

# Towards Practical Lattice-Based Public-Key Encryption on Reconfigurable Hardware

Thomas Pöppelmann<sup>(✉)</sup> and Tim Güneysu

Horst Görtz Institute for IT-Security, Ruhr University Bochum, Bochum, Germany  
thomas.poepelmann@rub.de

**Abstract.** With this work we provide further evidence that lattice-based cryptography is a promising and efficient alternative to secure embedded applications. So far it is known for solid security reductions but implementations of specific instances have often been reported to be too complex beyond any practicability. In this work, we present an efficient and scalable micro-code engine for Ring-LWE encryption that combines polynomial multiplication based on the Number Theoretic Transform (NTT), polynomial addition, subtraction, and Gaussian sampling in a single unit. This unit can encrypt and decrypt a block in 26.19  $\mu\text{s}$  and 16.80  $\mu\text{s}$  on a Virtex-6 LX75T FPGA, respectively – at moderate resource requirements of about 1506 slices and a few block RAMs. Additionally, we provide solutions for several practical issues with Ring-LWE encryption, including the reduction of ciphertext expansion, error rate and constant-time operation. We hope that this contribution helps to pave the way for the deployment of ideal lattice-based encryption in future real-world systems.

**Keywords:** Ideal lattices · Ring-LWE · FPGA implementation

## 1 Introduction and Motivation

Resistance against quantum computers and long term security has been an issue that cryptographers are trying so solve for some time [12]. However, while quite a few alternative schemes and problem classes are available, not many of them received the attention both from cryptanalysts and implementers that would be needed to establish the confidence and efficiency for their deployment in real-world systems. In the field of patent-free lattice-based public-key encryption there are a few promising proposals such as a provably secure NTRU variant [49] or the cryptosystem based on the (Ring) LWE problem [32, 36]. For the latter scheme Göttert et al. presented a proof-of-concept implementation in [22] demonstrating that LWE encryption is feasible in software. However, their corresponding hardware implementation is quite large and can only be placed fully on a Virtex-7 2000T and does not even fit onto the largest Xilinx Virtex-6 FPGA

for secure parameters.<sup>1</sup> Several other important aspects for Ring-LWE encryption have also not been regarded yet, such as the reduction of the extensive ciphertext expansion and constant-time operation to withstand timing attacks.

*Contribution.* In this work we aim to resolve the aforementioned deficiencies and present an efficient hardware implementation of Ring-LWE encryption that can be placed even on a low-cost Xilinx Spartan-6 FPGA. Our implementation of Ring-LWE encryption achieves significant performance, namely 42.88  $\mu\text{s}$  to encrypt and 27.51  $\mu\text{s}$  to decrypt a block, even with very moderate resource requirements on the low-cost Spartan-6 family. Providing the evidence that Ring-LWE encryption can be both fast and cheap in hardware, we hope to complement the work by Göttert et al. [22] and demonstrate that lattice-based cryptography is indeed a promising and practical alternative for asymmetric encryption in future real-world systems. In summary, the contributions of this work are as follows:

1. *Efficient hardware implementation of Ring-LWE encryption.* We present a micro-code processor implementing Ring-LWE encryption as proposed by [32, 36] in hardware, capable to perform the Number Theoretic Transform (NTT), polynomial additions and subtractions as well as Gaussian sampling. For a fair comparison of our implementation with previous work, we use the same parameters as in [22] and improve their results by at least an order of magnitude considering throughput/area on a similar reconfigurable platform. Moreover, our processor is designed as a versatile building block for the implementation of future ideal lattice-based schemes and is not solely limited to Ring-LWE encryption. All parts of our implementation have constant runtime and inherently provide resistance against timing attacks.
2. *Efficient Gaussian sampling.* We present a constant-time Gaussian sampler implementing the inverse transform method. The sampler is optimized for sampling from narrow Gaussian distributions and is the first hardware implementation of this method in the context of lattice-based cryptography.
3. *Reducing ciphertext expansion and decryption failure rates.* A major drawback of Ring-LWE encryption is the large expansion of the ciphertext<sup>2</sup> and the occurrence of (rare) decryption errors. We analyze different approaches to reduce the impact of both problems and harden Ring-LWE encryption for deployment in real-world systems.

In order to allow third-party evaluation of our results we will make source code files, test-benches and documentation available on our website.<sup>3</sup>

<sup>1</sup> The authors report that the utilization of LUTs required for LWE encryption exceeds the number of available LUTs on a Virtex-6 LX240T by 197 % and 410 % for parameters  $n = 256$  and  $n = 512$ , respectively. Note that the Virtex-6 LX240T is a very expensive (above €1000 as of August 2013) and large FPGA.

<sup>2</sup> For example, the parameters used for implementation in [22] result in a ciphertext expansion by a factor of 26.

<sup>3</sup> See our web page at <http://www.sha.rub.de/research/projects/lattice/>

*Outline.* In Sect. 2 we introduce the implemented ring-based encryption scheme. The implementation of our processor, the Gaussian sampler and the cryptosystem are discussed in Sect. 3. In Sect. 4 we give detailed results including a comparison with previous and related works and conclude with Sect. 5.

## 2 The Ring-LWEEncryptCryptosystem

In this section we briefly introduce the original definition of the implemented Ring-LWE public key encryption system (RING-LWEEncrypt) and propose modifications in order to decrease ciphertext expansion and error rate without affecting the security properties of the scheme.

### 2.1 Background on LWE

Since the seminal result by Ajtai [2] who proved a worst-case to average-case reduction between several lattice problems, the whole field of lattice-based cryptography has received significant attention. The reasons for this seems to be that the underlying lattice problems are very versatile and allow the construction of hierarchical identity based encryption (HIBE) [1] or homomorphic encryption [19, 42] but have also led to the introduction of reasonably efficient public-key encryption systems [22, 32, 36], signature schemes [14, 23, 34], and even hash functions [35]. A significant long-term advantage of such schemes is that quantum algorithms do not seem to yield significant improvements over classical ones and that some schemes exhibit a security reduction that relates the hardness of breaking the scheme to the presumably intractable problem of solving a worst-case (ideal) lattice problem. This is a huge advantage to heuristic and patent-protected schemes like NTRU [29], which are just related to lattice problems but might suffer from yet not known weaknesses and had to repeatedly raise their parameters as immediate reaction to attacks [28]. A particular example is the NTRU signature scheme NTRUSign which has been completely broken [17, 43]. As a consequence, while NTRU with larger parameters can be considered secure, it seems to be worthwhile to investigate possible alternatives.

