by $\mathcal{B}_c$ both the distribution and a generic random variable that follows that distribution independently of all others (thus we may write, for example, $\mathcal{B}_a \oplus \mathcal{B}_b = \mathcal{B}_{a+b-2ab}$). As a first step, to avoid explicit computation of exp, we use the simple fact that for an integer x in binary form $x = x_1 \cdots x_n$ we have $\mathcal{B}_{\exp(-x/f)} = \bigwedge_{i \text{ s.t. } x_i=1} \mathcal{B}_{\exp(-2^i/f)}$. This allows us to sample according to $\mathcal{B}_{\exp(-x/f)}$ using only logarithmically many precomputed values $\exp(-2^i/f)$. Similarly, we also design another algorithm to sample according to $\mathcal{B}_{1/\cosh(x/f)}$, using a Markov chain that makes less than two calls to $\mathcal{B}_{\exp(-x/f)}$ on average.

The second step is to replace the uniform distribution from which one chooses an integer by a more adapted one (as in Figure 3(b)) to decrease the rejection rate. It is essential, though, that the rejection rate retains an easily samplable form. To do this, we build on a specific discrete Gaussian of variance $\sigma_2^2 = 1/(2\ln 2)$ for which the distribution $D_{\sigma_2}(x)$ is proportional to 2^{-x^2}. This makes it very easily samplable, and the rejection rate still has the required form $\exp(\cdot/f)$. The final algorithm has bounded repetition rate of 1.5 rather than $2\tau/\sqrt{2\pi} \approx 10$. All the operations are very simple, requiring only small integer arithmetic, and are therefore well-suited for constrained devices. The efficiency of our sampling algorithm compared to standard techniques is detailed in Table 1.2. Algorithms and correctness proofs are detailed in Section 6.

Cryptanalysis and Experiments on NTRU Lattices. Previous cryptanalytic efforts against schemes based on SIS and LWE mostly involved computing the Hermite factor of the underlying average-case instance, as in the work of Gama and Nguyen [GN08], and making sure that its value is below the level required for the desired security guarantees. In this work we undertake a more careful cryptanalysis by using the results on BKZ 2.0 of Chen and Nguyen [CN11] in combination with other techniques – namely dual lattice reduction and the combinatorial meet-in-the-middle attack of Howgrave-Graham [HG07].

For optimal efficiency, the security of our scheme relies on the hardness of a type of NTRU problem that has recently (re-)appeared in the literature [LATV12] and which, we believe, could play a major role in the future of lattice-based cryptography (see Section 2 for the precise definition of the problem). The only cryptanalysis of which we are aware of that studies NTRU lattices deals with instances where the modulus is very close in size to the dimension of the lattice [GN08,HHGPW09]. It is thus unclear as to what roles each of the variables plays when looked at independently.

In our work, and also in the previously-mentioned work of [LATV12], the modulus is required to be substantially larger than the dimension. As far as we are aware, no previous cryptanalysis was done for these types of instances. The most complete study of the behavior of BKZ in the presence of unusually short vector(s) is due to Gama and Nguyen [GN08] who thoroughly analyzed the algorithm's running time in the presence of *one* such vector. Their experiments show that the hardness of finding this vector depends on the ratio λ_2/λ_1, that is, the gap between the second-



(a) Shortest vector not found

(b) Shortest vector found

Fig. 4. Results BKZ-20 for $n \in [48, 150]$, $q \in [6000, 25000]$ and binary search on the λ_1-threshold. On horizontal axis is the value of $n + \texttt{random(0,5)}$ and on vertical axis is $\left(\frac{1}{.40}\sqrt{\frac{qm}{2\pi e}}/\lambda_1\right)^{1/2n}$

shortest and the shortest vectors in the m-dimensional lattice. In practice, for BKZ-20, the shortest vector was found when $\lambda_2/\lambda_1 > .48 \cdot 1.01^m$.

We ran similar experiment of BKZ-20 in the case of $2n$-dimensional NTRU lattices where $\lambda_1 = \ldots = \lambda_n$. In NTRU lattices, the gap normally occurs between the n-th and the $n+1$-st successive minima, and one might think that the ratio between these two quantities would somehow determine the hardness of the instance. Our experiments showed that this is not the case, and the shortest vector was found when $\sqrt{qm/2\pi e}/\lambda_1$ was greater than $.40 \cdot 1.012^m$ (see Figure 4). Despite the fact that there is no vector in the lattice having length $\sqrt{qm/2\pi e}$ this is actually consistent with the results of [GN08]! The reason is that $\sqrt{qm/2\pi e}$ is the *expected* length of the shortest vector according to the Gaussian heuristic,[5] and we would also expect $\lambda_2 \approx \sqrt{qm/2\pi e}$ in a random q-ary lattice analyzed in [GN08]. Thus one could say that the hardness of finding a short vector in q-ary lattices depends not on the gap, but rather on the ratio between the Gaussian heuristic and the actual length of the shortest vector.

Similar to the results in [GN08], when the ratio was smaller than $.40 \cdot 1.012^m$, the resulting shortest vector had length about $\sqrt{q} \cdot 1.012^m$. In other words, BKZ-20 behaved as if the lattice were truly random. Because of our experiments with BKZ-20, it seems reasonable to assume that BKZ behaves analogously for larger block sizes. Thus we can measure its efficacy according to the BKZ 2.0 methodology in [CN11]. We would like to stress that we have no explanation for the reason why the ratio between the Gaussian heuristic and the actual length of the vector seems to dictate the hardness of finding short vectors in NTRU lattices. We are equally unsure whether this phenomenon implies that these lattices are indeed as hard as the random lattices that have been more exhaustively studied [GN08,CN11].

The general dearth of lattice cryptanalysis papers stands in contrast to the vast number of articles proposing theoretical lattice-based constructions. Our belief is that this lack of cryptanalytic effort is in part due to the fact that most of the papers with scheme proposals give no concrete

[5] The Gaussian heuristic says that for certain types of random lattices $\mathcal{L}$, we will have $\lambda_1(\mathcal{L}) \approx det(\mathcal{L})^{1/m} \cdot \sqrt{\frac{m}{2\pi e}}$ [GN08].

targets to attack. One of the proposed instantiations in the present work is a "toy example" that we estimate has approximately 60 bits of security. Thus if it turns out that NTRU lattices are weaker than believed, it is wholly possible that this example could be broken on a personal computer, and we think this would be of great interest to the practical community. In addition, it could be argued that we do not yet know enough about lattice reduction to be able to propose such "fine-grained" security estimates like 160-bit or 192-bit. But one of the main reasons that we make these proposals is to make it "worthwhile" for cryptanalysts to work on these problems. In short, one of our hopes is that this work spurs on the cryptanalysis that is currently much needed in the field.

1.3 Acknowledgments

We thank the CRYPTO 2013 reviewers for their careful reading of the paper and their diligent comments. We also thank Steven Galbraith and Pascal Paillier for useful comments on previous versions of this work.

2 Preliminaries

2.1 Notation

For any integer q, we identify the ring $\mathbb{Z}_q$ with the interval $[-q/2, q/2) \cap \mathbb{Z}$, and in general for a ring $\mathcal{R}$, we define $\mathcal{R}_q$ to be the quotient ring $\mathcal{R}/(q\mathcal{R})$. Whenever working in the quotient ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n+1)$(or $\mathcal{R}_{2q} = \mathbb{Z}_{2q}[x]/(x^n+1)$), we will assume that n is a power of 2 and q is a prime number such that $q = 1 \pmod{2n}$. We define $\mathbb{B} = \{0, 1\}$ and $\mathbb{T} = \{-1, 0, 1\}$ the set of binary and ternary integers and $\mathbb{B}_w^n$ (resp. $\mathbb{T}_w^n$) the set of binary vectors (resp. ternary vectors) of length n and Hamming weight w (*i.e.* vectors with exactly w out of n non-zero entries).Vectors, considered as column vectors, will be written in bold lower case letters. Matrices will be written in bold upper case letters. For a positive integer n, we write $\mathbf{I}_n$ to be the identity matrix of dimension n. We may also more generally use it as the identity element in the ring $\mathcal{R}$.

We recall that the ℓ_p-norm of a vector $\mathbf{v}$ is defined as $\|\mathbf{v}\|_p = (\sum_i |v_i|^p)^{1/p}$ for $p > 0$, and its ℓ_∞-norm as $\|\mathbf{v}\|_\infty = \max_i |v_i|$. By default, we use $\|\cdot\|$ for the ℓ_2-norm.

We now state a general rejection sampling lemmathat will be used throughout the paper to show the equivalence of two distributions. The proof of this lemma is quite standard (cf. [Lyu12]).

Lemma 2.1 (Rejection Sampling). *Let V be an arbitrary set, and $h\colon V \to \mathbb{R}$ and $f\colon \mathbb{Z}^m \to R$ be probability distributions. If $g_v\colon \mathbb{Z}^m \to R$ is a family of probability distributions indexed by $v \in V$ with the property that there exists a $M \in \mathbb{R}$ such that*

$$\forall v \in V, \forall \mathbf{z} \in \mathbb{Z}^m, M \cdot g_v(\mathbf{z}) \geqslant f(\mathbf{z}) \ ,$$

then, the output distributions of the following two algorithms are identical:

1. *$v \leftarrow h$, $z \leftarrow g_v$, output $(\mathbf{z}, v)$ with probability $f(\mathbf{z})/\big(M \cdot g_v(\mathbf{z})\big)$.*
2. *$v \leftarrow h$, $z \leftarrow f$, output $(\mathbf{z}, v)$ with probability $1/M$.*

2.2 Discrete Gaussian Distribution

Gaussian Distribution. The (un-normalized) Gaussian distribution with standard deviation $\sigma \in \mathbb{R}$ and center $c \in \mathbb{R}$ evaluated at $x \in \mathbb{R}$ is defined by $\rho_{c,\sigma}(x) = \exp\left(\frac{-(x-c)^2}{2\sigma^2}\right)$, and more generally by $\rho_{\mathbf{c},\sigma}(\mathbf{x}) = \exp\left(\frac{-\|\mathbf{x}-\mathbf{c}\|^2}{2\sigma^2}\right)$ for $\mathbf{x}, \mathbf{c} \in \mathbb{R}^n$. When the center $\mathbf{c}$ is $\mathbf{0}$, we generally omit it from the notation and simply write $\rho_\sigma(\mathbf{x})$. The discrete Gaussian distribution over $\mathbb{Z}$ centered at 0 is defined by $D_\sigma(x) = \rho_\sigma(x)/\rho_\sigma(\mathbb{Z})$, and more generally, over $\mathbb{Z}^m$ by $D_\sigma^m(\mathbf{x}) = \rho_\sigma(\mathbf{x})/\rho_\sigma(\mathbb{Z})^m$.

Tailcutting. It is generally useful to ignore large values which are unlikely to appear when drawing according to a Gaussian distribution.

Lemma 2.2 ([MR07]). *For any dimension m, $\sigma > 0$ and $\tau > 1$, $\rho_\sigma(\mathbb{Z}^m \setminus \tau\sigma\sqrt{m}\mathfrak{B}) < 2C(\tau)^m \cdot \rho_\sigma(\mathbb{Z})^m$, where $C(\tau) = \tau \exp\left(\frac{1-\tau^2}{2}\right) < 1$, and $\mathfrak{B}$ is the centered ℓ_2 unit ball.*

Therefore, to tailcut less than $2^{-\lambda}$ of a 1-dimensional Gaussian, one should choose $\tau \approx \sqrt{\lambda \cdot 2\ln 2}$, the typical value being $\tau = 12$ for $\lambda = 100$.

2.3 Hardness Assumptions

All the constructions in this paper are based on the hardness of the *generalized SIS* (Short Integer Solution) problem, which we define below.

Definition 2.3 ($\mathcal{R}$-SIS$^{\mathcal{K}}_{q,n,m,\beta}$ problem). *Let $\mathcal{R}$ be some ring and $\mathcal{K}$ be some distribution over $\mathcal{R}_q^{n\times m}$, where $\mathcal{R}_q$ is the quotient ring $\mathcal{R}/(q\mathcal{R})$. Given a random $\mathbf{A} \in \mathcal{R}_q^{n\times m}$ drawn according to the distribution $\mathcal{K}$, find a non-zero $\mathbf{v} \in \mathcal{R}_q^m$ such that $\mathbf{A}\mathbf{v} = \mathbf{0}$ and $\|\mathbf{v}\|_2 \leqslant \beta$.*

If we let $\mathcal{R} = \mathbb{Z}$ and $\mathcal{K}$ be the uniform distribution, then the resulting problem is the classical SIS problem first defined by Ajtai [Ajt96] in his seminal paper showing connections between worst-case lattice problems and the average-case SIS problem. By the pigeonhole principle, if $\beta \geqslant \sqrt{m}q^{n/m}$ then the SIS instances are guaranteed to have a solution. Using Gaussian techniques, Micciancio and Regev [MR07] improved Ajtai's result to show that, for a large enough q as a function of n and β, the SIS$_{q,n,m,\beta}$ problem is as hard (on the average) as the $\tilde{\mathcal{O}}(\sqrt{n}\beta)$-SIVP problem for *all* lattices of dimension n.

In 2006, a ring variant of SIS was introduced independently by Peikert and Rosen [PR06] and Lyubashevsky and Micciancio [LM06]. In [LM06] it was shown that if $\mathcal{R} = \mathbb{Z}[x]/(x^n + 1)$, where n is a power of 2, then the $\mathcal{R}$-SIS$^{\mathcal{K}}_{q,1,m,\beta}$ problem is as hard as the $\tilde{\mathcal{O}}(\sqrt{n}\beta)$-SVP problem in all lattices that are ideals in $\mathcal{R}$ (where $\mathcal{K}$ is again the uniform distribution over $\mathcal{R}_q^{1\times m}$).

NTRU Lattices. In the NTRU cryptosystem over the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$ [HPS98], the key generation procedure picks two short secret keys $\mathbf{f}, \mathbf{g} \in \mathcal{R}_q$ (according to some distribution) and computes the public key as $\mathbf{a} = \mathbf{g}/\mathbf{f}$.[6] When the norm of $\mathbf{f}, \mathbf{g}$ is large enough, it can be shown that $\mathbf{a}$ is actually uniformly random in $\mathcal{R}_q$ [SS11], but even when the secret keys do not have enough entropy, their quotient still appears to be pseudorandom, although no proof of this fact is known [LATV12]. In the NTRU cryptosystem (or its more secure modification of [SS11] which is based on the Ring-LWE problem), one encrypts a message μ, represented as a polynomial in $\mathcal{R}_q$ with $\{0, 1\}$

[6] In the original NTRU scheme, the ring was $\mathbb{Z}_q[x]/(x^n - 1)$, but lately researchers have also used $\mathbb{Z}_q[x]/(x^n + 1)$ when n is a power of 2. Indeed, the latter choice seems at least as secure.

coefficients, by picking two short vectors $\mathbf{r}, \mathbf{e} \in \mathcal{R}_q$ and outputting $\mathbf{z} = 2(\mathbf{ar} + \mathbf{e}) + \mu$. The security of the scheme relies on the fact that the distribution of $(\mathbf{a}, \mathbf{z})$ is pseudo-random in $\mathcal{R}_q^2$.

One can define an NTRU version of the SIS problem that is at least as hard as breaking the NTRU cryptosystem.[7] In particular, given an NTRU public key $\mathbf{a}$, find two polynomials $\mathbf{v}_1, \mathbf{v}_2 \in \mathcal{R}_q$ such that $\|(\mathbf{v}_1|\mathbf{v}_2)\| \leqslant \beta$ and $\mathbf{av}_1 + \mathbf{v}_2 = 0$ in $\mathcal{R}_q$. Notice that $(\mathbf{f}, -\mathbf{g})$ is a solution to this problem, but in fact, finding larger solutions can also be useful in breaking the NTRU cryptosystem. In particular, notice that for any solution $(\mathbf{v}_1|\mathbf{v}_2)$, one can compute $\mathbf{zv}_1 = 2(-\mathbf{rv}_2 + \mathbf{ev}_1) + \mu\mathbf{v}_1$. If β is sufficiently small with respect to $\|(\mathbf{r}|\mathbf{e})\|$, then $\mathbf{z} \cdot \mathbf{v}_1 \bmod 2 = \mu\mathbf{v}_1$, and μ can be recovered. Thus, for certain parameters, the NTRU version of the SIS problem is at least as hard as breaking the NTRU cryptosystem. As a side-note, we would like to point out that the NTRU encryption scheme remains hard even after 15 years of cryptanalysis. The weakness in the NTRU signature scheme, which uses the same key generation procedure, is due to the fact that signatures slowly leak the secret key [NR09,DN12b], but this is provably (*i.e.* information-theoretically) avoided in our scheme. In Appendix A, we analyze the hardness of the NTRU SIS problem using combinations of lattice [CN11] and hybrid attacks [HG07].

3 BLISS: A Lattice Signature Scheme using Bimodal Gaussians

In this section, we present our new signature scheme along with the proof of correctness. The security of the signature scheme is based on the hardness of the $\mathcal{R}\text{-SIS}^{\mathcal{K}}_{q,n,m,\beta}$ problem. We mention that this is the "simple" version of our algorithm, its specific implementation that uses numerous enhancements is presented in Section 4 . For simplicity, we present our algorithm for $\mathcal{R} = \mathbb{Z}$, but it works in exactly the same way for rings $\mathcal{R} = \mathbb{Z}[x]/(x^n + 1)$(see Section 4).

