On the Design of Hardware Building Blocks for Modern Lattice-Based Encryption Schemes

Norman Göttert, Thomas Feller, Michael Schneider, Johannes Buchmann, and Sorin Huss

CASED - Center for Advanced Security Research Darmstadt
Technische Universität Darmstadt, Germany
{norman.gottert,thomas.feller,michael.schneider, johannes.buchmann,sorin.huss}@cased.de

Abstract. We present both a hardware and a software implementation variant of the *learning with errors* (LWE) based cryptosystem presented by Lindner and Peikert. This work helps in assessing the practicality of lattice-based encryption. For the software implementation, we give a comparison between a matrix and polynomial based variant of the LWE scheme. This module includes multiplication in polynomial rings using Fast Fourier Transform (FFT). In order to implement lattice-based cryptography in an efficient way, it is crucial to apply the systems over polynomial rings. FFT speeds up multiplication in polynomial rings, which is the most critical operation in lattice-based cryptography, from quadratic to quasi-linear runtime. For the hardware variant, we show how this fundamental building block of lattice-based cryptography can be implemented and evaluated in terms of performance. A second important component for lattice-based cryptosystems is the sampling from discrete Gaussian distributions. We examine three different variants for sampling Gaussian distributed integers, namely rejection sampling, a rounding based approach, and a look-up table based approach in hardware.

Keywords: LWE, Lattice-Based Encryption, Hardware, FPGA.

1 Introduction

Lattice-based cryptography is currently enjoying high attention in the cryptographic community. Related systems offer an alternative security background to factoring and discrete logarithm based schemes. Moreover, while the latter two may be broken using quantum computers, so far there is no quantum computer algorithm known that solves hard lattice problems faster than classical algorithms. Unlike factoring and discrete logarithms, there are even no subexponential time attacks known against lattice systems on classical computers. Last, but not least, lattice-based cryptosystems only apply simple and fast arithmetic operations and asymptotically allow for quasi-linear runtimes, which is nearly optimal. Lattice-based schemes are usually accompanied by very strong security proofs, which relate breaking the system to solving worst-case problems in lattices (compared to basing the security on average-case problems only, like we know from other areas of cryptography). All these facts distinguish lattice-based cryptosystems as promising candidates to replace systems based on number theoretic problems, like factoring and computing discrete logarithms.

Originally, most lattice systems require the storage of huge matrices over integer rings and are quite inefficient both in runtime and storage space. The idea of replacing matrices by polynomials over ideals in integer rings allows to reduce both. Hence, replacing lattices by ideal lattices results in very efficient systems. Instead of storing huge matrices of space $\mathcal{O}(n^2)$, where n is larger than 128, it is sufficient to store just $\mathcal{O}(n \log n)$ elements. Moreover, the multiplication of elements of ideal lattices can be performed efficiently using the Fast Fourier Transform (FFT) [CT65] in time $\mathcal{O}(n \log n)$ for a serial and in $\mathcal{O}(\log n)$ for a parallel implementation, instead of $\mathcal{O}(n^2)$ for straightforward multiplication.

Based on lattice problems, many cryptographic primitives were already developed in theory. Among others, there are a hash function [ADL$^+$08], digital signatures [Lyu09], encryption schemes [SS11, LP11], fully homomorphic encrpytion [Gen09], and many more. The security of most encryption schemes is based on the *learning with errors* problem (LWE). Regev [Reg05] and Peikert [Pei09] proved that the LWE problem is at least as hard as solving certain lattice problems in the worst case, which is the background of the strong security of LWE-based cryptosystems. What is missing for nearly all lattice-based encryption systems, however, are implementations. To the best of our knowledge, there is no publicly available implementation of any provably secure lattice-based cryptosystem available yet. The NTRUEncrypt system [HPS98] is a special case, where implementations are provided, but they are protected by patents. Further, the original NTRUEncrypt scheme lacks a security proof. In order to show that lattice-based cryptography is ready for practical real-world applications, the schemes have to be implemented first. The asymptotic advantage gained by FFT is well-known, however, it lacks a practical evaluation for this application.

For sampling of Gaussian distributed integers, the situation is similar. The theoretical evaluation of rejection sampling is known meanwhile, but the practical efficiency of this approach is still unclear. We are also not aware of any comparison to the rounding-based approach as presented by Devroye in [Dev86].

When lattice-based cryptography is to be used in practice, efficient hardware components are required as well. Therefore, it is necessary to investigate into design optimizations of current cryptosystems as hardware modules. Hence, we present efficient hardware modules for the fundamental building blocks of lattice-based encryption schemes, such as the FFT-based polynomial multiplication and a Gaussian sampler.

In addition to providing a reference software implementation of the matrix and polynomial variants of the encryption scheme, we detail a fully engineered hardware implementation using FFT for polynomial multiplication. Furthermore, an evaluation of all these implementation variants is given.

1.1 Related Work

Regev introduced the first worst-case hardness proof for the LWE problem together with the first LWE-based encryption scheme in 2005 [Reg05]. Various improvements of this scheme appeared later, such as [ACPS09, LPR10, Mic10]. In 2011, Lindner and Peikert proposed in [LP11] an adaption of the system of [Mic10] and various efficiency improvements. This is the most recent and at the same time the most promising LWE-based encryption scheme. These authors detail an improved security analysis and multiple parameter sets for different security levels. They introduce a matrix-based variant as well as an instantiation based on polynomials over residue rings.

The encryption scheme of [SS11] is a variant of the NTRUEncrypt system equipped with a quantum security reduction. Its security is based on the LWE problem in polynomial rings in the standard model. To our knowledge there is no practical investigation of this scheme available so far. We expect its efficiency to be comparable to the LWE scheme of [LP11].

1.2 Our Contribution

To the best of our knowledge we present the first practical evaluation of an LWE-based cryptosystem. This paper details a performance comparison of two realization variants based on matrix and polynomial operations, whereas the latter one is using FFT for fast multiplication. Further, real-world implications are evaluated by a comparison of the measured error-rate to theoretical expectations. Moreover, we present an optimal set of parameters for hardware implementations, which allow for optimizations of the largest hardware modules. In our hardware implementation we apply the FFT approach of [CLRS09]. Additionally, well-known improvements related to the inverse FFT allow for the removal of half of the residue class multipliers, together with the reduction of the critical path and hence provide a higher performance.