However, the biggest practical problem of lattice-based cryptography are huge key sizes and also quite inefficient matrix-vector and matrix-matrix arithmetic. This led to the definition of cyclic [40] and more generalized ideal lattices [33] which correspond to ideals in the ring  $\mathbb{Z}[x]/\langle f \rangle$  for some irreducible polynomial  $f$  of degree  $n$ . While certain properties can be established for various rings, in most cases the ring  $R = \mathbb{Z}_q[\mathbf{x}]/\langle x^n + 1 \rangle$  is used. Some papers proposing parameters then also follow the methodology to choose  $n$  as a power of two and  $q$  a prime such that  $q \equiv 1 \pmod{2n}$  and thus support asymptotic quasi-linear runtime by direct usage of FFT techniques. Recent work also suggests that  $q$  does not have to be prime in order to allow security reductions [11].

Nowadays, the most popular average-case problem to base lattice-based cryptography on is presumably the learning with errors (LWE) problem [48]. In order to solve the decisional Ring-LWE problem in the ring  $R = \mathbb{Z}_q[\mathbf{x}]/\langle x^n + 1 \rangle$ , an

attacker has to decide whether the samples  $(a_1, t_1), \dots, (a_m, t_m) \in R \times R$  are chosen uniformly random or whether each  $t_i = a_i s + e_i$  with  $s, e_1, \dots, e_m$  has small coefficients from a Gaussian distribution  $D_\sigma$  [36].<sup>4</sup> This distribution  $D_\sigma$  is defined as the (one-dimensional) discrete Gaussian distribution on  $\mathbb{Z}$  with standard deviation  $\sigma$  and mean 0. The probability of sampling  $x \in \mathbb{Z}$  is  $\rho_\sigma(x)/\rho_\sigma(\mathbb{Z})$  where  $\rho_\sigma(x) = \exp(-\frac{x^2}{2\sigma^2})$  and  $\rho_\sigma(\mathbb{Z}) = \sum_{k=-\infty}^{\infty} \rho_\sigma(k)$ . In this simple case the standard deviation  $\sigma$  completely describes the Gaussian distribution. Note that some works, e.g., [22, 32] use the parameter  $s = \sqrt{2\pi}\sigma$  to describe the Gaussian.

## 2.2 Ring-LWEEncrypt

The properties of the Ring-LWE problem can be used to realize a semantically secure public key encryption scheme with a reduction to decisional Ring-LWE. The scheme has been introduced in the full version [37] of Lyubashevsky et al. [36] and parameters have been proposed by Lindner and Peikert [32] as well as Göttert et al. [22]. The scheme (**Gen**, **Enc**, **Dec**) is defined as follows and will from now on be referred to as RING-LWEENCRYPT:

- **Gen**( $a$ ): Choose  $r_1, r_2 \leftarrow D_\sigma$  and let  $p = r_1 - a \cdot r_2 \in R$ . The public key is  $p$  and the secret key is  $r_2$  while  $r_1$  is just noise and not needed anymore after key generation. The value  $a \in R$  can be defined as global constant or chosen uniformly random during key generation.
- **Enc**( $a, p, m \in \{0, 1\}^n$ ): Choose the noise terms  $e_1, e_2, e_3 \leftarrow D_\sigma$ . Let  $\bar{m} = \mathbf{encode}(m) \in R$ , and compute the ciphertext  $[c_1 = a \cdot e_1 + e_2, c_2 = p \cdot e_1 + e_3 + \bar{m}] \in R^2$
- **Dec**( $c = [c_1, c_2], r_2$ ): Output  $\mathbf{decode}(c_1 \cdot r_2 + c_2) \in \{0, 1\}^n$ .

During encryption the encoded message  $\bar{m}$  is added to  $pe_1 + e_3$  which is uniformly random and thus hides the message. Decryption is only possible with knowledge of  $r_2$  since otherwise the large term  $ae_1r_2$  cannot be eliminated when computing  $c_1r_2 + c_2$ . According to [32] the polynomial  $a$  can be chosen during key generation (as part of each public key) or regarded as a global constant and should then be generated from a public verifiable random generator (e.g., using a binary interpretation of  $\pi$ ). The encoding of the message of length  $n$  is necessary as the noise given by  $e_1r_1 + e_2r_2 + e_3$  is still present after calculating  $c_1r_2 + c_2$  and would prohibit the retrieval of the binary message after decryption. Note that the noise is relatively small as all noise terms are sampled from a narrow Gaussian distribution. With the simple threshold encoding  $\mathbf{encode}(m) = \frac{q-1}{2}m$  the value  $\frac{q-1}{2}$  is assigned only to each binary one of the string  $m$ . The corresponding decoding function needs to test whether a received coefficient  $z \in [0..q-1]$  is in the interval  $\frac{q-1}{4} \leq z < 3\frac{q-1}{4}$  which is interpreted as one and zero otherwise. As a consequence, the maximum error added to each coefficient must not be larger than  $\lfloor \frac{q}{4} \rfloor$  in order to decrypt correctly. The probability of a decryption error

<sup>4</sup> Note that this is the definition of Ring-LWE in **Hermite normal form** where the secret  $s$  is sampled from the noise distribution  $D_\sigma$  instead of uniformly random [37].

is mainly dominated by the tailcut and the standard deviation of the Gaussian  $\sigma = \frac{s}{\sqrt{2\pi}}$ . Decreasing  $s$  decreases the error probability but also negatively affects the security of the scheme.

*Parameter Selection.* For details regarding parameter selection we refer to the work by Lindner and Peikert [32] who propose the parameter sets  $(n, q, s)$  with  $(192, 4093, 8.87)$ ,  $(256, 4093, 8.35)$ , and  $(320, 4093, 8.00)$  for low, medium, and high security levels, respectively. In this context, Lindner and Peikert [32] state that medium security should be roughly considered equivalent to the security of the symmetric AES-128 block cipher as the decoding attack requires an estimated runtime of approximately  $2^{120}$  s for the best runtime/advantage ratio. However, they did not provide bit-security results due to the new nature of the problem and several trade-offs in their attack.

In this context, the authors of [22] introduced hardware-friendly parameter sets for medium  $(256, 7681, 11.31)$  and high security  $(512, 12289, 12.18)$ . With  $n$  being a power of two and  $q$  a prime such that  $q = 1 \pmod{2n}$ , the Fast Fourier Transform (FFT) in  $\mathbb{Z}_q$  (namely the Number Theoretic Transform (NTT)) can be directly applied for polynomial multiplication with a quasi-linear runtime of  $\mathcal{O}(n \log n)$ . Increased security parameters (e.g., a larger  $n$ ) have therefore much less impact on the efficiency compared to other schemes [36].