3.1 New Signature and Verification Algorithms

Key pairs. The secret key is a (short) matrix $\mathbf{S} \in \mathbb{Z}_{2q}^{m \times n}$ and the public key is given by the matrix $\mathbf{A} \in \mathbb{Z}_{2q}^{n \times m}$ such that $\mathbf{AS} = q\mathbf{I}_n \pmod{2q}$. A crucial property, for our new rejection sampling algorithm, satisfied by the key pair, is that $\mathbf{AS} = \mathbf{A}(-\mathbf{S}) = q\mathbf{I}_n \pmod{2q}$. Obtaining such a key pair is easy and can be done efficiently. In Appendix B , we explain the key-generation procedure which results in a scheme whose security is based on the classic $\text{SIS}_{q,n,m,\beta}$ problem and in Section 4 we present an "NTRU-like" variant of the key generation which yields a more efficient instantiation of the signature scheme.

Random Oracle Domain. We model the hash function H as a random oracle that has uniform output in $\mathbb{B}_\kappa^n$, the set of binary vectors of length n and weight κ. An efficient construction of such a random oracle can be found in Section 4.4.

The Signature Algorithm. The signer, who is given a message digest μ, first samples a vector $\mathbf{y}$ from the m-dimensional discrete Gaussian distribution D_σ^m and then computes $\mathbf{c} \leftarrow H(\mathbf{Ay} \bmod 2q, \mu)$. He then samples a bit b in $\{0, 1\}$ and computes the potential output $\mathbf{z} \leftarrow \mathbf{y} + (-1)^b\mathbf{Sc}$. Notice that $\mathbf{z}$ is distributed according to the bimodal discrete Gaussian distribution $\frac{1}{2}D^m_{\mathbf{Sc},\sigma} + \frac{1}{2}D^m_{-\mathbf{Sc},\sigma}$. At this point we perform rejection sampling and output the signature $(\mathbf{z}, \mathbf{c})$ with probability

[7] A way to state the NTRU SIS problem in terms of the $\mathcal{R}\text{-SIS}^{\mathcal{K}}_{q,1,2,\beta}$ problem is to set $\mathcal{R} = \mathbb{Z}[x]/(x^n + 1)$ and let $\mathcal{K}$ be the distribution that picks small $\mathbf{f}, \mathbf{g}$ and outputs the public key $\mathbf{A} = (\mathbf{a}, \mathbf{1}) \in \mathcal{R}_q^{1\times 2}$ for $\mathbf{a} = \mathbf{g}/\mathbf{f}$.

Algorithm 1: Signature Algorithm

Input: Message μ, public key $\mathbf{A} \in \mathbb{Z}_{2q}^{n \times m}$, secret key $\mathbf{S} \in \mathbb{Z}_{2q}^{m \times n}$, stand. dev. $\sigma \in \mathbb{R}$
Output: A signature $(\mathbf{z}, \mathbf{c})$ of the message μ

1: $\mathbf{y} \leftarrow D_\sigma^m$
2: $\mathbf{c} \leftarrow H(\mathbf{A}\mathbf{y} \bmod 2q, \mu)$
3: Choose a random bit $b \in \{0, 1\}$
4: $\mathbf{z} \leftarrow \mathbf{y} + (-1)^b \mathbf{S}\mathbf{c}$
5: **Output**$(\mathbf{z}, \mathbf{c})$ with probability $1 \Big/ \left(M \exp\left(-\frac{\|\mathbf{Sc}\|^2}{2\sigma^2}\right) \cosh\left(\frac{\langle \mathbf{z}, \mathbf{Sc} \rangle}{\sigma^2}\right)\right)$ otherwise **restart**

Algorithm 2: Verification Algorithm

Input: Message μ, public Key $\mathbf{A} \in \mathbb{Z}_{2q}^n$, signature $(\mathbf{z}, \mathbf{c})$
Output: Accept or Reject the signature

1: **if** $\|\mathbf{z}\| > B_2$ **then** Reject
2: **if** $\|\mathbf{z}\|_\infty \geqslant q/4$ **then** Reject
3: Accept iff $\mathbf{c} = H(\mathbf{A}\mathbf{z} + q\mathbf{c} \bmod 2q, \mu)$

$1 \Big/ \left(M \exp\left(-\frac{\|\mathbf{Sc}\|^2}{2\sigma^2}\right) \cosh\left(\frac{\langle \mathbf{z}, \mathbf{Sc} \rangle}{\sigma^2}\right)\right)$, where M is some fixed positive real that is set large enough to ensure that the preceding probability is always at most 1. We explain how to set M in accordance with the standard deviation σ in the next section. If the signing algorithm did not output the signature, then it is restarted and repeated until something is output. The expected number of iterations of the signing algorithm is M.

The Verification Algorithm. The verification algorithm will accept $(\mathbf{z}, \mathbf{c})$ as the signature for μ if the following three conditions hold:

1. $\|\mathbf{z}\| \leqslant B_2$
2. $\|\mathbf{z}\|_\infty < q/4$
3. $\mathbf{c} = H(\mathbf{A}\mathbf{z} + q\mathbf{c} \bmod 2q, \mu)$

The signer outputs signatures of the form $(\mathbf{z}, \mathbf{c})$ where $\mathbf{z}$ is distributed according to D_σ^m, thus the acceptance bound B_2 should be set a little bit higher than $\sqrt{m}\sigma$, which is the expected value around which the output of D_σ^m is tightly concentrated; denoting $B_2 = \eta\sqrt{m}\sigma$, one can set η so that $\|\mathbf{z}\| \leqslant B_2$ is verified with probability $1 - 2^{-\lambda}$ [Lyu12, Lemma 4.4] for the security parameter λ (in practice, $\eta \in [1.1, 1.4]$). For technical reasons in the security proof, we also need that $\|\mathbf{z}\|_\infty < q/4$, but this condition is usually verified whenever the first one is and does not restrict the manner in which we choose the parameters for the scheme(see Section 3.3). Condition 3 will also hold for valid signatures because

$$\mathbf{A}\mathbf{z} + q\mathbf{c} = \mathbf{A}(\mathbf{y} + (-1)^b\mathbf{S}\mathbf{c}) + q\mathbf{c} = \mathbf{A}\mathbf{y} + \big((-1)^b\mathbf{A}\mathbf{S}\big)\mathbf{c} + q\mathbf{c} = \mathbf{A}\mathbf{y} + (q\mathbf{I}_n)\mathbf{c} + q\mathbf{c} = \mathbf{A}\mathbf{y} \bmod 2q.$$

3.2 Rejection Sampling: Correctness and Efficiency

We now explain how to pick the standard deviation σ and positive real M so that the signing algorithm in the preceding section produces vectors $\mathbf{z}$ according to the distribution D_σ^m. Because $\mathbf{y}$ is distributed according to D_σ^m, it is easy to see that in Step 4 of the signing algorithm, $\mathbf{z}$ is

distributed according to $g_{\mathbf{Sc}} = \frac{1}{2}D^m_{\mathbf{Sc},\sigma} + \frac{1}{2}D^m_{-\mathbf{Sc},\sigma}$ for fixed $\mathbf{Sc}$ and over the space of all $(b, \mathbf{y})$. Thus for any $\mathbf{z}^* \in \mathbb{R}^m$, we have

$$\begin{aligned}
\Pr[\mathbf{z} = \mathbf{z}^*] &= \frac{1}{2}D^m_{\mathbf{Sc},\sigma}(\mathbf{z}^*) + \frac{1}{2}D^m_{-\mathbf{Sc},\sigma}(\mathbf{z}^*) \\
&= \frac{1}{2\rho_\sigma(\mathbb{Z}^m)} \exp\left(-\frac{\|\mathbf{z}^* - \mathbf{Sc}\|^2}{2\sigma^2}\right) + \frac{1}{2\rho_\sigma(\mathbb{Z}^m)} \exp\left(-\frac{\|\mathbf{z}^* + \mathbf{Sc}\|^2}{2\sigma^2}\right) \\
&= \frac{1}{2\rho_\sigma(\mathbb{Z}^m)} \exp\left(-\frac{\|\mathbf{z}^*\|^2}{2\sigma^2}\right) \exp\left(-\frac{\|\mathbf{Sc}\|^2}{2\sigma^2}\right) \left(e^{-\frac{\langle \mathbf{z}^*, \mathbf{Sc}\rangle}{\sigma^2}} + e^{\frac{\langle \mathbf{z}^*, \mathbf{Sc}\rangle}{\sigma^2}}\right) \\
&= \frac{1}{\rho_\sigma(\mathbb{Z}^m)} \exp\left(-\frac{\|\mathbf{z}^*\|^2}{2\sigma^2}\right) \exp\left(-\frac{\|\mathbf{Sc}\|^2}{2\sigma^2}\right) \cosh\left(\frac{\langle \mathbf{z}^*, \mathbf{Sc}\rangle}{\sigma^2}\right) .
\end{aligned}$$

The desired output distribution is the centered Gaussian distribution $f(\mathbf{z}^*) = \rho_\sigma(\mathbf{z}^*)/\rho_\sigma(\mathbb{Z}^m)$. Thus, by Lemma 2.1, one should accept the sample $\mathbf{z}^*$ with probability:

$$p_{\mathbf{z}*} = \frac{f(\mathbf{z}^*)}{M g_{\mathbf{Sc}}(\mathbf{z}^*)} = 1 \Big/ \left(M \exp\left(-\frac{\|\mathbf{Sc}\|^2}{2\sigma^2}\right) \cosh\left(\frac{\langle \mathbf{z}^*, \mathbf{Sc}\rangle}{\sigma^2}\right)\right) ,$$

where M is chosen large enough so that $p_{\mathbf{z}*} \leqslant 1$. Note that $\cosh(x) \geqslant 1$ for any x, so it suffices that

$$M = e^{\frac{1}{2\alpha^2}} \tag{3}$$

where α is such that $\sigma \geqslant \alpha \cdot \|\mathbf{Sc}\|$.

Bound on $\|\mathbf{Sc}\|$. Notice that if we fix the repetition rate M, then the standard deviation of the signature $\mathbf{z}$, and therefore also its size, only depend on the maximum possible norm of the vector $\mathbf{Sc}$. For this reason, it is important to obtain a bound as tight as possible on this product. Several upper bounds on $\|\mathbf{Sc}\|$ can be used such as $\|\mathbf{Sc}\| \leqslant \|\mathbf{c}\|_1 \cdot \|\mathbf{S}\| = \kappa \|\mathbf{S}\|$ (as in [Lyu12]) or $\|\mathbf{Sc}\| \leqslant s_1(\mathbf{S}) \cdot \|\mathbf{c}\| = s_1(\mathbf{S}) \cdot \sqrt{\kappa}$ where $s_1(\mathbf{S})$ is the singular norm of $\mathbf{S}$. Here we introduce a new measure of $\mathbf{S}$, adapted to the form of $\mathbf{c}$, which helps us achieve a tighter bound than with all previous methods. We believe that this norm and the technique for bounding it could be of independent interest.

Definition 3.1. *For any integer κ, we define $N_\kappa \colon \mathbb{R}^{m\times n} \to \mathbb{R}$ as:*

$$N_\kappa(\mathbf{X}) = \max_{\substack{I\subset\{1,\dots,n\}\\ \#I=\kappa}} \sum_{i\in I} \left(\max_{\substack{J\subset\{1,\dots,n\}\\ \#J=\kappa}} \sum_{j\in J} T_{i,j} \right) \quad \textit{where } \mathbf{T} = \mathbf{X}^t \cdot \mathbf{X} \in \mathbb{R}^{n\times n} .$$

The following proposition states that $\sqrt{N_\kappa(\mathbf{S})}$ is also an upper bound for $\|\mathbf{Sc}\|$.

Proposition 3.2. *Let $\mathbf{S} \in \mathbb{R}^{m\times n}$ be a real matrix. For any $\mathbf{c} \in \mathbb{B}^n_\kappa$, we have $\|\mathbf{Sc}\|^2 \leqslant N_\kappa(\mathbf{S})$.*

Proof. Set $I = J = \{i \in \{1, \dots, n\} : \mathbf{c}_i = 1\}$, which implies $\#I = \#J = \kappa$. Rewriting $\|\mathbf{S} \cdot \mathbf{c}\|^2 = \mathbf{c}^t \cdot \mathbf{S}^t \cdot \mathbf{S} \cdot \mathbf{c} = \mathbf{c}^t \cdot \mathbf{T} \cdot \mathbf{c} = \sum_{i\in I} \sum_{j\in J} T_{i,j}$, we can conclude from the definition of N_κ. □

In practice, we will use this upper bound (see Section 4)to bound $\|\mathbf{Sc}\|$ and derive the parameters. Some secret keys $\mathbf{S}$ will be rejected according to the value of $N_\kappa(\mathbf{S})$, which is easily computable. In addition to the gain from the use of bimodal Gaussians, this new upper bound lowers the standard deviation σ by a factor $\approx \sqrt{\kappa}/2$ compared to [Lyu12].

3.3 Security Proof

Any existential forger against our signature scheme can solve the $\mathcal{R}\text{-SIS}^{\mathcal{K}}_{q,n,m,\beta}$ problem for $\beta = 2B_2$ where $\mathcal{K}$ is the distribution induced by the public-key generation algorithm.

Theorem 3.3. *Suppose there is a polynomial-time algorithm $\mathcal{F}$ which makes at most s queries to the signing oracle and h queries to the random oracle H, and succeeds in forging with non negligible probability δ. Then there exists a polynomial-time algorithm which can solve the $\mathcal{R}\text{-SIS}^{\mathcal{K}}_{q,n,m,\beta}$ problem for $\beta = 2B_2$ with probability $\approx \frac{\delta^2}{2(h+s)}$. Moreover the signing algorithm produces a signature with probability $\approx 1/M$ and the verifying algorithm accepts a signature produced by an honest signer with probability at least $1 - 2^m$.*

The proof of the theorem follows from standard arguments, and is simpler and tighter than the proof of [Lyu12]. In a nutshell, the fact that the distribution of the signatures in the scheme does not depend on the secret key means that the simulator can "sign" arbitrary messages without having the secret key by programming the random oracle. Then when the adversary produces a forgery, the simulator can extract a solution to the SIS problem. It is proved in a sequence of two lemmas. In Lemma 3.4, we show that our signing algorithm can be replaced with Hybrid 2 (Algorithm 4), and the statistical distance between the two outputs will be at most $\epsilon = s(s+h)2^{-n+1}$. Since Hybrid 2 produces an output with probability exactly $1/M$, the signing algorithm produces an output with probability at least $(1-\epsilon)/M$. Then in Lemma 3.5 we show that if a forger can produce a forgery with probability δ when the signing algorithm is replaced with Hybrid 2, then we can use him to recover a vector $\mathbf{v} \neq \mathbf{0}$ such that $\|\mathbf{v}\| \leqslant \beta = 2B_2$ and $\mathbf{A}\mathbf{v} = \mathbf{0} \bmod q$ with probability at least $\delta^2/(2(s+h))$.

Algorithm 3: Hybrid 1

$\mathbf{y} \leftarrow \mathcal{D}^m_\sigma$
$\mathbf{c} \leftarrow \mathbb{B}^n_\kappa$
Choose a random bit b
$\mathbf{z} \leftarrow (-1)^b \mathbf{S}\mathbf{c} + \mathbf{y}$
With probability $1/\big(M \exp(-\frac{\|\mathbf{S}\mathbf{c}\|^2}{2\sigma^2}) \cosh(\frac{\langle \mathbf{z}, \mathbf{S}\mathbf{c}\rangle}{\sigma^2})\big)$:
    output $(\mathbf{z}, \mathbf{c})$
    program $H(\mathbf{A}\mathbf{z} + q\mathbf{c}, \mu) = \mathbf{c}$

Algorithm 4: Hybrid 2

$\mathbf{c} \leftarrow \mathbb{B}^n_\kappa$
$\mathbf{z} \leftarrow \mathcal{D}^m_\sigma$
With probability $\frac{1}{M}$:
    output $(\mathbf{z}, \mathbf{c})$
    program $H(\mathbf{A}\mathbf{z} + q\mathbf{c}, \mu) = \mathbf{c}$

Lemma 3.4. *Let $\mathcal{D}$ be a distinguisher which can query the random oracle H and either the actual signing algorithm (Algorithm 1) or Hybrid 2 (Algorithm 4). If she makes h queries to H and s queries to the signing algorithm that she has access to, then for all but a $1 - e^{\Omega(n)}$ fraction of all possible matrices $\mathcal{A}$, her advantage in distinguishing the actual signing algorithm from the one in Hybrid 2 is at most $s(s+h)2^{-n+1}$.*

Proof. First, we show that the distinguisher $\mathcal{D}$ has advantage at most $s(s+h)2^{-n+1}$ in distinguishing the real signature scheme from an output of Hybrid 1 (Algorithm 3). The only difference between these algorithms is that, in Hybrid 1, the output of the random oracle is chosen at random from $\mathbb{B}^n_\kappa$ and then programmed as the answer to $H(\mathbf{A}\mathbf{z} + q\mathbf{c}, \mu) = H(\mathbf{A}\mathbf{y}, \mu)$ without checking whether the value of $(\mathbf{A}\mathbf{y}, \mu)$ was already set. Now, each time Hybrid 1 is called, the probability of generating a

$\mathbf{y}$ such that $\mathbf{Ay}$ is equal to one of the previous values that was queried is at most 2^{-n+1}. Indeed, let us notice that at most $s+h$ values of $(\mathbf{Ay}, \mu)$ will ever be set. With probability at least $1 - e^{\Omega(n)}$, the matrix $\mathbf{A}$ can be written in Hermite Normal Form as $\mathbf{A} = [\bar{\mathbf{A}} \| \mathbf{I}]$. Finally, for any $\mathbf{t} \in \mathbb{Z}_{2q}^n$, since $\sigma \geqslant 3/\sqrt{2\pi}$, we have

$$\Pr[\mathbf{Ay} = \mathbf{t};\ \mathbf{y} \leftarrow \mathcal{D}_\sigma^m] = \Pr[\mathbf{y_1} = (\mathbf{t} - \bar{\mathbf{A}}\mathbf{y}_0);\ \mathbf{y} \leftarrow \mathcal{D}_\sigma^m] \leqslant \max_{\mathbf{t}' \in \mathbb{Z}_{2q}^n} \Pr[\mathbf{y_1} = \mathbf{t}';\ \mathbf{y_1} \leftarrow \mathcal{D}_\sigma^n] \leqslant 2^{-n}\ .$$

Thus if Hybrid 1 is accessed s times, and the probability of getting a collision each time is at most $(s+h)2^{-n+1}$, the probability that a collision occurs after s queries is at most $s(s+h)2^{-n+1}$.