We propose efficient hardware modules for evaluation of the FFT as well as for the discrete Gaussian sampler. Both modules can be used for numerous lattice-based encryption [LPR10, LP11] and signature schemes as in, e.g. [GPV08, Lyu12].

Our experiments illustrate the sizes of the keys, the message expansion factor, and the timing results for the hardware and software implementations. The polynomial variant of LWE performes as expected: The size of private and public keys grows linearly in the security parameter. For the medium security parameters given in [LP11], the size of the secret key is 0.5 KB, whereas the public key is 1 KB in size. In software, key generation for the same security level takes $3.1ms$, whereas encryption and decryption take $1.5ms$ and $0.6ms$, respectively. In contrast, dedicated hardware modules speed-up encryption by a factor of nearly 200 and decryption by nearly 70 for this level of security. The message expansion factor, i.e., the size of a ciphertext divided by the size of a plaintext, is about 50. While the polynomial variant is superior in all other measured characteristics, the matrix variant features smaller message expansion factors.

2 Preliminaries

The authors of [LP11] propose two implementation variants, which exploit either a matrix-based or a polynomial-based representation. For this reason we refer to these representations as LWE-Matrix and LWE-Polynomial, respectively.

2.1 The LWE Problem

The security background of the cryptosystem under examination is the LWE problem, which was introduced by Regev in 2005.

Consider a dimension $n > 1$, an integer module $p \geq 2$ and an error distribution χ. The distribution χ will be the discrete Gaussian error distribution. Given a vector $\mathbf{s} \in \mathbb{Z}_q^n$, a vector $\mathbf{a} \in \mathbb{Z}_q^n$ is chosen uniformly at random. Further, an error term $e \leftarrow \chi$ is chosen and the pair $(\mathbf{a}, t = \langle \mathbf{a} | \mathbf{s} \rangle + e \mod p)$ is computed. The search version of LWE asks to find $\mathbf{s}$ given an arbitrary number of sample pairs $(\mathbf{a}_i, t_i)$. The decision version of LWE asks to distinguish between arbitrary numbers of sample pairs $(\mathbf{a}_i, t_i)$ and uniformly drawn samples from $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

For hardness results on LWE we refer the reader to the work of [Reg05, Pei09]. Practical attacks on the LWE-based cryptosystems were described in [LP11]. The ring LWE problem defined in [LPR10] is the adaption of LWE to polynomial rings. It is important as security background for the LWE-Polynomial scheme. An attacker breaking the LWE-Polynomial encryption system is able to solve the ring LWE problem instance, and thus is able to solve certain lattice problems in *all* lattices of a certain smaller dimension (the so-called *worst-case hardness*).

2.2 LWE-Based Encryption

Here we recall the more efficient polynomial variant, for the matrix variant we refer to [Mic10, LP11]. Define the polynomial rings $R = \mathbb{Z}[X]/\langle f(x) \rangle$ and $R_q = \mathbb{Z}_q[X]/\langle f(x) \rangle$ for a polynomial $f(x)$ that is monic and irreducible. Example choices are $f(x) = x^n + 1$ for n being a power of 2. Further, χ_k and χ_e are error distributions over R for key generation and encryption, respectively. Useful parameters for different levels of security were presented in [LPR10, LP11].

The LWE-Polynomial encryption is denoted as $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, where

- $\mathsf{KeyGen}(a)$: choose r_1, $r_2 \leftarrow \chi_k$ and let $p = r_1 - a \cdot r_2$. Output public key p and secret key r_2.
- $\mathsf{Enc}(a, p, m \in \Sigma^n)$: choose $e_1, e_2, e_3 \leftarrow \chi_e$. Let $\bar{m} = \mathsf{encode}(m) \in R_q$. The ciphertext is then $(c_1 = a \cdot e_1 + e_2, c_2 = p \cdot e_1 + e_3 + \bar{m}) \in R_q^2$.
- $\mathsf{Dec}((c_1, c_2), r_2)$: output $\mathsf{decode}(c_1 \cdot r_2 + c_2)$.

Decoding fails if $|e_1 \cdot r_1 + e_2 \cdot r_2 + e_3|$ is bigger than the threshold $t = \lfloor q/4 \rfloor$. This per-symbol error probability is denoted δ. It is depending on the error distributions χ_k and χ_e. More exactly, δ is an upper bound on the error probability per symbol. Following the proposal of [LP11], we choose $\chi_k = \chi_e = \chi$. The Gaussian standard deviation s for the distribution χ is selected depending on the dimension n, threshold t, parameter c, and the error probability δ by means of

$$s^2 = \frac{\sqrt{2}\pi}{c} \cdot \frac{t}{\sqrt{2n \cdot ln(2/\delta)}} . \tag{1}$$

The variant LWE-Matrix is denoted in a similar manner, it uses matrices over $\mathbb{Z}_q$ instead of polynomials over the ring R. Since the arithmetic in polynomial rings can be performed more efficiently, the polynomial variant seems to be more appropriate in practice. A possible disadvantage of LWE-Polynomial concerns security. The system is provably secure as long as the decision ring LWE problem is hard. LWE-Matrix only requires the decision LWE problem to be hard, which is a weaker assumption, since it is unknown if the additional ring structure influences the hardness of the LWE problem.

Message Encoding. Error-tolerant encoder and decoder functions are required by the presented encryption system. In the following the message encoding of LWE-Polynomial is detailed, which can analogously be applied to LWE-Matrix.

A message m, represented as a bit-vector $m \in \Sigma^n = \{0,1\}^n$, is transformed into a vector $\bar{m} \in R_q$. Therefore, the encoding and decoding are functions encode:$\Sigma \rightarrow R_q$ and decode:$R_q \rightarrow \Sigma$, respectively. The equation $encode(decode(m) + e \ mod \ q) = m$ is satisfied as long as all coefficients of the error-polynomial $e \in R_q$ are within the threshold t, for which we selected $[-t,t) = [-\lfloor\frac{q}{4}\rfloor, \lfloor\frac{q}{4}\rfloor)$.

2.3 Fast Fourier Transform

The FFT is used to convert the coefficient representation of the polynomial to a point-value representation. The multiplication using the coefficients of two polynomials with degree n takes $\mathcal{O}(n^2)$ time. In point-value representation, the multiplication is performed in $\mathcal{O}(n \log n)$ as serial implementation and in $\mathcal{O}(\log n)$ if implemented in parallel, which we selected for hardware implementation.