*Security Implications of Gaussian Sampling.* For practical and efficiency reasons it is common to bound the tail of the Gaussian. As an example, the authors of the first proof-of-concept implementation of RING-LWEENCRYPT [22] have chosen to bound their sampler to  $[-\lceil 2s \rceil, \lceil 2s \rceil]$ . Unfortunately, they do not provide either a security analysis or justification for this specific value. In this context, the probability of sampling  $\pm 24$  which is out of this bound (recall that  $\lceil 2s \rceil = \lceil 2 \cdot 11.32 \rceil = 23$ ) is  $6.505 \cdot 10^{-8}$  and thus not negligible. However, when increasing the tail-cut up to a certain level it can be ensured that certain values will only occur with a negligible probability. For  $[-48, 48]$ , the probability of sampling an  $x = \pm 49$  is  $2.4092 \cdot 10^{-27} < 2^{-80}$  which is unlikely to happen in a real world scenario. The overall quality of a Gaussian random number generator (GRNG) can be measured by computing the statistical distance  $\Delta(X, Y) = \frac{1}{2} \sum_{\omega \in \Omega} |X(\omega) - Y(\omega)|$  over a finite domain  $\Omega$  between the probability of sampling a value  $x$  by the GRNG and the probability given by  $\rho_\sigma(x)/\rho_\sigma(\mathbb{Z})$ .

Since in general attacks on LWE work better for smaller secrets (see [3, 4] for a survey on current attacks) the tail-cut will certainly influence the security level of the scheme. However, we are not aware of any detailed analysis whether short tails or certain statistical distances lead to better attacks. Moreover, a recent trend in lattice-based cryptography is to move away from Gaussian to very small uniform distributions (e.g.,  $-1/0/1$ ) [23, 41]. It is therefore not clear whether a sampler has to have a statistical distance of  $2^{-80}$  or  $2^{-100}$  (which is required for a worst-case to average-case reductions) in order to withstand practical attacks. Moreover the parameter choices for the RING-LWEENCRYPT scheme and for most other practical lattice-based schemes already sacrifice the worst-case to average-case reduction in order to obtain practical parameters (i.e., small

keys). As a consequence, we primarily implemented a  $\pm[2s]$  bound sampler for straightforward comparison with the work by Göttert et al. [22] but also provide details and implementation results for larger sampler instantiations that support a much larger tail.

### 2.3 Improving Efficiency

In this section we propose efficient modifications to RING-LWEECRYPT to decrease the undesirable ciphertext expansion and the error rate at the same level of security.

*Reducing the Ciphertext Expansion.* Threshold encoding was proposed in [22, 32] to transfer  $n$  bits resulting in an inflated ciphertext of size  $2n \log_2 q$ . Efficiency is further reduced if only a part of the  $n$  bits is used, for example to transfer a 128-bit AES key. Moreover, the RING-LWEECRYPT scheme suffers from random decryption errors so that redundancy in the message  $m$  is required to correct those errors. In the following we analyze a simple but effective way to reduce the ciphertext expansion without significantly affecting the error rate. This approach has been previously applied to homomorphic encryption schemes [9, Sect. 6.4], [10, Sect. 4.2] and the idea is basically to cut-off a certain number of least significant bits of  $c_2$  since they mostly carry noise but only few information supporting the threshold decoding. We experimentally verified the applicability of this approach in practice with regard to concrete parameters by measuring the error rates for reduced versions of  $c_2$  as shown in Table 1 ( $u = 1$ ).

**Table 1.** Bit-error rate for the encryption and decryption of 160,000,000 bytes of plaintext when cutting off a certain number  $x$  of least significant bits of every coefficient of  $c_2$  for the parameter set ( $n = 256, q = 7681, s = 11.31$ ) where  $u$  is the parameter of the additive threshold encoding (see Algorithm 1) and  $\pm[2s]$  the tailcut bound. For a cutoff of 12 or 13 bits almost no message can be recovered.

u Cut-off $x$ bits	0	1	2	3	4	5	6	7	8	9	10	11
1 Errors ( $10^3$ )	46	46	45.5	45.6	46	46.5	48.6	56.1	94.4	381	5359	135771
Error rate ( $10^{-5}$ )	3.59	3.59	3.56	3.57	3.59	3.63	3.80	4.38	7.38	29.81	418.7	10610
2 Errors	26	20	26	27	23	21	21	32	71	957	125796	$44 \cdot 10^6$
Error rate ( $10^{-8}$ )	2.03	1.56	2.03	2.11	1.80	1.64	1.64	2.5	5.55	74.7	9830	$34 \cdot 10^5$

As it turns out the error rate does not significantly increase – even if we remove 7 least significant bits of every coefficient and thus have halved the size of  $c_2$ . It would also be possible to cut-off very few bits (e.g., 1 to 3) of  $c_1$  at the cost of an higher error rate. A further extreme option to reduce ciphertext expansion is to omit whole coefficients of  $c_2$  in case they are not used to transfer message bits (e.g., to securely transport a symmetric key). Note that this approach does not affect the concrete security level of the scheme as the modification does not involve any knowledge of the secret key or message and thus

does not leak any further information. When compared with much more complicated and hardware consuming methods, e.g., the compression function for the Lyubashevsky signature scheme presented in [23], this straightforward approach is much more practical.

*Decreasing the Error Rate.* As noted above decryption of RING-LWEEncrypt is prone to undesired message bit-flips with some small probability. Such a faulty decryption is certainly highly undesirable and can also negatively affect security properties. One solution can be the subsequent application of forward error correcting codes but such methods obviously introduce additional complexity in hardware or software. As another approach, the error probability can be lowered by modifying the threshold encoding scheme, i.e., instead of encoding one bit into each coefficient of  $c_2$ , a plaintext bit is now encoded into  $u$  coefficients of  $c_2$ . This additive threshold encoding algorithm is shown in Fig. 1 where **encode** takes as input a plaintext bit-vector  $m$  of length  $\lfloor \frac{n}{u} \rfloor$  and outputs the threshold encoded vector  $\tilde{m}$  of size  $m$ . The decoding algorithm is given the encoded message vector  $\tilde{m}$  affected by an unknown error vector. The impact on the error rate by using additive threshold encoding ( $u = 2$ ) jointly with the removal of least significant bits is shown in Table 1. Note that this significantly lowers the error rate without any expensive encoding or decoding operations and is much more efficient than, e.g., a simple repetition code [38].