We next emphasize that the outputs of Hybrid 1 and Hybrid 2 exactly follows the same distribution.This is a direct consequence of Lemma 2.1: Hybrid 1 exactly plays the role of algorithm $\mathcal{A}$ and Hybrid 2 corresponds to $\mathcal{F}$, where $M = \exp(1/(2\alpha^2))$ and

$$f(\mathbf{z}) = \exp\left(-\|\mathbf{z}\|^2/(2\sigma^2)\right), \quad g_{\mathbf{c}}(\mathbf{z}) = \exp\left(-\|\mathbf{z}\|^2/(2\sigma^2)\right)\exp\left(-\|\mathbf{Sc}\|^2/(2\sigma^2)\right)\cosh\left(\langle \mathbf{z}, \mathbf{Sc}\rangle/\sigma^2\right)\ .$$

By Lemma 2.1 the outputs of Hybrid 1 and Hybrid 2 follow the same distribution (since we have $M \cdot g_{\mathbf{c}} \geqslant f$ for all $\mathbf{v}$). □

Lemma 3.5. *Suppose there exists a polynomial-time algorithm $\mathcal{F}$ which makes at most s queries to the signer in Hybrid 2, h queries to the random oracle H, and succeeds in forging with probability δ. Then there exists an algorithm with the same time-complexity as $\mathcal{F}$ which, for a given $\mathbf{B} \leftarrow \mathcal{K}$, finds with probability at least $\approx \delta^2/(2(s+h))$ a non-zero $\mathbf{v} \in \mathbb{Z}^m$ such that $\|\mathbf{v}\| \leqslant 2B_2$ and $\mathbf{Bv} = 0$.*

Proof. Let $\mathbf{B} \leftarrow \mathcal{K}$ be the matrix for the generalized SIS instance we want to solve. We slightly transform $\mathbf{B}$ to create a public key $\mathbf{A}$ as in the signature scheme and publish it as the public key $\mathbf{A} \in \mathbb{Z}_{2q}^{n \times m}$; notice that this modification is such that $\mathbf{A} \bmod q = 2\mathbf{B}$ for our key generation procedures. Therefore finding a vector $\mathbf{v}$ such that $\mathbf{Av} = \mathbf{0} \bmod q$ yields $\mathbf{Bv} = \mathbf{0} \bmod q$ because 2 is invertible modulo q.[8] Denote by $t = s + h$ the bound on the number of times the random oracle H is called or programmed during $\mathcal{F}$'s attack.

First, we pick random coins ϕ and ψ respectively for the forger and the signer. We also pick the values that will correspond to the responses of the random oracle $\mathbf{c}_1, \ldots, \mathbf{c}_t \xleftarrow{\$} \mathbb{B}_\kappa^n$. We now consider a subroutine $\mathcal{A}$ taking as input $(\mathbf{A}, \phi, \psi, \mathbf{c}_1, \ldots, \mathbf{c}_t)$.

The first step of the subroutine is to initialize $\mathcal{F}$ by giving it the public-key $\mathbf{A}$ and the random coins ϕ. Then, it proceeds to run $\mathcal{F}$. Whenever $\mathcal{F}$ wants some message signed, $\mathcal{A}$ runs the signing algorithm of Hybrid 2 using the signer random coins ψ to produce a signature. During signing or when $\mathcal{F}$ will make queries to the random oracle, the random oracle H will have to be programmed, and the response of H will be the first $\mathbf{c}_i$ in the list $(\mathbf{c}_1, \ldots, \mathbf{c}_t)$ that has not been used yet. (Of course, $\mathcal{A}$ keeps a table of all queries to H, so in case the same query is made twice, the previously answered $\mathbf{c}_i$ will be replied.) When $\mathcal{F}$ finishes running and outputs a forgery (with probability δ), our subroutine $\mathcal{A}$ simply outputs $\mathcal{F}$'s output $\{(\mathbf{z}, \mathbf{c}), \mu\}$.

[8] More precisely, for the SIS-based generation detailed in Appendix B, denoting $\mathbf{B} = (\mathbf{B}_1 | -\mathbf{B}_2) \in \mathbb{Z}_q^{n \times (m-n)} \times \mathbb{Z}_q^{n \times n}$, define $\mathbf{A} = (2\mathbf{B}_1 | q\mathbf{I}_n - 2\mathbf{B}_2)$. The matrix $\mathbf{A}$ follows the same distribution as in the key generation of Appendix B because of the use of the leftover hash lemma, and $\mathbf{A} \bmod q = 2\mathbf{B}$. For the "NTRU-like" key generation used to instantiate our scheme in Section 4, we get from the NTRU SIS assumption a matrix $\mathbf{B} = (\mathbf{B}_1 | -\mathbf{B}_2) = (\mathbf{a} | -1)$ where $\mathbf{a} = (2\mathbf{g} + 1)/\mathbf{f}$ and we define $\mathbf{A} = (2\mathbf{B}_1 | q\mathbf{1} - 2\mathbf{B}_2) = (2\mathbf{a} | q - 2)$ that is, a public key with the same distribution as in Section 4. Moreover we get $\mathbf{A} \bmod q = 2\mathbf{B} = (2\mathbf{a} | -2)$.

Recall that the output of $\mathcal{A}$ verifies $\|\mathbf{z}\|_\infty < q/4$ and $\|\mathbf{z}\| \leqslant B_2$ and $\mathbf{c} = H(\mathbf{Az} + q\mathbf{c}, \mu)$. Notice that if the random oracle H was not queried or programmed on some input $\mathbf{w} = \mathbf{Az} + q\mathbf{c}$, then $\mathcal{F}$ has only a $1/|\mathbb{B}_\kappa^n|$ chance of producing a $\mathbf{c}$ such that $\mathbf{c} = H(\mathbf{w}, \mu)$. Thus with probability $1 - 1/|\mathbb{B}_\kappa^n|$, $\mathbf{c}$ must be one of the $\mathbf{c}_i$'s, and so the probability that $\mathcal{F}$ succeeds in a forgery. Thus the probability that $\mathbf{c} = \mathbf{c}_j$ for some j is $\delta - 1/|\mathbb{B}_\kappa^n|$.

Type 1 Forgery. Suppose that $\mathbf{c}_j$ was a response to a signing query made by $\mathcal{F}$ on $(\mathbf{w}', \mu') = (\mathbf{Az}' + q\mathbf{c}_j, \mu')$. Then we would have

$$H(\mathbf{Az} + q\mathbf{c}_j, \mu) = H(\mathbf{Az}' + q\mathbf{c}_j, \mu').$$

If $\mu \neq \mu'$ or $\mathbf{Az} + q\mathbf{c}_j \neq \mathbf{Az}' + q\mathbf{c}_j$, it means that $\mathcal{F}$ found a pre-image of $\mathbf{c}_j$. Therefore with overwhelming probability, we have $\mu = \mu'$ and $\mathbf{Az} + q\mathbf{c}_j = \mathbf{Az}' + q\mathbf{c}_j$. This yields $\mathbf{A}(\mathbf{z} - \mathbf{z}') = \mathbf{0} \bmod 2q$. We know that $\mathbf{z} \neq \mathbf{z}'$ (otherwise the signatures would be the same). Moreover, since $\|\mathbf{z}\|_\infty, \|\mathbf{z}'\|_\infty \leqslant q/4$, we have $\mathbf{z} - \mathbf{z}' \neq 0 \bmod q$. Finally, the condition on the ℓ_2-norm of $\mathbf{z}$ and $\mathbf{z}'$ gives $\|\mathbf{z} - \mathbf{z}'\| \leqslant 2B$.

Type 2 Forgery. Assume now that $\mathbf{c}_j$ was a response to a random oracle query made by $\mathcal{F}$. In this case we record this signature $(\mathbf{z}, \mathbf{c}_j)$ on the message μ, and we generate fresh random elements $\mathbf{c'}_j, \ldots, \mathbf{c'}_t \xleftarrow{\$} \mathbb{B}_\kappa^n$. By the General Forking Lemma of Bellare and Neven [BN06], we obtain that the probability that $\mathbf{c'}_j \neq \mathbf{c}_j$ and the forger uses the random oracle response $\mathbf{c'}_j$ (and the query associated to it) in the forgery is at least

$$\left(\delta - \frac{1}{|\mathbb{B}_\kappa^n|}\right) \cdot \left(\frac{\delta - 1/|\mathbb{B}_\kappa^n|}{t} - \frac{1}{|\mathbb{B}_\kappa^n|}\right).$$

Thus, with the above probability, $\mathcal{F}$ outputs a signature $(\mathbf{z}', \mathbf{c'_j})$ of the message μ and $\mathbf{Az} + q\mathbf{c}_j = \mathbf{Az}' + q\mathbf{c'_j}$. We finally obtain

$$\mathbf{A}\big(\mathbf{z} - \mathbf{z}'\big) = q\big(\mathbf{c}_j - \mathbf{c'}_j\big) \bmod 2q\ .$$

Since $\mathbf{c}_j - \mathbf{c'}_j \neq \mathbf{0} \bmod 2$, we have $\mathbf{z} - \mathbf{z}' \neq \mathbf{0} \bmod 2q$. Moreover, we have $\|\mathbf{z} - \mathbf{z}'\|_\infty < q/2$: this implies that $\mathbf{v} = \mathbf{z} - \mathbf{z}' \neq \mathbf{0} \bmod q$. Finally, we have

$$\mathbf{Av} = \mathbf{0} \bmod q \qquad \text{and} \qquad \|\mathbf{v}\| \leqslant 2B_2\ ,$$

that is $\mathbf{v}$ is a solution to a $SIS^{\mathcal{K}}_{q,n,m,\beta}$ with $\beta = 2B_2$. □

4 Practical Instantiation of BLISS

In this section, we present a practical instantiation of our signature scheme inspired by the NTRU key-generation. We present optimizations and discuss implementation issues for each step of the signing algorithm (Algorithm 1). The signature scheme was implemented as a proof of concept on a desktop computer. Parameters proposals and timings are provided in Section 5.

4.1 Key-Generation

Given densities δ_1 and δ_2, we generate random polynomials $\mathbf{f}$ and $\mathbf{g}$ with $d_1 = \lceil \delta_1 n \rceil$ coefficients in $\{\pm 1\}$, $d_2 = \lceil \delta_2 n \rceil$ coefficients in $\{\pm 2\}$ and all other coefficients set to 0 until $\mathbf{f}$ is invertible.[9] The secret key is given by $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2)^t = (\mathbf{f}, 2\mathbf{g}+1)^t$.

The public key is then computed as follows: set $\mathbf{a}_q = (2\mathbf{g}+1)/\mathbf{f} \in \mathcal{R}_q$ ($\mathbf{a}_q$ is defined as a quotient *modulo* q). Next, define $\mathbf{A} = (2\mathbf{a_q}, q-2) \in \mathcal{R}_{2q}^{1\times 2}$. One easily verifies that:

$$\begin{aligned} \mathbf{AS} &= 2\mathbf{a}_q \cdot \mathbf{f} - 2(2\mathbf{g}+1) = 0 \mod q \\ \mathbf{AS} &= q(2\mathbf{g}+1) = q \cdot 1 = 1 \mod 2 \ , \end{aligned}$$

that is $\mathbf{AS} = q \bmod 2q$. Finally, $(\mathbf{A}, \mathbf{S})$ is a valid key pair for our scheme.

Denote by $\mathcal{K}_{n,\delta_1,\delta_2}$ the distribution that picks small $\mathbf{f}$ and $\mathbf{g}$ as uniform polynomials with exactly d_1 entries in $\{\pm 1\}$ and d_2 entries in $\{\pm 2\}$ and outputs the public key $\mathbf{B} = (\mathbf{a}, 1) \in \mathcal{R}_q^{1\times 2}$ for $\mathbf{a} = (2\mathbf{g}+1)/\mathbf{f}$.

The public key generated above $\mathbf{A}$ taken modulo q follows the distribution $2\mathcal{K}_{n,\delta_1,\delta_2}$; that is, such key-pair generation algorithm gives a scheme based on $\mathcal{R}\text{-SIS}_{q,1,2,\beta}^{\mathcal{K}_{n,\delta_1,\delta_2}}$.

Rejection According to $N_\kappa(\mathbf{S})$. In practice after generating $\mathbf{S}$, we restart when $N_\kappa(\mathbf{S}) \geqslant C^2 \cdot 5 \cdot (d_1 + 4d_2) \cdot \kappa$ for a fixed constant C. This constant is chosen so that 25% of the keys are accepted, decreasing the overall security by at most 2 bits.

Computation of $N_\kappa(\mathbf{S})$. Recall that

$$N_\kappa(\mathbf{S}) = \max_{\substack{I \subset \{1,\ldots,n\} \\ \#I = \kappa}} \sum_{i \in I} \left(\max_{\substack{J \subset \{1,\ldots,n\} \\ \#J = \kappa}} \sum_{j \in J} T_{i,j} \right) \quad \text{where } \mathbf{T} = \mathbf{S}^t \cdot \mathbf{S} \in \mathbb{R}^{n\times n} \ .$$

However, in order to obtain $N_\kappa(\mathbf{S})$, it is not required to compute the $2 \cdot \binom{n}{\kappa}$ sums in the definition. Indeed, it suffices to compute $\mathbf{T} = \mathbf{S}^t \cdot \mathbf{S}$, then sort the columns of $\mathbf{T}$, sum the κ larger values in each line, sort the resulting vector and to sum its κ larger components. Notice moreover that working in $\mathbb{Z}_{2q}[x]/(x^n+1)$ implies that $\mathbf{S}$ is composed of rotations (possibly with opposed coefficients) of $\mathbf{s}_i$'s, and this (ideal) structure is thus also present in $\mathbf{T}$. Thus it suffices to compute the vector

$$\mathbf{t} = \left(\langle \mathbf{s}_1, \mathbf{s}_1 \rangle + \langle \mathbf{s}_2, \mathbf{s}_2 \rangle, \langle \mathbf{s}_1, \mathbf{x} \cdot \mathbf{s}_1 \rangle + \langle \mathbf{s}_2, \mathbf{x} \cdot \mathbf{s}_2 \rangle, \ldots, \langle \mathbf{s}_1, \mathbf{x}^{n-1} \cdot \mathbf{s}_1 \rangle + \langle \mathbf{s}_2, \mathbf{x}^{n-1} \cdot \mathbf{s}_2 \rangle \right) \ ,$$

and derive $\mathbf{T} = (\mathbf{t}, \mathbf{x} \cdot \mathbf{t}, \ldots, \mathbf{x}^{n-1} \cdot \mathbf{t})$.

Theoretical Bound. We provide below a (theoretical) asymptotic bound on $N_\kappa(\mathbf{S})$ for the purposes of completeness. The following proposition easily generalizes to the form of our secret keys (see Corollary 4.2).

Proposition 4.1. *For a fixed density $\delta \in (0,1)$, and $w = \lceil \delta n \rceil$, let $\mathbf{s} \in \mathbb{Z}[x]/(x^n+1)$ be chosen uniformly in $\mathbb{T}_w^n$, and $\mathbf{S} \in \mathbb{Z}^{n\times n}$ denotes its matrix representation. Then, for any $\epsilon > 0$, we have:*

$$N_\kappa(\mathbf{S}) \leqslant w\kappa + \kappa^2 \mathcal{O}\left(w^{1/2+\epsilon}\right)$$

except with negligible probability.

[9] In order to get a better entropy/length ratio, we include a few entries in $\{\pm 2\}$ in the secret key, increasing resistance to the Hybrid attack.

