Practical Lattice-Based Cryptography: A Signature Scheme for Embedded Systems

Tim Güneysu[1*], Vadim Lyubashevsky[2†], and Thomas Pöppelmann[1*]

[1] Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany
[2] INRIA / ENS, Paris

Abstract. Nearly all of the currently used and well-tested signature schemes (e.g. RSA or DSA) are based either on the factoring assumption or the presumed intractability of the discrete logarithm problem. Further algorithmic advances on these problems may lead to the unpleasant situation that a large number of schemes have to be replaced with alternatives. In this work we present such an alternative – a signature scheme whose security is derived from the hardness of lattice problems. It is based on recent theoretical advances in lattice-based cryptography and is highly optimized for practicability and use in embedded systems. The public and secret keys are roughly 12000 and 2000 bits long, while the signature size is approximately 9000 bits for a security level of around 100 bits. The implementation results on reconfigurable hardware (Spartan/Virtex 6) are very promising and show that the scheme is scalable, has low area consumption, and even outperforms some classical schemes.

Keywords: Post-Quantum Cryptography, Lattice-Based Cryptography, Ideal Lattices, Signature Scheme Implementation, FPGA

1 Introduction

Due to the yet unpredictable but possibly imminent threat of the construction of a quantum computer, a number of alternative cryptosystems to RSA and ECC have gained significant attention during the last years. In particular, it has been widely accepted that relying solely on asymmetric cryptography based on the hardness of factoring or the (elliptic curve) discrete logarithm problem is certainly not sufficient in the long term [7]. This has been mainly due to the work of Shor [35], who demonstrated that both classes of problems can be efficiently attacked with quantum computers. As a consequence, first steps towards the required diversification and investigation of alternative fundamental problems and schemes have been taken. This has already led to efficient implementations of various schemes based on multivariate quadratic systems [5, 3] and the code-based McEliece cryptosystem [10, 36].

[*] This work was partially supported by European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.

[†] Work supported in part by the European Research Council.

Another promising alternative to number-theoretic constructions are lattice-based cryptosystems because they admit security proofs based on well-studied problems that currently cannot be solved by quantum algorithms. For a long time, however, lattice constructions have only been considered secure for inefficiently large parameters that are well beyond practicability[3] or were, like GGH [14] and NTRUSign [16], broken due to flaws in the ad-hoc design approach [31]. This has changed since the introduction of cyclic and ideal lattices [27] and related computationally hard problems like Ring-SIS [32, 23, 25] and Ring-LWE [26] which enabled the constructions of a great variety of theoretically elegant and efficient cryptographic primitives.

In this work we try to further close the gap between the advances in theoretical lattice-based cryptography and real-world implementation issues by constructing and implementing a provably-secure digital signature scheme based on ideal lattices. While maintaining the connection to hard ideal lattice problems we apply several performance optimizations for practicability that result in moderate signature and key sizes as well as performance suitable for embedded and hardware systems.

Digital Signatures and Related Work. Digital signatures are arguably the most used public-key cryptographic primitive in practical applications, and a lot of effort has gone into trying to construct such schemes from lattice assumptions. Due to the success of the NTRU encryption scheme, it was natural to try to design a signature scheme based on the same principles. Unlike the encryption scheme, however, the proposed NTRU signature scheme [19, 17] has been completely broken by Nguyen and Regev [31]. Provably-secure digital signatures were finally constructed in 2008, by Gentry, Peikert, and Vaikuntanathan [13], and, using different techniques, by Lyubashevsky and Micciancio [24]. The scheme in [13] was very inefficient in practice, with outputs and keys being megabytes long, while the scheme in [24] was only a one-time signature that required the use of Merkle trees to become a full signature scheme. The work of [24] was extended by Lyubashevsky [21, 22], who gave a construction of a full-fledged signature scheme whose keys and outputs are currently on the order of 15000 bits each, for an 80-bit security level. The work of [13] was also recently extended by Micciancio and Peikert [28], where the size of the signatures and keys is roughly $100,000$ bits.

Our Contribution. The main contribution of this work is the implementation of a digital signature scheme from [21, 22] optimized for embedded systems. In addition, we propose an improvement to the above-mentioned scheme which preserves the security proof, while lowering the signature size by approximately a factor of two. We demonstrate the practicability of our scheme by implementing a scalable and efficient signing and verification engine. For example, on the low-cost Xilinx Spartan-6 we are 1.5 times faster and use only half of the resources

[3] One notable exception is the NTRU public-key encryption scheme [18], which has essentially remained unbroken since its introduction.

of the optimized RSA implementation of Suzuki [39]. With more than 12000 signatures and over 14000 signature verifications per second, we can satisfy even high-speed demands using a Virtex-6 device.

Outline. The paper is structured as follows. First we give a short overview on our hardness assumption in Section 2 and then introduce the highly efficient and practical signature scheme in Section 3. Based on this description, we introduce our implementation and the hardware architecture of the signing and signature verification engine in Section 4 and analyze its performance on different FPGAs in Section 5. In Section 6 we summarize our contribution and present an outlook for future work.

2 Preliminaries

2.1 Notation

Throughout the paper, we will assume that n is an integer that is a power of 2, p is a prime number congruent to 1 modulo $2n$, and $\mathcal{R}^{p^n}$ is the ring $\mathbb{Z}_p[\mathbf{x}]/(\mathbf{x}^n + 1)$. Elements in $\mathcal{R}^{p^n}$ can be represented by polynomials of degree $n-1$ with coefficients in the range $[-(p-1)/2, (p-1)/2]$, and we will write $\mathcal{R}_k^{p^n}$ to be a subset of the ring $\mathcal{R}^{p^n}$ that consists of all polynomials with coefficients in the range $[-k, k]$. For a set S, we write $s \overset{\$}{\leftarrow} S$ to indicate that s is being chosen uniformly at random from S.

2.2 Hardness Assumption

In a particular version of the Ring-SIS problem, one is given an ordered pair of polynomials $(\mathbf{a}, \mathbf{t}) \in \mathcal{R}^{p^n} \times \mathcal{R}^{p^n}$ where $\mathbf{a}$ is chosen uniformly from $\mathcal{R}^{p^n}$ and $\mathbf{t} = \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$, where $\mathbf{s}_1$ and $\mathbf{s}_2$ are chosen uniformly from $\mathcal{R}_k^{p^n}$, and is asked to find an ordered pair $(\mathbf{s}'_1, \mathbf{s}'_2)$ such that $\mathbf{a}\mathbf{s}'_1 + \mathbf{s}'_2 = \mathbf{t}$. It can be shown that when $k > \sqrt{p}$, the solution is not unique and finding any one of them, for $\sqrt{p} < k \ll p$, was proven in [32, 23] to be as hard as solving worst-case lattice problems in ideal lattices. On the other hand, when $k < \sqrt{p}$, it can be shown that the only solution is $(\mathbf{s}_1, \mathbf{s}_2)$ with high probability, and there is no classical reduction known from worst-case lattice problems to finding this solution. In fact, this latter problem is a particular instance of the Ring-LWE problem. It was recently shown in [26] that if one chooses the $\mathbf{s}_i$ from a slightly different distribution (i.e., a Gaussian distribution instead of a uniform one), then solving the Ring-LWE problem (i.e., recovering the $\mathbf{s}_i$ when given $(\mathbf{a}, \mathbf{t})$) is as hard as solving worst-case lattice problems using a quantum algorithm. Furthermore, it was shown that solving the decision version of Ring-LWE, that is distinguishing ordered pairs $(\mathbf{a}, \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2)$ from uniformly random ones in $\mathcal{R}^{p^n} \times \mathcal{R}^{p^n}$, is still as hard as solving worst-case lattice problems.

