Area Optimization of Lightweight Lattice-Based Encryption on Reconfigurable Hardware

Thomas Pöppelmann
Horst Görtz Institute for IT-Security
Ruhr University Bochum
Bochum, Germany
Email: thomas.poeppe1mann@rub.de

Tim Güneysu
Horst Görtz Institute for IT-Security
Ruhr University Bochum
Bochum, Germany
Email: tim.gueneysu@rub.de

Abstract—Ideal lattice-based cryptography gained significant attraction in the last years due to its versatility, simplicity and performance in implementations. Nevertheless, existing implementations of encryption schemes reported only results trimmed for high-performance what is certainly not sufficient for all applications in practice. To the contrary, in this work we investigate lightweight aspects and suitable parameter sets for Ring-LWE encryption and show optimizations that enable implementations even with very few resources on a reconfigurable hardware device. Despite of this restriction, we still achieve reasonable throughput that is sufficient for many today’s and future applications.

I. INTRODUCTION

Lattice-based cryptography is currently emerging as a promising and efficient alternative to discrete logarithm (elliptic curve cryptography) or factoring-based (RSA) schemes. Sound hardness results, resistance against quantum computers, and versatile average-case problems (e.g., LWE or SIS) have created a lot attention by theoretical cryptographers and cryptanalysts [14]. However, it is still not clear how efficient schemes based on modern lattice assumptions will be in practice.1 While large key sizes can be mitigated by using the more structured ideal lattices [12], there are still implementation challenges to be solved, especially, when tuning schemes for low-cost application scenarios.

In this work we investigate such a low-cost scenario and implement semantically secure ideal lattice-based public key encryption [11], [12] on reconfigurable hardware with only very limited resources. While implementations using FFT-techniques have shown high performance [1], [6], [8], [15] they tend to be too large for lightweight scenarios and very resource constrained applications. Employing available embedded multipliers (DSP) and a result proposed by Brakerski et al. [2] on parameter selection, we are able to provide a decryption circuit implemented with only 32 slices, 1 BRAM, and 1 DSP block of a Xilinx Spartan-6 FPGA. The encryption module is slightly larger — the reason is that the scheme requires costly sampling from a discrete Gaussian distribution. To implement this costly operation we combine rejection sampling with Bernoulli trials in order to evaluate the \( \exp() \) function as proposed in [3]. For the necessary standard deviation of \( \sigma = 3.33 \) we just use 37 slices for the sampler (excluding a random bit source).

1An exception is the NTRU public key encryption scheme [10] which is still considered secure and very efficient but protected by patents and defined in \( \mathbb{Z}_q[x]/(x^n - 1) \).

This approach complements the work of Roy, Vercauteren, and Verbauwhede [17] who proposed a slightly larger sampler for the same application scenario.

II. RING-LWE ENCRYPTION

Current lattice-based cryptography has led to the construction of a huge number of cryptosystems. The most relevant proposals for practice are efficient public-key encryption schemes [11], [12] and signatures [4], [7]. A major advantage of lattice-based encryption is that no quantum algorithms are known to solve the underlying problems in polynomial time — contrary to popular schemes like ECC or RSA [14]. In order to gain reasonable efficiency, ideal lattices [12] are often used as they allow significantly shorter key sizes and faster computation by introducing certain algebraic structure into previously random lattices. A very convenient and well studied problem that can be used to construct encryption systems and (partly) signature schemes is the ring learning with errors (Ring-LWE) problem [12], [16]. In its Hermite normal form defined over ideal lattices in the ring \( R = \mathbb{Z}_q[x]/(x^n+1) \), the problem requires one to decide whether the samples \( (a_1, t_1), \ldots, (a_m, t_m) \in R \times R \) are chosen uniformly random or whether each \( t_i = a_i s + e_i \) where \( s, e_1, \ldots, e_m \) have small coefficients from the (one-dimensional) discrete Gaussian distribution \( D_\sigma \) [12]. The distribution \( D_\sigma \) is defined such that a value \( x \in \mathbb{Z} \) is sampled from \( D_\sigma \) with the probability \( \rho_x(x) = \exp(\frac{-x^2}{2\sigma^2}) \) and \( \sum_{k=\infty}^{\infty} \rho_\sigma(k) \). Note that some authors do not define the discrete Gaussian by the standard deviation \( \sigma \) but instead use the parameter \( s = \sqrt{2\pi\sigma} \).

A simple public key encryption scheme whose semantic security directly follows from the Ring-LWE problem has been introduced in the full version [13] of Lyubashevsky et al. [12]. The scheme (GEN, ENC, DEC) is defined as follows:

- **GEN(a):** Choose \( r_1, r_2 \leftarrow D_\sigma \) and let \( p = r_1 - ar_2 \in R \). 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 \( \mathbf{\bar{m}} = \text{ENC}(m) \in R \),
and compute the ciphertext \( c_1 = a e_1 + e_2, c_2 = p e_1 + e_3 + m \) \( \in R^2 \).

- \( \text{DEC}(c = [c_1, c_2], r_2) \) : Output \( \text{DECD}(c_1, r_2 + c_2) \in \{0, 1\}^n \).