Proof. The first term $w\kappa$ arises from the diagonal coefficients of $\mathbf{T} = \mathbf{S}^t \cdot \mathbf{S}$, equals to $\|\mathbf{s}\|^2 = w$. It remains to bound the non-diagonal terms of T. For $i \neq j$,

$$Y_{i,j} = \sum_{1 \leqslant k \leqslant n} \varepsilon_{i,j,k} \cdot s_{i+k} \cdot s_{j+k} \ ,$$

where $\varepsilon_{i,j,k} \in \{\pm 1\}$ are some fixed coefficients, and the indices are taken modulo n. The key argument is to split this sum into two parts, so that each part contains only independent terms. This is possible when $i - j \neq 0$ and n is a power of 2: one easily checks that there exists a set $K \subset \mathbb{Z}_n$ such that $K + i$ and $K + j$ form a partition of $\mathbb{Z}_n$. Thus, we rewrite

$$Y_{i,j} = \sigma_{i,j} + \bar{\sigma}_{i,j} \quad \text{where } \sigma_{i,j} = \sum_{k \in K} \varepsilon_{i,j,k} \cdot s_{i+k} \cdot s_{j+k} \text{ and } \bar{\sigma}_{i,j} = \sum_{k \in \mathbb{Z}_n \setminus K} \varepsilon_{i,j,k} \cdot s_{i+k} \cdot s_{j+k} \ .$$

Focusing on the sum $\sigma_{i,j}$ (a similar argument holds for $\bar{\sigma}_{i,j}$), one can restrict the sum to its non-zero terms and notice that the remaining terms are uniformly random in $\{-1, 1\}$ and independent from each other. Finally $\sigma_{i,j}$ is the sum of at most w uniform variables over $\{-1, 1\}$ and therefore $\sigma_{i,j} \leqslant w^{1/2+\epsilon}$ except with negligible probability.[10] □

Corollary 4.2. *Let $\mathbf{f}, \mathbf{g} \in \mathbb{Z}[x]/(x^n + 1)$ be chosen uniformly in $\mathbb{T}_w^n$, $\mathbf{F}, \mathbf{G} \in \mathbb{Z}^{n \times n}$ be their matrix representations, and set $\mathbf{S}^t = (\mathbf{F} | 2\mathbf{G} + \mathbf{I}_n) \in \mathbb{Z}^{n \times 2n}$. Then,*

$$N_\kappa(\mathbf{S}) \leqslant (5w + 1)\kappa + \kappa^2 \mathcal{O}\left(w^{1/2+\epsilon}\right) \ .$$

Proof. This follows easily from the fact that $\mathbf{S}^t \cdot \mathbf{S} = \mathbf{F}^T \cdot \mathbf{F} + 4\mathbf{G}^T \cdot \mathbf{G} + \mathbf{G} + \mathbf{G}^T + \mathbf{I}_n$, yielding $N_\kappa(\mathbf{S}) \leqslant N_\kappa(\mathbf{F}) + 4N_\kappa(\mathbf{G}) + 2\kappa^2 + \kappa$. □

4.2 Gaussian Sampling

In Line 1 of Algorithm 1, we want to produce $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2)^t$ where $\mathbf{y}_1, \mathbf{y}_2$ are polynomials over $\mathbb{Z}_{2q}[x]/(x^n + 1)$ with coefficients distributed according to a centered discrete Gaussian distribution of standard deviation σ. In Section 6, we provide a new technique to perform *efficiently* discrete Gaussian sampling on constrained devices. However in an environment with enough memory, using the cumulative distribution table algorithm is the easiest and fastest solution (see Table 1.2). Namely, we tabulate the approximate cumulative distribution of the desired distribution, *i.e.* the probabilities $p_z = \Pr[x \leqslant z : x \leftarrow \mathcal{D}_\sigma]$ for $z \in [-\tau\sigma, \tau\sigma]$, precomputed with λ bits of precision. At sampling time, one generates $y \in [0, 1)$ uniformly at random, then performs a binary search through the table to locate some $z \in \mathbb{Z}$ such that $y \in [p_{z-1}, p_z)$ and outputs z. A GCC-profiling of our program reveals that this step takes about 35% of the entire running-time, including the entropy generation using `sha-512`.

4.3 Multiplication of two polynomials

In Line 2 of Algorithm 1, the element $\mathbf{Ay} = \mathbf{a}_1 \cdot \mathbf{y}_1 + \mathbf{a}_2 \cdot \mathbf{y}_2 \in \mathbb{Z}_{2q}[x]/(x^n + 1)$ is given as input to the random oracle. Since $\mathbf{a}_2 = q - 2$ is a constant, $\mathbf{a}_2 \cdot \mathbf{y}_2$ is straightforward to obtain. It remains to (efficiently) compute the product of $\mathbf{a}_1$ by $\mathbf{y}_1$ over $\mathbb{Z}_{2q}[x]/(x^n + 1)$.

[10] By Hoeffding bound for example, or classical properties of random walks.

Because of the particular shape of $\mathbf{a}_1$ in the NTRU-like key generation, namely that $\mathbf{a}_1$ is lifted from $\mathbb{Z}_q[x]/(x^n+1)$ to $\mathbb{Z}_{2q}[x]/(x^n+1)$ by multiplying its coefficients by 2 (*i.e.* $\mathbf{a}_1 = 2\cdot\mathbf{a'}_1$), computing $\mathbf{a_1}\cdot\mathbf{y}_1$ over $\mathbb{Z}_{2q}[x]/(x^n+1)$ can be done by computing the product $\mathbf{a'}_1\cdot\mathbf{y}_1$ over $\mathbb{Z}_q[x]/(x^n+1)$ and then multiplying the coefficients of the result by 2. Now multiplying two polynomials of $\mathbb{Z}_q[x]/(x^n+1)$ for q prime is made efficient by choosing a modulus q such that $q = 1 \bmod 2n$: there exists then a primitive $2n$-th root ω of unity modulo q. Finally, the multiplication can be done in complexity $\mathcal{O}(n \log n)$ via a Number Theoretic Transform (*i.e.* Fast Fourier Transform over a finite field). Details on these standard techniques can be found for example in [PG12,Ber08]. Notice that one does not need to work with vectors of size $2n$ as the component-wise multiplication of the NTT representations of size n of $\mathbf{a'}_1(\omega x)$ and $\mathbf{y}_1(\omega x)$ gives the NTT representation of $[\mathbf{a'}_1 \cdot \mathbf{y}_1](\omega x) \in \mathbb{Z}_q[x]/(x^n+1)$.

4.4 Hashing to $\mathbb{B}^n_\kappa$

We discuss how to build a hash function outputting uniform vectors in $\mathbb{B}^n_\kappa$ from a standard hash function H (used in Line 2 of Algorithm 1). In [Lyu12], it was suggested to use a Hash function with $\kappa \log_2 \binom{n}{\kappa}$ bits of output (recall that $\#\mathbb{B}^n_\kappa = \binom{n}{\kappa}$, and thus $\#\mathbb{T}^n_\kappa = 2^\kappa \binom{n}{\kappa}$), and then apply a one-to-one map to $\mathbb{T}^n_\kappa$. Such a mapping can be found in [FS96] but its complexity is quadratic in n; this is quite inefficient especially for large parameters. To avoid this costly algorithm, the authors of [GLP12] used an efficient procedure injectively mapping 160-bit strings to $\mathbb{T}^{512}_{32}$; they increased the value κ from 20 to 32 to gain efficiency, yielding a larger signature size.

Overview. We here give an alternative solution that is both efficient and optimal (*i.e.* κ is minimal for a target entropy) to produce random elements in $\mathbb{B}^n_\kappa$. In a few words, our approach consists in obtaining $\kappa' > \kappa$ values $x_1 \dots x_{\kappa'}$ in $\mathbb{Z}_n$, and setting the coordinates $\mathbf{c}_{x_i}$ of the challenge $\mathbf{c}$ to 1, starting from $i = 1$, and until $\|\mathbf{c}\|_1 = \kappa$. If some coordinate $\mathbf{c}_{x_j}$ is already set to 1 one just ignores this x_j. If we run out of values x_j, we would restart the process using a different seed. In the following we describe more precisely this algorithm and show it indeed produces a uniform random function over $\mathbb{B}^n_\kappa$ if H is indeed a uniform random function over $\mathbb{Z}^k_n$.

Detailed Construction and Correctness. Let n be a power of 2 and $H_0 \colon \{0,1\}^* \to \mathbb{Z}^{\kappa'}_n$ with $\kappa' > \kappa$ be a random function outputting $\kappa' \log_2 n$ bits (parsed as κ' elements in $\mathbb{Z}_n$). We consider the set $S \subset \mathbb{Z}^{\kappa'}_n$ of vectors that have at least κ different entries. The probability that a uniform element in $\mathbb{Z}^{\kappa'}_n$ lies in S is:

$$A = 1 - \frac{|\mathbb{Z}^{\kappa'}_n \setminus S|}{|\mathbb{Z}^{\kappa'}_n|}.$$

When A is not negligible, one can efficiently build a random function $\widetilde{H} \colon \{0,1\}^* \to S$ as $\widetilde{H}(x) = H(x|i)$, where i is the smallest index such that $H(x|i) \in S$. This is somehow a rejection sampling technique applied to a random function. Finally, in average, one call of $\widetilde{H}$ requires $1/A$ calls to H.

Fact 4.3. *With the notation above,* $|\mathbb{Z}^{\kappa'}_n \setminus S| \leqslant \binom{n}{\kappa-1}(\kappa-1)^{\kappa'}$.

Proof. Notice that $|\mathbb{Z}^{\kappa'}_n \setminus S|$ is the set of vectors over $\mathbb{Z}_n$ of length κ' with at most $\kappa - 1$ distinct coordinates. To obtain this set, one may first choose a subset $K \subset \mathbb{Z}_n$ of size $\kappa - 1$ ($\binom{n}{\kappa-1}$ choices),

and then chooses the κ' coordinates in K ($(\kappa-1)^{\kappa'}$ choices). Note that vectors with strictly less than $\kappa-1$ coordinates have been counted several times. More formally:

$$\mathbb{Z}_n^{\kappa'} \setminus S = \bigcup_{\substack{K\subset\mathbb{Z}_n \\ |K|<\kappa-1}} K^{\kappa'} = \bigcup_{\substack{K\subset\mathbb{Z}_n \\ |K|=\kappa-1}} K^{\kappa'} . \qquad \square$$

For our parameters, this gives $1/A \leqslant 1.00001$ using a 512-bit hash function for H (*e.g.* BLISS-IV: $n = 2^9$, $\kappa' = 56$, $\kappa = 39$).

It remains to map the domain S to $\mathbb{B}_\kappa^n$. For $\mathbf{x} \in S$, let I be the set of the κ first distinct coordinates values of $\mathbf{x}$, and set $f(\mathbf{x}) = \sum_{i\in I} \mathbf{e}_i \in \mathbb{B}_\kappa^n$ where $\mathbf{e}_1, \ldots, \mathbf{e}_n$ are the canonical vectors of $\mathbb{Z}^n$. Each image $\mathbf{y} \in \mathbb{B}_\kappa^n$ has the same amount of f-preimages in S, therefore $\widehat{H}\colon \{0,1\}^* \to \mathbb{B}_\kappa^n$ defined as $\widehat{H}(x) = f \circ \widetilde{H}(x)$ is also a random function.

4.5 Multiplication of S by a sparse vector c

In Line 4 of Algorithm 1, one should compute $\mathbf{Sc}$. Let $\mathbf{S}_i$, $i = 1, 2$ denotes the $n \times n$ matrix over $\mathbb{Z}_{2q}$ whose columns vectors are the $x^j \cdot \mathbf{s}_i$'s for $j = 0, \ldots n-1$ In particular we have that

$$\mathbf{s}_i \cdot \mathbf{c} = \mathbf{S}_i\mathbf{c} .$$

Now, since $\mathbf{c}$ is a sparse binary vector, one should not use the NTT to compute $\mathbf{s}_i \cdot \mathbf{c}$ for this step (contrary to Section 4.3). Indeed, the absolute value of the coefficients of $\mathbf{s}_1$ and $\mathbf{s}_2$ is smaller than 5, yielding $\|\mathbf{s}_i \cdot \mathbf{c}\|_\infty \leqslant 5\kappa \ll 2q$, $i = 1, 2$. Therefore, computing $(\mathbf{s}_1 \cdot \mathbf{c})$ and $(\mathbf{s}_2 \cdot \mathbf{c})$ can be performed very efficiently by additions over $\mathbb{Z}$ (*i.e.* without reduction modulo $2q$) of κ pre-stored columns of $\mathbf{S}_i$. Notice moreover that working over $\mathbb{Z}_{2q}[x]/(x^n+1)$ allows to reduce the memory storage overhead to zero: all the columns of $\mathbf{S}_i$ are rotations (possibly with opposite coefficients) of $\mathbf{s}_i$.

4.6 Rejection Sampling according to 1/ exp and 1/ cosh

In Line 5 of Algorithm 1, one should reject with probability $1/(M \exp(-x/f) \cosh(x'/f))$. To avoid floating-point computations of the transcendental functions exp and cosh, we use the techniques described in Section 6 to do it efficiently with a very small memory footprint. Notice that the precomputed values can be the same both for Gaussian sampling and this rejection sampling step.

4.7 Signature Compression

Recall that the signature is a pair $(\mathbf{z}, \mathbf{c})$ where $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2)^t$ follows the Gaussian distribution D_σ^{2n}.

Working with A in Hermite Normal Form. As in [GLP12], in order to compress our signature, we need to have $\mathbf{A}$ in Hermite Normal Form. Now, during the key-generation process, we explicitly constructed $\mathbf{A} = (\mathbf{a}_1, q-2)$ such that

$$\mathbf{a}_1 \cdot \mathbf{s}_1 + (q-2)\mathbf{s}_2 = q \bmod 2q .$$

Let us define ζ such that $\zeta \cdot (q-2) = 1 \bmod 2q$. Next, instead of calling the random oracle on $(\mathbf{Ay} \bmod 2q, \mu)$, we can call it on $\big((\zeta\mathbf{A})\mathbf{y} \bmod 2q, \mu\big)$ because $\zeta\mathbf{A} = (\zeta\mathbf{a}_1, 1)$ is in Hermite Normal Form.

Dropping the low-order bits of $\mathbf{z}_2$. We denote by d the number of bits we would like to drop in $\mathbf{z}_2$. For every integer x in the range $[-q, q)$ and any positive integer d, x can be uniquely written

$$x = \lfloor x \rceil_d \cdot 2^d + [x \bmod 2^d] \ ,$$

where $[x \bmod 2^d] \in [-2^{d-1}, 2^{d-1})$. Thus $\lfloor x \rceil_d$ can be viewed as the "high-order bits" of x and $[x \bmod 2^d]$ as its "low-order bits".

In [GLP12], the signature size is reduced by dropping almost all the information about $\mathbf{z}_2$ in the signature. Such a strategy impacts security, as it reduces to an easier SIS problem (it may allow an attacker to forge using longer vectors). Let us describe a similar feature for our signature scheme. First, we replace the random oracle H input by

$$\begin{aligned}\left(\lfloor (\zeta\mathbf{A})\mathbf{y} \rceil_d, \mu\right) &= \left(\lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{y}_1 + \mathbf{y}_2 \bmod 2q \rceil_d, \mu\right) \\ &= \left(\lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} + \mathbf{z}_2 \bmod 2q \rceil_d, \mu\right). \end{aligned} \tag{4}$$

The idea of [GLP12], transposed to our settings, was to define a vector $\widetilde{\mathbf{z}_2}$ with coefficients in $\{0, \pm 2^d\}$ and a limited number of coefficients $\widetilde{\mathbf{z}_2}[i] = \mathbf{z}_2[i]$ (coming from the need of reduction modulo $2q$ after the addition with small but non negligible probability) such that

$$\lfloor (\zeta\mathbf{A})\mathbf{y} \rceil_d = \lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} + \widetilde{\mathbf{z}_2} \bmod 2q \rceil_d \ .$$

Unfortunately the workaround which consists in storing some coefficients uncompressed, *i.e.* of the form $\widetilde{\mathbf{z}_2}[i] = \mathbf{z}_2[i]$, yields a signature scheme which is not strongly unforgeable. Indeed it is easy to forge a signature by modifying the least-significant bit of one of the uncompressed values, and this does not modify the high-order bits of the sum with very high probability.[11]

Let us describe how to solve this issue for our signature scheme. We want to replace $\mathbf{z}_2$ by a small vector $\mathbf{z}_2^\dagger$ such that

$$\lfloor (\zeta\mathbf{A})\mathbf{y} \rceil_d = \lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} \bmod 2q \rceil_d + \mathbf{z}_2^\dagger \ .$$

Unfortunately without additional modification, the security proof does not go through because of a similar issue as in [GLP12], *i.e.* the coefficients $\mathbf{z}_2[i]$ of $\mathbf{z}_2$ which, added to $\left(\zeta \cdot (\mathbf{a}_1 \cdot \mathbf{z}_1)[i] + \zeta \cdot q \cdot \mathbf{c}[i]\right)$, force us to reduce the result modulo $2q$ in Equation (4). Let us define $p = \lfloor 2q/2^d \rfloor$; we have $2q = p \cdot 2^d + \nu$ with a small ν (typically $\nu = 1$ in our parameters). Now we modify the random oracle H input by

$$\left(\lfloor (\zeta\mathbf{A})\mathbf{y} \rceil_d \bmod p, \mu\right) \ ,$$

and define

$$\mathbf{z}_2^\dagger = \left(\lfloor (\zeta\mathbf{A})\mathbf{y} \rceil_d - \lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} \bmod 2q \rceil_d \bmod p\right) \in [0, p)^n \ .$$

The coefficients of $\mathbf{z}_2^\dagger$ are small modulo p. We redefine the signature to be $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$ instead of $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$, and during the verification, we check that

$$H\left(\mathbf{z}_2^\dagger + \lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} \bmod 2q \rceil_d \bmod p, \mu\right) = \mathbf{c} \ ,$$

that $\left\|(\mathbf{z}_1 \| 2^d \mathbf{z}_2^\dagger)\right\| \leqslant B_2$ and that $\left\|(\mathbf{z}_1 \| 2^d \mathbf{z}_2^\dagger)\right\|_\infty \leqslant B_\infty$.

Finally, we have the following theorem:

[11] As a direct consequence, the scheme of [GLP12] is not strongly unforgeable.

Theorem 4.4. *Let us consider the signature scheme of Section 4.8. Assume that $d \geqslant 3$, $q \equiv 1 \bmod 2^{d-1}$, and $2B_\infty + (2^d + 1) < q/2$. Suppose there is a polynomial-time algorithm $\mathcal{F}$ which succeeds in forging with non negligible probability. Then there exists a polynomial-time algorithm which can solve the $\mathcal{R}$-SIS$^{\mathcal{K}}_{q,n,m,\beta}$ problem for $\beta = 2B_2 + (2^d + 1)\sqrt{n}$.*

Proof. The differences with the proof of Theorem 3.3 are detailed in Appendix C. □

Compressing Most Significant Bits of $\mathbf{z_1}$ and $\mathbf{z_2^\dagger}$. The simplest representation of the entries of $\mathbf{z}_1$ then requires $\lceil \log_2(8\sigma) \rceil \leqslant \log_2(16\sigma)$ bits. Yet, the entropy of these entries is actually smaller:

Fact 4.5. *Let X be distributed as D_σ, that is a centered discrete Gaussian variable. Then the entropy of X is upper-bounded by:*

$$\mathcal{H}(X) \leqslant \frac{1}{\sigma^3} + \log_2(\sqrt{2\pi e}\sigma) \approx \log_2(4.1\sigma) \ .$$

Now, Huffman coding provides (almost) optimal encoding for data when their distribution is exactly known. More precisely:

Theorem 4.6 (Huffman Coding). *For any random variable X over a finite support S, there exist an injective prefix-free code $C \colon S \to \{0,1\}^*$ such that:*

$$\mathcal{H}(X) \leqslant E\left[|C(X)|\right] < \mathcal{H}(X) + 1 \ .$$

To keep the compression efficient, we choose to only encode the highest bits of all entries; the lower are almost uniform and therefore we do not loose anything by not compressing them. Moreover, if by packing several independent variables $X_1, \ldots, X_k$, we can decrease the overhead to $1/k$.