Algorithm 1 characterizes the polynomial multiplication, which applies the FFT to convert the polynomials from coefficient to point-value representation.

To exploit the full potential of the FFT and speed-up the polynomial reduction (cf. Appendix A), we restrict the choice of the irreducible polynomial $f(x)$ to a cyclotomic one with the form $f(x) = x^n + 1$, where n is a power of 2. This allows for further improvements which are detailed in Section 4.3.

There are multiple reasons why we favour FFT over other polynomial multiplication approaches (e.g., Toom-Cook[Coo66]). FFT is easily parallelizable, and the asymptotic runtime of the parallel FFT is $\mathcal{O}(\log n)$ compared to $\mathcal{O}(n^{1+\epsilon})$, where $0 < \epsilon < 1$ for Toom-Cook. Additionally, the FFT hardware implementation greatly benefits from the utilized polynomial $f(x) = x^n + 1$ with n a power of 2, as this saves a lot of hardware resources. However, the Toom-Cook approach might be faster for practical parameters, but the comparison of polynomial multiplication algorithms is out of scope of this work.

Algorithm 1. Polynomial Multiplication using FFT

```
Input: $a, b \in \mathbb{Z}_q^{2n}$, $\omega_m$
Output: $c \in \mathbb{Z}_q^{2n}$
1 $A = FFT(a, \omega_m)$
2 $B = FFT(b, \omega_m)$
3 for $i = 0$ to $2n - 1$ do
4     $C[i] = A[i] \cdot B[i]$
5 end
6 $c = FFT^{-1}(C, \omega_m)$
7 return c
```

3 Software Implementation

The general purpose of the software implementation presented herein was to provide a reference implementation of the LWE based encryption scheme and to assess its real-world properties. We are using C++ on a Linux-based operating system, with the GCC 4.6.1 compiler. We integrated the NTL library [Sho] in version 5.5.2, which comprises data types for matrices and vectors over residue classes $\mathbb{Z}_q$ as well as elements in polynomial rings $\mathbb{Z}_q[X]$ and in factor rings $\mathbb{Z}_q[X]/\langle f(x)\rangle$. NTL applies FFT for its polynomial multiplication routines. Our outlined software implementation contains both variants – LWE-Matrix as well as LWE-Polynomial, which are available online.[1] The tested parameter sets of n, q, c, and s, for different values of δ are denoted in Table 1. The first column is taken from [LP11] and the values for the standard deviation of Gaussian sampling s are computed according to (1). It should be noted that the toy parameter set for $n = 128$ can not be considered secure in practice. For $n = 256$ and $\delta = 10^{-2}$ the estimated runtime/advantage ratio of the strongest (so-called decoding) attack is 2^{120} seconds, which is compared to the security of AES-128 in [LP11]. Unfortunately, an approach to compute "real" security estimates (bit-security) for lattice-based cryptosystems is not known so far.

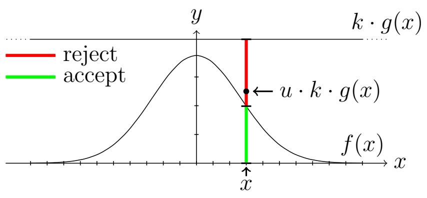


Fig. 1. Rejection sampling for $f(x) = \frac{1}{s} \cdot e^{\frac{-\pi \cdot x^2}{s^2}}$, $g(x) = \frac{1}{n}$ and $k = \lceil \frac{n}{s} \rceil$.

Algorithm 2. Rejection Sampling

```
1 repeat
2     $x \xleftarrow{\$} \mathbb{Z} \cap [-t, t]$
3     $u \xleftarrow{\$} \mathbb{R} \cap [0, 1]$
4 until $u \cdot k \cdot g(x) < f(x)$
5 return $x \in \mathbb{Z}_q$ following a Gaussian distribution
```

[1] https://www.cdc.informatik.tu-darmstadt.de/de/cdc/personen/michael-schneider/

Table 1. Computed values for parameter s in dependence of combinations of all used tuples (n, c, q) and δ, as proposed in [LP11]. For $\delta = 10^{-2}$ this reflects the classifications toy, low, medium, and high for n set to 128, 192, 256, and 320, respectively

(n, c, q)	$s_{\delta=10^{-2}}$	class[LP11]	$s_{\delta=10^{-3}}$	$s_{\delta=10^{-4}}$	$s_{\delta=10^{-5}}$	$s_{\delta=10^{-6}}$
$(128, 1.35, 2053)$	6.77	toy	6.19	5.79	5.5	5.26
$(192, 1.28, 4093)$	8.87	low	8.11	7.59	7.2	6.9
$(256, 1.25, 4093)$	8.35	medium	7.63	7.15	6.78	6.5
$(320, 1.22, 4093)$	8.0	high	7.31	6.84	6.5	6.22
$(384, 1.2, 4451)$	8.04	–	7.34	6.87	6.52	6.25
$(448, 1.184, 4723)$	8.02	–	7.33	6.86	6.51	6.23
$(512, 1.172, 4987)$	8.01	–	7.32	6.85	6.5	6.23

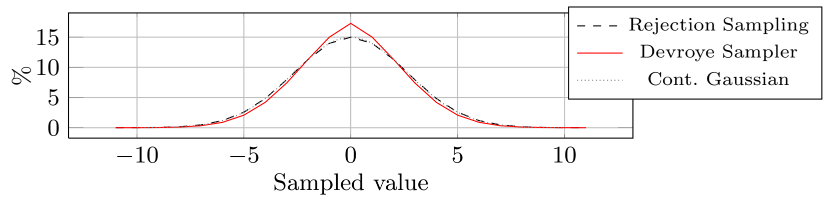


Fig. 2. Histogram of 10^8 samples of rejection sampling and the sampler of Devroye

We denote the expression $x \overset{\$}{\leftarrow} S$ for a value of x that is being sampled uniformly at random from the set S. For sampling Gaussian distributed integers, we apply the *rejection sampling* approach. This method is, among others, exploited in [GPV08] already. Algorithm 2 and Fig. 1 illustrate the rejection sampling approach. An exercise of this method revealed that for the chosen parameters the sampling success rate is approx. 20%. This is due to the fact that in the second sampling step in Algorithm 2, values far from the origin being accepted with a very small probability. Tests with the Gaussian Sampler of Devroye [Dev86, Chap. 3, Exercise 3] with standard deviation of $\sigma = \frac{1}{\sqrt{2\pi}} \cdot s$ showed a success rate of 85.2% for this sampler. The generation of 10^8 samples on our test platform took 19s compared to 75s for the rejection sampler. Therefore, using Devroye's sampler would allow for faster sampling in the encryption system. Unfortunately, the output of this sampler differs from the continuous Gaussian distribution, as shown in Fig. 2. Further, the performance benefits by using a Devroye sampler were negligible in our tests, although the success rates differ significantly.