In this paper, we implement our signature scheme based on the presumed hardness of the decision RING-LWE problem with particularly "aggressive" parameters. We define the $\mathbf{DCK}_{p,n}$ problem (Decisional Compact Knapsack problem) to be the problem of distinguishing between the uniform distribution over $\mathcal{R}^{p^n} \times \mathcal{R}^{p^n}$ and the distribution $(\mathbf{a}, \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2)$ where $\mathbf{a}$ is uniformly random in $\mathcal{R}^{p^n}$ and $\mathbf{s}_i$ are uniformly random in $\mathcal{R}_1^{p^n}$. As of now, there are no known algorithms that take advantage of the fact that the distribution of $\mathbf{s}_i$ is uniform (i.e., not Gaussian) and consists of only $-1/0/1$ coefficients[4], and so it is very reasonable to conjecture that this problem is still hard. In fact, this is essentially the assumption that the NTRU encryption scheme is based on. Due to lack of space, we direct the interested reader to Section 3 of the full version of [22] for a more in-depth discussion of the hardness of the different variants of the SIS and LWE problems.

2.3 Cryptographic Hash Function H with Range D_{32}^n

Our signature scheme uses a hash function, and it is quite important for us that the output of this function is of a particular form. The range of this function, D_{32}^n, for $n \geq 512$ consists of all polynomials of degree $n-1$ that have all zero coefficients except for at most 32 coefficients that are ± 1.

We denote by H the hash function that first maps $\{0,1\}^*$ to a 160-bit string and then *injectively* maps the resulting 160-bit string r to D_{32}^n via an efficient procedure we now describe. To map a 160-bit string into the range D_{32}^n for $n \geq 512$, we look at 5 bits of r at a time, and transforms them into a 16-digit string with at most one non-zero coefficient as follows: let $r_1r_2r_3r_4r_5$ be the five bits we are currently looking at. If r_1 is 0, then put a -1 in position number $r_2r_3r_4r_5$ (where we read the 4-digit string as a number between 0 and 15) of the 16-digit string. If r_1 is 1, then put a 1 in position $r_2r_3r_4r_5$. This converts a 160-bit string into a 512-digit string with at most 32 ± 1's.[5] We then convert the 512-bit string into a polynomial of degree at least 512 in the natural way by assigning the i^{th} coefficient of the polynomial the i^{th} bit of the bit-string. If the polynomial is of degree greater than 512, then all of its higher-order terms will be 0.

3 The Signature Scheme

In this section, we will present the lattice-based signature scheme whose hardware implementation we describe in Section 4. This scheme is a combination of

[4] For readers familiar with the Arora-Ge algorithm for solving LWE with small noise [2], we would like to point out that it is does not apply to our problem because this algorithm requires polynomially-many samples of the form $(\mathbf{a}_i, \mathbf{a}_i\mathbf{s} + \mathbf{e}_i)$, whereas in our problem, only one such sample is given.

[5] There is a more "compact" way to do it (see for example [11] for an algorithm that can convert a 160-bit string into a 512-digit one with at most 24 ± 1 coefficients), but the resulting transformation algorithm is quadratic rather than linear.

the schemes from [21] and [22] as well as an additional optimization that allows us to reduce the signature length by almost a factor of two. In [21], Lyubashevsky constructed a lattice-based signature scheme based on the hardness of the Ring-SIS problem, and this scheme was later improved in two ways [22]. The first improvement results in signatures that are asymptotically shorter, but unfortunately involves a somewhat more complicated rejection sampling algorithm during the singing procedure, involving sampling from the normal distribution and computing quotients to a very high precision, which would not be very well supported in hardware. We do not know whether the actual savings achieved in the signature length would justify the major slowdown incurred, and we do leave the possibility of efficiently implementing this rejection sampling algorithm to future work. The second improvement from [22], which we do use, shows how the size of the keys and the signature can be made significantly smaller by changing the assumption from Ring-SIS to Ring-LWE.

3.1 The Basic Signature Scheme

For ease of exposition, we first present the basic combination scheme of [21] and [22] in Figure 1, and sketch its security proof. Full security proofs are available in [21] and [22]. We then present our optimization in Sections 3.2 and 3.3.

```
Signing Key: $\mathbf{s}_1, \mathbf{s}_2 \overset{\$}{\leftarrow} \mathcal{R}_1^{p^n}$
Verification Key: $\mathbf{a} \overset{\$}{\leftarrow} \mathcal{R}^{p^n}, \mathbf{t} \leftarrow \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$
Cryptographic Hash Function: $\mathrm{H} : \{0,1\}^* \rightarrow D_{32}^n$

Sign($\mu, \mathbf{a}, \mathbf{s}_1, \mathbf{s}_2$)
  1: $\mathbf{y}_1, \mathbf{y}_2 \overset{\$}{\leftarrow} \mathcal{R}_k^{p^n}$
  2: $\mathbf{c} \leftarrow \mathrm{H}(\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2, \mu)$
  3: $\mathbf{z}_1 \leftarrow \mathbf{s}_1\mathbf{c} + \mathbf{y}_1, \mathbf{z}_2 \leftarrow \mathbf{s}_2\mathbf{c} + \mathbf{y}_2$
  4: if $\mathbf{z}_1$ or $\mathbf{z}_2 \notin \mathcal{R}_{k-32}^{p^n}$, then goto step 1
  5: output $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$

Verify($\mu, \mathbf{z}_1, \mathbf{z}_2, \mathbf{c}, \mathbf{a}, \mathbf{t}$)
  1: Accept iff
     $\mathbf{z}_1, \mathbf{z}_2 \in \mathcal{R}_{k-32}^{p^n}$ and
     $\mathbf{c} = \mathrm{H}(\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c}, \mu)$
```

Fig. 1. The Basic Signature Scheme

The secret keys are random polynomials $\mathbf{s}_1, \mathbf{s}_2 \overset{\$}{\leftarrow} \mathcal{R}_1^{p^n}$ and the public key is $(\mathbf{a}, \mathbf{t})$, where $\mathbf{a} \overset{\$}{\leftarrow} \mathcal{R}^{p^n}$ and $\mathbf{t} \leftarrow \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$. The parameter k in our scheme which first appears in line 1 of the signing algorithm controls the trade-off between the security and the runtime of our scheme. The smaller we take k, the more secure the scheme becomes (and the shorter the signatures get), but the time to sign will increase. We explain this as well as the choice of parameters below.

To sign a message μ, we pick two "masking" polynomials $\mathbf{y}_1, \mathbf{y}_2 \overset{\$}{\leftarrow} \mathcal{R}_k^{p^n}$ and compute $\mathbf{c} \leftarrow \mathrm{H}(\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2, \mu)$ and the potential signature $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$ where $\mathbf{z}_1 \leftarrow \mathbf{s}_1\mathbf{c} + \mathbf{y}_1, \mathbf{z}_2 \leftarrow \mathbf{s}_2\mathbf{c} + \mathbf{y}_2$[6]. But before sending the signature, we must perform

[6] We would like to draw the reader's attention to the fact that in step 3, reduction modulo p is not performed since all the polynomials involved have small coefficients.

a rejection-sampling step where we only send if $\mathbf{z}_1, \mathbf{z}_2$ are both in $\mathcal{R}^{p^n}_{k-32}$. This part is crucial for security and it is also where the size of k matters. If k is too small, then $\mathbf{z}_1, \mathbf{z}_2$ will almost never be in $\mathcal{R}^{p^n}_{k-32}$, whereas if its too big, it will be easy for the adversary to forge messages[7]. To verify the signature $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$, the verifier simply checks that $\mathbf{z}_1, \mathbf{z}_2 \in \mathcal{R}^{p^n}_{k-32}$ and that $\mathbf{c} = \mathrm{H}(\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c}, \mu)$.