4.8 Final KeyGen, Sign and Verify Algorithms

In this section, we describe the final algorithms to instantiate BLISS with the parameters of Section 5. Notice that to obtain the signature size indicated Table 1 (page 2), one need to use Huffman Coding to compress the highest bits of $\mathbf{z}_1$ and $\mathbf{z}_2^\dagger$. Let us define $p = \lfloor 2q/2^d \rfloor$ where d is the number of dropped bits.

Algorithm 5: BLISS Key Generation

Output: Key pair $(\mathbf{A}, \mathbf{S})$ such that $\mathbf{AS} = q \bmod 2q$

1: Choose $\mathbf{f}, \mathbf{g}$ as uniform polynomials with exactly d_1 entries in $\{\pm 1\}$ and d_2 entries in $\{\pm 2\}$
2: $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2)^t \leftarrow (\mathbf{f}, 2\mathbf{g} + 1)^t$
3: **if** $N_\kappa(\mathbf{S}) \geqslant C^2 \cdot 5 \cdot (\lceil \delta_1 n \rceil + 4\lceil \delta_2 n \rceil) \cdot \kappa$ **then**
4: **restart**
5: **end if**
6: $\mathbf{a}_q = (2\mathbf{g} + 1)/\mathbf{f} \bmod q$ (**restart** if $\mathbf{f}$ is not invertible)
7: **Output**$(\mathbf{A}, \mathbf{S})$ where $\mathbf{A} = (2\mathbf{a}_q, q - 2) \bmod 2q$

Algorithm 6: BLISS Signature Algorithm

Input: Message μ, public key $\mathbf{A} = (\mathbf{a}_1, q-2) \in \mathcal{R}_{2q}^{1\times 2}$, secret key $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2)^t \in \mathcal{R}_{2q}^{2\times 1}$

Output: A signature $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$ of the message μ

1: $\mathbf{y}_1, \mathbf{y}_2 \leftarrow D_{\mathbb{Z}^n,\sigma}$
2: $\mathbf{u} = \zeta \cdot \mathbf{a}_1 \cdot \mathbf{y}_1 + \mathbf{y}_2 \bmod 2q$
3: $\mathbf{c} \leftarrow H(\lfloor \mathbf{u} \rceil_d \bmod p, \mu)$
4: Choose a random bit b
5: $\mathbf{z}_1 \leftarrow \mathbf{y}_1 + (-1)^b \mathbf{s}_1 \mathbf{c}$
6: $\mathbf{z}_2 \leftarrow \mathbf{y}_2 + (-1)^b \mathbf{s}_2 \mathbf{c}$
7: **Continue** with probability $1 \Big/ \left(M \exp\left(-\frac{\|\mathbf{Sc}\|^2}{2\sigma^2}\right) \cosh\left(\frac{\langle \mathbf{z}, \mathbf{Sc}\rangle}{\sigma^2}\right)\right)$ otherwise **restart**
8: $\mathbf{z}_2^\dagger \leftarrow (\lfloor \mathbf{u} \rceil_d - \lfloor \mathbf{u} - \mathbf{z}_2 \rceil_d) \bmod p$
9: **Output** $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$

Algorithm 7: BLISS Verification Algorithm

Input: Message μ, public key $\mathbf{A} = (\mathbf{a}_1, q-2) \in \mathcal{R}_{2q}^{1\times 2}$, signature $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$

Output: Accept or Reject the signature

1: **if** $\|(\mathbf{z}_1 | 2^d \cdot \mathbf{z}_2^\dagger)\|_2 > B_2$ **then** Reject
2: **if** $\|(\mathbf{z}_1 | 2^d \cdot \mathbf{z}_2^\dagger)\|_\infty > B_\infty$ **then** Reject
3: Accept iff $\mathbf{c} = H\big(\lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} \rceil_d + \mathbf{z}_2^\dagger \bmod p, \mu\big)$

5 Parameters and Benchmarks

In this section, we first propose parameters sets for the scheme BLISS described in Section 4. Next, we compare the benchmarks of our proof-of-concept implementations with the `openssl` running times of RSA and ECDSA.

5.1 Parameters Sets

In Table 5, we propose several sets of parameters to implement the $\mathcal{R}$-SIS$^{\mathcal{K}}$ variant of our scheme described in Section 4. The signature schemes BLISS-I and BLISS-II are respectively optimized for speed and compactness and offer 128 bits of security (*i.e.* long-term protection [NIS11,ECR12]). The signature schemes BLISS-III and BLISS-IV offer respectively 160 and 192 bits of security. The two last lines provide typical security measurement against direct lattice attack in terms of Hermite factor, but slightly better attacks exist. Therefore, our security claims are derived from an extensive analysis based on BKZ-2.0 simulation [CN11] in interaction with other techniques [MR09,MM11,HG07] detailed in Appendix A.

One of the objectives of this work was to determine whether the scheme from [Lyu12] could be improved so as it remains sufficiently secure for a dimension $n = 256$. Even though this seems possible when only considering *direct* lattice attacks, it turns out to be slightly out of reach according to the analysis of Appendix A. Any additional trick might unlock an extremely efficient 80-bit secure signature scheme; it seems to us a challenging but worthwhile goal. We do however propose a toy variant BLISS-0 in this dimension for which we expect up to 60 bits of security. Yet, we believe it would require a significant effort to break this toy variant; we leave it as a challenge to motivate

Name of the scheme	**BLISS-0**	**BLISS-I**	**BLISS-II**	**BLISS-III**	**BLISS-IV**
Security	Toy ($\leqslant$ 60 bits)	128 bits	128 bits	160 bits	192 bits
Optimized for	Fun	Speed	Size	Security	Security
n	256	512	512	512	512
Modulus q	7681	12289	12289	12289	12289
Secret key densities δ_1, δ_2	.55 , .15	.3 , 0	.3 , 0	.42 , .03	.45, .06
Gaussian standard deviation σ	100	215	107	250	271
α	.5	1	.5	.7	.55
κ	12	23	23	30	39
Secret key N_κ-Threshold C	1.5	1.62	1.62	1.75	1.88
Dropped bits d in $\mathbf{z}_2$	5	10	10	9	8
Verification thresholds B_2, B_∞	2492, 530	12872, 2100	11074, 1563	10206,1760	9901, 1613
Repetition rate	7.4	1.6	7.4	2.8	5.2
Entropy of challenge $\mathbf{c} \in \mathbb{B}_\kappa^n$	66 bits	132 bits	132 bits	161 bits	195 bits
Signature size	**3.3kb**	**5.6kb**	**5kb**	**6kb**	**6.5kb**
Secret key size	1.5kb	2kb	2kb	3kb	3kb
Public key size	3.3kb	7kb	7kb	7kb	7kb
SIS parameter $\beta/\sqrt{q}$ (as in Theorem 4.4)	$63 = 1.0083^m$	$441 = 1.0060^m$	$409 = 1.0059^m$	$289 = 1.0055^m$	$231 = 1.0053^m$
Ring-Unique-SVP parameter $\sqrt{\frac{qm}{2\pi e}}\Big/\lambda_1$	$14 = 1.0051^m$	$46 = 1.0037^m$	$46 = 1.0037^m$	$30 = 1.0033^m$	$25 = 1.0031^m$

Table 3. Parameter proposals

further advance in lattice cryptanalysis. Notice that choosing a non power-of-two dimension n would have been possible but yields several unwelcome consequences: on efficiency first as NTT becomes at least twice slower and the geometry is worse (our constant C grows), but also on simplicity as one will no longer work as on the simple quotient by $x^n + 1$. However, it is possible to get about 100 bits of security in dimension $n = 379$ for signatures of size 4kb. In comparison [Lyu12, **Set-IV**] and [GLP12, **Set-I**] have respective signature sizes of 15kb and 9.5kb, for a claimed security of 100 bits.[12]

5.2 Timings

In Table 1, we provide running times of our proof-of-concept implementation of our signature scheme with the parameters provided above, on a desktop computer. We also provide running times for the `openssl` implementations of RSA and ECDSA. Notice that, despite the lack of optimization on our proof-of-concept implementation, we derived interesting timings. First, our verification time is nearly the same for each of our variants, and is much faster than the RSA and the (even worse) ECDSA verifications by a factor 10 to 30. Secondly, excluding RSA which is really slow, the signature algorithm of BLISS-I is as fast as ECDSA-256 (with the same claimed security). We refer to [NIS11,ECR12] to get the equivalence between the key length of RSA and ECDSA and the expected security in bits.

Besides, we expect our scheme to be much more suitable to embedded devices than both RSA and ECDSA, mainly because our operation are done with a very small modulus (less than 16 bits). By design, the binary representation of q is `11 0000 0000 0001`, that is q has a very small

[12] Our analysis in Appendix A shows that the security of [GLP12, **Set-I**] may actually a little lower than claimed.

Hamming weight; this structure might yield interesting hardware optimizations. The main issue for such architectures is the generation of discrete Gaussian, addressed in Section 6.

6 Efficient Gaussian Sampling for Lightweight Devices

Since its introduction, and with the noticeable exception of NTRU, lattice-based cryptosystems operating at a standard security level have remained out of reach of constrained devices by *several orders of magnitude*. A first step towards a practical lattice-based signature scheme was achieved by [GLP12] with an implementation on a low-cost FPGA, by avoiding Gaussians, at the cost of some compactness and security compared to [Lyu12].

At this time, all known algorithms to sample according to a distribution statistically close to a discrete Gaussian distribution on a lattice [GPV08] require either long-integer arithmetic [DN12a] at some point or large memory storage [Pei10,GD12]. Some progress was made in [DN12a], showing that "lazy techniques" can limit the need for high precision; one can use floating-point numbers at double precision (53 bits) most of the time, native on high-end architectures but costly on embedded devices.

Section Outline. The main goal of this section is to show how to efficiently sample discrete Gaussian without resorting to large precomputed tables, nor evaluations of transcendental function. The first step is being able to sample according to a Bernoulli distribution with bias of the form $\exp(-x/f)$ (and $1/\cosh(x/f)$) without actually computing transcendental functions (Section 6.2). The second step is to build an appropriate and efficient distribution (see Figure 3(b)) as input of rejection sampling to reduce its rejection rate (Section 6.3). Our new algorithm still requires precomputed tables, but of much smaller size; precisely of size logarithmic in σ rather than linear.

6.1 Discrete Gaussian Sampling: Prior Art

Laziness. Laziness is an algorithmic trick saving both computation and entropy consumption; for our purpose, it is used in two cases of application. First, as in many compilers, when computing $a \wedge b$ and $a \vee b$, b is not always evaluated depending on the value of a. The second concerns the comparisons $r < c$: the result might be decided only knowing their first different bit; for a uniform $r \in [0, 1)$, only 2 bits are needed on average. In practice however, one may apply this technique word by word rather than bit by bit.

Sampling with a Constant Bias. Sampling from a distribution statistically close to a Bernoulli variable $\mathcal{B}_c$ for a given bias c is easy: to get a variable $(2^{-\lambda})$-close to $\mathcal{B}_c$, take an approximation of c up to λ correct bits, then sample a uniform real $r \in [0, 1)$ up to λ bits of precision and answer 1 if and only if $r < c$.

General Algorithm. A general algorithm to sample according to a discrete Gaussian distribution $\mathcal{D}_{\sigma,c}$ centered in $c \in \mathbb{R}$ was proposed in [GPV08] and is depicted on Figure 3(a) for $c = 0$. It uses rejection sampling from the uniform distribution $\mathcal{U}$ over $[c - \tau\sigma, c + \tau\sigma]$ by outputting a uniform integer x with probability $p(x) = \exp(-(x-c)^2/(2\sigma^2))$. This algorithm requires about $2\tau/\sqrt{2\pi}$ trials in average, and thus $\mathcal{O}(\tau \log_2(\sigma))$ bits of entropy using laziness. The main drawback is the need to compute the exp function with very high-precision. Additionally, an average of $2\tau/\sqrt{2\pi} \approx 10$ trials until acceptance is rather expensive. We address those issues in Sections 6.2 and 6.3, where we show how to avoid explicit computations of exp and decrease the repetition factor to 1.47.

Cumulative Distribution Table (CDT). In [Pei10], Peikert suggested to use a cumulative distribution table to sample more efficiently (with complexity $O(\log_2 \sigma)$) when c is known in advance. One tabulates the approximate cumulative distribution of the desired distribution, *i.e.* the probabilities $p_z = \Pr[x \leqslant z : x \leftarrow \mathcal{D}_{\sigma,c}]$ for $z \in [c-\tau\sigma, c+\tau\sigma]$, precomputed with λ bits of precision. At sampling time, one generates $y \in [0,1)$ uniformly at random, performs a binary search through the table to locate some $z \in \mathbb{Z}$ such that $y \in [p_{z-1}, p_z)$ and outputs z. This approach consumes $\mathcal{O}(\log_2(\sigma))$ bits of entropy, which is optimal up to a constant factor. The main drawback of this approach resides in the size of the table. Taking our set of parameters (see Section 5), the storage requirement is $(\lambda\tau\sigma)$ bits, *i.e.* up to 630kb, which is unsuitable for many embedded devices.

Combination with the Knuth-Yao Algorithm. In an extensive study on discrete Gaussian distributions [GD12], Galbraith and Dwarakanath suggest to combine the previous method with the Knuth-Yao algorithm. This leads to a significant decrease of the table size by a factor slightly less than 2. Unfortunately, the obtained tables remain prohibitively large.

In the following, we show how to achieve a much smaller precomputation storage (up to 4kb for our parameters) at the expense of more input entropy (see Table 1.2, page 7).

6.2 Efficient Sampling of $\mathcal{B}_{\exp(-x/f)}$ and $B_{1/\cosh(x/f)}$

Requirements. To implement the rejection step, it is enough to sample according $\mathcal{B}_{\exp(-x/f)}$ and $B_{1/\cosh(x/f)}$ where x is a bounded integer and f a fixed real. Our sampler for $\mathcal{B}_{\exp(-x/f)}$ will also be useful later to build our efficient Gaussian Sampler.

Main Idea. Our solution uses the fact that appropriate combinations of Bernoulli variables can easily produce new Bernoulli variables with combined biases. We make use of that observation to avoid an explicit computation of c and require much fewer precomputed values. Typically, if one has access to Bernoulli variables $\mathcal{B}_a, \mathcal{B}_b$ three new Bernoulli variables are easily derived from them: $\mathcal{B}_{1-a} = \neg\mathcal{B}_a$, $\mathcal{B}_{ab} = \mathcal{B}_a \wedge \mathcal{B}_b$ and $\mathcal{B}_{a+b-ab} = \mathcal{B}_a \vee \mathcal{B}_b$. We build a new operator $\oslash$ such that $\mathcal{B}_a \oslash \mathcal{B}_b = \mathcal{B}_{a/(1-(1-a)b)} = \mathcal{B}_a \oslash \mathcal{B}_b$, allowing one to homomorphically introduce fractions in the Bernoulli algebra.

Efficient Bernoulli Sampling with Exponential Biases. The problem is as follows: for a fixed real f, a positive integer $x \leqslant 2^\ell$ given as input, sample a random Boolean according to $\mathcal{B}_{\exp(-x/f)}$. Using the simple homomorphic property of the exponential function, our approach, implemented by Algorithm 8, requires only ℓ precomputed entries, and *no* evaluation of transcendental functions.

Lemma 6.1. *For any integer $x \in [0, 2^\ell)$, Algorithm 8 outputs a bit according to $\mathcal{B}_{\exp(-x/f)}$.*

Proof. Denoting the binary decomposition of x by $x = \sum_{i=0}^{\ell-1} x_i 2^i$ with $x_i \in \{0,1\}$, we have

$$\mathcal{B}_{\exp(-x/f)} = \mathcal{B}_{\exp\left(-\sum_i x_i 2^i/f\right)} = \mathcal{B}_{\prod_i \exp(-x_i 2^i/f)} = \bigwedge_{i \text{ s.t. } x_i=1} \mathcal{B}_{\exp(-2^i/f)} \ . \qquad \square$$

Algorithm 8: Sampling $\mathcal{B}_{\exp(-x/f)}$ for $x \in [0, 2^\ell)$

Input: $x \in [0, 2^\ell)$ an integer in binary form $x = x_{\ell-1} \cdots x_0$
Precomputation: $c_i = \exp(-2^i/f)$ for $0 \leqslant i \leqslant \ell - 1$
  for $i = \ell - 1$ **to** 0
    if $x_i = 1$ **then**
      sample $A_i \leftarrow \mathcal{B}_{c_i}$
      if $A_i = 0$ **then** return 0
  return 1

Algorithm 9: Sampling $\mathcal{B}_a \oslash \mathcal{B}_b$

sample $A \leftarrow \mathcal{B}_a$
if A **then** return 1
sample $B \leftarrow \mathcal{B}_b$
if $\neg B$ **then** return 0
restart

Remark 6.2. Notice that Algorithm 8 is defined so that the smallest probabilities are checked first, so that the algorithm can terminate faster. Notice that this algorithm is very fast, and uses at worst $2\lceil \log_2(x) \rceil$ bits of entropy, and much less on average for random x.