The performance tests presented in this paper have been executed on an Intel Core 2 Duo CPU running at 3.00 GHz and 4Gb of RAM. As clearly visible from Fig. 3, LWE-Polynomial benefits from the fewer coefficients and outperforms LWE-Matrix by at least a factor of 4. The superior performance results, the smaller memory footprint, and less key data were the reasons to consider only LWE-Polynomial for hardware implementation. To be more specific, the analysis of the memory footprint revealed that LWE-Polynomial utilized 3.8 to 23 times less memory during key generation, 2.6 to 17.2 times while encrypting, and decryption took 2 to 5 times less memory resources than LWE-Matrix.

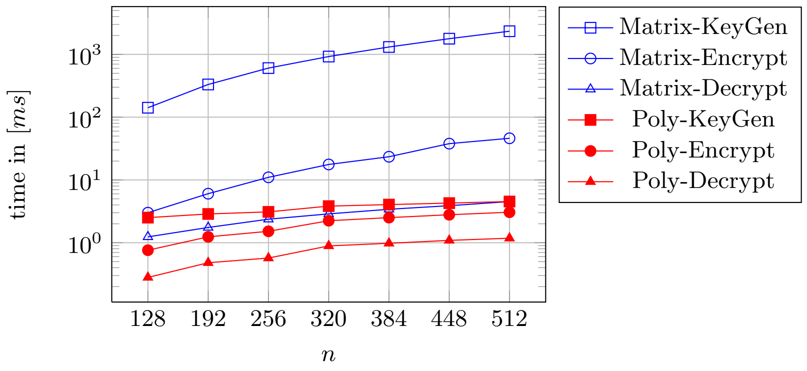


Fig. 3. Computation times of LWE software variants for all basic functions KeyGen, Encrypt, and Decrypt (cf. Table 2).

Table 2. LWE-Matrix vs. LWE-Polynomial: Time in milliseconds for key generation, encryption, and decryption of a 16 byte plaintext.

	KeyGen			Encrypt			Decrypt		
n	t_{Matrix}	t_{Poly}	t_{Matrix}/t_{Poly}	t_{Matrix}	t_{Poly}	t_{Matrix}/t_{Poly}	t_{Matrix}	t_{Poly}	t_{Matrix}/t_{Poly}
128	141.3	2.51	56.2	3.01	0.76	3.98	1.24	0.28	4.40
256	604.9	3.10	195.3	11.01	1.52	7.23	2.37	0.57	4.15
384	1311.2	4.05	323.6	23.41	2.51	9.34	3.41	0.98	3.46
512	2338.5	4.53	516.5	46.05	3.06	15.04	4.52	1.18	3.84

Table 3. Filesizes in bytes of LWE public and private keys as well as size of a ciphertext for a 16 byte plaintext. It is remarkable that the ciphertext of the matrix variant is smaller than that of the polynomial variant.

	Public Key			Private Key			Cyphertext		
n	$Matrix$	$Poly$	$Matrix/Poly$	$Matrix$	$Poly$	$Matrix/Poly$	$Matrix$	$Poly$	$Matrix/Poly$
128	146811	1154	127.22	53602	394	136.05	1142	1143	1.00
256	465851	2435	191.31	108298	883	122.65	1816	2423	0.75
384	935676	3659	255.72	162232	1271	127.64	2433	3650	0.67
512	1567516	4912	319.12	216624	1665	130.10	3058	4893	0.62

The filesize of the generated key material is depicted in Table 3. We directly used the NTL [Sho] output, which is on the one hand clearly not an optimal representation for the data, but allowed for interoperability with other tools. The theoretical key and ciphertext sizes for the software implementation are given in (2) and (3), respectively. Here, n denotes the dimension and l the message length in bits. For software based approaches a short integer (16 bit) length has been selected instead of $\lceil \log_2(q) \rceil$ for the required bit width, which has been chosen for the hardware implementation. The ratios between the filesizes of LWE-Matrix and LWE-Polynomial are on the other hand good estimations for the real values.

$$\begin{aligned} Size_{Matrix,Public} &= (n \cdot l + n^2) \cdot \lceil log_2(q) \rceil & Size_{Matrix,Private} &= n \cdot l \cdot \lceil log_2(q) \rceil \\ Size_{Poly,Public} &= 2 \cdot n \cdot \lceil log_2(q) \rceil & Size_{Poly,Private} &= n \cdot \lceil log_2(q) \rceil \end{aligned} \tag{2}$$

$$Size_{Matrix,Cipher} = (n+l) \cdot \lceil log_2(q) \rceil \qquad Size_{Poly,Cipher} = 2n \cdot \lceil log_2(q) \rceil \qquad (3)$$

Another noteable result of our software evaluation revealed that the ciphertext for higher dimensions is smaller in LWE-Matrix compared to LWE-Polynomial, as denoted in Table 3. Further on, we noticed that different values of the error probability δ have only a marginal impact on the related runtimes.

4 Hardware Implementation

As aforementioned, only the LWE-Polynomial variant has been selected for hardware implementation, as the software evaluation indicated the performance benefits of this representation (see Fig. 3). The evaluation platform of the hardware implementation is a Xilinx ML-605 evaluation board providing a Virtex-6 LX240T FPGA. Hardware modules have been designed in a vendor independent manner, which allows the utilization of FPGAs from other vendors as well as a representation as an Application Specific Integrated Circuit (ASIC) implementation. Additionally, we also present related synthesis results using a Virtex-7 device in Table 6.

The parameters exploited for the hardware implementation are given in Table 4 and we assigned $\delta = 10^{-2}$. Recall that the parameter set with $n = 128$ does not supply sufficient security guarantees ("toy" parameters). The FFT requires for all roots of unity that $q-1$ is a multiple of $2n$ (see Corollary 30.4 [CLRS09]).

