High-Speed Signatures from Standard Lattices

Özgür Dagdelen[1], Rachid El Bansarkhani[1], Florian Göpfert[1], Tim Güneysu[2], Tobias Oder[2], Thomas Pöppelmann[2](✉), Ana Helena Sánchez[3], and Peter Schwabe[3]

[1] Technische Universität Darmstadt, Darmstadt, Germany
oezguer.dagdelen@cased.de,
{elbansarkhani,fgoepfert}@cdc.informatik.tu-darmstadt.de
[2] Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Bochum, Germany
thomas.poeppelmann@rub.de
[3] Digital Security Group, Radboud University Nijmegen, Nijmegen, The Netherlands
ahsanchez@cs.ru.nl, peter@cryptojedi.org

Abstract. At CT-RSA 2014 Bai and Galbraith proposed a lattice-based signature scheme optimized for short signatures and with a security reduction to hard standard lattice problems. In this work we first refine the security analysis of the original work and propose a new 128-bit secure parameter set chosen for software efficiency. Moreover, we increase the acceptance probability of the signing algorithm through an improved rejection condition on the secret keys. Our software implementation targeting Intel CPUs with AVX/AVX2 and ARM CPUs with NEON vector instructions shows that even though we do not rely on ideal lattices, we are able to achieve high performance. For this we optimize the matrix-vector operations and several other aspects of the scheme and finally compare our work with the state of the art.

Keywords: Signature scheme · Standard lattices · Vectorization · Ivy bridge

1 Introduction

Most practical lattice-based signatures [7,16,21], proposed as post-quantum [9] alternatives to RSA and ECDSA, are currently instantiated and implemented using structured ideal lattices [30] corresponding to ideals in rings of the form

P. Schwabe—This work was supported by the German Research Foundation (DFG) through the DFG Research Training Group GRK 1817/1, by the German Federal Ministry of Economics and Technology through Grant 01ME12025 SecMobil), by the Netherlands Organisation for Scientific Research (NWO) through Veni 2013 project 13114, and by the German Federal Ministry of Education and Research (BMBF) through EC-SPRIDE. Permanent ID of this document: c5e2da3f0d05a056a5490a5c9b88baa9. Date: 2014-09-04.

$\mathbb{Z}[x]/\langle \mathbf{f} \rangle$, where $\mathbf{f}$ is a degree-n irreducible polynomial (usually $\mathbf{f} = x^n + 1$). With those schemes one is able to achieve high speeds on several architectures as well as reasonably small signatures and key sizes. However, while no attacks are known that perform significantly better against schemes based on ideal lattices, it is still possible that further cryptanalysis will be able to exploit the additional structure[1]. Especially, if long-term security is an issue, it seems that standard lattices and the associated problems—e.g., the Learning With Errors (LWE) [34] or the Small Integer Solution (SIS) problem—offer more confidence than their ring counterparts.

The situation for code-based cryptography [9] is somewhat similar. The use of more structured codes, such as quasi-dyadic Goppa codes [31], has been the target of an algebraic attack [15] which is effective against certain (but not all) proposed parameters. This is an indication that the additional structure used to improve the efficiency of such cryptosystems might be also used by adversaries to improve their attack strategies. Moreover, basing a scheme on the plain LWE or SIS problem seems much more secure than using stronger assumptions on top of ideal lattices like the discrete-compact-knapsack (DCK) [21] or NTRU-related assumptions [16] that have not been studied extensively so far.

While results for ideal-lattice-based signatures have been published recently [11,22,32,33], currently no research is available dealing with implementation and performance issues of standard-lattice-based signatures. While the large keys of such schemes might prevent their adoption on constrained devices or reconfigurable hardware, the size of the keys is much less an issue on current multi-core CPUs which have access to large amounts of memory. In this context, the scheme by Bai and Galbraith [6] (from now on referred to as BG signature) is an interesting proposal as it achieves small signatures and is based on the standard LWE and SIS problems.

An interesting question arising is also the performance of schemes based on standard lattices and how to choose parameters for high performance. While FFT-techniques have been used successfully for ideal lattices on various architectures [22,35] there are no fast algorithms to speed up the necessary matrix-vector arithmetic. However, matrix-vector operations can be parallelized very efficiently and there are no direct restrictions on the parameters (for efficiency of ideal lattices n is usually chosen as power of two) so that there is still hope for high speed. The only results currently available dealing with the implementation of standard lattice-based instantiations rely on arithmetic libraries [7,20] and can thus not fully utilize the power of their target architectures.

An additional feature of the BG signature is that sampling of Gaussian noise is only needed during the much less performance-critical key-generation phase but not for signing[2]. While there was some progress on techniques for efficient

[1] There exists sieving algorithms which can exploit the ideal structure, but the speed-up is of no significance [24,36]. Some first ideas towards attacks with lower complexity were sketched by Bernstein in his blog [8].

[2] Omitting costly Gaussian sampling was also the motivation for the design of the GLP signature [21].

discrete Gaussian sampling [16,17,33] it is still not known how to implement the sampling efficiently[3] without leaking information on the sampled values through the runtime of the signing process (contrary to uniform sampling [22]).

While we cannot present a direct attack, careful observation of the runtime of software implementations (even remotely over a network) has led to various attacks in the past and thus it is desirable to achieve constant runtime or at least a timing independent from secret data [13,25].

Our Contribution. The contribution of this paper is twofold. First, we study the parameter selection of the BG signature scheme in more detail than in the original paper and assess its security level[4]. Based on our analysis of the currently most efficient attack we provide a new 128-bit security parameter set chosen for efficient software implementation and long-term security. We compare the runtimes of several attacks on LWE with and without a limit on the number of samples available. Since the behavior of the attacks in a suboptimal attack dimension is not well understood at this point, our analysis may be of independent interest for the hardness assessment of other LWE instances. Additionally, we introduce an optimized rejection sampling procedure and rearrange operations in the signature scheme.

The second part of the paper deals with the implementation of this parameter set on the ARM NEON and Intel AVX architectures optimized for high speed. By using parallelization, interleaving, and vectorization we achieve on average 1203924 cycles for signing and 335072 cycles for verification on the Haswell architecture. This corresponds to roughly 2824 signing and 10147 verification operations per second on one core of a CPU clocked with 3.4 GHz. While we do not set a speed record for general lattices, we are able to present the currently fastest implementation of a lattice-bases signature scheme that relies solely on standard assumptions and is competitive in terms of performance compared to classical and post-quantum signature schemes.

Availability of Software. We will place all software described in this paper into the public domain to maximize reusability of our results. We will submit the software to the eBACS benchmarking project [10] for public benchmarking.

Road Map. The paper is organized as follows: In Sect. 3 we introduce the original BG signature scheme and our modifications for efficiency. The security analysis is revisited and appropriate parameters are selected in Sect. 4. In Sect. 5 we discuss our NEON and AVX software implementation and finish with results and a comparison in Sect. 6.

[3] A software implementation of a constant time discrete Gaussian sampler using the Cumulative Distribution Table (CDT) approach was recently proposed by Bos et al. [12]. However, even for the small standard deviation required for lattice-based encryption schemes, the constant time requirement leads to a significant overhead.

[4] We note here that there was some vagueness in the parameter selection in the original work [6], also noticed later by the authors of the paper [5].