Efficient Bernoulli Sampling with Inverse Hyperbolic Cosine Biases. During the final rejection step of our signing procedure, one needs to reject with probability $1/\cosh(x/f)$ for a given f. Recall that

$$\frac{1}{\cosh(x/f)} = \frac{2}{\exp(|x|/f) + \exp(-|x|/f)} = \frac{\exp(-|x|/f)}{1/2 + 1/2 \cdot \exp(-2|x|/f)} \,. \tag{5}$$

To sample efficiently according to the Bernoulli distribution $\mathcal{B}_{1/\cosh(x/f)}$, we reuse the previous generator for $\mathcal{B}_{\exp(-x/f)}$ with no explicit evaluation of exp or cosh. In order to deal with the fraction in Equation (5), we introduce a new operation denoted $\oslash$ and computed according to Algorithm 9.

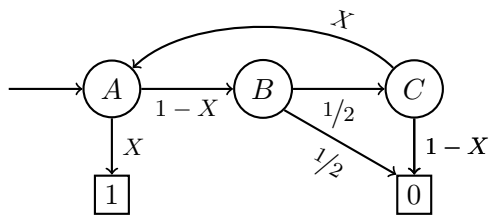
Lemma 6.3 (Correctness and Efficiency of Algorithm 9). *For any $a, b \in (0, 1]$ we have, $\mathcal{B}_a \oslash \mathcal{B}_b = \mathcal{B}_{a/(1-(1-a)b)}$ and Algorithm 9 terminates after an average of $1/(1-(1-a)b)$ trials.*

Proof. At each trial, the probability of restarting is $(1-a)b$. Now, the probability that it outputs 1 is easily computed as the sum over each trial:

$$\Pr[\mathcal{B}_a \oslash \mathcal{B}_b = 1] = a \sum_{k=0}^{\infty} (1-a)^k b^k = \frac{a}{1-(1-a)b} \qquad \square$$

Corollary 6.4. *For any $x \in \mathbb{R}$ we have: $\mathcal{B}_{1/\cosh(x)} = \mathcal{B}_{\exp(-|x|)} \oslash \left(\mathcal{B}_{1/2} \vee \mathcal{B}_{\exp(-|x|)}\right)$ and Algorithm 9 requires less than 1.53 calls to $\mathcal{B}_{\exp(-|x|)}$ on average.*

Proof. Correctness is a direct application of the previous lemma. Set $X = \exp(-|x|)$. Algorithm 9 for the computation of the Bernoulli variable $\mathcal{B}_X \oslash \left(\mathcal{B}_{1/2} \vee \mathcal{B}_X\right)$ can be seen as the following Markov chain:



Let $\mathbf{M}$ denote the restriction of the transition matrix to the states A, B and C (indexed in that order), and let $\mathbf{v} = (1, 0, 0)^t$ be the initial density vector. The density vector after k steps is $\mathbf{M}^k \cdot \mathbf{v}$, so the average number of steps through each state A, B and C is given by the vector

$$\mathbf{w} = (w_A, w_B, w_C) = \sum_{k=0}^{\infty} \mathbf{M}^k \cdot \mathbf{v} = (\mathbf{I}_3 - \mathbf{M})^{-1} \cdot \mathbf{v}$$

where

$$\mathbf{M} = \begin{pmatrix} 0 & 0 & X \\ 1-X & 0 & 0 \\ 0 & \frac{1}{2} & 0 \end{pmatrix} \quad \text{and} \quad (\mathbf{I}_3 - \mathbf{M})^{-1} \cdot \mathbf{v} = \frac{1}{2 + X(X-1)} \begin{pmatrix} 2 \\ -2X+2 \\ 1-X \end{pmatrix}.$$

Since the calls to $\mathcal{B}_{\exp(-|x|/f)}$ are performed during the states A and C, the average number of calls to this Bernoulli sampling is $C(X) := w_A + w_C = \frac{3-X}{2+X(X-1)}$. One easily derives an upper bound for $C(X)$:

$$C(X) \leqslant \frac{5 + 4\sqrt{2}}{7} \approx 1.523,$$

reached when $X = 3 - 2\sqrt{2}$. □

6.3 Sampling Centered Discrete Gaussian Variables

Based on Algorithm 8 to sample efficiently from $\mathcal{B}_{\exp(-x/f)}$, it is now possible to obtain a Gaussian distribution via generic rejection sampling algorithm as in [GPV08], trading high-precision evaluation of transcendental functions against a table of $\log_2(\tau^2\sigma^2)$ precomputed values (see Figure 3(a)). However, the algorithm still requires $(2\tau/\sqrt{2\pi}) \approx 10$ trials on average to output an x statistically close to the correct distribution. This is due to the significant distance between the uniform distribution and the target distribution.

In what follows, we introduce a new sampling algorithm with an average number of rejections smaller than 1.47. We achieve that result by sampling from a specific distribution denoted D_{k,σ_2}, for which sampling is easy. The distribution D_{k,σ_2} is much closer to the target distribution $D_{k\sigma_2}$ than the uniform distribution (see Figure 3(b)), leading to a huge acceleration of rejection sampling.

The Binary Discrete Gaussian Distribution. Let us introduce the binary discrete Gaussian distribution $\mathcal{D}_{\sigma_2}$, which is a discrete Gaussian with specific variance $\sigma_2 = \sqrt{1/(2\ln 2)} \approx 0.849$ and probability density

$$\rho_{\sigma_2}(x) = e^{-x^2/(2\sigma_2^2)} = 2^{-x^2} \quad \text{for } x \in \mathbb{Z}\,.$$

We will combine $\mathcal{D}_{\sigma_2}$ with the uniform distribution to produce the distribution D_{k,σ_2} (see Figure 3(b)). We will only focus on the positive half of $\mathcal{D}_{\sigma_2}$ denoted $\mathcal{D}^+_{\sigma_2} = \{x \leftarrow \mathcal{D}_{\sigma_2} : x \geqslant 0\}$. Algorithm 10 is designed to sample according to $D^+_{\sigma_2}$ very efficiently using only unbiased random bits.

Lemma 6.5. *Algorithm 10 outputs positive integers according to $D^+_{\sigma_2}$. On average, the algorithm terminates after $2/\rho_{\sigma_2}(\mathbb{Z}^+) < 1.3$ trials, consuming 2.6 bits of entropy overall.*

Proof. We denote $\rho_{\sigma_2}(I) = \sum_{i \in I} 2^{-i^2}$ for $I \subseteq \mathbb{Z}^+$, the probability that the algorithm returns $x \in \mathbb{Z}^+$ is $\rho_{\sigma_2}(x)/\rho_{\sigma_2}(\mathbb{Z}^+)$ where $\rho_{\sigma_2}(\mathbb{Z}^+) = \sum_{i=0}^{\infty} 2^{-i^2} \approx 1.564$. We now observe that the binary expansion of $\rho_{\sigma_2}(\{0, \ldots, j\})$ is of the form

$$\rho_{\sigma_2}(\{0, \ldots, j\}) = \sum_{i=0}^{j} 2^{-i^2} = 1.1001\underbrace{0\ldots0}_{4}1\underbrace{0\ldots0}_{6}1 \quad \ldots \quad \underbrace{0\ldots0}_{2(j-2)}1\underbrace{0\ldots0}_{2(j-1)}1\,.$$

Thus, each trial of Algorithm 10 implicitly chooses a random real $r \in [0, 2)$ that will be rejected if $r > \rho_{\sigma_2}(\mathbb{Z}^+)$. It then computes the cumulative table (scaled by $\rho_{\sigma_2}(\mathbb{Z}^+)$) on the fly and reject if necessary. On average, the algorithm completes after $2/\rho_{\sigma_2}(\mathbb{Z}^+) < 1.3$ trials, consuming 2.6 bits of entropy. □

Building the Centered Discrete Gaussian Distribution. Based on our efficient sampling for the distribution $\mathcal{D}^+_{\sigma_2}$, we can now easily build the positive discrete Gaussian distribution with standard deviation $\sigma = k\sigma_2$ for $k \in \mathbb{Z}^+$. We refer to our Algorithm 11 based on the property

$$D^+_{k,\sigma_2} = k \cdot \mathcal{D}^+_{\sigma_2} + \mathcal{U}(0, k-1),$$

and where we reject the result with probability $\exp(-y(y+2kx)/(2\sigma^2))$ where x and y respectively follow the distributions $D^+_{\sigma_2}$ and $\mathcal{U}(0, k-1)$.

Theorem 6.6. *For any integer input k, Algorithm 11 outputs positive integers according to D^+_σ for $\sigma = k\sigma_2$. On average, it requires less than 1.47 trials. Consequently, Algorithm 12 output integers according to D_σ, and requires about $1 + \frac{1}{5\sigma}$ trials.*

Remark 6.7. Entropy consumption for each trials is: 2.6 bits for $x \leftarrow D^+_{\sigma_2}$, $\log_2 k$ bits for $y \leftarrow \mathcal{U}(\{0 \ldots k-1\})$, and $\approx 1 + \log_2 \sigma$ for rejection bit $b \leftarrow \mathcal{B}_{\exp(-y(y+2kx)/(2\sigma^2))}$ (measured in practice for this particular distribution), for a total of $\approx 4 + 2\log_2 \sigma$.

Proof. Let us start with the fact that any output z is uniquely written as $kx+y$ for $y \in \{0, \ldots, k-1\}$. The input (resp. desired output) distribution weight function g (resp f) is

$$g(z) = g(kx+y) = \frac{\rho_{\sigma_2}(x)}{k\rho_{\sigma_2}(\mathbb{Z}^+)} \quad \text{and} \quad f(z) = f(kx+y) = \frac{\rho_{k\sigma_2}(kx+y)}{\rho_{k\sigma_2}(\mathbb{Z}^+)}.$$

Since we restrict the distribution to non-negative integers, we have $\exp\left(-\frac{y(y+2kx)}{2\sigma^2}\right) \leqslant 1$ since x and y are both positive. Therefore, the probability to output some integer $z = kx+y$ is proportional to

$$\rho_{\sigma_2}(x)\exp\left(-\frac{y(y+2kx)}{2\sigma^2}\right) = \exp\left(-\frac{x^2}{2\sigma_2^2} - \frac{2kxy+y^2}{2\sigma^2}\right) = \exp\left(-\frac{(kx+y)^2}{2\sigma^2}\right) = \rho_{k\sigma_2}(z)\,.$$

The repetition rate M is upper-bounded by

$$M = \max\frac{f}{g} \leqslant \frac{k\rho_{\sigma_2}(\mathbb{Z}^+)}{\rho_{k\sigma_2}(\mathbb{Z}^+)} \leqslant \frac{k\rho_{\sigma_2}(\mathbb{Z}^+)}{k\sigma_2\sqrt{\pi/2}} \leqslant 1.47\,,$$

where the second inequality follows from the sum-integral comparison ($\rho_{k\sigma_2}$ is decreasing over $[0, \infty)$)

$$\rho_{k\sigma_2}(\mathbb{Z}^+) \geqslant \int_{x=0}^{\infty} \rho_{k\sigma_2}(x)dx = k\sigma_2\sqrt{\pi/2}.$$

Finally, we apply Algorithm 12 to build the (full) discrete Gaussian distribution $\mathcal{D}_\sigma$ over $\mathbb{Z}$.

Algorithm 10: Sampling $D^+_{\sigma_2}$

Output: An integer $x \in \mathbb{Z}^+$ according to $D^+_{\sigma_2}$
  generate a bit $b \leftarrow \mathcal{B}_{1/2}$
  if $b = 0$ **then** return 0
  for $i = 1$ **to** ∞ **do**
    draw random bits $b_1 \dots b_k$ for $k = 2i - 1$
    if $b_1 \dots b_{k-1} \neq 0 \dots 0$ **then** restart
    if $b_k = 0$ **then** return i
  end for

Algorithm 11: Sampling $D^+_{k\sigma_2}$ for $k \in \mathbb{Z}$

Input: An integer $k \in \mathbb{Z}$
Output: An integer $z \in \mathbb{Z}^+$ according to D^+_σ
  sample $x \in \mathbb{Z}$ according to $D^+_{\sigma_2}$
  sample $y \in \mathbb{Z}$ uniformly in $\{0, \dots, k-1\}$
  $z \leftarrow kx + y$
  sample $b \leftarrow \mathcal{B}_{\exp(-y(y+2kx)/(2\sigma^2))}$
  if $\neg b$ **then** restart
  return z

Algorithm 12: Sampling $D_{k\sigma_2}$ for $k \in \mathbb{Z}$

generate an integer $z \leftarrow D^+_{k\sigma_2}$
if $z = 0$ restart with probability $1/2$
generate a bit $b \leftarrow \mathcal{B}_{1/2}$ and return $(-1)^b z$

References

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th Annual ACM Symposium on Theory of Computing*, pages 99–108, Philadephia, Pennsylvania, USA, May 22–24, 1996. ACM Press.

[Ber08] Daniel J. Bernstein. Fast multiplication and its applications. In Joe Buhler and Peter Stevenhagen, editors, *Algorithmic number theory: lattices, number fields, curves and cryptography*, pages 325–384. Cambridge University Press, 2008.

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06: 13th Conference on Computer and Communications Security*, pages 390–399, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press.

[BP02] Mihir Bellare and Adriana Palacio. GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 162–177, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Berlin, Germany.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20, Seoul, South Korea, December 4–8, 2011. Springer, Berlin, Germany.

[DL] Léo Ducas and Tancrède Lepoint. *A Proof-of-concept Implementation of BLISS*. Available under the CeCILL License at `http://bliss.di.ens.fr`.

[DN12a] Léo Ducas and Phong Q. Nguyen. Faster gaussian lattice sampling using lazy floating-point arithmetic. In Wang and Sako [WS12], pages 415–432.

[DN12b] Léo Ducas and Phong Q. Nguyen. Learning a zonotope and more: Cryptanalysis of ntrusign countermeasures. In Wang and Sako [WS12], pages 433–450.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662, Santa Barbara, CA, USA, August 19–23, 2012. Springer, Berlin, Germany.

[ECR12] ECRYPT II. Ecrypt II yearly report on algorithms and keysizes (2011-2012). Available on `http://www.ecrypt.eu.org/`, 2012.

[FS96] Jean-Bernard Fischer and Jacques Stern. An efficient pseudo-random generator provably as secure as syndrome decoding. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 245–255, Saragossa, Spain, May 12–16, 1996. Springer, Berlin, Germany.

[GD12] Steven D. Galbraith and Nagarjun C. Dwarakanath. Efficient sampling from discrete gaussians for lattice-based cryptography on a constrained device. Available on `http://www.math.auckland.ac.nz/~sgal018/pubs.html`, 2012.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, Maryland, USA, May 31 – June 2, 2009. ACM Press.

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 112–131, Santa Barbara, CA, USA, August 17–21, 1997. Springer, Berlin, Germany.

[GGH13] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

[GL89] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *21st Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, Washington, USA, May 15–17, 1989. ACM Press.

[GLP12] Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 530–547, Leuven, Belgium, September 9–12, 2012. Springer, Berlin, Germany.

[GN08] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 31–51, Istanbul, Turkey, April 13–17, 2008. Springer, Berlin, Germany.

[GNR10] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 257–278, French Riviera, May 30 – June 3, 2010. Springer, Berlin, Germany.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206, Victoria, British Columbia, Canada, May 17–20, 2008. ACM Press.

[GS02] Craig Gentry and Michael Szydlo. Cryptanalysis of the revised NTRU signature scheme. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 299–320, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Berlin, Germany.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, pages 545–554, 2013.

[HG07] Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In Alfred Menezes, editor, *Advances in Cryptology – CRYPTO 2007*, volume 4622 of *Lecture Notes in Computer Science*, pages 150–169, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Berlin, Germany.

[HHGPW09] Jeff Hoffstein, Nick Howgrave-Graham, Jill Pipher, and William Whyte. *The LLL Algorithm: Survey and Applications*, chapter Practical Lattice-Based Cryptography: NTRUEncrypt and NTRUSign. Information Security and Cryptography. Springer, 2009.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HNHGSW03] Jeffrey Hoffstein, Jill Pipher Nick Howgrave-Graham, Joseph H. Silverman, and William Whyte. NTRUSIGN: Digital signatures using the NTRU lattice. In Marc Joye, editor, *Topics in Cryptology*

– *CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 122–140, San Francisco, CA, USA, April 13–17, 2003. Springer, Berlin, Germany.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe Buhler, editor, *ANTS*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer, 1998.

[HPS01] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NSS: An NTRU lattice-based signature scheme. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 211–228, Innsbruck, Austria, May 6–10, 2001. Springer, Berlin, Germany.

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 447–464, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Berlin, Germany.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th Annual ACM Symposium on Theory of Computing*, pages 1219–1234, New York, NY, USA, May 19–22, 2012. ACM Press.

[LM06] Vadim Lyubashevsky and Daniele Micciancio. Generalized compact Knapsacks are collision resistant. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP 2006: 33rd International Colloquium on Automata, Languages and Programming, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 144–155, Venice, Italy, July 10–14, 2006. Springer, Berlin, Germany.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23, French Riviera, May 30 – June 3, 2010. Springer, Berlin, Germany.

[Lyu08] Vadim Lyubashevsky. Lattice-based identification schemes secure under active attacks. In Ronald Cramer, editor, *PKC 2008: 11th International Conference on Theory and Practice of Public Key Cryptography*, volume 4939 of *Lecture Notes in Computer Science*, pages 162–179, Barcelona, Spain, March 9–12, 2008. Springer, Berlin, Germany.

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 598–616, Tokyo, Japan, December 6–10, 2009. Springer, Berlin, Germany.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 738–755, Cambridge, UK, April 15–19, 2012. Springer, Berlin, Germany.

[MM11] Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 465–484, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Berlin, Germany.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718, Cambridge, UK, April 15–19, 2012. Springer, Berlin, Germany.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007.

[MR09] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In *Post-quantum cryptography*, pages 147–191. Springer, Berlin, 2009.