The dataflow for the three primitive operations of the LWE-based scheme are depicted in Fig. 4. In contrast to the rejection sampling approach applied in the software variant, which would require floating point arithmetic, the Gaussian sampler has been implemented by means of a look-up table. Integers in the range of $[-\lceil 2 \cdot s \rceil, \lceil 2 \cdot s \rceil]$ are selected by applying the random output of a linear feedback shift register (LFSR) as an address to an array of Gaussian distributed values. In order to save resources, this array has been embodied using only start and end addresses of the values. An unoptimized Gaussian array would require (resolution $\cdot \lceil log_2(q) \rceil$) bits, whereas the optimized version requires $3 \cdot (2 \cdot \lceil 2s \rceil + 1) \cdot \lceil log_2(q) \rceil$ only. For example, with $s = 6.67$, $q = 3329$ and a resolution of 1023 the optimized version requires 1044 bits (c.f. Table 5) whereas a straight forward array would require 12288 bits, which saves in this case approx. 92% of the memory. Additionally, the uniform sampler, required during key generation is realized by an LFSR (cf. Sect. 4.1). For the look-up-table with interval size of $4s$, the probability of a sample outside of this interval is $4.6 \cdot 10^{-6}$. This probability can be lowered further by choosing a larger interval.

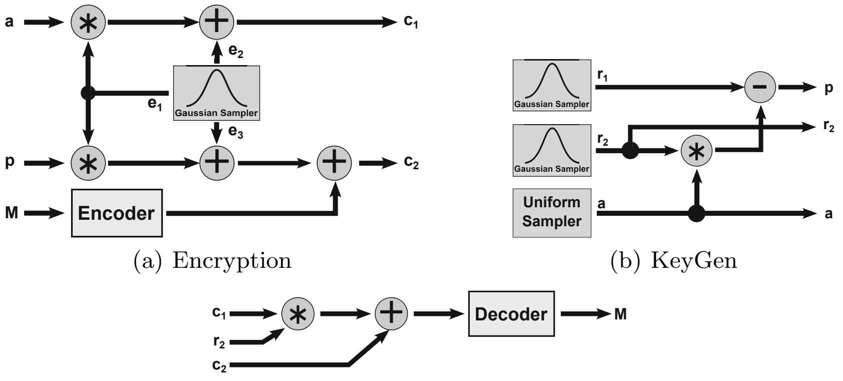
Table 4. LWE parameters for hardware tests using $\delta = 10^{-2}$ and the bit width ($\lceil log_2(q) \rceil$) for representing coefficient values.

n	q	s	ω	$\lceil log_2(q) \rceil$
128	3329	8.62	17	12
256	7681	11.31	62	13
512	12289	12.18	49	14

Table 5. Implemented approach of the Gaussian array

Start/End-address	...	...	157	237	238	337	338	451	452	570	571	684	685	784	785	865	...	...
Gaussian value	...		-3		-2		-1		0		1		2		3		...	

The message encoding, as outlined in Sect. 2.2, is performed by the encode and decode modules depicted in Fig. 4(a) and Fig. 4(c), respectively. The encryption datapath, as displayed in Fig. 4(a), shows how Gaussian distributed random errors are introduced within the cipertext by multiplication and addition. As a result of the encryption, the two vectors c_1 and c_2 contain the ciphertext. Decryption of the ciphertext (c_1, c_2) is performed by a multiplication of the private key r_2 with c_1 followed by an addition of c_2 as detailed in Fig. 4(c).



(a) Encryption

(b) KeyGen

(c) Decryption

Fig. 4. Overview of LWE Encryption Scheme Datapaths

The modular multiplication of polynomials is by far the most expensive operation in this scheme. Therefore, we apply the FFT for polynomial reduction and a Montgomery multiplier [Mon85] to realize the modular multiplication of the polynomial coefficients as depicted in Fig. 5.

4.1 Random Numbers for Keys and Errors

Uniformly distributed random numbers are required by the Gaussian sampler. In this work we emphasize on the implementation of the Gaussian sampler and therfore the quality of the LFSR-based RNG is not in the scope of this paper. For the practical use of the herein proposed scheme, attacks on the random number generators, such as frequency injection [MM09], have to be considered. Novel concepts for RNGs addressing reconfigurable hardware have been presented in previous works, such as [KG04], [Gün10], [VD10] and [MKD11].

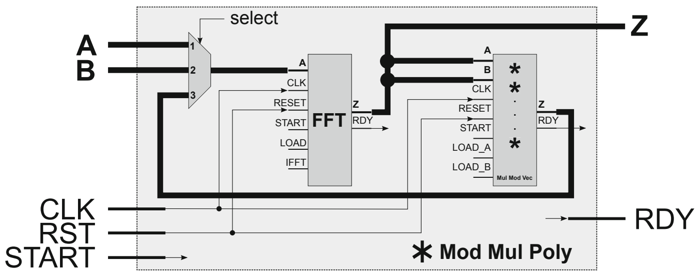


Fig. 5. Arithmetic unit for polynomial multiplication

4.2 Resource Utilization

The utilization of device resources are depicted in Fig. 6 and correspond to the data of the Virtex-6 columns in Table 6. Taking a closer look at the actual values, one can find that dimensions $n > 128$ did not fit into the Virtex-6 device, which we used for evaluation. Therefore, we additionally provide synthesis results for a Virtex-7 series device (cf. Table 6). Using this device enabled the implementation of the whole scheme for the largest dimension considered in this paper. Our primary goal was performance, which naturally leads to larger implementations.

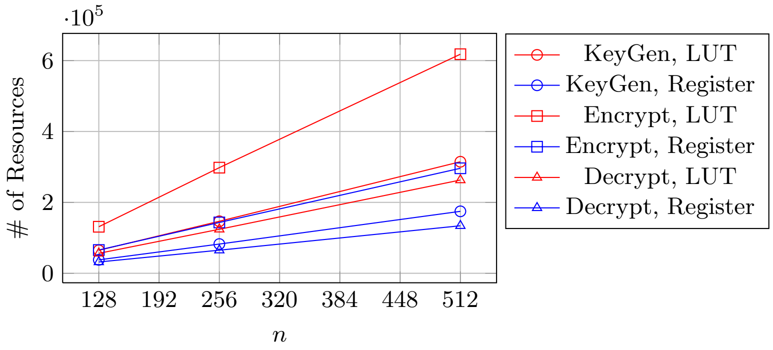


Fig. 6. Hardware resource utilization of the top-modules LWE-KeyGen, LWE-Encrypt, and LWE-Decrypt for $n = \{128, 256, 512\}$. Detailed data is denoted in Table 6.