<p><b>Algorithm Encode</b>(<math>m = \{0, 1\}^{\lfloor \frac{n}{u} \rfloor}, u</math>)</p> <pre> 1: for i=0 to <math>\lfloor \frac{n}{u} \rfloor - 1</math> do 2:   for j=0 to u-1 do 3:     <math>\tilde{m}[u \cdot i + j] = m[i] \cdot \frac{q-1}{2}</math> 4:   end for 5: end for 6: return <math>\tilde{m}</math> </pre>	<p><b>Algorithm Decode</b>(<math>\tilde{m} = \{-\frac{q-1}{2}, \frac{q-1}{2}\}^n, u</math>)</p> <pre> 1: for i=0 to <math>\lfloor \frac{n}{u} \rfloor</math> do 2:   <math>s = 0</math> 3:   for j=0 to u-1 do 4:     <math>s = s +  \tilde{m}[u \cdot i + j] </math> 5:   end for 6:   if <math>s &lt; \frac{u \cdot q}{4}</math> then 7:     <math>m[i] = 0</math> 8:   else 9:     <math>m[i] = 1</math> 10:  end if 11: end for 12: return <math>m</math> </pre>
---	--

Fig. 1. Additive threshold encoding.

### 3 Implementation of Ring-LWEEncrypt

In this section we describe the design and implementation of our processor with special focus on the efficient and flexible implementation of Gaussian sampling.

### 3.1 Gaussian Sampling

Beside its versatile applicability in lattice-based cryptography, sampling of Gaussian distributed numbers is also crucial in electrical engineering and information technology, e.g., for the simulation of complex communication systems (see [51] for a survey from this perspective). However, it is not clear how to adapt continuous Gaussian samplers, like the ones presented in [25, 31, 54], for the requirements of lattice-based cryptography. In the context of discrete Gaussian sampling for lattice-based cryptography the most straightforward method is rejection sampling. In this case an uniform integer  $x \in \{-\tau\sigma, \dots, \tau\sigma\}$ , where  $\tau$  is the “tail-cut” factor, is chosen from a certain range depending on the security parameter and then accepted with probability proportional to  $e^{-x^2/2\sigma^2}$  [20]. This method has been implemented in software in [22] but the success rate is only approximately 20% and requires costly floating point arithmetic (cf. to the laziness approach in [16]). Another method is a table-based approach where a memory array is filled with Gaussian distributed values and selected by a randomly generated address. Unfortunately, a large resolution – resulting in a very large table – is required for accurate sampling. It is not explicitly addressed in [22] how larger values such as  $x = \lceil 2s \rceil$  for  $s = 6.67$  with a probability of  $\Pr[x = 14] = 1.46 \cdot 10^{-7}$  are accurately sampled from a table with a total resolution of only 1024 entries. We further refer to [15, Table 2] for a comparison of different methods to sample from a Gaussian distribution and a new approach.

*Hardware Implementation Using the Inverse Transform Method.* Since the aforementioned methods seem to be unsuitable for an efficient hardware implementation we decided to use the inverse transform method. When applying this method in general a table of cumulative probabilities  $p_z = \Pr(x \leq z : x \leftarrow D_\sigma)$  for integers  $z \in [-\tau\sigma, \dots, \tau\sigma]$  is computed with a precision of  $\lambda$  bits. For a uniformly random chosen value  $x$  from the interval  $[0, 1)$  the integer  $y \in \mathbb{Z}$  is then returned (still requiring costly floating point arithmetic) for which it holds that  $p_{z-1} \leq x < p_z$  [15, 18, 44].

In hardware we operate with integers instead of floats by feeding a uniformly random value into a parallel array of comparators. Each comparator  $c_i$  compares its input to the commutative distribution function scaled to the range of the PRNG outputting  $r$  bits. As we have to cut the tail at a certain point, we compute the accumulated probability over the positive half (as it is slightly smaller than 0.5) until we reach the maximum value  $j$  (e.g.,  $j = \lceil 2s \rceil$ ) so that  $w = \sum_{k=0}^j \rho_\sigma(x) / \rho_\sigma(\mathbb{Z})$ . We then compute the values fed into the comparators as  $v_k = \frac{2^{r-1}-1}{w}(v_{k-1} + \sum_{k=0}^j \rho_\sigma(x) / \rho_\sigma(\mathbb{Z}))$  for  $0 < k \leq j$  and with  $v_0 = \frac{2^{r-1}-1}{2w} \rho_\sigma(0) / \rho_\sigma(\mathbb{Z})$ . Each comparator  $c_i$  is preloaded with the rounded value  $v_i$  and outputs a one bit if the input was smaller or equal to  $v_i$ . A subsequent circuit then identifies the first comparator  $c_i$  which returned a one bit and outputs either  $l$  or  $-l$ .

The block diagram of the sampler is shown in Fig. 2 for the concrete parameter set ( $n = 256, q = 7681, s = 11.32$ ) where the output of the sampler is bound to  $[-\lceil 2s \rceil, \lceil 2s \rceil] = [-5.09\sigma, 5.09\sigma]$  and the amount of required randomness is



25 bits per sample. These random bits are supplied by a PRNG for which we used the output of an AES block cipher operating in counter mode. Each 128-bit output block of our AES-based PRNG allows sampling of 5 coefficients. One random bit is used for sign determination while the other 24 bits form a uniformly random value. Finally, the output of the sampler is buffered in a FIFO. When leaving the FIFO, the values are lifted to the target domain  $[0, q - 1]$ . Although it is possible to generate a sampler directly in VHDL by computing the cumulative distribution function on-the-fly during synthesis, we have implemented a Python script for this purpose. The reason is that the VHDL floating point implementation only provides double accuracy while the `Decimal`<sup>5</sup> data type supports arbitrary precision. The Python script also performs a direct evaluation of the properties of the sampler (e.g., statistical distance).

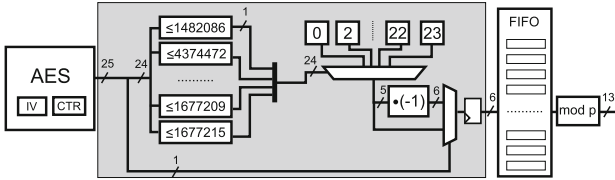


Fig. 2. Gaussian sampler using the inverse transform sampling method.