[NIS11] NIST Special Publication 800-131A. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. Available on `http://csrc.nist.gov`, 2011.

[NR09] Phong Q. Nguyen and Oded Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. *Journal of Cryptology*, 22(2):139–160, April 2009.

[Pei10] Chris Peikert. An efficient and parallel gaussian sampler for lattices. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 80–97, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Berlin, Germany.

[PG12] Thomas Pöppelmann and Tim Güneysu. Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware. In Alejandro Hevia and Gregory Neven, editors, *Progress in Cryptology - LATINCRYPT 2012: 2nd International Conference on Cryptology and Information Security in*

Latin America, volume 7533 of *Lecture Notes in Computer Science*, pages 139–158, Santiago, Chile, October 7–10, 2012. Springer, Berlin, Germany.

[PR06] Chris Peikert and Alon Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In Shai Halevi and Tal Rabin, editors, *TCC 2006: 3rd Theory of Cryptography Conference*, volume 3876 of *Lecture Notes in Computer Science*, pages 145–166, New York, NY, USA, March 4–7, 2006. Springer, Berlin, Germany.

[Rüc10] Markus Rückert. Lattice-based blind signatures. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 413–430, Singapore, December 5–9, 2010. Springer, Berlin, Germany.

[SS11] Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 27–47, Tallinn, Estonia, May 15–19, 2011. Springer, Berlin, Germany.

[vN51] John von Neumann. Various techniques used in connection with random digits. *J. Research Nat. Bur. Stand., Appl. Math. Series*, 12:36–38, 1951.

[WS12] Xiaoyun Wang and Kazue Sako, editors. *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *Lecture Notes in Computer Science*. Springer, 2012.

A Security Analysis

In this section, we describe how known attacks apply to our scheme. First, we describe in Appendix A.1 combinatorial attacks on the secret key, namely brute-force and meet-in-middle attacks.

Then we consider lattice reduction attacks. Typical measurements of lattice problem hardness (the so called Hermite factor, see [CN11]) are given in Table 5 (page 24), measuring how hard it is to find vectors of a given norm in a random lattice. We first apply this measure to the hardness of the underlying SIS problem, as if the lattice used was truly random (*c.f.* Section A.2).

Yet, the lattice is not truly random, as by design it contains unusually short vectors. Therefore, one may try to directly recover the secret by lattice reduction: find the secret key $(\mathbf{f}, \mathbf{g})$ as a short vector in the primal lattice $\Lambda = \{(\mathbf{x}, \mathbf{y}) \in \mathcal{R}^2 : \mathbf{a}_q\mathbf{x} + \mathbf{y} = 0 \bmod q\}$. Unfortunately, the only study [GN08] of the behavior of lattice algorithms in the presence of unusually short vectors only considers the unique-SVP problem, in which there is only one unusually short vector. In the NTRU-like case, there is a basis of n of them. We provide new experiments showing that the behavior is similar; that it is dictated by the ratio between the actual shortest vector, its expected length in a random lattice and the Hermite factor (*c.f.* Section A.3).

An alternative attack to recover unusually short vectors of a lattice is to use short (but quite larger) dual lattice $\Lambda^\times$ vectors to detect its presence, and then recover it [MR09,MM11] using search-to-decision reduction; quantification of this attack is detailed in Section A.4.

Finally, it is possible to combine lattice reduction and combinatorial techniques: Howgrave-Graham designed in [HG07] an attack against NTRU keys combining a meet-in-the-middle strategy with lattice reduction. This attack applies to our scheme, as detailed in Section A.5, but also on the previous related schemes [Lyu12,GLP12]. Notice that there is no mention of this attack in the security analysis of the latter schemes; therefore in order to compare, we also include security measurements for those schemes.

We base our security projection on the BKZ 2.0 simulation methodology [CN11] that models the behavior of BKZ including the latest improvements [GNR10,HPS11].

Note that we only sketch the attack principles; we refer the interested reader to the original articles [HNHGSW03,HG07,CN11,MR09,MM11] for more detail. We emphasize that the statistical attacks [NR09,DN12b] provably (*i.e.* information-theoretically) do not apply here because of

rejection sampling: the output distribution of the signature scheme is independent of the secret key.

A.1 Brute-force and Meet-in-the-Middle Key Recovery Attack

The key-recovery problem is as follows: given $\mathbf{a} \in \mathbb{Z}_q[x]/(x^n+1)$, find small polynomials $\mathbf{f}, \mathbf{g}$ such that $\mathbf{a}(2\mathbf{g}+1) - \mathbf{f} = \mathbf{0}$ (knowing that such a solution exists). Precisely, we know that both $\mathbf{f}$ and $\mathbf{g}$ have respectively $d_1 = \lceil n\delta_1 \rceil$ entries in $\{-1, +1\}$ and $d_2 = \lceil n\delta_2 \rceil$ entries in $\{-2, +2\}$.

Brute-force Key Recovery. The brute-force attack simply consists in picking a random vector $\mathbf{g}$ according to the key-generation distribution, and checking whether $\mathbf{f} = \mathbf{a}(2\mathbf{g}+1)$ is a polynomial with ternary coefficients. To measure the complexity of this attack, one simply measures the entropy of $\mathbf{g}$: this entropy yields a lower bound on the time to exhaust all possible values. The time complexity of this attack is therefore $T = 2^{d_1+d_2}\binom{n}{d_1}\binom{n-d_1}{d_2}$.

For more complex attacks, it may be simpler to model all the entries of the secret key as independent random variables, each of them having entropy:

$$e = \delta_0 \log_2 \delta_0 + \delta_1 \log_2 \frac{\delta_1}{2} + \delta_2 \log_2 \frac{\delta_2}{2}.$$

In this model, the total entropy is $n \cdot e$, which is at most $\log n$ greater than the true entropy.

Meet-in-the-Middle Attack. Odlyzko proposed a MiM attack of running time the square root of the latter attack (but with additional memory consumption). It was designed against the NTRU signature scheme, but it also applies here. We refer to [HNHGSW03] for details, and give only a short explanation of a simplified version: exhaust $\mathbf{g}_1$ as the first half bits of $\mathbf{g}$ and store $\mathbf{g}_1$ in several labeled boxes (of an hash table) according to the values of $\mathbf{f}_1 = \mathbf{a}(2\mathbf{g}_1+1)$. Then search for the second half $\mathbf{g}_2$ of $\mathbf{g}$ by computing $\mathbf{f}_2 = \mathbf{a}(2\mathbf{g}_2) \bmod q$: the labeling is designed so that to ensure a collision whenever $\mathbf{f}_1 + \mathbf{f}_2$ is ternary.

This attacks runs in time and memory about $2^{n \cdot e/2}$, since the entropy of a half of the vector is $n \cdot e/2$.

A.2 Hardness of the underlying SIS problem

Attack Overview. In this section we measure the hardness of forging a signature according to our security proof. We will consider the running time necessary for the BKZ algorithm to find a vector of norm $\beta = 2B_2 + (2^d+1)\sqrt{n}$ in a random q-ary lattice according to the latest analysis in [CN11]. While the lattice Λ is not perfectly random because of the presence of unusually short vectors, the next section analyzes how hard it is to detect and find those unusually short vectors.

Remark A.1. Note that we have $\beta > q$, yet the q-vectors give no proper solution to the SIS instance since it is required that the short solution is non-null modulo q. This is one of the reasons our scheme constraints the ℓ_2 and the ℓ_∞ norms of signature vectors; this ensures that the reduction provides a vector $\mathbf{v}$ such that $\|\mathbf{v}\|_\infty < q/2$, and thus is non-null modulo q. While we could have chosen larger values for B_∞ and still have a valid security reduction, choosing it as small as possible for correctness can only make the scheme more secure.

Table 4. Hardness of the underlying SIS instance

Scheme	**BLISS-0**	**BLISS-I**	**BLISS-II**	**BLISS-III**	**BLISS-IV**
SIS parameter $\beta/\sqrt{q}$ (as in Theorem 4.4)	$63 = 1.0083^m$	$441 = 1.0060^m$	$409 = 1.0059^m$	$289 = 1.0055^m$	$231 = 1.0053^m$
Required Block Size	125	215	220	245	260
Enum. Cost $\log_2 T$	**53**	**130**	**136**	**168**	**188**

Quantification. The hardness of this SIS problem is dictated by the ratio $\beta/\sqrt{q}$ and the dimension m, precisely it is necessary to run BKZ with a blocksize providing a Hermite factor $\delta^m < \beta/\sqrt{q}$. The relation between the block-size δ and the running time is interpolated from [CN11].

Margins. The cost given in the last line of Table 4 is expressed as the number of nodes to visit in the enumeration tree of the enumeration subroutine of BKZ. Each visit requires about 100 CPU cycles, and BKZ needs to perform at least $2n$ such enumerations, adding an additional 10 bits to those numbers. Yet, those numbers do not directly give rise to an attack as they are derived from a security reduction; actually forging seems to require finding vectors smaller by a factor 2.

A.3 Primal Lattice Reduction Key Recovery

Attack Overview. The attack consists in applying lattice reduction to the primal lattice Λ hoping that the short vector found will be the secret key. This problem can be seen as a ring variant of the unique-SVP problem. Recall that we ran experiments (see Section 1.2) suggesting that BKZ behaves similarly in the presence of either only one unusually short vector or a basis of n of them, even for larger block sizes; therefore we measure hardness according to the BKZ 2.0 methodology [CN11].

Table 5. Cost of finding the Ring-unique shortest vector via primal lattice reduction

Scheme	**BLISS-0**	**BLISS-I**	**BLISS-II**	**BLISS-III**	**BLISS-IV**
Ring-Unique-SVP parameter $\sqrt{\frac{qm}{2\pi e}}\Big/\lambda_1$	$14 = 1.0051^m$	$46 = 1.0037^m$	$46 = 1.0037^m$	$30 = 1.0033^m$	$25 = 1.0031^m$
Required Block Size	270	> 300	> 300	> 300	> 300
Enum. Cost $\log_2 T$	**200**	$>$ **240**	$>$ **240**	$>$ **240**	$>$ **240**

A.4 Dual Lattice Reduction Key Recovery

Attack Overview. The attack consists in using short dual lattice vectors as a distinguisher for the existence of a very short vector $\mathbf{s}$ in a lattice [MR09]. Then, one may use the distinguisher to completely recover this very short vector using the reduction of Micciancio and Mol [MM11], inspired by the Goldreich-Levin Theorem [GL89].

Quantification. For a q-ary lattice Λ of dimension m, using a vector $\mathbf{v} \in \Lambda^\times$ (where $\Lambda^\times$ is the dual lattice) and assuming its direction is random, one is able to distinguish the existence of an unusual short vector $\mathbf{s}$ in the dual with probability $\epsilon = e^{-\pi\tau^2}$, where $\tau = \|\mathbf{v}\| \cdot \|\mathbf{s}\| / (q\sqrt{m})$.

Next, using this distinguisher as an oracle, it is possible to recover one entry of the private key except with small fixed probability, using $1/\epsilon^2$ calls to that oracle. We then iterated over different block-sizes (5 by 5) to minimize the total cost T/ϵ^2, where T is the running time of the enumeration subroutine of BKZ.

Remark A.2. Rather than trying to find the proper secret key $\mathbf{s} = (\mathbf{f}| - 2\mathbf{g} + 1)$ as a short solution to $(2\mathbf{a}_q, 2)^t\mathbf{s} = \mathbf{0} \bmod q$, one would search directly $\mathbf{s}' = (\mathbf{f}|\mathbf{g})$ as a shorter solution to $(\mathbf{a}_q, -1)^t\mathbf{s}' = 1 \bmod q$.

Table 6. Cost of distinguish the existence of the shortest vector via primal lattice reduction

Scheme	**BLISS-0**	**BLISS-I**	**BLISS-II**	**BLISS-III**	**BLISS-IV**	[Lyu12, **Set-IV**]	[GLP12, **Set-I**]
Best Block Size b	110	220	220	240	245	190	130
Enum. Cost: $\log_2 T$	45	136	136	162	168	103	56
Hermite Factor: δ	1.0088^m	1.0059^m	1.0059^m	1.0056^m	1.0056^m	1.0067^m	1.0081^m
Dist. Advantage: $\log_2 \epsilon$	-5.5	-20	-20	-19	-21	-7	-5
Total Cost: $\log_2(T/\epsilon^2)$	**56**	**177**	**177**	**201**	**211**	**118**	**67**

Margins. To stay on the safe side, we do not include the additional n^2 factor to the running time of this attack: indeed there are n coordinates to guess, and each BKZ reduction requires at least n enumerations; one might then be tempted to claim an additional 20 bits of security. Yet it is unclear whether one needs to run the full BKZ reduction to get new short vectors, neither if one can reuse the same short dual vector to guess each coordinate. Even though we do not claim an attack in time 2^{67} on [GLP12], we believe that claiming more than 90 bits of security is a long shot. The difference between our measurement and theirs might be explained by the fact that the authors only considered the case where ϵ was close to 1.

A.5 Hybrid MiM-Lattice Key Recovery

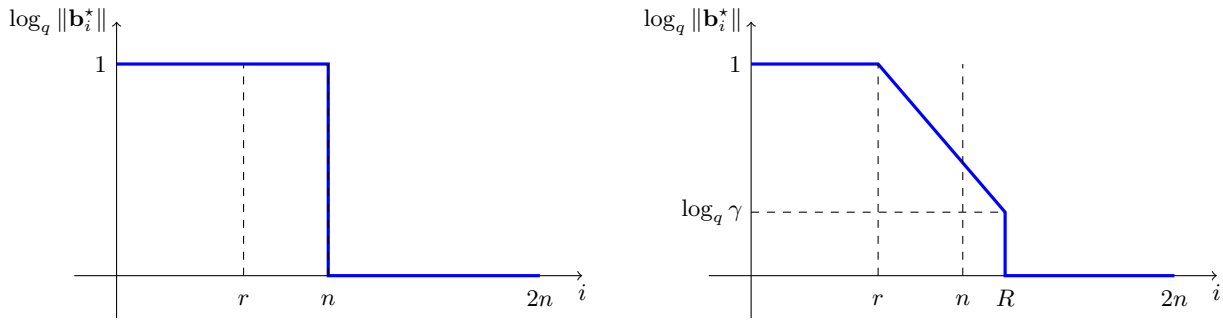
Attack Overview. The attack from [HG07] uses lattice reduction as a preprocessing step, in order to decrease the search space of combinatorial attacks. Precisely, one first chooses parameters r and R, and applies lattice reduction on the sub-lattice generated by the vectors of the sub-basis $\mathbf{b}_r, \ldots, \mathbf{b}_{R-1}$ (see Figure 5), in order to run the MiM attack only over the $2n - R$ last coordinates.

In order to perform the combinatorial attack, one needs to obtain a basis whose last orthogonalized vector is large enough. Precisely, the basis needs to be good enough so that Babai's algorithm properly solves BDD on the error $\mathbf{s}' = (s_1, \ldots, s_R, 0, \ldots, 0)$. A necessary condition is therefore:

$$\left\langle \mathbf{s}', \mathbf{b}_i^\star \right\rangle / \left\| \mathbf{b}_i^\star \right\|^2 \leqslant 1/2 \,, \tag{6}$$

where the $\mathbf{b}_1^*, \ldots, \mathbf{b}_R^*$ is the Gram-Schmidt orthogonalization of $\mathbf{b}_1, \ldots, \mathbf{b}_R$.

Quantification. Once again, we assume that the lattice reduction algorithm provides a basis of random direction. Therefore, we model the quantity $\langle \mathbf{s}', \mathbf{b}_i^\star \rangle / \left\| \mathbf{b}_i^\star \right\|^2$ as a Gaussian of standard deviation $\left\| \mathbf{s}' \right\| / (\sqrt{R} \left\| \mathbf{b}_i^\star \right\|)$. Denoting $\gamma = \left\| \mathbf{b}_{R-1}^\star \right\|$, one models by the GSA (geometric series assumption) that $\left\| \mathbf{b}_{R-1-i}^\star \right\| = \gamma \times \delta^{2i}$, where $\delta \leqslant 1.007$ is the Hermite factor. To verify Equation (6) with reasonable probability (say at least 0.01), it is required that $\gamma \geqslant 2.5 \left\| \mathbf{s}' \right\| / \sqrt{R}$.



(a) Before reduction (b) After reduction

Fig. 5. Basis Profile during the Hybrid Attack

We thus determine the security against this attack as follows: to claim λ bits of security, set R so that it takes 2^λ time and memory to exhaust the last $2n - R$ entries of the secret. Recall that e denotes the entropy of a single entry, each step of the Meet in the Middle attack requires $O(n^2)$ operations, and at least $e \cdot R$ bits of storage, therefore we set R such that $R \cdot e = 2\lambda - \log_2(e \cdot R) - \log_2(n^2)$.

Then we determine γ, and run BKZ 2.0 simulation according to [CN11], increasing block-size until $\gamma \geqslant 2.5 \|\mathbf{s}'\| / \sqrt{R}$. Finally, deduce the cost of lattice reduction and verify it is greater than 2^λ. Note that r is derived from the behavior of this simulation. Analysis results are described in Table 7.