4.3 Design Improvements

The choice of parameters allows for some optimizations of certain modules. As an example, the inverse FFT can be considerably improved in the case that if the reduction polynomial for the residue ring follows the structure of $f(x) = x^n + 1$ for which n is a power of 2, i.e., it is cyclotomic. The *root of unity* $\omega \in \mathbb{Z}_q$ is selected such that $\omega^{2n} = 1$ *mod* q with q is prime and $q - 1$ is a multiple of $2n$. Based on Collary 30.4 in [CLRS09], ω is determined by $\omega^n = -1$ *mod* q

Table 6. Top-level module resource utilization of KeyGen, Encrypt, and Decrypt on a Xilinx XC6VLX240T FPGA and XC7V2000T for $n = \{128, 256, 512\}$

(a) LWE-KeyGen

	Virtex-6 LX240T						Virtex-7 2000T					
	n = 128	%	n = 256	%	n = 512	%	n = 128	%	n = 256	%	n = 512	%
# Registers	37918	12	82463	27	174757	57	37918	1	85472	3	174757	7
# LUTs	64804	42	146718	97	314635	208	69140	5	163209	13	348204	28

(b) LWE-Encrypt

	Virtex-6 LX240T						Virtex-7 2000T					
	n = 128	%	n = 256	%	n = 512	%	n = 128	%	n = 256	%	n = 512	%
# Registers	65680	21	143396	47	296207	98	65680	2	143396	5	296207	12
# LUTs	131254	87	298016	197	618934	410	131187	10	320816	26	634893	51

(c) LWE-Decrypt

	Virtex-6 LX240T						Virtex-7 2000T					
	n = 128	%	n = 256	%	n = 512	%	n = 128	%	n = 256	%	n = 512	%
# Registers	31884	10	65174	21	134036	44	31884	1	65174	2	134036	5
# LUTs	56311	37	124158	82	263083	174	56313	4	124265	10	260772	21

throughout this paper and it is used as a common parameter of all butterfly modules. Generally speaking, this eliminates the final polynomial reduction step, resulting in a cut of half the residue class multipliers.

Assume that the inputs of a butterfly module are denoted by x_i and x_{i+n} and the outputs are denoted by y_i and y_{i+n}. Then the inner calculation of the butterfly module is given by

$$\begin{aligned} y_i &= x_i + \omega_j x_{i+n} \\ y_{i+n} &= x_i - \omega_j x_{i+n} \,. \end{aligned} \tag{4}$$

In the last step of the inverse FFT each coefficient is multiplied by the inverse element of $2n$ in the residue class ring $\mathbb{Z}_q$. Applying this multiplication to every output of a butterfly module results in

$$\begin{aligned} y_i \cdot (2n)^{-1} &= x_i \cdot (2n)^{-1} + \omega_j x_{i+n} \cdot (2n)^{-1} \\ y_{i+n} \cdot (2n)^{-1} &= x_i \cdot (2n)^{-1} - \omega_j x_{i+n} \cdot (2n)^{-1} \,. \end{aligned} \tag{5}$$

Applying the reduction step of cyclotomic polynomials (cf. (10)), both outputs of a butterfly module are subtracted as follows

$$\begin{aligned} y_i \cdot (2n)^{-1} - y_{i+n} \cdot (2n)^{-1} =& x_i \cdot (2n)^{-1} + \omega_j x_{i+n} \cdot (2n)^{-1} \\ &- x_i \cdot (2n)^{-1} + \omega_j x_{i+n} \cdot (2n)^{-1} \\ =& 2\omega_j (2n)^{-1} \cdot x_{i+n} \,. \end{aligned} \tag{6}$$

An important consequence of (6) is that not every input x_i is required to calculate the inverse FFT. If this consequence is considered for the inputs of the inverse FFT, only those inputs characterized by an odd index are used. So, each input with an odd index is connected to a wire that is placed in the lower half of the parallel FFT depicted as in Fig. 7. The term $2\omega_j(2n)^{-1}$ in (6) is precomputed in order to further reduce the ammount of utilized resources.

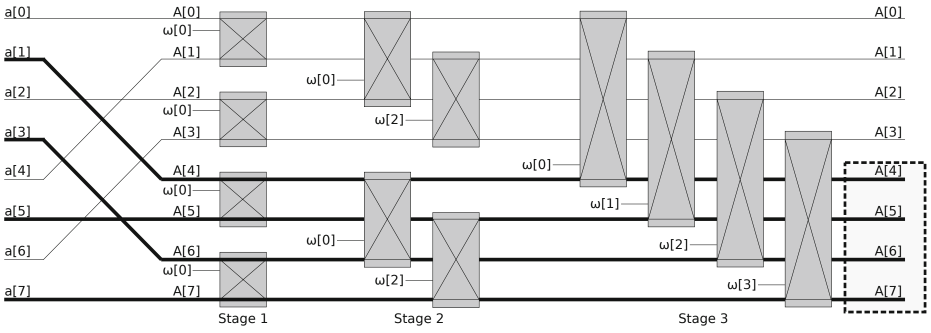


Fig. 7. Optimization of the inverse FFT followed by a polynomial reduction (cf. (6))

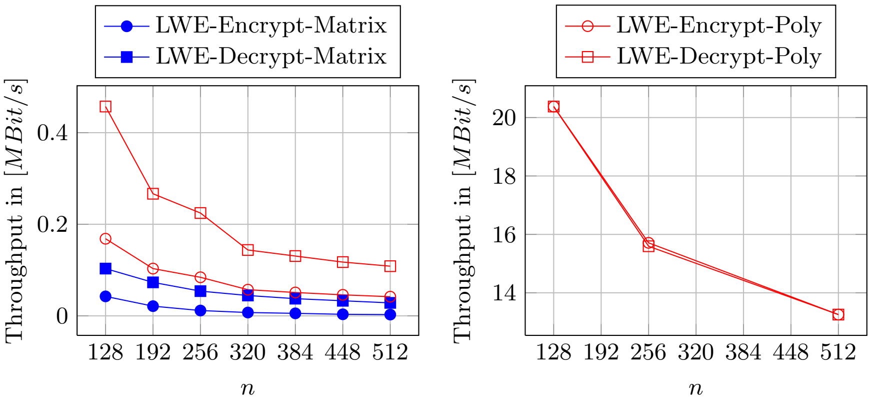
5 Evaluation and Practical Implications

An evaluation of the presented results shows that for the LWE-Polynomial implementation, encryption and decryption differ only by a factor of roughly 2.6 for the software version. The hardware implementation as presented in this paper is, because of its full parallel structure, able to perform encryption and decryption at the same speed. The LWE-Matrix does not share this property as the size of the matrix grows quadratically with the dimension n.