2 Preliminaries

Notation. We mainly follow the notation of [6] and denote column vectors by bold lower case letters (e.g., $\mathbf{v} = (v_1, \ldots, v_n)^T$ where $\mathbf{v}^T$ is the transpose) and matrices by bold upper case letters (e.g., $\mathbf{M}$). The centered discrete Gaussian distribution $\mathcal{D}_\sigma$ for $\sigma > 0$ associates the probability $\rho_\sigma(x)/\rho_\sigma(\mathbb{Z})$ to $x \in \mathbb{Z}$ for $\rho_\sigma(x) = \exp(\frac{-x^2}{2\sigma^2})$ and $\rho_\sigma(\mathbb{Z}) = 1 + 2\sum_{x=1}^{\infty} \rho_\sigma(x)$. We denote by $d \overset{\$}{\leftarrow} \mathcal{D}_\sigma$ the process of sampling a value d randomly according to $\mathcal{D}_\sigma$. In case S is a finite set, then $s \overset{\$}{\leftarrow} S$ means that the value s is sampled according to a uniform distribution over the set S. For an integer $c \in \mathbb{Z}$, we define $[c]_{2^d}$ to be the integer in the set $(-2^{d-1}, 2^{d-1}]$ such that $c \equiv [c]_{2^d} \mod 2^d$ which is basically extraction of the least significant bits. For $c \in \mathbb{Z}$ we define $\lfloor c \rceil_d = (c - [c]_{2^d})/2^d$ to drop the d least significant bits. Both operators can also be applied to vectors.

Lattices. A k-dimensional lattice Λ is a discrete additive subgroup of $\mathbb{R}^m$ containing all integer linear combinations of k linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_k$ with $k \leq m$ and $m \geq 0$. More formally, we have $\Lambda = \{ \mathbf{B} \cdot \mathbf{x} \mid \mathbf{x} \in \mathbb{Z}^k \}$. Throughout this paper we are mostly concerned with q-ary lattices $\Lambda_q^\perp(\mathbf{A})$ and $\Lambda_q(\mathbf{A})$, where $q = poly(n)$ denotes a polynomially bounded modulus and $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is an arbitrary matrix. $\Lambda_q^\perp(\mathbf{A})$ resp. $\Lambda_q(\mathbf{A})$ are defined by

$$\Lambda_q^\perp(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m \mid \mathbf{A}\mathbf{x} \equiv \mathbf{0} \mod q\}$$
$$\Lambda_q(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m \mid \exists \mathbf{s} \in \mathbb{Z}^m \text{ s.t. } \mathbf{x} = \mathbf{A}^\top \mathbf{s} \mod q\}.$$

By $\lambda_i(\Lambda)$ we denote the *i-th successive minimum*, which is the smallest radius r such there exist i linearly independent vectors of norm r (typically l_2 norm) in Λ. For instance, $\lambda_1(\Lambda) = \min_{\mathbf{x} \neq \mathbf{0}} ||\mathbf{x}||_2$ denotes the minimum distance of a lattice determined by the length of its shortest nonzero vector.

The SIS and LWE Problem. In the following we recall the main problems used in order to construct secure lattice-based cryptographic schemes.

Definition 1 (SIS-Problem). *Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, a modulus $q > 0$, and a real β, the small-integer-solution problem (l-norm typically $l = 2$) $\mathsf{SIS}_{n,m,\beta}$ asks to find a vector $\mathbf{x}$ such that $\mathbf{A}\mathbf{x} \equiv \mathbf{0} \mod q$ and $||\mathbf{x}||_l \leq \beta$.*

Let χ be a distribution over $\mathbb{Z}$. We define by $A_{\mathbf{s},\chi}$ the distribution of $(\mathbf{a}, \mathbf{a}^\top \cdot \mathbf{s} + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ for $n, q > 0$, where $\mathbf{a} \overset{\$}{\leftarrow} \mathbb{Z}_q^n$ is chosen uniformly at random and $e \leftarrow \chi$.

Definition 2 (LWE-Problem). *For a modulus $q = poly(n)$ and given vectors $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ sampled according to $A_{\mathbf{s},\chi}$ the learning-with-errors problem $\mathsf{LWE}_{\chi,q}$ asks to distinguish $A_{\mathbf{s},\chi}$, where $\mathbf{s}$ is chosen uniformly at random, from the uniform distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$.*

It is also possible to sample $\mathbf{s}$ according to the error distribution χ^n [3].

Departing from the original definition of LWE, that gives access to arbitrary many samples, an attacker has often only access to a maximum number of samples. Typically, this number of samples is denoted by m. In this case, one typically "collects" all samples $\mathbf{a}_i, b_i \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ to $\mathbf{A}, \mathbf{b} \in \mathbb{Z}_q^{m\times n} \times \mathbb{Z}_q^m$, and the LWE problem is to decide whether the entries of $\mathbf{b}$ were sampled uniformly at random and independently from $\mathbf{A}$ or according to the LWE distribution.

3 The Bai-Galbraith Signature Scheme

The Bai-Galbraith digital signature scheme [6] (BG signature) is based on the Fiat-Shamir paradigm which transforms an identification scheme into a signature scheme [18] and closely follows previous proposals by Lyubashevsky et al. [16,21,28,29]. The hardness of breaking the BG signature scheme, in the random oracle model, is reduced to the hardness of solving standard worst-case computational assumptions on lattices. The explicit design goal of Bai and Galbraith is having short signatures.

3.1 Description of the BG Signature Scheme

For easy reference, the key generation, signing, and the verification algorithm of the BG signature scheme are given in Fig. 1. Our proposed parameter set is summarized in Table 1. An analysis of the original parameter sets can be found in the full online version of this paper. However, the algorithms have been simplified and redundant definitions have been removed (e.g., we just use σ as standard deviation and do not differentiate between $\sigma_{\mathbf{E}}, \sigma_{\mathbf{S}}$ and set $n = k$).

During key generation two secret matrices $\mathbf{S} \in \mathbb{Z}^{n\times n}, \mathbf{E} \in \mathbb{Z}^{m\times n}$ are sampled from a discrete Gaussian distribution $D_\sigma^{n\times n}$ and $D_\sigma^{m\times n}$, respectively. A rejection condition CHECK_E enforces certain constraints on $\mathbf{E}$, which are necessary for correctness and short signatures (see Sect. 3.2). Finally, the public key $\mathbf{T} = \mathbf{AS} + \mathbf{E}$ and the secret key matrices $\mathbf{S}, \mathbf{E}$ are returned where $\mathbf{AS}$ is the only matrix-matrix multiplication necessary in the scheme. As we choose $\mathbf{A} \in \mathbb{Z}^{m\times n}$ as a global constant, it does not have to be sampled during key generation and is also not included in the public key and secret key.