Our security proof follows that in [22] except that it uses the rejection sampling algorithm from [21]. Given a random polynomial $\mathbf{a} \in \mathcal{R}^{p^n}$, we pick two polynomials $\mathbf{s}_1, \mathbf{s}_2 \overset{\$}{\leftarrow} \mathcal{R}^{p^n}_{k'}$ for a sufficiently large k' and return $(\mathbf{a} \in \mathcal{R}^{p^n}, \mathbf{t} = \mathbf{a}\mathbf{s}'_1 + \mathbf{s}'_2)$ as the public key. By the $\mathbf{DCK}_{p,n}$ assumption (and a standard hybrid argument), this looks like a valid public key (i.e., the adversary cannot tell that the $\mathbf{s}_i$ are chosen from $\mathcal{R}^{p^n}_{k'}$ rather than from $\mathcal{R}^{p^n}_1$). When the adversary gives us signature queries, we appropriately program the hash function outputs so that our signatures are valid even though we do not know a valid secret key (in fact, a valid secret key does not even exist). When the adversary successfully forges a new signature, we then use the "forking lemma" [34] to produce two signatures of the message μ, $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$ and $(\mathbf{z}'_1, \mathbf{z}'_2, \mathbf{c}')$, such that

$$\mathrm{H}(\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c}, \mu) = \mathrm{H}(\mathbf{a}\mathbf{z}'_1 + \mathbf{z}'_2 - \mathbf{t}\mathbf{c}', \mu), \tag{1}$$

which implies that

$$\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c} = \mathbf{a}\mathbf{z}'_1 + \mathbf{z}'_2 - \mathbf{t}\mathbf{c}' \tag{2}$$

and because we know that $\mathbf{t} = \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$, we can obtain

$$\mathbf{a}(\mathbf{z}_1 - \mathbf{c}\mathbf{s}_1 - \mathbf{z}'_1 + \mathbf{c}'\mathbf{s}_1) + (\mathbf{z}_2 - \mathbf{c}\mathbf{s}_2 - \mathbf{z}'_2 + \mathbf{c}'\mathbf{s}_2) = \mathbf{0}.$$

Because $\mathbf{z}_i, \mathbf{s}_i, \mathbf{c}$, and $\mathbf{c}'$ have small coefficients, we found two polynomials $\mathbf{u}_1, \mathbf{u}_2$ with small coefficients such that $\mathbf{a}\mathbf{u}_1 + \mathbf{u}_2 = 0$[8] By [22, Lemma 3.7], knowing such small $\mathbf{u}_i$ allows us to solve the $\mathbf{DCK}_{p,n}$ problem.

We now explain the trick that we use to lower the size of the signature as returned by the optimized scheme presented in Section 3.3. Notice that if Equation (2) does not hold exactly, but only approximately (i.e., $\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c} - (\mathbf{a}\mathbf{z}'_1 + \mathbf{z}'_2 - \mathbf{t}\mathbf{c}') = \mathbf{w}$ for some small polynomial $\mathbf{w}$), then we can still obtain small $\mathbf{u}_1, \mathbf{u}_2$ such that $\mathbf{a}\mathbf{u}_1 + \mathbf{u}_2 = 0$, except that the value of $\mathbf{u}_2$ will be larger by at most the norm of $\mathbf{w}$. Thus if $\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c} \approx \mathbf{a}\mathbf{z}'_1 + \mathbf{z}'_2 - \mathbf{t}\mathbf{c}'$, we will still be able to produce small $\mathbf{u}_1, \mathbf{u}_2$ such that $\mathbf{a}\mathbf{u}_1 + \mathbf{u}_2 = \mathbf{0}$. This could make us consider only sending $(\mathbf{z}_1, \mathbf{c})$ as a signature rather than $(\mathbf{z}_1, \mathbf{z}_2, \mathbf{c})$, and the proof will go through fine. The problem with this approach is that the verification algorithm will no longer work, because even though $\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c} \approx \mathbf{a}\mathbf{z}_1 - \mathbf{t}\mathbf{c}$, the output of the hash function H will be different. A way to go around the problem is to only evaluate H on the "high order bits" of the coefficients comprising the polynomial $\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c}$ which we could hope to be the same as those of the polynomial $\mathbf{a}\mathbf{z}_1 - \mathbf{t}\mathbf{c}$. But in practice, too many bits would be different (because of the carries caused by $\mathbf{z}_2$) for this to be a useful trick. What we do instead is

[7] The exact probability that $\mathbf{z}_1, \mathbf{z}_2$ will be in $\mathcal{R}^{p^n}_{k-32}$ is $\left(1 - \frac{64}{2k+1}\right)^{2n}$.

[8] It is also important that these polynomials are non-zero.

send $(\mathbf{z}_1, \mathbf{z}'_2, \mathbf{c})$ as the signature where $\mathbf{z}'_2$ only tells us the carries that $\mathbf{z}_2$ would have created in the high order bits in the sum of $\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2 - \mathbf{t}\mathbf{c}$, and so $\mathbf{z}'_2$ can be represented with much fewer bits than $\mathbf{z}_2$. In the next subsection, we explain exactly what we mean by "high-order bits" and give an algorithm that produces a $\mathbf{z}'_2$ from $\mathbf{z}_2$, and then provide an optimized version of the scheme in this section that uses the compression idea.

3.2 The Compression Algorithm

For every integer y in the range $\left[-\frac{p-1}{2}, \frac{p-1}{2}\right]$ and any positive integer k, y can be uniquely written as $y = y^{(1)}(2k+1) + y^{(0)}$ where $y^{(0)}$ is an integer in the range $[-k, k]$ and $y^{(1)} = \frac{y - y^{(0)}}{2k+1}$. Thus $y^{(0)}$ are the "lower-order" bits of y, and $y^{(1)}$ are the "higher-order" ones[9]. For a polynomial $\mathbf{y} = \mathbf{y}[0] + \mathbf{y}[1]\mathbf{x} + \ldots + \mathbf{y}[n-1]\mathbf{x}^{n-1} \in \mathcal{R}^{p^n}$, we define $\mathbf{y}^{(1)} = \mathbf{y}[0]^{(1)} + \mathbf{y}[1]^{(1)}\mathbf{x} + \ldots + \mathbf{y}[n-1]^{(1)}\mathbf{x}^{n-1}$ and $\mathbf{y}^{(0)} = \mathbf{y}[0]^{(0)} + \mathbf{y}[1]^{(0)}\mathbf{x} + \ldots + \mathbf{y}[n-1]^{(0)}\mathbf{x}^{n-1}$.

The Lemma below states that given two vectors $\mathbf{y}, \mathbf{z} \in \mathcal{R}^{p^n}$ where the coefficients of $\mathbf{z}$ are small, we can replace $\mathbf{z}$ by a much more compressed vector $\mathbf{z}'$ while keeping the higher order bits of $\mathbf{y} + \mathbf{z}$ and $\mathbf{y} + \mathbf{z}'$ the same. The algorithm that satisfies this lemma is presented in Figure 5 in Appendix A.