To compare the performance of each implementation, the achieved throughputs of both, the hardware and software implementation variants are depicted in Fig. 8. Due to the increased parallelism and the structure of the encryption scheme, the hardware outperforms the software by a factor up to 316 for encryption and of 122 for decryption. For the key generation the results are even better and show a performance gain of roughly 400 in all three dimensions ($n = \{128, 256, 512\}$). The reason for this considerable gain is the difference between the rejection sampling approach in software and the look-up table method in hardware. Additionally, the hardware benefits from a full parallel implementation for sampling values, in contrast to the serial software implementation.

5.1 Message Expansion Factors

The estimated message expansion factor for the dimension $n = 128$ and the parameters $q = 2053$ and $s = 6.77$ is 22. For the software-based implementations of LWE-Matrix and LWE-Polynomial a short integer (16 bit) has been chosen to represent the coefficient values, resulting in the difference in Table 7. As aforementioned, the parameters for the hardware variant have been selected to improve the FFT, hence the message expansion factors are also different (as q is set to 3329). A comparison of all implementation variants is detailed in Table 7. Obviously, using the optimal representation for the polynomial coefficients, the message expansion factor of 24 is still very close to the expected factor of 22.



(a) Software (b) Hardware

Fig. 8. Throughput values of evaluated implementation variants

Table 7. Ciphertext in bytes for 16 byte plaintext and corresponding expansion factors

	LWE-Matrix		LWE-Polynomial		LWE-Hardware	
n	Cipher	Cipher/Plain	Cipher	Cipher/Plain	Cipher	Cipher/Plain
128	512	32	512	32	384	24
192	640	40	768	48	–	–
256	768	48	1024	64	832	52
320	896	56	1280	80	–	–
384	1024	64	1536	96	–	–
448	1152	72	1792	112	–	–
512	1280	80	2048	128	1792	112

5.2 Error Rates

An interesting result of the software implementaion tests, is the practical error rate that can be observed during decryption, as depicted in Figure 9. As a part of the evaluation we assessed the error rates, which represent the error probability per symbol, against the upper bound δ outlined in Sect. 2.2. We state that the practical error rate is more than a factor 100 smaller than its upper bound δ, whereas the error rate increases with the dimension n.

The exact measurement values for the encountered bit errors for LWE-Matrix and LWE-Polynomial are given in Table 8 and Table 9, respectively.

A reduction of the error probability may be achieved by changing the parameters which determine s (cf. (1)) q and n. The Gaussian standard deviation s decreases when δ decreases as well. For the security guarantee to hold, it is necessary that $s \cdot q > 2\sqrt{n}$ holds. This implies that, for the same security level n, q has to be increased when smaller δ (and with this smaller s) is required. A second way to deal with decryption errors is the application of error correcting codes. This approach allows to keep the system parameters unchanged, but enlarges the message expansion factor.

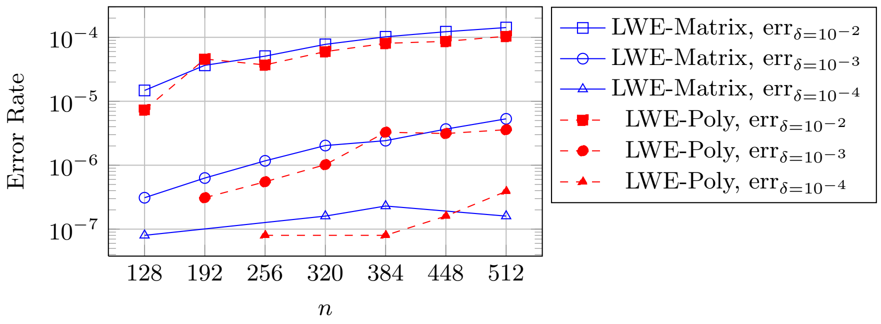


Fig. 9. Rate of bit-errors for LWE-Matrix and LWE-Polynomial for $\delta \in \{10^{-2}, 10^{-3}, 10^{-4}\}$ for 1,600,000 byte plaintext, as detailed in Table 8 and Table 9

Table 8. LWE-Matrix: Bit errors and error rate for a 1,600,000 byte plaintext

	$\delta = 10^{-2}$		$\delta = 10^{-3}$		$\delta = 10^{-4}$		$\delta = 10^{-5}$		$\delta = 10^{-6}$	
n	Errors	%	Errors	%	Errors	%	Errors	%	Errors	%
128	189	0.001477	4	0.000031	1	0.000008	0	0.00	0	0.00
192	467	0.003648	8	0.000063	0	0.00	0	0.00	0	0.00
256	650	0.005078	15	0.000117	0	0.00	0	0.00	0	0.00
320	994	0.007766	26	0.000203	2	0.000016	0	0.00	0	0.00
384	1304	0.010188	31	0.000242	3	0.000023	0	0.00	0	0.00
448	1567	0.012242	47	0.000367	0	0.00	0	0.00	0	0.00
512	1820	0.014219	68	0.000531	2	0.000016	0	0.00	0	0.00

Table 9. LWE-Polynomial: Bit errors and error rate for a 1,600,000 byte plaintext

	$\delta = 10^{-2}$		$\delta = 10^{-3}$		$\delta = 10^{-4}$		$\delta = 10^{-5}$		$\delta = 10^{-6}$	
n	Errors	%	Errors	%	Errors	%	Errors	%	Errors	%
128	94	0.000734	0	0.00	0	0.00	0	0.00	0	0.00
192	586	0.004578	4	0.000031	0	0.00	0	0.00	0	0.00
256	474	0.003703	7	0.000055	1	0.000008	0	0.00	0	0.00
320	766	0.005984	13	0.000102	0	0.00	0	0.00	0	0.00
384	1031	0.008055	42	0.000328	1	0.000008	0	0.00	0	0.00
448	1108	0.008656	40	0.000313	2	0.000016	0	0.00	0	0.00
512	1329	0.010383	46	0.000359	5	0.000039	0	0.00	0	0.00

6 Future Work

Usage of error correcting codes, such as Viterbi[Vit67], in order to overcome decryption errors will be addressed in future work. Another not yet addressed aspect is that error detection itself is not sufficient since it allows for a correlation with the private key if decryption fails; error corection however may interact with the security guarantees of the whole scheme. Applying the central limit theorem enables the precomputation of expected error rates instead of upper bounds, which leads to a better estimation than the upper bound δ.