For signing, the global constant $\mathbf{A}$ as well as secret keys $\mathbf{S}, \mathbf{E}$ are required (no usage of $\mathbf{T}$ in this variant). The vector $\mathbf{y}$ is sampled uniformly random from $[-B, B]^n$. For the instantiation of the random oracle H (using a hash function) only the higher order bits of $\mathbf{Ay}$ are taken into account and hashed together with the message μ. The algorithm $F(c)$ takes the binary output of the hash c and produces a vector $\mathbf{c}$ of weight ω (see [16] for a definition of $F(c)$). In a different way than [6] $\mathbf{w}$ is computed following an idea that has also been applied in [21]. Instead of computing $\mathbf{w} = \mathbf{Az} - \mathbf{Tc} \pmod q$ we calculate $\mathbf{w} = \mathbf{v} - \mathbf{Ec} \pmod q$, where $\mathbf{v} = \mathbf{Ay} \pmod q$. This is also the reason why $\mathbf{E}$ has to be included into the secret key $sk = (\mathbf{S}, \mathbf{E}) \in \mathbb{Z}^{n\times n} \times \mathbb{Z}^{m\times n}$. Thus, the large public key $\mathbf{T} \in \mathbb{Z}^{m\times n}$ is not needed anymore for signing and the operations become simpler (see further discussion in Sect. 5). The test whether

Algorithm KeyGen	Algorithm Sign	Algorithm Verify
INPUT:	INPUT:	INPUT:
$\mathbf{A}, n, m, q, \sigma$	$\mu, \mathbf{A}, \mathbf{S}, \mathbf{E}, B, U, d, w, \sigma$	$\mu, \mathbf{z}, c, \mathbf{A}, \mathbf{T}, B, U, d$
OUTPUT: $sk = (\mathbf{S}, \mathbf{E}), pk = (\mathbf{T})$	OUTPUT: $(\mathbf{z}, c)$	OUTPUT: Accept/Reject
1. $\mathbf{S} \overset{\$}{\leftarrow} D_\sigma^{n \times n}$	1. $\mathbf{y} \overset{\$}{\leftarrow} [-B, B]^n$	1. $\mathbf{y} \overset{\$}{\leftarrow} [-B, B]^n$
2. $\mathbf{E} \overset{\$}{\leftarrow} D_\sigma^{m \times n}$	2. $\mathbf{v} = \mathbf{Ay} \pmod q$	2. $\mathbf{v} = \mathbf{Ay} \pmod q$
3. **if** CHECK_E($\mathbf{E}$) $= 0$	3. $c = H(\lfloor \mathbf{v} \rceil_d, \mu)$	3. $c = H(\lfloor \mathbf{v} \rceil_d, \mu)$
then Restart	4. $\mathbf{c} = F(c)$	4. $\mathbf{c} = F(c)$
4. $\mathbf{T} = \mathbf{AS} + \mathbf{E} \pmod q$	5. $\mathbf{z} = \mathbf{y} + \mathbf{Sc}$	5. $\mathbf{z} = \mathbf{y} + \mathbf{Sc}$
5. **return** $sk = (\mathbf{S}, \mathbf{E}), pk = (\mathbf{T})$	6. $\mathbf{w} = \mathbf{v} - \mathbf{Ec} \pmod q$	6. $\mathbf{w} = \mathbf{v} - \mathbf{Ec} \pmod q$
	7. **if** $\lvert[\mathbf{w}_i]_{2^d}\rvert > 2^{d-1} - L$	7. **if** $\lvert[\mathbf{w}_i]_{2^d}\rvert > 2^{d-1} - L$
	then Restart	**then** Restart
	8. **return** $(\mathbf{z}, c)$	8. **return** $(\mathbf{z}, c)$
	if $\lvert\lvert\mathbf{z}\rvert\rvert_\infty \leq B - U$	**if** $\lvert\lvert\mathbf{z}\rvert\rvert_\infty \leq B - U$

Fig. 1. The BG signature scheme [6]; see Sect. 3.2 for implementations of CHECK_E.

$|[\mathbf{w}_i]_{2^d}| > 2^{d-1} - L_{BG}$ ($L_{BG} = 7\omega\sigma$ in [6]) ensures that the signature verification will not fail on a generated signature ($\mathbf{w}$ is never released) and the last line ensures that the signature is uniformly distributed within the allowed range $[-B+U, B-U]^n$ for $U = 14 \cdot \sigma\sqrt{\omega}$.

For verification the higher order bits of $\mathbf{w} = \mathbf{Az} - \mathbf{Tc} = \mathbf{Ay} - \mathbf{Ec}$ are hashed and a valid signature $(\mathbf{z}, c)$ is accepted if and only if $\mathbf{z}$ is small, i.e., $||\mathbf{z}||_\infty \leq B - U$, and $c = c'$ for $c' := H(\lfloor \mathbf{w} \rceil_d, \mu)$. For the security proof and standard attacks we refer to the original work [6].

3.2 Optimizing Rejection Sampling

In the original signature scheme [6] CHECK_E$_{BG}$ restarts the key generation if $|\mathbf{E}_{i,j}| > 7\sigma$ for any (i, j) and the rejection condition in Line 7 of Sign is $|[\mathbf{w}_i]_{2^d}| > 2^{d-1} - L_{BG}$ for $L_{BG} = 7w\sigma$. This ensures that it always holds that $\lfloor \mathbf{Ay} \rceil_d = \lfloor \mathbf{Ay} - \mathbf{Ec} \rceil_d$ and thus verification works even for the short signature. However, in practice the acceptance probability of $(1 - 14\omega\sigma/2^d)^m$ has a serious impact on performance and leaves much room for improvement. On first sight it would seem most efficient to test during signing whether $\lfloor \mathbf{Ay} \rceil_d = \lfloor \mathbf{Ay} - \mathbf{Ec} \rceil_d$ and just reject signatures that would not be verifiable. However, in this case the proof structure given in the full version of [6] does not work anymore. In Game 1, sign queries are replaced by a simulation (in the random oracle model) which is not allowed to use the secret key and later on has to produce valid signatures even for an invalidly chosen public key (Game 2).

Our optimization (similar to [16]) is to reject $\mathbf{E}$ during key generation only if the error generated by $\mathbf{Ec}$ in $\lfloor \mathbf{Ay} \rceil_d = \lfloor \mathbf{Ay} - \mathbf{Ec} \rceil_d$ for the worst-case $\mathbf{c}$ is larger than a threshold L. Thus, our CHECK_E$_{new}$ algorithm works the following: Using $\max_k(\cdot)$ which returns the k-th largest value of a vector we compute thresholds $t_h = \sum_{k=1}^{\omega} \max_k(|\mathbf{E}_h|), \forall h \in [0, m]$ where $\mathbf{E}_h$ is the h-th row of $\mathbf{E}$ and reject if one or more t_h are larger than L. Thus the rejection probability for the close-to-uniform $\mathbf{w}$ is independent of $\mathbf{c}$ and $\mathbf{E}$ and does not leak any information. When L is chosen such that only a small percentage of secret keys are rejected the LWE instances generated by the public key are still hard due to the same argument on the bounded number of samples as in [6,16]. The acceptance probability of $\mathbf{w}$ in Line 7 of Sign is $(1 - 2L/2^d)^m$. Table 1 shows concrete values for our choice of L_{new} and the original L_{BG}.

4 Security Analysis and Parameter Selection

In the original work [6], Bai and Galbraith proposed five different parameter sets to instantiate their signature scheme. In this section we revisit their security analysis and propose a new instantiation that is optimized for software implementations on modern server and desktop computers (Intel/AMD) and also mobile processors (ARM). The security analysis has been refined due to the following reasons: First, a small negligence in the assessment of the underlying LWE instances leads to a slightly wrong hardness estimation, which was acknowledged by the authors after publication [5]. Second, an important attack, namely the decoding attack, was not considered in [6]. We justify that indeed the decoding attack is less efficient than the one considered if one takes into account the limited number of samples m given to the attack algorithms.

In Table 1 we propose a parameter set for an instantiation of the signature scheme from Sect. 3 with 128 bits of security, for which we provide evidence in the next section.

4.1 Hardness of LWE

The decoding attack dates back to the nearest-plane algorithm by Babai [4] and was further improved by Lindner and Peikert in [26] and Liu and Nguyen in [27]. While it is often the fastest known approach, it turns out that it is not very suitable for our instances, because an attacker has only access to a few samples. Thus we concentrate on the embedding approach here and an analysis of the behavior of the decoding attack can be found in Appendix A.