Lemma 3.1. *There exists a linear-time algorithm* Compress$(\mathbf{y}, \mathbf{z}, p, k)$ *that for any* p, n, k *where* $2nk/p > 1$ *takes as inputs* $\mathbf{y} \overset{\$}{\leftarrow} \mathcal{R}^{p^n}, \mathbf{z} \in \mathcal{R}_k^{p^n}$*, and with probability at least* $.98$ *(over the choices of* $\mathbf{y} \in \mathcal{R}^{p^n}$*), outputs a* $\mathbf{z}' \in \mathcal{R}_k^{p^n}$ *such that*

1. $(\mathbf{y} + \mathbf{z})^{(1)} = (\mathbf{y} + \mathbf{z}')^{(1)}$
2. $\mathbf{z}'$ *can be represented with only* $2n + \lceil \log(2k+1) \rceil \cdot \frac{6kn}{p}$ *bits.*

3.3 A Signature Scheme for Embedded Systems

We now present the version of the signature scheme that incorporates the compression idea from Section 3.2 (see Figure 2). We will use the following notation that is similar to the notation in Section 3.2: every polynomial $\mathbf{Y} \in \mathcal{R}^{p^n}$ can be written as

$$\mathbf{Y} = \mathbf{Y}^{(1)}(2(k-32)+1) + \mathbf{Y}^{(0)}$$

where $\mathbf{Y}^{(0)} \in \mathcal{R}_{k-32}^{p^n}$ and k corresponds to the k in the signature scheme in Figure 2. Notice that there is a bijection between polynomials $\mathbf{Y}$ and this representation $(\mathbf{Y}^{(1)}, \mathbf{Y}^{(0)})$ where

$$\mathbf{Y}^{(0)} = \mathbf{Y} \bmod (2(k-32)+1),$$

and

$$\mathbf{Y}^{(1)} = \frac{\mathbf{Y} - \mathbf{Y}^{(0)}}{2(k-32)+1}.$$

Intuitively, $\mathbf{Y}^{(1)}$ is comprised of the higher order bits of $\mathbf{Y}$.

[9] Note that these only roughly correspond to the notion of most and least significant bits.

Signing Key: $\mathbf{s}_1, \mathbf{s}_2 \xleftarrow{\$} \mathcal{R}_1^{p^n}$

Verification Key: $\mathbf{a} \xleftarrow{\$} \mathcal{R}^{p^n}, \mathbf{t} \leftarrow \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$

Cryptographic Hash Function: $\mathrm{H} : \{0,1\}^* \to D_{32}^n$

```
Sign($\mu, \mathbf{a}, \mathbf{s}_1, \mathbf{s}_2$)
1: $\mathbf{y}_1, \mathbf{y}_2 \xleftarrow{\$} \mathcal{R}_k^{p^n}$
2: $\mathbf{c} \leftarrow \mathrm{H}\left((\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2)^{(1)}, \mu\right)$
3: $\mathbf{z}_1 \leftarrow \mathbf{s}_1\mathbf{c} + \mathbf{y}_1, \mathbf{z}_2 \leftarrow \mathbf{s}_2\mathbf{c} + \mathbf{y}_2$
4: if $\mathbf{z}_1$ or $\mathbf{z}_2 \notin \mathcal{R}_{k-32}^{p^n}$, then goto step 1
5: $\mathbf{z}_2' \leftarrow$ Compress $(\mathbf{a}\mathbf{z}_1 - \mathbf{t}\mathbf{c}, \mathbf{z}_2, p, k-32)$
6: if $\mathbf{z}_2' = \perp$, then goto step 1
7: output $(\mathbf{z}_1, \mathbf{z}_2', \mathbf{c})$
```

```
Verify($\mu, \mathbf{z}_1, \mathbf{z}_2', \mathbf{c}, \mathbf{a}, \mathbf{t}$)
1: Accept iff
   $\mathbf{z}_1, \mathbf{z}_2' \in \mathcal{R}_{k-32}^{p^n}$ and
   $\mathbf{c} = \mathrm{H}\left((\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2' - \mathbf{t}\mathbf{c})^{(1)}, \mu\right)$
```

Fig. 2. Optimized Signature Scheme

The secret key in our scheme consists of two polynomials $\mathbf{s}_1, \mathbf{s}_2$ sampled uniformly from $\mathcal{R}_1^{p^n}$ and the public key consists of two polynomials $\mathbf{a} \xleftarrow{\$} \mathcal{R}^{p^n}$ and $\mathbf{t} = \mathbf{a}\mathbf{s}_1 + \mathbf{s}_2$. In step 1 of the signing algorithm, we choose the "masking polynomials" $\mathbf{y}_1, \mathbf{y}_2$ from $\mathcal{R}_k^{p^n}$. In step 2, we let $\mathbf{c}$ be the hash function value of the high order bits of $\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2$ and the message μ. In step 3, we compute $\mathbf{z}_1, \mathbf{z}_2$ and proceed only if they fall into a certain range. In step 5, we compress the value $\mathbf{z}_2$ using the compression algorithm implied in Lemma 3.1, and obtain a value $\mathbf{z}_2'$ such that $(\mathbf{a}\mathbf{z}_1 - \mathbf{t}\mathbf{c} + \mathbf{z}_2)^{(1)} = (\mathbf{a}\mathbf{z}_1 - \mathbf{t}\mathbf{c} + \mathbf{z}_2')^{(1)}$ and send $(\mathbf{z}_1, \mathbf{z}_2', \mathbf{c})$ as the signature of μ. The verification algorithm checks whether $\mathbf{z}_1, \mathbf{z}_2'$ are in $\mathcal{R}_{k-32}^{p^n}$ and that $\mathbf{c} = \mathrm{H}\left((\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2' - \mathbf{t}\mathbf{c})^{(1)}, \mu\right)$.

The running time of the signature algorithm depends on the relationship of the parameter k with the parameter p. The larger the k, the more chance that $\mathbf{z}_1$ and $\mathbf{z}_2$ will be in $\mathcal{R}_{k-32}^{p^n}$ in step 4 of the signing algorithm, but the easier the signature will be to forge. Thus it is prudent to set k as small as possible while keeping the running time reasonable.

3.4 Concrete Instantiation

We now give some concrete instantiations of our signature scheme from Figure 2. The security of the scheme depends on two things: the hardness of the underlying $\mathbf{DCK}_{p,n}$ problem and the hardness of finding pre-images in the random oracle H[10]. For simplicity, we fixed the output of the random oracle to 160 bits and so finding pre-images is 160 bits hard. Judging the security of the lattice problem, on the other hand, is notoriously more difficult. For this part, we rely on the extensive experiments performed by Gama and Nguyen [12] and Chen and Nguyen [8] to determine the hardness of lattice reductions for certain classes of

[10] It is generally considered folklore that for obtaining signatures with λ bits of security using the Fiat-Shamir transform, one only needs random oracles that output λ bits (i.e., collision-resistance is not a requirement). While finding collisions in the random oracle does allow the *valid* signer to produce two distinct messages that have the same signature, this does not constitute a break.