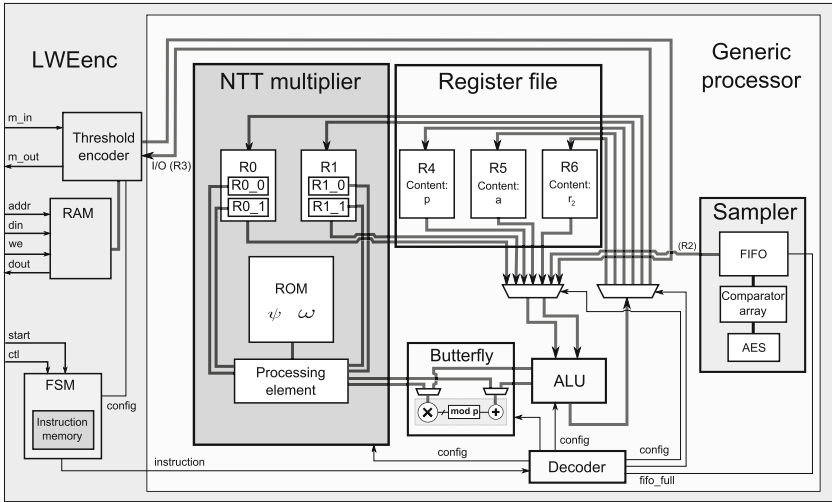
### 3.2 Ring-LWE Processor Architecture

The core of our processor is built around an NTT-based polynomial multiplier which is described in [45]. The freely available implementation has been further optimized and the architecture has been extended from a simple polynomial multiplier into a full-blown and highly configurable micro-code engine. Note that Aysu et al. [6] recently proposed some improvements to the architecture of [45] in order to increase the efficiency and area usage of the polynomial multiplier. While some improvements rely on their decision to fix the modulus  $q$  to  $2^{16} + 1$  other ideas are clearly applicable in future work and revisions of our implementations. However, we do not fix  $q$  as the design goal of our hardware processor is the native support for a large variety of ideal lattice-based schemes, including the most common operations on polynomials like addition, subtraction, multiplication by the NTT as well as sampling of Gaussian distributed polynomials. By supporting an arbitrary number of internal registers (each can store one polynomial) realized in block RAMs and by reusing the data path of the NTT multiplier for other arithmetic operations we achieve high performance at low resource consumption.

*General Description and Instruction Set.* The datapath of our engine depicted in Fig. 3 depends on the size of the reduction prime  $q$  and is thus  $\log_2 q$  as polynomial coefficients are processed serially in a pipeline. Four registers are

<sup>5</sup> <http://docs.python.org/2/library/decimal.html>

fixed where register R0 and R1 are part of the NTT block, while the Gaussian sampler is connected to register R2. Register R3 is exported to upper layers and operates as I/O port. More registers R4 to  $R_x$  can be flexibly enabled during synthesis where each additional register can hold a polynomial with  $n$  elements of size  $\log_2 q$ . The **Switch matrix** is a dynamic multiplexer that connects registers to the ALU and the external interface and is designed to process statements in two-operand form like  $R1 \leftarrow R1 + R2$ . All additional registers  $R_x$  for  $x > 4$  are placed inside of the **Register array** component. The **Decoder** unit is responsible for interpreting instructions that configure the switch matrix, determines whether the ALU has to be used (SUB, ADD, MOV) or if NTT specific commands need to invoke the **NTT multiplier**. To improve resource utilization of the overall system, the butterfly unit of the NTT core is shared between the NTT multiplier and the ALU.



**Fig. 3.** Architecture of our implementation of the RING-LWEENCRYPT engine with a particular instance of our generic lattice processor with three additional registers R4-6.

The most important instructions supported by the processor are the iterative forward (NTT\_NTT) as well as the backward transform (NTT\_INTT) which take  $\approx \frac{n}{2} \log_2 n$  cycles. Other instructions are for example used for the bit-reversal step (NTT\_REV), point-wise multiplication (NTT\_PW\_MUL), addition (ADD), or subtraction (SUB) – each consuming  $\approx n$  cycles. Note that the sampler and the I/O port are just treated as general purpose registers. Thus no specific I/O or sampling instructions are necessary and for example the MOV command can be used. Note also that the implementation of the NTT is performed in place and commands for the backward transformation (e.g., NTT\_PW\_MUL, or NTT\_INTT) modify only register R1. Therefore, after a backward transform a value in R0 is still available.

*Implementation of Ring-LWEEncrypt.* For our implementation we used the medium and high security parameter sets as proposed in [22] which are specifically optimized for hardware. We further exploit the general characteristic of the NTT which allows it to “decompose” a multiplication into two forward transforms and one backward transform. If one coefficient is fixed or needed twice it is wise to directly store it in NTT representation to save subsequent transformations. In Fig. 4 the modified algorithm is given which is more efficient since the public constant  $a$  as well as the public and private keys  $p$  and  $r_2$  are stored in NTT representation.

As a consequence, an encryption operation consists of a certain overhead, one forward NTT transformation ( $n + \frac{1}{2}n \log_2 n$  cycles), two backward transforms ( $2 \cdot (2n + \frac{1}{2}n \log_2 n)$  cycles), two coefficient-wise multiplications ( $2n$  cycles), three calls to the Gaussian sampling routine ( $3n$  cycles) and some additions as well as data movement operations ( $3n$  cycles) which return the error vectors. For decryption, we just need two NTT transformations, one coefficient-wise multiplications and one addition.

### Domain Parameters

Temporary value:  $r_1 = \text{sample}()$ , Global constant:  $\tilde{a} = \text{NTT}(a)$

Secret key:  $\tilde{r}_2 = \text{NTT}(\text{sample}())$ , Public key:  $\tilde{p} = \text{NTT}(r_1 - \text{INTT}(\tilde{a} \circ \tilde{r}_2))$

#### Algorithm Enc( $\tilde{a}, \tilde{p}, m \in \{0, 1\}^n$ )

- 1:  $e_1, e_2, e_3 = \text{sample}()$
- 2:  $\tilde{e}_1 = \text{NTT}(e_1)$
- 3:  $\tilde{h}_1 = \tilde{a} \circ \tilde{e}_1, h_2 = \tilde{p} \circ \tilde{e}_1$
- 4:  $h_1 = \text{INTT}(\tilde{h}_1), h_2 = \text{INTT}(\tilde{h}_2)$
- 5:  $c_1 = h_1 + e_2$
- 6:  $c_2 = h_2 + e_3 + \text{encode}(m)$

#### Algorithm Dec( $c_1, c_2, \tilde{r}_2$ )

- 1:  $\tilde{h}_1 = \text{NTT}(c_1)$
- 2:  $\tilde{h}_2 = \tilde{c}_1 \circ \tilde{r}_2$
- 3:  $m = \text{decode}(\text{INTT}(\tilde{h}_2) + c_2)$

**Fig. 4.** NTT-aware algorithms for Ring-LWEEncrypt.

The top-level module (`LWEEnc`) in Fig. 3 instantiates the ideal lattice processor and uses a block RAM as external interface to export or import ciphertexts  $c_1, c_2$ , keys  $r_2, p$  or messages  $m$  with straightforward clock domain separation (see again Fig. 3). The processor is controlled by a finite state machine (FSM) issuing commands to the lattice processor to perform encryption, decryption, key import or key generation. It is configured with three general purpose registers R4-R6 in order to permanently store the public key  $p$ , the global constant  $a$  and the private key  $r_2$ . More registers for several key-pairs are also supported but optional. The implementation supports pre-initialization of registers so that all constant values and keys can be directly included in the (encrypted) bit-stream. Note that, for encryption, the core is run similar to a stream cipher as  $c_1$  and  $c_2$  can be computed independently from the message which is then only added in the last step (e.g., comparable to the XOR operation used within stream ciphers).