The embedding approach solves LWE via a reduction to the unique-shortest-vector problem (uSVP). We will analyze two variants, the standard embedding approach [26] and the variant that is very suitable for LWE instances with small m that was already considered in [6]. Unfortunately, it is necessary to re-do the analysis, because the hardness evaluation in the original work [6] set some constant – namely τ – in the attack wrong yielding up to 17 bits more security for their parameters than actually offered. We will focus on the security of our parameter set in this section. Updated values for some of the parameter sets proposed in the original paper can be found in the full version of this paper.

Embedding Approach. Given an LWE instance $(\mathbf{A}, \mathbf{b})$ such that $\mathbf{A}\mathbf{s} = \mathbf{b} \bmod q$, the idea of the embedding approach proposed in [19] is to use the

Table 1. The parameter set we use for 128 bits of security. Note that signature and key sizes refer to fully compressed signature and keys. Our software uses slightly a larger (padded) signature and keys to support faster loads and stores aligned to byte boundaries.

Parameter Selection		
Parameter	Bound	Value
n		532
m		840
σ		43
ω	$2^{\omega}\binom{n}{\omega} \geq 2^{128}$	18
d	d is s.t. $(1 - 14\sigma\omega/2^{d})^{m} \geq 1/3$	23
B	power of two $\geq 14\sqrt{\omega}\sigma(n-1)$	$2^{21}-1$
q	$\geq \left(2^{(d+1)m+\kappa}/(2B)^{n}\right)^{1/(m-n)}$	$2^{29}-3$
U	$14 \cdot \sigma\sqrt{\omega}$ (Prob. of acceptance Line 8 of Sign: 0.51)	2554.1
L_{BG}	$7w\sigma$ (Prob. of acceptance Line 3 of KeyGen: ≈ 1) (Prob. of acceptance Line 7 of Sign: 0.337)	5418
L_{new}	$3w\sigma$ (Prob. of acceptance Line 3 of KeyGen: 0.99) (Prob. of acceptance Line 7 of Sign: 0.628)	2322
public-key size	$m \cdot n \cdot \lceil \log_2(q) \rceil$	1.54 Mb
secret-key size	$(n^2 + n \cdot m)\lceil \log_2(14 \cdot \sigma) \rceil$	0.87 Mb
signature size	$n \cdot \lceil \log_2(2B) \rceil + 256$	11960 bits

embedding lattice $\Lambda_q(\mathbf{A}_e)$ defined as

$$\Lambda_q(\mathbf{A}_e) = \{\mathbf{v} \in \mathbb{Z}^m \mid \exists \mathbf{x} \in \mathbb{Z}^n : \mathbf{A}_e \cdot \mathbf{x} = \mathbf{v} \mod q\},$$

where $\mathbf{A}_e = \begin{pmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{0} & 1 \end{pmatrix}$. Throughout the paper the subscript stands for the technique used in an attack such as e denoting the standard embedding approach. Since

$$\mathbf{A}_e \begin{pmatrix} -\mathbf{s} \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{0} & 1 \end{pmatrix} \begin{pmatrix} -\mathbf{s} \\ 1 \end{pmatrix} = \begin{pmatrix} -\mathbf{A}\mathbf{s} + \mathbf{b} \\ 0 \cdot s + 1 \cdot 1 \end{pmatrix} = \begin{pmatrix} \mathbf{e} \\ 1 \end{pmatrix} =: \mathbf{v}$$

is a very short lattice vector, one can apply a solver for uSVP to recover $\mathbf{e}$. We estimate the norm of $\mathbf{v}$ via $||\mathbf{v}|| \approx ||\mathbf{e}|| \approx \sqrt{m}\sigma_E$, and for the determinant of the lattice we have $\det(\Lambda_q(\mathbf{A}_e)) = q^{m+1-n}$ with very high probability [9].

It is known that the hardness of uSVP depends on the gap between the first and the second successive minimum $\lambda_1(\Lambda)$ and $\lambda_2(\Lambda)$, respectively. Gama and Nguyen [19] claim that an attack with a lattice-reduction algorithm that achieves Hermite factor δ succeeds with high probability if $\lambda_2(\Lambda)/\lambda_1(\Lambda) \geq \tau \cdot \delta^{\dim(\Lambda)}$,

Table 2. Security of our parameter set

Security level		
Problem	Attack	Bit security
LWE	Decoding [26]	271
	Embedding [2]	192
	Embedding [6]	130
SIS	Lattice reduction [6]	159

where $\tau \approx 0.4$ is a constant that depends on the reduction algorithm used. In fact, this factor is missing in the analysis by Bai and Galbraith, which causes too optimistic (i.e., too large) runtime predictions.

The successive minima of a random lattice Λ can be predicted by the Gaussian heuristic via

$$\lambda_i(\Lambda) \approx \frac{\Gamma(1+\dim(\Lambda)/2)^{1/\dim(\Lambda)}}{\sqrt{\pi}} \det(\Lambda)^{1/\dim(\Lambda)}.$$

Consequently, a particular short vector $\mathbf{v}$ of length $||\mathbf{v}|| = l$ can be found if

$$\delta^{\dim(\Lambda)} \leq \frac{\lambda_2(\Lambda)}{\lambda_1(\Lambda)\cdot\tau} \approx \frac{\Gamma(1+\dim(\Lambda)/2)^{1/\dim(\Lambda)}}{l\cdot\sqrt{\pi}\cdot\tau} \det(\Lambda)^{1/\dim(\Lambda)}. \qquad (1)$$

We can therefore estimate the necessary Hermite delta to break LWE with the embedding approach to be

$$\delta \approx \left(\frac{\Gamma(1+\frac{m+1}{2})^{1/(m+1)}}{\sqrt{\pi}\cdot m\cdot\tau\cdot\sigma_E} q^{\frac{m+1-n}{m+1}} \right)^{1/(m+1)},$$

where the dimension is set to $\dim(\Lambda_q(\mathbf{A}_e)) = m+1$. Note that it is possible to apply this attack in a smaller subdimension. In fact, there exists an optimal dimension that minimizes δ in Eq. (1). Our parameters, however, do not provide enough LWE samples to allow an attack in the optimal dimension, and in this case choosing the highest possible dimension seems to be optimal.

To achieve a small Hermite delta, it is necessary to run a basis-reduction algorithm like BKZ [37] or its successor BKZ 2.0 [14]. Lindner and Peikert [26] proposed the function

$$\log_2(T(\delta)) = 1.8/\log_2(\delta) - 110$$

to predict the time necessary to achieve a given Hermite delta by BKZ. More recently, Albrecht et al. [2] proposed the prediction

$$\log_2(T(\delta)) = 0.009/\log_2(\delta)^2 - 27$$

based on data taken from experiments with BKZ 2.0 [27]. We will stick to this estimation in the following, since it takes more recent improvements into consideration. Combining it with the fact that they run their experiments on a machine that performs about $2.3 \cdot 10^9$ operations per second, we estimate the number of operations necessary to achieve a given Hermite factor with

$$T(\delta) = \frac{2.3 \cdot 10^9}{2^{27}} \cdot 2^{0.009/\log(\delta)^2}. \tag{2}$$

We can therefore conclude that our LWE instance provides about 192 bits of security against the embedding attack, which corresponds to a Hermite delta of approximately 1.0048.