Table 1. Signature Scheme Parameters

Aspect	**Set I**	**Set II**
n	512	1024
p	8383489	16760833
k	2^{14}	2^{15}
Approximate signature bit size	$8,950$	$18,800$
Approximate secret key bit size	$1,620$	$3,250$
Approximate public key bit size	$11,800$	$25,000$
Expected number of repetitions	7	7
Approximate root Hermite factor	1.0066	1.0035
Equivalent symmetric security in bits	≈ 100	> 256

lattices. The lattices that were used in the experiments of [12] were a little different than ours, but we believe that barring some unforeseen weakness due to the added algebraic structure of our lattices and the parameters, the results should be quite similar. We consider it somewhat unlikely that the algebraic structure causes any weaknesses since for certain parameters, our signature scheme is as hard as Ring-LWE (which has a quantum reduction from worst-case lattice problems [26]), but we do encourage cryptanalysis for our particular parameters because they are somewhat smaller than what is required for the worst-case to average-case reduction in [38, 26] to go through.

The methodology for choosing our parameters is the same as in [22], and so we direct the interested reader to that paper for a more thorough discussion. In short, one needs to make sure that the length of the secret key $[\mathbf{s}_1|\mathbf{s}_2]$ as a vector is not too much smaller than $\sqrt{p}$ and that the allowable length of the signature vector, which depends on k, is not much larger than $\sqrt{p}$. Using these quantities, one can perform the now-standard calculation of the "root Hermite factor" that lattice reduction algorithms must achieve in order to break the scheme (see [12, 29, 22] for examples of how this is done). According to experiments in [12, 8] a factor of 1.01 is achievable now, a factor of 1.007 seems to have around 80 bits of security, and a factor of 1.005 has more than 256-bit security. In Figure 1, we present two sets of parameters. According to the aforementioned methodology, the first has somewhere around 100 bits of security, while the second has more than 256.

We will now explain how the signature, secret key, and public key sizes are calculated. We will use the concrete numbers from set I as example. The signature size is calculated by summing the bit lengths of $\mathbf{z}_1, \mathbf{z}_2'$, and $\mathbf{c}$. Since $\mathbf{z}_1$ is in $\mathcal{R}_{k-32}^{p^n}$, it can be represented by $n\lceil\log(2(k-32)+1)\rceil \leq n\log k + n = 7680$ bits. From Lemma 3.1, we know that $\mathbf{z}_2'$ can be represented with $2n + \lceil\log(2(k-32)+1)\rceil \cdot \frac{6(k-32)n}{p} \leq 2n + 6\log(2k) = 1114$ bits. And $\mathbf{c}$ can be represented with 160 bits, for a total signature size of 8954 bits. The secret key consists of polynomials $\mathbf{s}_1, \mathbf{s}_2 \in \mathcal{R}_1^{p^n}$, and so they can be represented with $2\lceil n\log(3)\rceil = 1624$ bits, but a simpler representation can be used that requires 2048 bits. The public key consists of the polynomials $(\mathbf{a}, \mathbf{t})$, but the polynomial $\mathbf{a}$ does not need to be unique for every secret key, and can in fact be some randomness that is agreed upon by everyone who uses the scheme. Thus the public key can be just $\mathbf{t}$, which can be represented using $\lceil n \log p \rceil = 11776$ bits.

We point out that even though the signature and key sizes are larger than in some number theory based schemes, the signature scheme in Figure 2 is quite efficient, (in software and in hardware), with all operations taking quasi-linear time, as opposed to at least quadratic time for number-theory based schemes. The most expensive operation of the signing algorithm is in step 2 where we need to compute $\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2$, which also could be done in quasilinear time using FFT. In step 3, we also need to perform polynomial multiplication, but because $\mathbf{c}$ is a very sparse polynomial with only 32 non-zero entries, this can be performed with just 32 vector additions. And there is no multiplication needed in step 5 because $\mathbf{a}\mathbf{z}_1 - \mathbf{t}\mathbf{c} = \mathbf{a}\mathbf{y}_1 + \mathbf{y}_2 - \mathbf{z}_2$.

4 Implementation

In this section we provide a detailed description of our FPGA implementation of the signature scheme's signing and verification procedures for parameter set **I** with about 100 bits of equivalent symmetric security. In order to improve the speed and resource consumption on the FPGA, we utilize internal block memories (BRAM) and DSP hardcores spanning over three clock domains. We designed dedicated implementations of the signing and verification operation that work with externally generated keys.

Roughly speaking, the signing engine is composed out of a scalable amount of area-efficient polynomial multipliers to compute $\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2$. Fresh randomness for $\mathbf{y}_1, \mathbf{y}_2$ is supplied each run by a random number generator (in this prototype implementation an LFSR). To ensure a steady supply of fresh polynomials from the multiplier for the subsequent parts of the design and the actual signing operation, we have included a buffer of a configurable size that pre-stores pairs $(\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2, \mathbf{y}_1 || \mathbf{y}_2)$. The hash function H saves its state after the message has been hashed and thus prevents rehashing of the (presumably long) message in each new rejection-sampling step. The sparse multiplication of $\mathbf{s}\mathbf{c}$ works coefficient-wise and thus allows immediate testing for the rejection condition. If an out-of-bound coefficient occurs (line 4 and 6 of Figure 2), the multiplication and compression is immediately interrupted and a new polynomial pair is retrieved from the buffer. For the verification engine, we rely on the polynomial multiplier used to compute $\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2$ twice as we compute $\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2^{'}$ first, maintain the internal state and therefore add $\mathbf{t}(-\mathbf{c})$ in a second round to produce the input for the hash function. When signatures are fed into or returned by both engines, they are encoded in order to meet the signature size (see Lemma A.2 for a detailed algorithm).

4.1 Message Signing

The detailed top-level design of the signing engine is depicted in Figure 3. The computation of $\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2$ is implemented in clock domain `(1)` and carried out by a number of `PolyMul` units (three units are shown in the depicted setup). The BRAMs storing the initial parameters $\mathbf{y} = \mathbf{y}_1||\mathbf{y}_2$ are refilled by a random number generator (`RNG`) running independently in clock domain `(3)` and the constant polynomial $\mathbf{a}$ is loaded during device initialization. When a `PolyMul` unit has finished the computation of $\mathbf{r} = \mathbf{a}\mathbf{y}_1 + \mathbf{y}_2$, it requests exclusive access to the `Buffer` and stores $\mathbf{r}$ and $\mathbf{y}$ when free space is available. Internally the `Buffer` consists of the two configurable FIFOs `FIFO(`$\mathbf{r}$`)` and `FIFO(`$\mathbf{y}$`)`. As all operations in clock domain `(1)` and `(3)` are independent of the secret key or message, they are triggered when space in the `Buffer` becomes available. As described in Section 3.4, the polynomial $\mathbf{r} = \mathbf{a}\mathbf{y}_1 + \mathbf{y}_2$ is needed as input to the hashing as well as for the compression components and is thus stored in `BRAM_BUF`($\mathbf{r}$) while the coefficients of $\mathbf{y}_1, \mathbf{y}_2$ are only needed once and therefore taken directly out of the FIFOs.