We consider as additional goals for the hardware implementation to investigate into an architecture which exploits resource sharing in order to reduce the amount of required resources. Further, a hardware version without the use of an FFT is envisaged to quantify the tradeoff between resource utilization and throughput. An investigation on the benefits of constant multipliers, to further improve the design, is part of future work.

Acknowledgements. This work was supported by CASED (`www.cased.de`). We thank Richard Linder for his contributions and fruitful discussions. We also thank the anonymous reviewers for their helpful comments.

References

[ACPS09] Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 595–618. Springer, Heidelberg (2009)

[ADL$^+$08] Arbitman, Y., Dogon, G., Lyubashevsky, V., Micciancio, D., Peikert, C., Rosen, A.: SWIFFTX: A proposal for the SHA-3 standard. In: The First SHA-3 Candidate Conference (2008)

[CLRS09] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. The MIT Press (2009)

[Coo66] Cook, S.A.: On the minimum computation time of functions. PhD thesis. Harvard Univ., Cambridge (1966)

[CT65] Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex fourier series. Mathematics of Computation 19(90) (1965)

[Dev86] Devroye, L.: Non-uniform random variate generation. Springer-Verlag New York (1986)

[Gen09] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC. ACM (2009)

[GPV08] Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: STOC. ACM (2008)

[Gün10] Güneysu, T.: True random number generation in block memories of reconfigurable devices. In: FPT. IEEE (2010)

[HPS98] 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)

[KG04] Kohlbrenner, P., Gaj, K.: An embedded true random number generator for FPGAs. In: ACM/SIGDA FPGA (2004)

[LP11] 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)

[LPR10] Lyubashevsky, V., Peikert, C., Regev, O.: On Ideal Lattices and Learning with Errors over Rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010)

[Lyu09] Lyubashevsky, V.: Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 598–616. Springer, Heidelberg (2009)

[Lyu12] Lyubashevsky, V.: Lattice Signatures without Trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (2012)

[Mic10] Micciancio, D.: Duality in lattice cryptography (2010) (Invited talk)

[MKD11] Majzoobi, M., Koushanfar, F., Devadas, S.: FPGA-Based True Random Number Generation Using Circuit Metastability with Adaptive Feedback Control. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 17–32. Springer, Heidelberg (2011)

[MM09] Markettos, A.T., Moore, S.W.: The Frequency Injection Attack on Ring-Oscillator-Based True Random Number Generators. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 317–331. Springer, Heidelberg (2009)

[Mon85] Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation 44 (1985)

[Paa94] Paar, C.: Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields. PhD thesis, Universität Essen (1994)

[Pei09] Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In: STOC. ACM (2009)

[Reg05] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: STOC. ACM (2005)

[Sho] Shoup, V.: Number theory library (NTL), `http://www.shoup.net/ntl/`

[SS11] 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)

[VD10] Varchola, M., Drutarovsky, M.: New High Entropy Element for FPGA Based True Random Number Generators. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 351–365. Springer, Heidelberg (2010)

[Vit67] Viterbi, A.J.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Transactions on Information Theory 13(2) (1967)

A Mathematical Background

The polynomial reduction can be written as $r(x) = g(x)\ mod\ f(x)$ and represented in matrix notation [Paa94] as a multiplication of $g(x)$ with the reduction matrix $\mathbf{M}$ as follows:

$$\begin{pmatrix} r_0 \\ r_1 \\ \vdots \\ r_{n-1} \end{pmatrix} = \begin{pmatrix} 1\ 0 \cdots 0 & \mu_{0,0} & \cdots & \mu_{0,n-2} \\ 0\ 1 \cdots 0 & \mu_{1,0} & \cdots & \mu_{1,n-2} \\ \vdots\ \vdots \ddots \vdots & \vdots & \ddots & \vdots \\ 0\ 0 \cdots 1 & \mu_{n-1,0} & \cdots & \mu_{n-1,n-2} \end{pmatrix} \cdot \begin{pmatrix} g_0 \\ g_1 \\ \vdots \\ g_{n-1} \\ g_n \\ \vdots \\ g_{2n-2} \end{pmatrix} . \qquad (7)$$

The elements $\mu_{j,i}$ of matrix $\mathbf{M}$ are calculated from

$$\mu_{j,i} = \begin{cases} -f_j & , \text{ for } j = 0, \ldots, n-1; \ i = 0 \\ \mu_{j-1,i-1} + \mu_{n-1,i-1} \cdot \mu_{j,0} & , \text{ for } j = 0, \ldots, n-1; i = 1, .., n-2; \end{cases} \tag{8}$$

where $\mu_{j-1,i-1} = 0$, if $j = 0$. This procedure gets very simple if $f(x)$ is a cyclotomic polynomial of the form $x^n + 1$, with n is a power of 2

$$\mathbf{M} = \begin{pmatrix} 1 & 0 & \cdots & 0 & -1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \ddots & \vdots \\ \vdots & \vdots & & \vdots & 0 & \cdots & 0 & -1 \\ 0 & 0 & \cdots & 1 & 0 & \cdots & 0 & 0 \end{pmatrix} . \tag{9}$$

By means of this simplified matrix, the calculation of the matrix-vector multiplication can be reduced to:

$$\begin{pmatrix} r_0 \\ r_1 \\ \vdots \\ r_{n-1} \end{pmatrix} = \begin{pmatrix} g_0 - g_n \\ g_1 - g_{n+1} \\ \vdots \\ g_{n-2} - g_{2n-2} \\ g_{n-1} \end{pmatrix} . \tag{10}$$

This simplification (10) leads to the fact that the polynomial reduction takes only linear time.