The efficacy of the standard embedding approach decreases significantly if the instance does not provide enough samples for the attack to run in the optimal dimension. Another attack, which is very suitable for LWE instances with few samples, reduces LWE to an uSVP instance defined by the lattice $\Lambda_q^\perp(\mathbf{A}_o) = \{\mathbf{v} \in \mathbb{Z}^{m+n+1} \mid \mathbf{A}_o \cdot \mathbf{v} = \mathbf{0} \mod q\}$ for $\mathbf{A}_o = \left[\mathbf{A} \mid \mathbf{I} \mid \mathbf{b}\right]$ (we use the index o because this attack runs in the lattice of the vectors that are orthogonal to $\mathbf{A}_o$). The main advantage of this attack is that it runs in dimension $n+m+1$ (recall that the standard embedding approach runs in dimension $m+1$). For $\mathbf{v} = \left(\mathbf{s}\ , \mathbf{e}\ , -1\right)^T$, we have $\mathbf{A}_o \cdot \mathbf{v} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e} - \mathbf{b} = \mathbf{0}$ and therefore $\mathbf{v} \in \Lambda_q^\perp(\mathbf{A}_o)$ is a small vector in the lattice. We estimate its length via $||\mathbf{v}|| \approx \sqrt{||\mathbf{s}||^2 + ||\mathbf{e}||^2} \approx \sqrt{m+n} \cdot \sigma$. Since $\det(\Lambda_q(\mathbf{A}_o)) = q^m$ with high probability [9], Eq. (1) predicts the necessary Hermite delta to be approximately

$$\delta \approx \left(\frac{\Gamma(1 + \frac{n+m+1}{2})^{1/(n+m+1)}}{\sqrt{n+m}\sigma \cdot \sqrt{\pi} \cdot \tau} q^{\frac{m}{n+m+1}} \right)^{1/(n+m+1)}.$$

Using Eq. (2), we can estimate the hardness of our instance against this attack to be about 130 bits (the Hermite delta is close to 1.0059).

4.2 Hardness of SIS

Instead of recovering the secret key, which corresponds to solving an instance of LWE, an attacker could also try to forge a signature directly and thus solve an SIS instance. We predict the hardness of SIS for the well-known lattice-reduction attack (see for example [9]) like it was done in [6]. This attack views SIS as a variant of the (approximate) shortest-vector problem and finds the short vector by applying a basis reduction. Forging a signature through this attack requires to find a reduced basis with Hermite factor

$$\delta = (D/q^{m/(m+n)})^{1/(n+m+1)}, \tag{3}$$

with $D = (\max(2B, 2^{d-1}) + 2E'\omega)$ for E' satisfying $(2E')^{m+n} \geq q^m 2^{132}$. Applying Eq. (2), we estimate that a successful forger requires to perform about 2^{159} operations (see Table 2).

4.3 An Instantiation for Software Efficiency

Choosing optimal parameters for the scheme is a non-trivial multi-dimensional optimization problem and our final parameter set is given in Table 1. Since the probability that the encoding function F maps two random elements to the same value must be negligible (i.e. smaller than 2^{-128}), we choose ω such that $2^{\omega}\binom{n}{\omega} \geq 2^{128}$. Since $\mathbf{Sc}$ is distributed according to a Gaussian distribution with parameter $\sqrt{\omega}\sigma$, we can bound its entries by $14\sqrt{\omega}\sigma$. Consequently, $B-U$ is lower bounded by $14\sqrt{\omega}\sigma(n-1)$ such that the acceptance probability of a signature P_{acc} (Line 8 in Fig. 1) is at least

$$P_{acc} = \left(\frac{2(B-U)+1}{2B}\right)^m = \left(\frac{2 \cdot 14\sqrt{\omega}\sigma(n-1)+1}{2 \cdot 14\sqrt{\omega}\sigma n+1}\right)^m \approx \left(1-\frac{1}{n}\right)^m \approx 1/e\,.$$

The next important choice to be made is the value for the parameter d. It has a determining influence on the trade-off between runtime and key sizes: The success probability in the signing algorithm (Line 7 in Fig. 1) is given by $(1-2L/2^d)^m$, which means that large values for d lead to a high success probability, and thereby to fewer rejections implying better running times. On the other hand, the security proof requires $(2B)^n q^{m-n} \geq 2^{(d+1)m+\kappa}$ to be satisfied, which means that increasing d implies larger values for q, hence, worsening runtime and key sizes.

Our goal is to come up with a parameter set that ensures at least 128 bits of security. We will focus on n,m and σ in this paragraph, since the other parameters depend on them. For easy modular reduction we choose a modulus slightly smaller than a power of two (like $2^{29}-3$). Furthermore, dimensions n resp. m are multiples of 4 to support four parallel operations in vector registers. In a way, n determines the overall security level, and the choice of σ and n can be used to balance the security of the scheme and the size of the second-order parameters q and B. Using our parameters we have set $L = L_{new} = 3\omega\sigma$ and thus reject a secret key with probability 0.025 and accept with probability $(1-2L/2^d)^m$ where we get $\approx$0.63 instead of $\approx$0.34 for $L_{BG} = 7\sigma\omega$.

For instance, Fig. 2 shows for $n = 532$ how the lower bound on q depends on σ for various values of m. Since too small values of σ lead to LWE-instances that are significantly easier than 128 bits, the best possible choice that allows $q = 2^{29}-3$ is $m = 840$ and $\sigma = 43$. We further choose $n = 532$ which leads to $\omega = 18$. This results in the lower bound $\log_2(B) \geq 20.4$, which allows our choice $B = 2^{21}-1$.

5 Implementation Details

In this section we discuss our techniques used for high performance on modern desktop and mobile CPUs with fast vector units. More specifically, we optimized the signature scheme for Intel Ivy Bridge CPUs with AVX, for Intel Haswell CPUs with AVX2 and for ARMv7 CPUs with NEON vector instructions. We first describe various high-level (platform-independent) optimizations for signing and

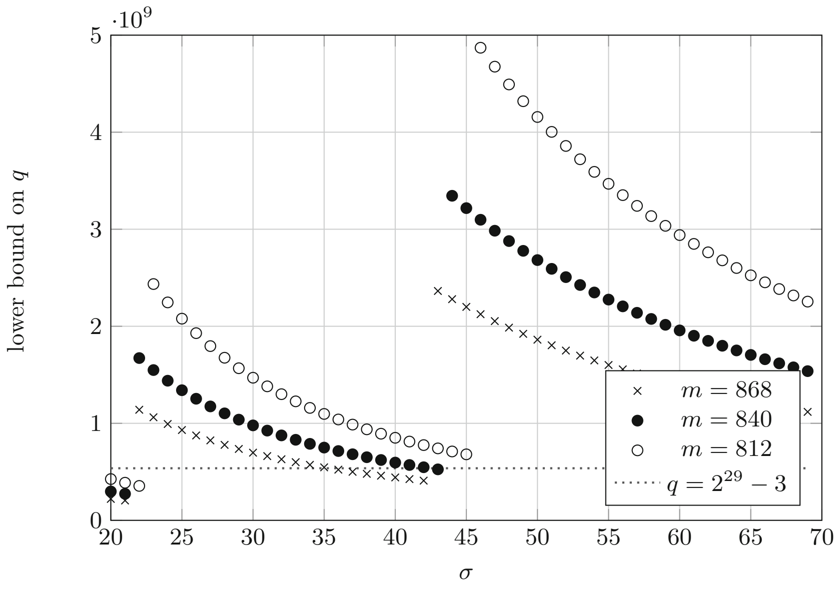

Fig. 2. Lower bound on q for $n = 532$ and various values of m

verification and then detail the low-level implementation techniques for the three target platforms. Our implementation only optimizes signing and verification speeds; our implementation includes a (slow) routine for key generation but we will not discuss key generation here.