When a signature for a message stored in `FIFO(m)` is requested, the sampling-rejection is repeated in clock domain `(2)` until a valid signature has been written into `FIFO(`σ`)`. The message to be signed is first hashed and its internal state saved. Therefore, it is only necessary to rehash $\mathbf{r}$ in case the computed signature is rejected (but not the message again). When the hash $\mathbf{c}$ is ready, the `Compression` component is started. In this component, the values $\mathbf{z}_1 = \mathbf{s}_1\mathbf{c}+\mathbf{y}_1$ and $\mathbf{z}_2 = \mathbf{s}_2\mathbf{c}+\mathbf{y}_2$ are computed column/coefficient-wise with a Comba-style sparse multiplier [9] followed by an addition so that coefficients of $\mathbf{z}_1$ or $\mathbf{z}_2$ are sequentially generated. Rejection-sampling is directly performed on these coefficients and the whole pair $(\mathbf{r}, \mathbf{y})$ is rejected once a coefficient is encountered that is not in the desired range. The secret key $\mathbf{s} = \mathbf{s}_1||\mathbf{s}_2$ is stored in the block RAM `BRAM(s)` which can be initialized during device initialization or set from the outside during runtime. The whole signature $\sigma = (\mathbf{z_1}, \mathbf{z_2}^{'}, \mathbf{c})$ is encoded by the `Encoder` component in order to meet the desired signature size (max. 8954 bits) and then written into the FIFO `FIFO(`σ`)`. The usage of FIFOs and BRAMs as I/O port allows easy integration of our engine into other designs and provides the ability for clock domain separation.

Polynomial Multiplication The most time-consuming operation of the signature scheme is the polynomial multiplication $\mathbf{a}\cdot\mathbf{y}_1$ (with the addition of $\mathbf{y}_2$ being rather simple). Recall that $\mathbf{a} \in \mathcal{R}^{p^n}$ has 512 23-bit wide coefficients and that $\mathbf{y}_1 \in \mathcal{R}_k^{p^n}$ consists of 512 16-bit wide coefficients. We are aware that the selected schoolbook algorithm (complexity of $\mathcal{O}(n^2)$) is theoretically inferior compared to Karatsuba [20] ($\mathcal{O}(n^{\log 3})$) or the FFT [30] ($\mathcal{O}(n \log n)$). However, its regular structure and iterative nature allows very high clock frequencies and an area efficient implementation on very small and cheap devices. The polynomial reduction with $f = x^n + 1$ is performed in place which leads to the negacyclic

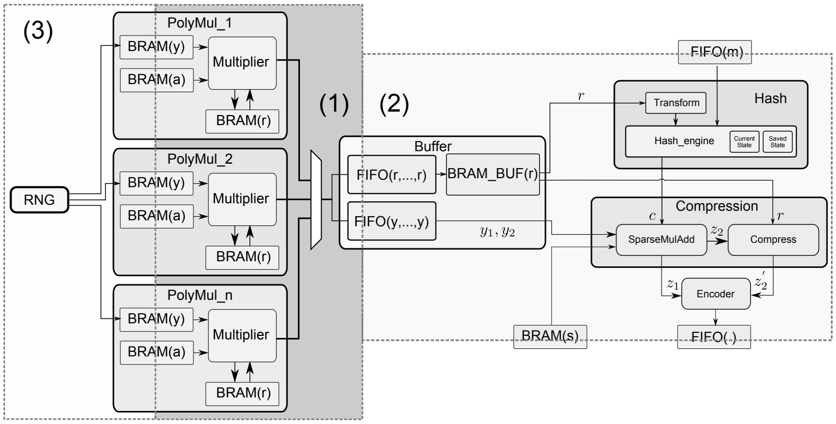


Fig. 3. Block structure of the implemented signing engine. The three different clock domains are denoted by (1), (2), (3).

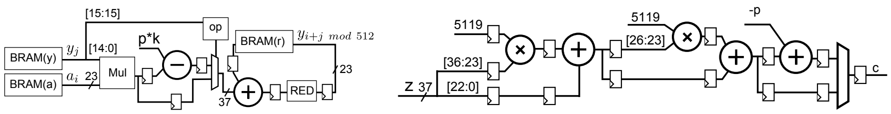
convolution

$$\mathbf{r} = \sum_{i=0}^{511}\sum_{j=0}^{511}(-1)^{\left\lfloor \frac{i+j}{n} \right\rfloor}\mathbf{a}_i\mathbf{y}_j x^{i+j \mod 512}$$

of $\mathbf{a}$ and $\mathbf{y_1}$. The data path for the arithmetic is depicted in Figure 4(a). The computation of $\mathbf{a}_i\mathbf{y}_j$ is realized in a multiplication core. We avoid dealing with signed values by determining the sign of the value added to the intermediate coefficient from the MSB sign bit of $\mathbf{y}_j$ and if a reduction modular $x^n + 1$ is necessary. As all coefficients of $\mathbf{a}$ are stored in the range $[0, p-1]$ they do not affect the sign of the result. Modular reduction (see Figure 4(b)) by $p = 8383489$ is implemented based on the idea of Solinas [37] as $2^{23} \mod 8383489 = 5119$ is very small. For the modular addition of y_2 the multiplier's arithmetic pipeline is reused in a final round in which the output of `BRAM(a)` is being set to 1 and the coefficients of $\mathbf{y}_2$ are being fed into the `BRAM(y)` port. Each `PolyMul` unit also acts as an additional buffer as it can hold one complete result of $\mathbf{r}$ in its internal temporary BRAM and thus reduces latency further in a scenario with precomputation. All in all, one `PolyMul` unit requires 204 slices, 3 BRAMs, 4 DSPs and is able to generate approx. 1130 pairs of (r,y) per second at a clock frequency of 300 MHz on a Spartan-6.

4.2 Signature Verification

In the previous sections we discussed the details of the signing algorithm. When dealing with the signature verification, we can reuse most of the previously described components. In particular, the `PolyMul` component only needs a slight modification in order to compute $\mathbf{a}\mathbf{z}_1 + \mathbf{z}_2' - \mathbf{t}\mathbf{c}$ which allows efficient resource sharing for both operation. It is easy to see that we can split the computation



(a) Pipelined data-path of `PolyMul`. (b) DSP based modular reduction with $p = 8383489$.

Fig. 4. Implementation of `PolyMul`.

of the input to the hash instantiation into $\mathbf{t}_1 = \mathbf{a}\mathbf{z}_1 + \mathbf{z}_2^{'}$, $\mathbf{t}_2 = \mathbf{t}(-\mathbf{c}) + 0$ and $\mathbf{t} = \mathbf{t}_1 + \mathbf{t}_2$. We see that the first equation can be performed by the `PolyMul` core as $\mathbf{a} \in \mathcal{R}^{p^n}$ and $\mathbf{z}_1, \mathbf{z}_2^{'} \in \mathcal{R}_k^{p^n}$. The same is true for the second equation with $\mathbf{t}$ being in $\mathcal{R}^{p^n}$ and the inverted c being also in the range $[-k, k]$ ($\mathbf{c}$ is even much smaller). The only problem is the final addition of the last equation as a third call to `PolyMul` would not work due to the fact that both inputs are from $\mathcal{R}^{p^n}$ which `PolyMul` cannot handle. However, note that `PolyMul` stores the intermediate state of the schoolbook multiplication in `BRAM`($\mathbf{r}$) but initializes the block RAM with zero coefficients prior to the next computation of a new $\mathbf{a}\mathbf{y}_1 + \mathbf{y}_2$. As a consequence, `PolyMul` supports a special flag that triggers a multiply-accumulate behavior in which the content of `BRAM`($\mathbf{r}$) is preserved after a full run of the schoolbook multiplication ($\mathbf{a}\mathbf{y}_1$) and an addition of $\mathbf{y}_2$. Therefore, the intermediate values $\mathbf{t}_1$ and $\mathbf{t}_2$ are summed up in `BRAM`($\mathbf{r}$) and we do not need the final addition. This enabled us to design a verification engine that performs its arithmetic operations with just two runs of the `PolyMul` core.