Table 7. Hybrid MiM+Lattice Reduction Attack Parameters

Scheme	**BLISS-0**	**BLISS-I**	**BLISS-II**	**BLISS-III**	**BLISS-IV**	[Lyu12, **IV**]	[GLP12, **I**]
MiM Search Cost $\log_2 M$	**60**	**128**	**128**	**160**	**192**	**100**	**80**
Entropy per Secret Key Entry	2.11	1.18	1.18	1.60	1.77	1.58	1.58
MiM Search Dimension R	46	194	194	183	201	110	85
Required Block Size	165	245	245	> 300	> 300	220	140
BKZ Enum. Cost $\log_2 T$	**84**	**168**	**168**	$>$ **200**	$>$ **200**	**150**	**60**

Margins. There is a small security margin coming from the fact that we set the parameters so that the attack succeeds with probability 0.01, which would add about 7 bits of security, and again 10 extra bits because BKZ requires at least $2n$ enumeration. More importantly we considered that the attacker has 2^λ memory available; in practice it is unlikely that an attacker may have as much memory available as the number of bit-operations.[13]

B Key Generation for a SIS-Based Scheme

In this section, we explain how to generate the key pair $(\mathbf{A}, \mathbf{S})$ so that

$$\mathbf{AS} = q\mathbf{I}_n \in \mathbb{Z}_{2q}^{n \times n} ,$$

[13] In 2007, there were no more than 2^{71} bits of storage globally, while all general-purpose computers could execute 2^{87} operations in a year. Storage growth is 23% a year versus 58% for computing power (see `http://news.usc.edu/\#!/article/29360/How-Much-Information-Is-There-in-the-World`). There are about 2^{160} atoms on earth.

where the distribution of $\mathbf{A}$ is statistically close to the uniform distribution, in order to obtain a general SIS-based variant of our scheme.

Leftover Hash Lemma. We first recall the classical Leftover Hash Lemma. A distribution D is ε-uniform if its statistical distance from the uniform distribution is at most ε, where the statistical distance $\Delta(D_1, D_2)$ between two distributions D_1, D_2 over a finite domain X is given by $\Delta(D_1, D_2) = \frac{1}{2}\sum_{x\in X} |D_1(x) - D_2(x)|$.

Let X and Y be finite sets. A family $\mathcal{H}$ of hash functions from X to Y is said to be *pairwise-independent* if for all distinct $x, x' \in X$, $\Pr_{h\leftarrow H}[h(x) = h(x')] = 1/|Y|$.

Lemma B.1 (Leftover Hash Lemma [HILL99]). *Let $\mathcal{H}$ be a family of pairwise hash functions from X to Y. Suppose that $h \leftarrow \mathcal{H}$ and $x \leftarrow X$ are chosen uniformly and independently. Then, $(h, h(x))$ is $\frac{1}{2}\sqrt{|Y|/|X|}$-uniform over $\mathcal{H} \times Y$.*

From the LHL, one can deduce the following lemma for finite linear combinations modulo the prime q:

Lemma B.2. *Let $m \geqslant 2$. Set $x_1, \ldots, x_m \leftarrow \mathbb{Z}_q^n$ uniformly and independently, set $s_1, \ldots, s_m \leftarrow (-2^\alpha, 2^\alpha) \cap \mathbb{Z}$, and set $y = \sum_{i=1}^m s_i \cdot x_i \bmod q$. Then $(x_1, \ldots, x_m, y)$ is $1/2\sqrt{q^n/2^{(\alpha+1)\cdot m}}$-uniform over $\mathbb{Z}_q^{n\cdot(m+1)}$.*

Proof. Let us consider the hash function family $\mathcal{H}$ from $(-2^\alpha, 2^\alpha)^m$ to $\mathbb{Z}_q$ in which each member $h \in \mathcal{H}$ is parameterized by the element $(x_1, \ldots, x_m) \in \mathbb{Z}_q^m$. Given $\mathbf{s} \in (-2^\alpha, 2^\alpha)^m$, we define $h(\mathbf{s}) = \sum_{i=1}^m s_i \cdot x_i \in \mathbb{Z}_q$. The hash function family is clearly pairwise independent since q is prime. Therefore by Lemma B.1, $(h, h(x))$ is $1/2\sqrt{q^n/2^{(\alpha+1)\cdot m}}$-uniform over $\mathbb{Z}_q^{m+1}$. □

SIS-based Scheme. Define $m' = m+n$. Choose a uniform matrix $\mathbf{A'}_q \in \mathbb{Z}_q^{n\times m}$ and a random small $\mathbf{S'} \in \mathbb{Z}_q^{m\times n}$ with coefficients in $(-2^\alpha, 2^\alpha)$. Define $\mathbf{A}_q = (\mathbf{A'}_q | -\mathbf{A'}_q\mathbf{S'}) \in \mathbb{Z}_q^{n\times m'}$. By Lemma B.2, the statistical distance between the distribution of $\mathbf{A}_q$ and the uniform distribution over $\mathbb{Z}_q^{n\times m'}$ is at most $n \cdot 1/2\sqrt{q^n/2^{(\alpha+1)\cdot m}}$. Thus, for this statistical distance to be negligible in the security parameter λ, we need

$$m \geqslant \frac{2(\lambda - 1 + \lceil \log_2(n) \rceil) + n\lceil \log_2(q) \rceil}{\alpha + 1} .$$

Set the secret key as $\mathbf{S} = \begin{pmatrix} \mathbf{S'} \\ \mathbf{I}_n \end{pmatrix} \in \mathbb{Z}_{2q}^{m'\times n}$. One observes that $\mathbf{A}_q\mathbf{S} = \mathbf{0} \bmod q$. It remains to set the public key as $\mathbf{A} = (2\mathbf{A'}_q | q\mathbf{I}_n - 2\mathbf{A'}_q\mathbf{S'}) \in \mathbb{Z}_{2q}^{n\times m'}$. Then one easily checks that $\mathbf{AS} = q\mathbf{I}_n$. Also, we have that $\mathbf{A} \bmod q = 2\mathbf{A}_q$ is *uniform* modulo q. Notice that this construction is easily adaptable to the ring settings.

C Security Proof with Dropped Bits

Recall from Section 4.7 that a signature is a tuple $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$ with

$$\mathbf{z}_2^\dagger = \lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{y}_1 + \mathbf{y}_2 \rceil_d - \lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} \rceil_d \bmod p ,$$

where $p = \lfloor 2q/2^d \rfloor$ and that the random oracle is called on

$$\left(\lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{y}_1 + \mathbf{y}_2 \rceil_d \bmod p, \mu\right) = \left(\mathbf{z}_2^\dagger + \lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} \rceil_d \bmod p, \mu\right) .$$

We recall the theorem stating that our scheme is secure when dropping bits.

Theorem C.1 (Restatement of Theorem 4.4). *Let us consider the signature scheme of Section 4.8. Assume that $d \geqslant 3$, $q \equiv 1 \bmod 2^{d-1}$, and $2B_\infty + (2^d + 1) < q/2$. Suppose there is a polynomial-time algorithm $\mathcal{F}$ which succeeds in forging with non negligible probability. Then there exists a polynomial-time algorithm which can solve the $\mathcal{R}$-SIS$^{\mathcal{K}}_{q,n,m,\beta}$ problem for $\beta = 2B_2 + (2^d+1)\sqrt{n}$.*

The proof of this theorem follows the same blueprint than the proof of Theorem 3.3. Namely, by a straightforward adaptation of Lemma 3.4, one can show that our signing algorithm can be replaced by Hybrid 3 (Algorithm 13). Next, an adaptation of Lemma 3.5 states that if an algorithm can produce a forgery with non-negligible probability when the signing algorithm is replaced by Hybrid 3, then we can use it to recover a vector $\mathbf{v} \neq \mathbf{0} \bmod q$ such that $\|\mathbf{v}\| \leqslant \beta = 2B_2 + (2^d+1)\sqrt{n}$ and $\mathbf{A}\mathbf{v} = \mathbf{0} \bmod q$.

Algorithm 13: Hybrid 3

1: $\mathbf{c} \leftarrow \mathbb{B}_\kappa^n$
2: $\mathbf{z}_1, \mathbf{z}_2 \leftarrow \mathcal{D}_\sigma^n$
3: With probability $\frac{1}{M}$:
4: $\quad \mathbf{z}_2^\dagger \leftarrow (\lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} + \mathbf{z}_2 \rceil_d - \lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} \rceil_d) \bmod p$
5: $\quad$ output $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$
6: $\quad$ program $H(\lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c} + \mathbf{z}_2 \rceil_d \bmod p, \mu) = \mathbf{c}$

Throughout the rest of the section, we focus on the modifications in the proof of Lemma 3.5 to deal with the dropping bits, *i.e.* we assume that $\mathcal{F}$ succeeds in forging the signature by outputting $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$ where $\mathbf{c} = \mathbf{c}_j \in \{\mathbf{c}_1, \ldots, \mathbf{c}_t\}$ was obtained from either a previous signing query, or a previous random oracle query.

C.1 Preliminaries

We have the following facts:

Fact C.2. *Let q be an odd integer and define $\zeta \in [0, 2q-1]$ such that $\zeta \cdot (q-2) = 1 \bmod 2q$. Then $\zeta = \frac{q-1}{2}$ if $(q-1)/2$ is odd or $\zeta = \frac{q-1}{2} + q$ if $(q-1)/2$ is even.*

Proof. We have that

$$\frac{q-1}{2} \cdot (q-2) = q \cdot \frac{q-1}{2} - q + 1 = 1 \bmod q .$$

Therefore $\zeta = \frac{q-1}{2} \bmod q$ and the fact holds according to the parity of $(q-1)/2$. □

Fact C.3. *Let $d \geqslant 2$, q be an integer such that $q \equiv 1 \bmod 2^{d-1}$, and let $p = \lfloor 2q/2^d \rfloor$. Then $p \cdot 2^d = 2q - 2$.*

C.2 Proof

Assume the challenger has a signature $(\mathbf{z'}_1, \mathbf{z'}_2^\dagger, \mathbf{c'}_j)$ such that

$$\lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c}_j \rceil_d + \mathbf{z}_2^\dagger \bmod p = \lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z'}_1 + \zeta \cdot q \cdot \mathbf{c'}_j \rceil_d + \mathbf{z'}_2^\dagger \bmod p \ .$$

There exists $\mathbf{k} \in \{0, \pm 1\}^n$ such that the following equation holds over $\mathbb{Z}$:

$$\lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c}_j \rceil_d - \lfloor \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z'}_1 + \zeta \cdot q \cdot \mathbf{c'}_j \rceil_d + \mathbf{z}_2^\dagger - \mathbf{z'}_2^\dagger = \mathbf{k}p \ .$$

Now we multiply the previous equation by 2^d, and this yields modulo $2q$:

$$\zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c}_j - \mathbf{e} - \zeta \cdot \mathbf{a}_1 \cdot \mathbf{z'}_1 - \zeta \cdot q \cdot \mathbf{c'}_j + \mathbf{e'} + 2^d(\mathbf{z}_2^\dagger - \mathbf{z'}_2^\dagger) = \mathbf{k} \cdot p2^d \bmod 2q \ ,$$

where $\mathbf{e} = [\zeta \cdot \mathbf{a}_1 \cdot \mathbf{z}_1 + \zeta \cdot q \cdot \mathbf{c}_j \bmod 2q] \bmod 2^d$ and $\mathbf{e'} = [\zeta \cdot \mathbf{a}_1 \cdot \mathbf{z'}_1 + \zeta \cdot q \cdot \mathbf{c'}_j \bmod 2q] \bmod 2^d$. This yields by Fact C.3:

$$(\zeta \cdot \mathbf{a}_1) \cdot (\mathbf{z}_1 - \mathbf{z'}_1) + 2^d(\mathbf{z}_2^\dagger - \mathbf{z'}_2^\dagger) + \zeta \cdot q \cdot (\mathbf{c}_j - \mathbf{c'}_j) + (\mathbf{e'} - \mathbf{e}) + 2\mathbf{k} = \mathbf{0} \bmod 2q \ . \tag{7}$$

Thus, if we define

$$\mathbf{v} = \left(\mathbf{z}_1 - \mathbf{z'}_1, 2^d(\mathbf{z}_2^\dagger - \mathbf{z'}_2^\dagger) + (\mathbf{e'} - \mathbf{e}) + 2\mathbf{k}\right)^t \in \mathcal{R}^{2\times 1} \ ,$$

we have that

$$(\zeta \cdot \mathbf{a}_1, 1) \cdot \mathbf{v} = \mathbf{0} \bmod q \ ,$$

and thus multiplying by 2:

$$(\mathbf{a}_1, 2) \cdot \mathbf{v} = \mathbf{0} \bmod q \ .$$

Now, we have that $\|\mathbf{v}\|_2 \leqslant 2B_2 + (2^d + 1) \cdot \sqrt{n}$ and $\|\mathbf{v}\|_\infty \leqslant 2B_\infty + (2^d + 1) < q/2$. Indeed

$$\begin{aligned}
\|\mathbf{v}\|_2 &\leqslant \left\| \left(\mathbf{z}_1 - \mathbf{z'}_1, 2^d(\mathbf{z}_2^\dagger - \mathbf{z'}_2^\dagger)\right) \right\|_2 + \left\| \left(\mathbf{0}, (\mathbf{e'} - \mathbf{e} + 2\mathbf{k})\right) \right\|_2 \\
&\leqslant 2B_2 + \left\| \left(\mathbf{0}, (\mathbf{e'} - \mathbf{e} + 2\mathbf{k})\right) \right\|_\infty \cdot \sqrt{n} \\
&\leqslant 2B_2 + (\left\| \left(\mathbf{0}, (\mathbf{e'} - \mathbf{e})\right) \right\|_\infty + 2\,\|\mathbf{k}\|_\infty) \cdot \sqrt{n} \\
&\leqslant 2B_2 + (2^d - 1 + 2) \cdot \sqrt{n} \\
&\leqslant 2B_2 + (2^d + 1) \cdot \sqrt{n} \ .
\end{aligned}$$

Similarly for the infinite norm, we get

$$\|\mathbf{v}\|_\infty \leqslant 2B_\infty + (2^d + 1) < q/2 \ .$$

It remains to show that $\mathbf{v} \neq \mathbf{0} \bmod q$ to conclude. By the condition $\|\mathbf{v}\|_\infty < q/2$, it suffices to show that $\mathbf{v} \neq \mathbf{0} \bmod 2q$.

Case #1: $[\mathbf{z}_1 \neq \mathbf{z'}_1 \bmod 2q]$. Since

$$\mathbf{v} = \left(\mathbf{z}_1 - \mathbf{z'}_1, 2^d(\mathbf{z}_2^\dagger - \mathbf{z'}_2^\dagger) + (\mathbf{e'} - \mathbf{e}) + 2\mathbf{k}\right)^t \ ,$$

we have $\mathbf{v} \neq \mathbf{0} \bmod 2q$. This case includes both type-1 and type-2 forgeries.

Case #2: [$\mathbf{z}_1 = \mathbf{z}'_1 \bmod 2q$ and $\mathbf{c}_j = \mathbf{c'}_j$]. In that case, we have $\mathbf{e} = \mathbf{e}'$, and for the signatures to be different we have $\mathbf{z}_2^\dagger \neq \mathbf{z'}_2^\dagger$. Therefore

$$\mathbf{v} = \left(\mathbf{0}, 2^d(\mathbf{z}_2^\dagger - \mathbf{z'}_2^\dagger) + 2\mathbf{k}\right)^t .$$

Now $\|2\mathbf{k}\|_\infty < 2^d$, then $\mathbf{v} \neq \mathbf{0} \bmod 2q$. This case is only possible for type-1 forgeries.

Case #3: [$\mathbf{z}_1 = \mathbf{z}'_1 \bmod 2q$, $\mathbf{c}_j \neq \mathbf{c'}_j$ and $\mathbf{z}_2^\dagger = \mathbf{z'}_2^\dagger \bmod 2q$]. In that case, Equation (7) yields

$$\mathbf{e}' - \mathbf{e} + 2\mathbf{k} = \zeta \cdot q \cdot (\mathbf{c}_j - \mathbf{c'}_j) \bmod 2q .$$

Now $\mathbf{c}_j - \mathbf{c'}_j \neq \mathbf{0} \bmod 2$, therefore $\mathbf{e}' - \mathbf{e} + 2\mathbf{k} \neq \mathbf{0} \bmod 2q$. Since

$$\mathbf{v} = \left(\mathbf{0}, (\mathbf{e}' - \mathbf{e}) + 2\mathbf{k}\right)^t ,$$

we have $\mathbf{v} \neq \mathbf{0} \bmod 2q$. This case is only possible for type-2 forgeries.

Case #4: [$\mathbf{z}_1 = \mathbf{z}'_1 \bmod 2q$, $\mathbf{c}_j \neq \mathbf{c'}_j$ and $\mathbf{z}_2^\dagger \neq \mathbf{z'}_2^\dagger \bmod 2q$]. In that case

$$\mathbf{v} = \left(\mathbf{0}, 2^d(\mathbf{z}_2^\dagger - \mathbf{z'}_2^\dagger) + (\mathbf{e}' - \mathbf{e}) + 2\mathbf{k}\right)^t .$$

Since $\mathbf{c}_j \neq \mathbf{c'}_j$, there exists i such that $\mathbf{c}_j[i] \neq \mathbf{c'}_j[i]$. Without loss of generality, we can assume that $\mathbf{c'}_j[i] = 1$ and thus $\mathbf{c}_j[i] = 0$. Therefore,

$$\mathbf{e}'[i] = \left(x + \zeta \cdot q \bmod 2q\right) \bmod 2^d ,$$

and

$$\mathbf{e}[i] = x \bmod 2^d ,$$

where $x = \left(\zeta \cdot (\mathbf{a}_1 \cdot \mathbf{z}_1)[i]\right) \bmod 2q$. Now $\zeta \cdot q = q \bmod 2q$ because $\zeta = 1 \bmod 2$ by Fact C.2. Therefore

$$\mathbf{e}'[i] = \left(x + q \bmod 2q\right) \bmod 2^d .$$

Now, $\left(x + q \bmod 2q\right) = x \pm q$ over $\mathbb{Z}$. Therefore,

$$\left(\mathbf{e}'[i] - \mathbf{e}[i]\right) \bmod 2^d = \left((x \pm q) - x\right) \bmod 2^d$$

is odd. This proves that $\mathbf{v}[i]$ is odd, and therefore that $\mathbf{v} \neq \mathbf{0} \bmod 2q$. This case is only possible for type-2 forgeries.