5.1 High-Level Optimizations

Regarding platform independent high-level optimizations we follow the approach from [22] and would like to emphasize the changes to the algorithm (adding **E** to the private key and choosing **A** as global constant) and improved rejection sampling (usage of L_{new}) as discussed in Sect. 3. For uniform sampling of $\mathbf{y} \overset{\$}{\leftarrow} [-B, B]^n$ during signing we rely on the hybrid approach of seeding the Salsa20 stream cipher using true randomness from the Linux random number [22]. As $B = 2^{21} - 1$ we sample $3n + 68$ uniform bytes at once using Salsa20 and construct a sample r' from 3 bytes each. By computing $r = r' \bmod 2^{22}$ we bring r into the range $[0, 2^{22} - 1]$, reject if $r = 2^{22} - 1$ and return $r - (2^{22} - 1)$. The probability to discard an element is 2^{-22} and by oversampling 68 bytes it is highly unlikely that we have to sample additional randomness. We also exploit that **c** is sparse with weight ω. Thus, we store **c** not as a vector but as list with ω tuples containing the position and sign bits of entries which are non zero. Additionally, when multiplying **c** with **S** and **E**, only a small subset of coefficients from **S**, **E** is actually needed. As a consequence, we do not unpack the whole matrices **S**, **E** from the binary representation of the secret key (which is the usual approach) but just the coefficients that are required in this case. Additionally, during signing we perform rejection sampling on $\mathbf{w}$ before we actually compute $\mathbf{v}$ in order to be able to abort as early as possible (without leaking timing information). For hashing $H(\lfloor \mathbf{v} \rceil_d, \mu)$ and $H(\lfloor \mathbf{w} \rceil_d, \mu)$, respectively, we pack the input to the hash function after extraction of higher-order bits in order to keep the input buffer to the hash function as small as possible.

5.2 Low-Level Optimizations in AVX and AVX2

With the Sandy Bridge microarchitecture, Intel introduced the AVX instruction set. AVX extends the 16 128-bit `XMM` vector registers of the SSE instruction set to 256-bit `YMM` registers. Arithmetic instructions treat these registers either as vectors of 4 double-precision or 8-single precision floating-point numbers. Each cycle, the Intel Sandy Bridge and later Ivy Bridge CPUs can issue one addition instruction and one multiplication instruction on those vectors. The power of these vector-arithmetic units was exploited by [22] to achieve very high speeds for GLP signatures. We also use these floating-point vector operations for our software. With the Haswell microarchitecture, Intel introduced AVX2, which extends the AVX instruction set. There are two notable additions. One is that vector registers can now also be treated as vectors of integers (of various sizes); the other is that Intel added floating-point multiply-accumulate instructions. Haswell CPUs can issue two floating-point multiply-accumulate vector instructions per cycle.

The basic approach for our implementation is that all elements of $\mathbb{Z}_q$ are represented as double-precision floating-point numbers. The mantissa of a double-precision float has 53 bits and a 29-bit integer can thus obviously be represented exactly. One might think that 53 bits are still not enough, because *products* of elements of $\mathbb{Z}_q$ do not fit into the mantissa. However, the signature scheme never computes the product of two full-size field elements. The largest products appear in the matrix-vector multiplications $\mathbf{Ay}$ and $\mathbf{Az}$. The coefficients of $\mathbf{A}$ are full-size $\mathbb{Z}_q$ elements in the interval $[-(q-1)/2, (q-1)/2]$, but the coefficients of $\mathbf{y}$ are in $[-B, B]$ and the coefficients of $\mathbf{z}$ are in $[-(B-U), B-U]$. With $B = 2^{21} - 1$ each coefficient multiplication in $\mathbf{Ay}$ produces a result of at most 49 bits.

Matrix-vector multiplication. The matrix-vector multiplications $\mathbf{Ay}$ and $\mathbf{Az}$ are not only the operations which produce the largest intermediate results, they are also the operations which dominate the cost for signing and verification, respectively. The AVX and AVX2 implementations store the matrix $\mathbf{A}$ in transposed form which allows more efficient access to the elements of $\mathbf{A}$ in vector registers. One can think of the whole computation as a sequence of multiply-accumulate instructions, where one factor is a vector register containing 4 coefficients of $\mathbf{A}$, the other factor is a vector register containing 4 copies of the same coefficient of $\mathbf{y}$ (or $\mathbf{z}$) and the accumulator is a vector register containing 4 result coefficients. Loading the same coefficient of $\mathbf{y}$ into all 4 elements of a vector register can be done efficiently through the `vbroadcastsd` instruction. Latencies can be hidden by interleaving instructions from the computation of independent vectors of result coefficients.

One might think that $n \cdot m = 532 \cdot 840 = 446880$ multiplications and accumulations translate into 111720 AVX and 55860 AVX2 cycles (because AVX handles 4 vectorized multiplications and 4 vectorized additions per cycle and AVX2 handles 2×4 vectorized multiply-accumulates per cycle), but this is not the case. It turns out that arithmetic is not the bottleneck but access to matrix coefficients. Note that if we store $\mathbf{A}$ as 446880 double-precision floats, the matrix occupies about 3.5 MB of storage – way too much for the 32 KB L1 cache. Also note that each matrix coefficient is used exactly once, which is the most cache-unfriendly access pattern possible. We overcome this bottleneck to some extent by storing the coefficients of $\mathbf{A}$ as 32-bit integers. We then load 4 coefficients (and convert to double-precision floats on the fly) using the `vcvtdq2pd` instruction of the AVX instruction set. An additional cost stems from reductions modulo q of coefficients. We can use lazy-reduction, i.e., we do not have to reduce after every multiply-accumulate. For example in the computation of $\mathbf{Ay}$ we have to reduce after 16 multiply-accumulate operations. Our software is currently overly conservative and reduces after 7 multiply-accumulates in both cases. We perform modular reduction of floating-point coefficients in the same way as [22]: We produce a “carry” by multiplying by a floating-point approximation of q^{-1}, then use the `vroundpd` instruction to round that carry to the nearest integer, multiply by q and then subtract the carry from the original value.

In total, the matrix-vector multiplication takes 278912 cycles on a Haswell CPU and 488474 cycles on an Ivy Bridge CPU.

5.3 Low-Level Optimization in NEON

Fast vector units are not only present in large desktop and server CPUs but also in mobile CPUs. Most ARM Cortex-A processors include the NEON vector extensions. These extensions add 16 128-bit vector registers. The most powerful arithmetic instructions are addition and subtraction of vectors of 4 32-bit integers or 2 64-bit integers (one per cycle) and multiplication of vectors of 2 32-bit integers producing as a result a vector of 2 64-bit integers (one every two cycles). The NEON instruction set also includes multiply-accumulate at the same cost of a multiplication.

For our optimized NEON implementation we represent elements of $\mathbb{Z}_q$ as 32-bit signed integers. Products of coefficients in the matrix-vector multiplications $\mathbf{Ay}$ and $\mathbf{Az}$ are represented as 64-bit signed integers. Lazy reduction can go much further than in AVX and AVX2; we only have to perform one reduction modulo q at the very end of the computation.