5 Results and Comparison

All presented results below were obtained after post-place-and-route (PAR) and were generated with Xilinx ISE 13.3. We have implemented the signing and verification engine (parameter set **I**, buffer of size one) on two devices of the low-cost Spartan-6 device family and on one high-speed Virtex-6 (all speed grade -3). Detailed information regarding performance and resources consumption is given in Table 2 and Table 3, respectively. For the larger devices we instantiate multiple distinct engines as the `Compression` and `Hash` components become the bottleneck when a certain amount of `PolyMul` components are instantiated. Note also that our implementation is small enough to fit the signing (two `PolyMul` units) or verification engine on the second-smallest Spartan-6 LX9.

When comparing our results to other work as given in Table 4, we conservatively assume that RSA signatures (one modular exponentiation) with a key size of 1024 bit and ECDSA signatures (one point multiplication) with a key size of 160 bit are comparable to our scheme in terms of security (see Section 3.4 for details on the parameters). In comparison with RSA, our implementation on the low-cost Spartan-6 is 1.5 times faster than the high-speed implementation of Suzuki [39] – that still needs twice as many device resources and runs on the more expensive Virtex-4 device. Note however, that ECC over binary curves is

Table 2. Performance of signing and verification for different design targets.

	Aspect	Spartan-6 LX16	Spartan-6 LX100	Virtex-6 LX130
Signing	Engines/Multiplier	1/7	4/9	9/8
	Total Multipliers	7	36	72
	Max. freq. domain (1)	270 MHz	250 MHz	416 MHz
	Max. freq. domain (2)	162 MHz	154 MHz	204 MHz
	Throughput Ops/s	931	4284	12627
Verification	Independent engines	2	14	20
	Max. frequency domain (1)	272 MHz	273 MHz	402 MHz
	Max. frequency domain (2)	158 MHz	103 MHz	156 MHz
	Throughput Ops/s	998	7015	14580

Table 3. Resource consumption of signing and verification for different design targets.

	Aspect	Spartan 6 LX16	Spartan 6 LX100	Virtex 6 LX130
Signing	Slices	2273	11006	19896
	LUT/FF	7465/8993	30854/34108	67027/95511
	18K BRAM	29.5	138	234
	DPS48A1	28	144	216
Verification	Slices	2263	14649	18998
	LUT/FF	6225/6663	44727/45094	61360/57903
	18K BRAM	15	90	120
	DPS48A1	8	56	60

very well suited for hardware and even implementations on old FPGAs like the Virtex-2 [1] are faster than our lattice-based scheme. For the NTRUSign lattice-based signature scheme (introduced in [16] and broken by Nguyen [31]) and the XMSS [6] hash-based signature scheme we are not aware of any implementation results for FPGAs. Hardware implementations of Multivariate Quadratic (MQ) cryptosystems [5, 3] show that these schemes are faster (factor 2-50) than ECC but also suffer from impractical key sizes for the private and public key (e.g., 80 Kb for Unbalanced Oil and Vinegar (UOV)) [33]. While implementations of the McEliece encryption scheme offer good performance [10, 36] the only implementation of a code based signature scheme [4] is extremely slow with a runtime of 830 ms for signing.

6 Conclusion

In this paper we presented a provably secure lattice based digital signature scheme and its implementation on a wide scale of reconfigurable hardware. With moderate resource requirements and more than 12,000 and 14,000 signing and verification operations per second on a Virtex-6 FPGA, our prototype imple-

Table 4. Implementation results for comparable signature schemes (signing).

Operation	**Algorithm**	**Device**	**Resources**	**Ops/s**
RSA Signature [39]	RSA-1024; private key	XC4VFX12-10	3937 LS/ 17 DSPs	548
ECDSA [15]	NIST-P224; point mult.	XC4VFX12-12	1580 LS/ 26 DSPs	2,739
ECDSA [1]	NIST-B163; point mult.	XC2V2000	8300 LUTs/ 7 BRAMs	24,390
UOV-Signature [5]	UOV(60,20)	XC5VLX50-3	13437 LUTs	170,940

mentation even outperforms classical and alternative cryptosystems in terms of signature size and performance.

Future work consists of optimization of the rejection-sampling steps as well as evaluation of different polynomial multiplication methods like the FFT. We also plan to investigate practicability of the signature scheme on other platforms like microcontrollers or graphic cards.

References

1. B. Ansari and M. Hasan. High performance architecture of elliptic curve scalar multiplication. *CACR Research Report*, 1:2006, 2006.
2. S. Arora and R. Ge. New algorithms for learning in presence of errors. In *ICALP (1)*, pages 403–415, 2011.
3. S. Balasubramanian, H. Carter, A. Bogdanov, A. Rupp, and J. Ding. Fast multivariate signature generation in hardware: The case of rainbow. In *Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008.*, pages 25–30. IEEE, 2008.
4. J. Beuchat, N. Sendrier, A. Tisserand, G. Villard, et al. FPGA implementation of a recently published signature scheme. *Rapport de Recherche RR LIP 2004-14*, 2004.
5. A. Bogdanov, T. Eisenbarth, A. Rupp, and C. Wolf. Time-area optimized public-key engines: MQ-cryptosystems as replacement for elliptic curves? In *Proceedings of the 10th international workshop on Cryptographic Hardware and Embedded Systems*, CHES '08, pages 45–61, Berlin, Heidelberg, 2008. Springer-Verlag.
6. J. Buchmann, E. Dahmen, and A. Hülsing. Xmss - a practical forward secure signature scheme based on minimal security assumptions. In B.-Y. Yang, editor, *PQCrypto*, volume 7071 of *Lecture Notes in Computer Science*, pages 117–129. Springer, 2011.
7. J. Buchmann, A. May, and U. Vollmer. Perspectives for cryptographic long-term security. *Commun. ACM*, 49:50–55, September 2006.
8. Y. Chen and P. Q. Nguyen. Bkz 2.0: Better lattice security estimates. In *ASIACRYPT*, pages 1–20, 2011.
9. P. G. Comba. Exponentiation cryptosystems on the ibm pc. *IBM Syst. J.*, 29:526–538, October 1990.