## 4 Results and Performance

For performance analysis we primarily focus on Virtex-6 platforms (speed grade -2) but would also like to emphasize that our solution can be efficiently implemented even on a small and low-cost Spartan-6 FPGA. All results were obtained after post-place and route (Post-PAR) with Xilinx ISE 14.2.

### 4.1 Gaussian Sampling

In Table 2 we summarize resource requirements of six setups of the implemented comparator-based Gaussian sampler for different tail cuts and statistical distances. Our random number generator is a round based AES in counter mode that computes a 128-bit AES block in 13 cycles and comprises 349 slices, 1181/350 LUT/FF, two 18K block RAMs and runs with a maximum frequency of about 265 MHz. Combined with this PRNG<sup>6</sup>, Gaussian sampling based on the inverse transform method is efficient for small values of  $s$  (as typically used for RING-LWEENCRYPT) but would not be suitable for larger Gaussian parameters like, e.g.,  $s = \sqrt{2\pi}2688 = 6737.8$  for the treeless signature scheme presented in [34]. While our sampler needs a huge number of random inputs, the AES engine is still able to generate these numbers (for each encryption we need  $3n$  samples). Table 2 also shows that it is possible to realize an efficient sampler even for a small statistical distance  $<2^{-80}$  since its resource consumption of roughly 250 slices is quite moderate (setup III/IV). With additional register levels and pipelining for versions I/II we achieved the overall clock frequency for the whole core reported in Table 3 in this section. As the PRNG does not provide enough randomness to sample a value in every clock cycle it is not required to evaluate the comparator array in every single cycle so that in particular setups III-VI can use several clock cycles until output is provided. This lowers the critical path and thus allows higher clock frequencies without costs for pipelining registers. Setups V/VI are even more accurate and support (theoretical) requirements of a statistical distance smaller than  $2^{-100}$  [18]. However, then a faster PRNG would be required as for  $n = 256$  we would need  $105 \cdot 3n = 80640$  bits of random input.

### 4.2 Performance of Ring-LWEEncrypt

Table 3 lists the resource consumption and performance of our implementation of RING-LWEENCRYPT. As stated in Sect. 3.2 our implementation combines key generation, encryption and decryption in a holistic design and would not significantly benefit from removing any one of these functional units. The only exception might be a decryption-only core in which no Gaussian sampling is needed.

Table 4 compares the results achieved in this work with the implementation by Göttert et al. [22] as well as other relevant asymmetric schemes and also adds performance figures for a Spartan-6 instantiation. Note that a detailed

<sup>6</sup> Generation of true random numbers is not in the scope of this work; we refer to the survey by Varchola [52] how to achieve this.

**Table 2.** Performance, resource consumption, and quality of the core part (shaded grey in Fig. 2) of the Gaussian sampler on a Virtex-6 LX75T (Post-PAR). The entry *rnd* denotes the number of used random bits to sample one value.

Setup	$s$	Max $s$	<i>rnd</i>	Slices	LUT/FF	MHz	Stat. distance
I	11.32	23	25	42	136/5	115	$<2^{-22}$
II	12.18	25	25	46	149/5	118	$<2^{-22}$
III	11.32	48	85	231	863/6	61	$<2^{-80}$
IV	12.18	51	85	255	911/6	61	$<2^{-80}$
V	11.32	53	105	314	1157/6	58	$<2^{-100}$
VI	12.18	57	105	342	1248/6	50	$<2^{-100}$

**Table 3.** Resource consumption and performance of the combined key generation, encryption and decryption engine for the two different bits security levels on a Virtex-6 LX75T (Post-PAR). The public key requires  $n \log_2 q$  bits (when stored in NTT representation), the private key  $n \log_2 q$  bits and the ciphertext  $2n \log_2 q$  bits.

Aspect		Medium Security ( $n=256, q=7681, s=11.32$ )	High Security ( $n=512, q=12289, s=12.18$ )
Resources	Slices	1506	1887
	LUT/FF	4549/3624	5595/4760
	18K BRAM	12	14
	DSP48E1	1	1
Performance	MHz	262	251
	Key generation (cycles/time)	7235/27.61 $\mu$ s	14532/57.90 $\mu$ s
	Encryption (cycles/time)	6861/26.19 $\mu$ s	13769/54.86 $\mu$ s
	Decryption (cycles/time)	4404/16.80 $\mu$ s	8883/35.39 $\mu$ s

comparison with [22] is unfair due to inaccuracies of synthesis results (the Virtex-6 LX240T FPGA used in [22] was overmapped so that the subsequent place-and-route (PAR) step providing final results could not be performed). Figures for clock frequency, overall slice consumption, and cycles counts for individual operations or the whole encryption block are thus not given in [22]. We therefore can only refer to numbers providing the resource consumption of registers and LUT usage. For a rough comparison we apply the throughput to area (T/A) metric and define area equivalent to the usage of LUTs due to the restriction mentioned above. It turns out that our implementation for  $n = 256$  is 32 times smaller regarding key generation, 65 times smaller for encryption and 27 times smaller for decryption, at a loss of a factor of about 2 and 3.3 in performance. When employing the  $\frac{\text{Bit/s}}{\text{LUT}}$  metric for medium security encryption we achieve  $\frac{9.77 \cdot 10^6 \text{ Bits}}{4549 \text{ LUTs}} = 2147$  while the work presented in [22] gives  $\frac{31.8 \cdot 10^6 \text{ Bits}}{298016 \text{ LUTs}} = 106$ . This results in an improvement of a factor of roughly 20.<sup>7</sup>

<sup>7</sup> For this comparison we assumed that for each encryption 256 bits are transmitted.