In most aspects, the NEON implementation follows the ideas of the AVX and AVX2 implementations, but two aspects are different. One aspect is that simply storing the transpose of $\mathbf{A}$ is not sufficient for efficient vectorized access to the elements of $\mathbf{A}$. The reason is that the ARM-NEON addressing modes are by far not as flexible as the Intel addressing modes. Therefore, we store the matrix $\mathbf{A}$ such that each vector load instruction can simply pick up the next 4 coefficients of $\mathbf{A}$ and then increment the pointer to $\mathbf{A}$ as part of the load instruction.

The other aspect is modular reduction. In NEON we are operating on integers so the modular reduction technique we use for floats in AVX and AVX2 does not work. This is where the special shape of $q = 2^{29} - 3$ comes into play. Reduction modulo q on integers can be achieved with various different approaches, we currently use one shift, a logical and, and three additions to reduce modulo q. Obviously we always reduce two coefficients in parallel using vector instructions.

The penalty for access to coefficients of $\mathbf{A}$ is even higher than on the Intel platforms. Instead of 446880 cycles which one might expect from an arithmetic lower bound, matrix-vector multiplication takes 2448008 cycles.

6 Results and Comparison

Our software follows the eBACS API [10] and we will submit the software to eBACS for public benchmarking. In this section we do *not* report cycle counts obtained by running the eBACS benchmarking framework SUPERCOP. The reason is the same as in [22]: eBACS reports median cycle counts which is much too optimistic for the signing procedure which includes rejection sampling. Instead, we benchmark $10,000$ signature generations and report the average of those measurements. Verification does not include any rejection sampling and we thus report the more stable median of $10,000$ measurements.

We benchmarked our software on three machines, namely

– A machine with an Intel Core i7-4770K (Haswell) CPU running Debian GNU/Linux with gcc 4.6.3. Compilation used compiler flags `-msse2avx -march=corei7-avx -O3 -std=gnu99`.
– A machine with an Intel Core i5-3210M (Ivy Bridge) CPU running Ubuntu GNU/Linux with gcc 4.6.3. Compilation used compiler flags `-msse2avx -march=corei7-avx -O3 -std=gnu99`.
– A Beaglebone Black development board with a TI Sitara AM335x (ARM Cortex-A8) CPU running Debian GNU/Linux with gcc 4.6.3. Compilation used compiler flags `-O3 -flto -march=armv7-a -Ofast -funroll-all-loops -marm -mfpu=neon -fprefetch -loop-arrays-mvectorize-with-neon-quad -mthumb-interwork -mtune=cortex-a15`.

All benchmarks were carried out on just one core of the CPU and we followed the standard practice of turning off TurboBoost and hyperthreading.

Table 3 reports performance results of our software and compares it to previous implementations of lattice-based signatures. As an additional contribution of this paper we improved the performance of the software presented in [22]. We report both the original and the improved cycle counts in Table 3. For details on the improvement we refer to the full version of this paper. Compared with our work it becomes clear that usage of standard lattices only incurs a small performance penalty. This is remarkable, as no efficient and quasi-logarithmic-runtime arithmetic like the number-theoretic transform (NTT) is available for standard lattices. Moreover, for a security level matching the security level of GLP we expect our implementation to be much faster (m, n, q could be decreased). For BLISS performance we rely on the implementation given in [16]. However, an implementation of BLISS which uses techniques similar to those described in [22], should be much faster due to smaller parameters and lower rejection rates than in GLP. The main problem of BLISS is that it requires efficient (and secure) sampling of Gaussian noise not only for key generation but also for signing. All efficient techniques for Gaussian sampling rely heavily on secret branch conditions or lookup tables, which are both known to create timing leaks (see [12]).

Table 3. Comparison of lattice-based-signature software performance

Software	**CPU**	**Security**	**Cycles**
Software using standard lattices			
This work	Intel Core i7-4770K (Haswell)	128 bits	**sign:** 1203924 **verify:** 335072
This work	Intel Core i5-3210M (Ivy Bridge)	128 bits	**sign:** 1973610 **verify:** 608870
This work	TI Sitara AM335x (ARM Cortex-A8)	128 bits	**sign:** 10264721 **verify:** 2796433
`GPV-matrix` [7] ($n = 512, k = 27$)	AMD Opteron 8356 (Barcelona)	100 bits	**sign:** 287500000 **verify:** 48300000
Software using ideal lattices			
GLP [22]	Intel Core i5-3210M (Ivy Bridge)	75–80 bits	**sign:** 634988 **verify:** 45036
GLP [22] (see full version)	Intel Core i5-3210M (Ivy Bridge)	75–80 bits	**sign:** 452223 **verify:** 34004
`GPV-poly` [7] ($n = 512, k = 27$)	AMD Opteron 8356 (Barcelona)	100 bits	**sign:** 71300000 **verify:** 9200000
`BLISS` [16] (BLISS-I)	Intel Core i7	128 bits	**sign:** $\approx$ 421600 **verify:** $\approx$ 102000
`PASS` [23] ($N = 1153$)	Intel Core i7-2640M (Sandy Bridge)	130 bits	**sign:** 584230 **verify:** 172641

Conclusion and future work. With this work we have shown that the performance impact of using standard lattices over ideal lattices for short digital signatures is only small for signing and manageable for verification. Possible future work might consist in evaluating the performance of an independent time implementation of vectorized BLISS or PASS. Moreover, NTRUsign might become interesting again if it is possible to fix the security issues efficiently, as proposed in [1].

Acknowledgment. We would like to thank Patrick Weiden, Rafael Misoczki, Shi Bai, and Steven Galbraith for useful discussions. We would further like to thank the anonymous reviewers for their suggestions and comments.

References

1. Melchor, C.A., Boyen, X., Deneuville, J.-C., Gaborit, P.: Sealing the leak on classical NTRU signatures. In: Mosca, M. (ed.) PQCrypto 2014. LNCS, vol. 8772, pp. 1–21. Springer, Heidelberg (2014). 99
2. Albrecht, M.R., Fitzpatrick, R., Göpfert, F.: On the efficacy of solving LWE by reduction to unique-SVP. Cryptology ePrint Archive, Report 2013/602 (2013). http://eprint.iacr.org/2013/602/. 92
3. Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 595–618. Springer, Heidelberg (2009). 87
4. Babai, L.: On Lovász' lattice reduction and the nearest lattice point problem. Combinatorica **6**(1), 1–13 (1986). http://www.csie.nuk.edu.tw/~cychen/Lattices/Onlovaszlatticereductionandthenearestlatticepointproblem.pdf. 90, 102
5. Bai, S., Galbraith, S.: Personal communication and e-mail exchanges (2014). 86, 90
6. Bai, S., Galbraith, S.D.: An improved compression technique for signatures based on learning with errors. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 28–47. Springer, Heidelberg (2014). 85, 86, 87, 88, 89, 90, 92, 93, 102
7. El Bansarkhani, R., Buchmann, J.: Improvement and efficient implementation of a lattice-based signature scheme. In: Lange, T., Lauter, K., Lisoněk, P. (eds.) SAC 2013. LNCS, vol. 8282, pp. 48–67. Springer, Heidelberg (2014). 84, 85, 99
8. Bernstein, D.J.: A subfield-logarithm attack against ideal lattices, Feb 2014. http://blog.cr.yp.to/20140213-ideal.html. 85
9. Bernstein, D.J., Buchmann, J., Dahmen, E. (eds.): Post-Quantum Cryptography. Mathematics and Statistics. Springer, Heidelberg (2009). 84, 85, 91, 93
10. Bernstein, D.J., Lange, T.: eBACS: ECRYPT benchmarking of cryptographic systems. http://bench.cr.yp.to. Accessed 25 Jan 2013. 86, 98
11. Boorghany, A., Jalili, R.: Implementation and comparison of lattice-based identification protocols on smart cards and microcontrollers. IACR Cryptology ePrint Archive, 2014. http://eprint.iacr.org/2014/078/. 85
12. Bos, J.W., Costello, C., Naehrig, M., Stebila, D.: Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. IACR Cryptology ePrint Archive (2014). http://eprint.iacr.org/2014/599. 86, 99
13. Brumley, D., Boneh, D.: Remote timing attacks are practical. In: SSYM 2003 Proceedings of the 12th Conference on USENIX Security Symposium. USENIX Association (2003). http://crypto.stanford.edu/dabo/pubs/papers/ssl-timing.pdf. 86
14. Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better lattice security estimates. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 1–20. Springer, Heidelberg (2011). 92
15. Couvreur, A., Otmani, A., Tillich, J.P.: Polynomial time attack on wild McEliece over quadratic extensions. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 17–39. Springer, Heidelberg (2014). 85
16. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 40–56. Springer, Heidelberg (2013). 85, 86, 88, 89, 99
17. Dwarakanath, N.C., Galbraith, S.D.: Sampling from discrete Gaussians for lattice-based cryptography on a constrained device. Appl. Algebra Eng. Commun. Comput. **25**(3), 159–180 (2014). 86

18. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). 88
19. Gama, N., Nguyen, P.Q.: Predicting lattice reduction. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 31–51. Springer, Heidelberg (2008). 90, 91
20. Göttert, N., Feller, T., Schneider, M., Buchmann, J., Huss, S.: On the design of hardware building blocks for modern lattice-based encryption schemes. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 512–529. Springer, Heidelberg (2012). 85
21. Güneysu, T., Lyubashevsky, V., Pöppelmann, T.: Practical lattice-based cryptography: a signature scheme for embedded systems. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 530–547. Springer, Heidelberg (2012). 84, 85, 88
22. Güneysu, T., Oder, T., Pöppelmann, T., Schwabe, P.: Software speed records for lattice-based signatures. In: Gaborit, P. (ed.) PQCrypto 2013. LNCS, vol. 7932, pp. 67–82. Springer, Heidelberg (2013). 85, 86, 95, 96, 97, 98, 99
23. Hoffstein, J., Pipher, J., Schanck, J.M., Silverman, J.H., Whyte, W.: Practical signatures from the Partial Fourier recovery problem. In: Boureanu, I., Owesarski, P., Vaudenay, S. (eds.) ACNS 2014. LNCS, vol. 8479, pp. 476–493. Springer, Heidelberg (2014). 99
24. Ishiguro, T., Kiyomoto, S., Miyake, Y., Takagi, T.: Parallel Gauss Sieve algorithm: solving the SVP challenge over a 128-Dimensional ideal lattice. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 411–428. Springer, Heidelberg (2014). 85
25. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). 86
26. Lindner, R., Peikert, C.: Better key sizes (and attacks) for LWE-based encryption. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 319–339. Springer, Heidelberg (2011). 90, 92, 102, 103
27. Liu, M., Nguyen, P.Q.: Solving BDD by enumeration: an update. In: Dawson, E. (ed.) CT-RSA 2013. LNCS, vol. 7779, pp. 293–309. Springer, Heidelberg (2013). 90, 93, 102, 103
28. Lyubashevsky, V.: Fiat-Shamir with aborts: applications to lattice and factoring-based signatures. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 598–616. Springer, Heidelberg (2009). 88
29. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (2012). 88
30. Lyubashevsky, V., Peikert, C., Regev, O.: On Ideal Lattices and Learning with Errors over Rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010). 84
31. Misoczki, R., Barreto, P.S.L.M.: Compact McEliece keys from Goppa codes. In: Jacobson Jr, M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 376–392. Springer, Heidelberg (2009). 85
32. Oder, T., Pöppelmann, T., Güneysu, T.: Beyond ECDSA and RSA: Lattice-based digital signatures on constrained devices. In: DAC 2014 Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference, pp. 1–6. ACM (2014). https://www.sha.rub.de/media/attachments/files/2014/06/bliss_arm.pdf. 85

33. Pöppelmann, T., Ducas, L., Güneysu, T.: Enhanced lattice-based signatures on reconfigurable hardware. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 353–370. Springer, Heidelberg (2014). 85, 86
34. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) STOC 2005 Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of computing, pp. 84–93. ACM (2005). http://www.cims.nyu.edu/~regev/papers/qcrypto.pdf. 85
35. Roy, S.S., Vercauteren, F., Mentens, N., Chen, D.D., Verbauwhede, I.: Compact ring-LWE cryptoprocessor. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 371–391. Springer, Heidelberg (2014). 85
36. Schneider, M.: Sieving for shortest vectors in ideal lattices. In: Youssef, A., Nitaj, A., Hassanien, A.E. (eds.) AFRICACRYPT 2013. LNCS, vol. 7918, pp. 375–391. Springer, Heidelberg (2013). 85
37. Schnorr, C.-P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Math. Program. **66**, 181–199 (1994). http://www.csie.nuk.edu.tw/~cychen/Lattices/LatticeBasisReductionImprovedPracticalAlgorithmsandSolvingSubsetSumProblems.pdf. 92

A Decoding Attack

An approach for solving LWE that has not been considered in the original work [6] is the decoding attack. It is inspired by the nearest plane algorithm proposed by Babai [4]. For a given lattice basis and a given target vector, it returns a lattice vector that is relatively close to the target vector. Hence, improving the quality of the lattice basis yields a vector that is closer to the target vector. Lindner and Peikert [26] proposed the nearest planes algorithm, a generalization of the former that returns more than one vector and thereby enhances the previous algorithm with a trade-off between its runtime and the probability of returning the actual closest vector within the set of obtained vectors.

There is a continuous correspondence between the success probability of this attack and the Hermite delta. We follow the approach proposed by Lindner and Peikert [26] to predict this success probability. In short, they show how one can use the Geometric Series Assumption (GSA) in order to predict the length of the Gram-Schmidt vectors of a reduced basis, and this estimation in turn serves to predict the success probability of the attack. Together with an estimation of the running time of nearest plane – the authors propose 2^{-16} s – and the runtime estimation for basis reduction (see Eq. (2)), it is possible to predict the runtime and success probability of nearest planes.

Optimizing the trade-offs between the time spent on the attack and its success probability is not trivial, but simulations of the attack show that it is in most cases preferable to run multiple attacks with small success probabilities. This technique is called randomization and was investigated by Liu and Nguyen (see [27]), together with a further improvement called pruning. In comparison to the big improvement achieved with randomization, pruning leads only to a moderate speedup. The maximal speedup achieved in [27] is about 2^6, while randomization can reduce the cost by a factor of 2^{32}. Since it turned out that the decoding-attack is outperformed by other attacks by far (and pruning is furthermore very hard to analyze), we focused on the randomized version.

Briefly speaking, [26] provides the tools necessary to estimate the expected runtime of the attack for a given set of attack parameters, and [27] proposed to minimize the expected runtime (i.e. the time for one attack divided by the success probability of the attack). We applied this technique to our instance (cf. Table 2).