10. T. Eisenbarth, T. Güneysu, S. Heyse, and C. Paar. Microeliece: Mceliece for embedded devices. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '09, pages 49–64, Berlin, Heidelberg, 2009. Springer-Verlag.
11. J.-B. Fischer and J. Stern. An efficient pseudo-random generator provably as secure as syndrome decoding. In *EUROCRYPT*, pages 245–255, 1996.
12. N. Gama and P. Q. Nguyen. Predicting lattice reduction. In *EUROCRYPT*, pages 31–51, 2008.
13. C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.
14. O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. In *CRYPTO*, pages 112–131, 1997.
15. T. Güneysu and C. Paar. Ultra high performance ecc over nist primes on commercial FPGAs. *Cryptographic Hardware and Embedded Systems–CHES 2008*, pages 62–78, 2008.
16. J. Hoffstein, N. Howgrave-Graham, J. Pipher, J. Silverman, and W. Whyte. NTRUsign: Digital signatures using the ntru lattice. In *Proceedings of the 2003 RSA conference on The cryptographers' track*, pages 122–140. Springer-Verlag, 2003.
17. J. Hoffstein, N. Howgrave-Graham, J. Pipher, J. H. Silverman, and W. Whyte. Ntrusign: Digital signatures using the ntru lattice. In *CT-RSA*, pages 122–140, 2003.
18. J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In *ANTS*, pages 267–288, 1998.
19. J. Hoffstein, J. Pipher, and J. H. Silverman. NSS: An NTRU lattice-based signature scheme. In *EUROCRYPT*, pages 211–228, 2001.
20. A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. In *Soviet physics doklady*, volume 7, page 595, 1963.
21. V. Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In *ASIACRYPT*, pages 598–616, 2009.
22. V. Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT*, 2012. Full version at `http://eprint.iacr.org/2011/537`.
23. V. Lyubashevsky and D. Micciancio. Generalized compact knapsacks are collision resistant. In *ICALP (2)*, pages 144–155, 2006.
24. V. Lyubashevsky and D. Micciancio. Asymptotically efficient lattice-based digital signatures. In *TCC*, pages 37–54, 2008.
25. V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. SWIFFT: A modest proposal for FFT hashing. In *FSE*, pages 54–72, 2008.
26. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, pages 1–23, 2010.
27. D. Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Computational Complexity*, 16(4):365–411, 2007.
28. D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, 2012. Full version at `http://eprint.iacr.org/2011/501`.
29. D. Micciancio and O. Regev. Lattice-based cryptography. In D. J. Bernstein, J. Buchmann, and E. Dahmen, editors, *Chapter in Post-quantum Cryptography*, pages 147–191. Springer, 2008.
30. R. T. Moenck. Practical fast polynomial multiplication. In *Proceedings of the third ACM symposium on Symbolic and algebraic computation*, SYMSAC '76, pages 136–148, New York, NY, USA, 1976. ACM.

31. P. Nguyen and O. Regev. Learning a parallelepiped: Cryptanalysis of ggh and ntru signatures. *Journal of Cryptology*, 22:139–160, 2009.
32. C. Peikert and A. Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In *TCC*, pages 145–166, 2006.
33. A. Petzoldt, E. Thomae, S. Bulygin, and C. Wolf. Small public keys and fast verification for multivariate quadratic public key systems. In *Proceedings of the 13th international conference on Cryptographic hardware and embedded systems*, CHES'11, pages 475–490, Berlin, Heidelberg, 2011. Springer-Verlag.
34. D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3):361–396, 2000.
35. P. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE, 1994.
36. A. Shoufan, T. Wink, H. Molter, S. Huss, and E. Kohnert. A novel cryptoprocessor architecture for the mceliece public-key cryptosystem. *Computers, IEEE Transactions on*, 59(11):1533 –1546, nov. 2010.
37. J. Solinas. *Generalized mersenne numbers.* Faculty of Mathematics, University of Waterloo, 1999.
38. D. Stehlé, R. Steinfeld, K. Tanaka, and K. Xagawa. Efficient public key encryption based on ideal lattices. In *ASIACRYPT*, pages 617–635, 2009.
39. D. Suzuki. How to maximize the potential of FPGA resources for modular exponentiation. *Cryptographic Hardware and Embedded Systems-CHES 2007*, pages 272–288, 2007.

A Compression Algorithm

In this section we present our compression algorithm. For two vectors $\mathbf{y}, \mathbf{z}$, the algorithm first checks whether the coefficient $\mathbf{y}[i]$ of $\mathbf{y}$ is greater than $(p-1)/2-k$ in absolute value. If it is, then there is a possibility that $\mathbf{y}[i]+\mathbf{z}[i]$ will need to be reduced modulo p and in this case we do not compress $\mathbf{z}[i]$. Ideally there should not be many such elements, and we can show that for the parameters used in the signature scheme, there will be at most 6 (out of n) with high probability. It's possible to set the parameters so that there are no such elements, but this decreases the efficiency and is not worth the very slight savings in the compression.

Assuming that $\mathbf{y}[i]$ is in the range where $\mathbf{z}[i]$ can be compressed, we assign the value of k to $\mathbf{z}'[i]$ if $\mathbf{y}[i]^{(0)}+\mathbf{z}[i]>k$, assign $-k$ if $\mathbf{y}[i]^{(0)}+\mathbf{z}[i]<-k$, and 0 otherwise. We now move on to proving that the algorithm satisfies Lemma 3.1.

Lemma A.1. *Item 1 of Lemma 3.1 holds.*

Proof. Given in the full version of this paper.

Lemma A.2. *Item 2 of Lemma 3.1 holds.*

Proof. If $\mathbf{z}[i]'=0$, we represent it with the bit string $'00'$. If $\mathbf{z}[i]'=k$, we represent it with the bit string $'01'$. $\mathbf{z}[i]'=-k$, we represent it with the bit string $'10'$. If $\mathbf{z}[i]'=\mathbf{z}[i]$ (in other words, it is uncompressed), we represent it

```
Compress($\mathbf{y}, \mathbf{z}, p, k$)
 1: $uncompressed \leftarrow 0$
 2: for i=1 to n do
 3:   if $|\mathbf{y}[i]| > \frac{p-1}{2} - k$ then
 4:     $\mathbf{z}'[i] \leftarrow \mathbf{z}[i]$
 5:     $uncompressed \leftarrow uncompressed + 1$
 6:   else
 7:     write $\mathbf{y}[i] = \mathbf{y}[i]^{(1)}(2k+1) + \mathbf{y}[i]^{(0)}$ where $-k \leq \mathbf{y}[i]^{(0)} \leq k$
 8:     if $\mathbf{y}[i]^{(0)} + \mathbf{z}[i] > k$ then
 9:       $\mathbf{z}'[i] \leftarrow k$
10:     else if $\mathbf{y}[i]^{(0)} + \mathbf{z}[i] < -k$ then
11:       $\mathbf{z}'[i] \leftarrow -k$
12:     else
13:       $\mathbf{z}'[i] \leftarrow 0$
14:     end if
15:   end if
16: end for
17: if $uncompressed \leq \frac{6kn}{p}$ then
18:   return $\mathbf{z}'$
19: else
20:   return $\perp$
21: end if
```

Fig. 5. The Compression Algorithm

with the string $'11\mathbf{z}[i]'$ where $\mathbf{z}[i]$ can be represented by $2\log k$ bits (the $'11'$ is necessary to signify that the following $\log 2k$ bits represent an uncompressed value). Thus uncompressed values use $2 + \log 2k$ bits and the other values use just 2 bits. Since there are at most $6kn/p$ uncompressed values, the maximum number of bits that are needed is

$$(2 + \log 2k) \cdot \frac{6kn}{p} + 2\left(n - \frac{6kn}{p}\right) = 2n + \lceil \log(2k+1) \rceil \cdot \frac{6kn}{p}.$$

□

Finally, we show that if $\mathbf{y}$ is uniformly distributed in $\mathcal{R}^{p^n}$, then with probability at least .98, the algorithm will not have more than 6 uncompressed elements.

Lemma A.3. *If* $\mathbf{y}$ *is uniformly distributed modulo* p *and* $2nk/p \geq 1$*, then the compression algorithm outputs* $\perp$ *with probability less than* 2%.

Proof. The probability that the inequality in line 3 will be true is exactly $2k/p$. Thus the value of the "*uncompressed*$''$ variable follows the binomial distribution with n samples each being 1 with probability $2k/p$. Since we will always set $n >> 2k/p$, this distribution can be approximated by the Poisson distribution with $\lambda = 2nk/p$. If $\lambda \geq 1$ then the probability that the number of occurrences is greater than 3λ is at most 2% (this occurs for $\lambda = 1$). Since we output $\perp$ when *uncompressed* $> 6kn/p = 3\lambda$, it is output with probability at most 2%. □