In comparison with a recent implementations of the code-based Niederreiter scheme [27] we are faster for decryption and we also use fewer resources on the same platform. Another natural target for comparison is the patent-protected NTRU scheme which has been implemented on a large number of architectures [5, 7, 26]. The implementation in [30] is clearly faster than ours. However, the implemented NTRU(251,3,12) variant in [30] seems to be less secure than our scheme [28]. Unfortunately, we are not aware of any newer NTRU FPGA implementations in order to determine the impact of increased security parameters on runtime and area consumption. In software, NTRU even seems to be rather slow for higher security levels what can be obtained from the 256-bit secure NTRU software implementation (`ntruees787ep1`) benchmarked using the eBACS framework [8] with secret/public key sizes of 1854/1574 bytes and a ciphertext of 1574 bytes. For the ideal lattice-based NTRU version presented in [49], no implementation and concrete parameters have been published yet. In comparison with ECC over prime curves (i.e., a single point multiplication [24]) and RSA (random-exponent 1024-bit exponentiation [50]) our implementation is by an order of magnitude faster, scales better for higher security levels, and also consumes less resources. However, we are not able to beat the recent binary curve implementation of Rebeiro et al. [47] in terms of throughput and performance.

**Table 4.** Performance comparison of our proposal with other public key encryption schemes ( $\approx 80..128$  bit) comparable to the medium security ( $n = 256, q = 7681, s = 11.31$ ) parameter set which is capable of transferring 256-bit messages. Our implementation is versatile enough to perform encryption, decryption and key generation in a single core. Figures denoted with an asterisk (\*) are less accurate results obtained from synthesis due to extensive overmapping of resources.

Scheme	Device	Resources	Speed
Our work [Gen/Enc/Dec] ( $n=256$ )	S6LX16 @160 MHz	4121 LUT/3513 FF/	45.22 $\mu$ s
		14 BRAM(8K)/1 DSP48	42.88 $\mu$ s
			27.51 $\mu$ s
Our work [Gen/Enc/Dec] ( $n=256$ )	V6LX75T @262 MHz	4549 LUT/3624 FF/	27.61 $\mu$ s
		12 BRAM(18K)/1 DSP48	26.19 $\mu$ s
			16.80 $\mu$ s
RING-LWEENCRYPT [Gen/Enc/Dec] ( $n=256$ ) [22]	V6LX240T	146718 LUT/82463 FF	-
		298016 LUT/143396 FF	8.05 $\mu$ s*
		124158 LUT/65174 FF	8.10 $\mu$ s
Niederreiter [Enc/Dec] [27]	V6LX240T V6LX240T	888 LUT/875 FF/17 BRAM	0.66 $\mu$ s
		9409 LUT/12861 FF/ 12 BRAM	57.78 $\mu$ s
NTRU [Enc/Dec] [30]	XCV1600E	27292 LUT/5160 FF	1.54 $\mu$ s
			1.41 $\mu$ s
1024-bit mod. Exp. [50]	XC4VFX12	3937 SLICE/17 DSP48	1.71 ms
ECC-P224 [24]	XC4VFX12	1825 LUT/1892 FF/ 26 DSP48/ 11 BRAM	365.1 $\mu$ s
ECC-B233 [47]	XC5VLX85T	18097 LUT/5644 SLICE	12.3 $\mu$ s

### 4.3 Constant Time Operation

Side-channel attacks are a problem for all physical implementations [39]. A simple target for a side-channel attack is the use of timing information of the security algorithm by measuring execution time or cycles. Our implementation of RING-LWEENCRYPT is fully pipelined and has no data-dependent operations. The processor core does not support any branches and Gaussian sampling based on the inverse transform operates in constant time. Summarizing, all cryptographic operations of our core are timing-invariant.

## 5 Conclusions and Future Work

In this work we presented a novel implementation of the ideal lattice-based Ring-LWE encryption scheme that fits even on a low-cost Spartan-6 FPGA. According to our findings, we improved the results obtained in the previous work of [22] by at least an order of magnitude using the same FPGA platform and much less resources.

Future work can combine our hardware engine with error correction facilities and CCA2 conversion. Additionally, countermeasures against further side-channel and fault-injection attacks need to be considered. As we intend to make our implementation publicly available, our work also offers the chance for third-party side-channel evaluation and cryptanalysis (e.g., exploiting the concrete implementation of the Gaussian sampler). Since our processor could also be utilized by other lattice-based cryptosystems, the provably secure NTRU variant presented in [49] can be another target for implementation. Moreover, a recent proposal of a lattice-based signature scheme by Ducas et al. [14] uses exactly the same parameters ( $n = 512, q = 12289$ ) as RING-LWEENCRYPT and is thus a natural target for implementation based on our micro-code engine.

## References

1. Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: Gilbert [21], pp. 553–572
2. Ajtai, M.: Generating hard instances of lattice problems. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 99–108. ACM (1996)
3. Albrecht, M., Cid, C., Faugère, J.-C., Fitzpatrick, R., Perret, L.: On the complexity of BKW algorithm against LWE. In: SCC’12: Proceedings of the 3rd International Conference on Symbolic Computation and Cryptography, Castro-Urdiales, July 2012, pp. 100–107 (2012)
4. Albrecht, M., Cid, C., Faugère, J.-C., Fitzpatrick, R., Perret, L.: On the complexity of the Arora-Ge algorithm against LWE. In: SCC’12: Proceedings of the 3rd International Conference on Symbolic Computation and Cryptography, Castro-Urdiales, July 2012, pp. 93–99 (2012)
5. Atici, A.C., Batina, L., Fan, J., Verbaauwhede, I., Örs, S.B.: Low-cost implementations of NTRU for pervasive security. In: ASAP, pp. 79–84. IEEE Computer Society (2008)

6. Aysu, A., Patterson, C., Schaumont, P.: Low-cost and area-efficient FPGA implementations of lattice-based cryptography. In: IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 2013. IEEE (2013, to appear)
7. Bailey, D.V., Coffin, D., Elbirt, A., Silverman, J.H., Woodbury, A.D.: NTRU in constrained devices. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 262–272. Springer, Heidelberg (2001)
8. Bernstein, D.J., Lange, T.: eBACS: ECRYPT benchmarking of cryptographic systems. <http://bench.cr.yp.to>. Accessed 10 May 2013
9. Bos, J.W., Lauter, K., Loftus, J., Naehrig, M.: Improved security for a ring-based fully homomorphic encryption scheme. IACR Cryptol. ePrint Arch. **2013**, 75 (2013)
10. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 868–886. Springer, Heidelberg (2012)
11. Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) STOC, pp. 575–584. ACM (2013)
12. Buchmann, J., May, A., Vollmer, U.: Perspectives for cryptographic long-term security. Commun. ACM **49**(9), 50–55 (2006)
13. Canetti, R., Garay, J.A. (eds.): CRYPTO 2013, Part I. LNCS, vol. 8042. Springer, Heidelberg (2013)
14. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. In: Canetti and Garay [13], pp. 40–56. Proceedings version of [15]
15. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. IACR Cryptol. ePrint Arch. **2013**, 383 (2013). (Full version of [14])
16. Ducas, L., Nguyen, P.Q.: Faster Gaussian lattice sampling using lazy floating-point arithmetic. In: Wang and Sako [53], pp. 415–432
17. Ducas, L., Nguyen, P.Q.: Learning a zonotope and more: cryptanalysis of NTRUSign countermeasures. In: Wang and Sako [53], pp. 433–450
18. Galbraith, S.D., Dwarakanath, N.C.: Efficient sampling from discrete gaussians for lattice-based cryptography on a constrained device
19. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, pp. 169–178. ACM (2009)
20. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Dwork, C. (ed.) STOC, pp. 197–206. ACM (2008)
21. Gilbert, H. (ed.): EUROCRYPT 2010. LNCS, vol. 6110. Springer, Heidelberg (2010)
22. 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 and Schaumont [46], pp. 512–529
23. Güneysu, T., Lyubashevsky, V., Pöppelmann, T.: Practical lattice-based cryptography: a signature scheme for embedded systems. In: Prouff and Schaumont [46], pp. 530–547
24. Güneysu, T., Paar, C.: Ultra high performance ECC over NIST primes on commercial FPGAs. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 62–78. Springer, Heidelberg (2008)



25. Gutierrez, R., Torres, V., Valls, J.: Hardware architecture of a Gaussian noise generator based on the inversion method. *IEEE Trans. Circ. Syst.* **59-II**(8), 501–505 (2012)
26. Hermans, J., Vercauteren, F., Preneel, B.: Speed records for NTRU. In: Pieprzyk, J. (ed.) *CT-RSA 2010*. LNCS, vol. 5985, pp. 73–88. Springer, Heidelberg (2010)
27. Heyse, S., Güneysu, T.: Towards one cycle per bit asymmetric encryption: code-based cryptography on reconfigurable hardware. In: Prouff and Schaubert [46], pp. 340–355
28. Hirschhorn, P.S., Hoffstein, J., Howgrave-Graham, N., Whyte, W.: Choosing NTRUEncrypt parameters in light of combined lattice reduction and MITM approaches. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) *ACNS 2009*. LNCS, vol. 5536, pp. 437–455. Springer, Heidelberg (2009)
29. Hoffstein, J., Pipher, J., Silverman, J.H.: NTRU: a ring-based public key cryptosystem. In: Buhler, J.P. (ed.) *ANTS 1998*. LNCS, vol. 1423, pp. 267–288. Springer, Heidelberg (1998)
30. Kamal, A.A., Youssef, A.M.: An FPGA implementation of the NTRUEncrypt cryptosystem. In: *2009 International Conference on Microelectronics (ICM)*, pp. 209–212. IEEE (2009)
31. Lee, D.-U., Luk, W., Villasenor, J.D., Zhang, G., Leong, P.H.-W.: A hardware Gaussian noise generator using the Wallace method. *IEEE Trans. Very Large Scale Integr. VLSI Syst.* **13**(8), 911–920 (2005)
32. 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)
33. Lyubashevsky, V., Micciancio, D.: Generalized compact knapsacks are collision resistant. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) *ICALP 2006*. LNCS, vol. 4052, pp. 144–155. Springer, Heidelberg (2006)
34. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) *EUROCRYPT 2012*. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (2012)
35. Lyubashevsky, V., Micciancio, D., Peikert, C., Rosen, A.: SWIFFT: a modest proposal for FFT hashing. In: Nyberg, K. (ed.) *FSE 2008*. LNCS, vol. 5086, pp. 54–72. Springer, Heidelberg (2008)
36. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert [21], pp. 1–23. Proceedings version of [37]
37. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. *IACR Cryptol. ePrint Arch.* **2012**, 230 (2012). (Full version of [36])
38. MacWilliams, F.J., Sloane, N.J.A.: *The Theory of Error-Correcting Codes*. vol. 16, 762 pp, Elsevier Science Publishers B. V., North-Holland (2006). ISBN: 0-444-85193-3
39. Mangard, S., Oswald, E., Popp, T.: *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*, 3rd edn. Springer, New York (2007)
40. Micciancio, D.: Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Comput. Complex.* **16**(4), 365–411 (2007)
41. Micciancio, D., Peikert, C.: Hardness of SIS and LWE with small parameters. In: Canetti and Garay [13], pp. 21–39
42. Naehrig, M., Lauter, K., Vaikuntanathan, V.: Can homomorphic encryption be practical? In: *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop, CCSW '11*, pp. 113–124. ACM, New York (2011)

43. Nguyễn, P.Q., Regev, O.: Learning a parallelepiped: cryptanalysis of GGH and NTRU signatures. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 271–288. Springer, Heidelberg (2006)
44. Peikert, C.: An efficient and parallel Gaussian sampler for lattices. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 80–97. Springer, Heidelberg (2010)
45. Pöppelmann, T., Güneysu, T.: Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware. In: Hevia, A., Neven, G. (eds.) LatinCrypt 2012. LNCS, vol. 7533, pp. 139–158. Springer, Heidelberg (2012)
46. Prouff, E., Schaumont, P. (eds.): CHES 2012. LNCS, vol. 7428. Springer, Heidelberg (2012)
47. Rebeiro, C., Roy, S.S., Mukhopadhyay, D.: Pushing the limits of high-speed  $GF(2^m)$  elliptic curve scalar multiplication on FPGAs. In: Prouff and Schaumont [46], pp. 494–511
48. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) STOC, pp. 84–93. ACM (2005)
49. Stehlé, D., Steinfeld, R.: Making NTRU as secure as worst-case problems over ideal lattices. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 27–47. Springer, Heidelberg (2011)
50. Suzuki, D.: How to maximize the potential of FPGA resources for modular exponentiation. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 272–288. Springer, Heidelberg (2007)
51. Thomas, D.B., Luk, W., Leong, P.H.W., Villasenor, J.D.: Gaussian random number generators. *ACM Comput. Surv.* **39**(4), 11:1–11:38 (2007)
52. Varchola, M.: FPGA based true random number generators for embedded cryptographic applications. Ph.D. thesis, Technical University of Kosice (2008)
53. Wang, X., Sako, K. (eds.): ASIACRYPT 2012. LNCS, vol. 7658. Springer, Heidelberg (2012)
54. Zhang, G., Leong, P.H.-W., Lee, D.-U., Villasenor, J.D., Cheung, R.C.C., Luk, W.: Ziggurat-based hardware Gaussian random number generator. In: International Conference on Field Programmable Logic and Applications, 2005, pp. 275–280 (2005)