The required operations are Gaussian sampling and polynomial arithmetic, including addition and multiplication of polynomials with \( n \) coefficients where each coefficient is reduced modulo \( q \) and polynomials are reduced modulo \( x^n + 1 \). For encryption, the \( n \)-bit binary message \( m \in \{0, 1\}^n \) has to be encoded into a polynomial \( m \in R \). This is necessary as even after decryption small noise terms \( e_1 r_1 + e_2 r_2 + e_3 \) are present. By using threshold encryption and multiplying each one bit in the message by \( 2^{\frac{k-1}{2}} \) it is later on possible to recover from these errors.\(^2\) Decoding returns a one if a coefficient \( z \) is in the range \( q/4 < z < \frac{q}{2} \) and zero otherwise. Lindner and Peikert [11] proposed parameter sets \( (n, q, s) \) supporting low \((192, 4093, 8, 87)\), medium \((256, 4093, 8, 35)\), and high \((320, 4093, 8, 00)\) security levels. In this context, medium security can be compared to the hardness of breaking the symmetric AES-128 block cipher. Additionally, different parameter sets that allow the usage of straightforward FFT techniques for fast polynomial multiplication have been proposed by Göttert et al. [6]. In recent work Brakerski et al. [2] discovered that \( q \) is not necessarily required to be prime. As a power of two modulus allows a significantly more efficient implementation of modular reduction we propose, based on the work of Brakerski et al., the modified parameter set \((256, 4096, 8, 35)\). We thus also assume a medium security level roughly equivalent to AES-128 for this parameter set.

### III. HARDWARE IMPLEMENTATION

In this section we describe a lightweight implementation of separate encryption and decryption modules for the parameter sets \((256, 4093, 8, 35)\) and \((256, 4096, 8, 35)\) of the scheme described in Section II. We are using a pipelined DSP-enabled schoolbook polynomial multiplier and a very recently proposed method for efficient sampling from a discrete Gaussian distribution using the Bernoulli distribution and small tables (described in [4]).

#### A. Polynomial Arithmetic

For our implementation we used row-wise polynomial multiplication which can be implemented efficiently with just two counters and a \( \log_2 q \times \log_2 q \) modular multiplication core. An advantage over recursive algorithms is the low memory consumption and the immediate modular reduction modulo \( x^n + 1 \). In Figure 1 we give the algorithm used to compute the encryption operation. As we just have one polynomial multiplier we first sample a coefficient of \( e_1 \) (Line 7), compute a row of \( c_1 \) (Line 8), and then a row of \( c_2 \) (Line 12). In every execution of the loop in Line 4 we also sample one coefficient of \( e_2 \) and \( e_3 \). The reasons for mixing the sampling into the polynomial multiplication is that we want to minimize buffers in the sampler. Sampling the complete \( e_1, e_2 \) or \( e_3 \) polynomials at once would require additional storage space or a very fast sampler. The \( \log_2 q \times \log_2 q \) bit multiplier, accumulator, and modulo reduction circuit used in Line 10 and Line 14 is implemented as depicted in Figure 2. For the \( q = 4096 \) case no reduction is necessary as we just need the 12 lowest output bits of the DSP block. For \( q = 4093 \) we observe that \( 2^{12} \mod 4093 = 3 \). So we can write \( x_{23.0} \mod 4093 \equiv 2^{12} x_{23.12} + x_{11.0} \equiv 3 x_{23.12} + x_{11.0} \equiv (x_{23.12} \ll x_{11.0} = x_{23.12} + x_{11.0} \ll x_{23.12} + x_{11.0} \ll x_{23.12} + x_{11.0} (by \ll we denote a shift-left operation). This result can then be reduced into the range \([0, 4092]\) by at maximum two subtractions of \( q \). The implemented circuit is shown in Figure 2 and translates efficiently into hardware. Note that we are using a full \( \log_2 q \times \log_2 q \) bit multiplier, although the values sampled from \( D_\sigma \) are small and only in the range \([-\tau \sigma, \tau \sigma] \) (for the specific parameter set we set \( \tau = 12 \)). In general this would allow a smaller unsign- signed multiplier. However, in this case the reduction of both positive and negative multiplication results would be more complicated. Additionally, we already use a DSP block which natively supports a \( 18 \times 18 \)-bit multiplication without additional

---

\(^2\) There is still a very small possibility for decryption errors. See [6], [8], [11] for more details.
resource usage. A multiplication by a negative number is implemented by choosing the corresponding mode of operation of the DSP block. The mode \( AB + C \) is used for unsigned multiplication and the mode \( (D-A)B+C = (q-A)B+C \) for signed multiplication, respectively. As this is supported directly by the DSP block it does not require further resources (e.g., a multiplexer or subtraction circuit).

In Figure 3 we show the block diagram of the encryption core. We use one dual port 9 Kb Block RAM (BRAM9) to store the public key \( a, p \) (2 \( \cdot \) 256 \( \cdot \) 12 = 6144 bits) and one dual-port BRAM9 to hold temporary variables and the final ciphertext \( c_1, c_2 \) (2 \( \cdot \) 256 \( \cdot \) 12 = 6144 bits). The multiplication is basically controlled by two counters implementing the interleaved multiplication where the state machine can select whether \( c_1, a \) or \( c_2, p \) are accessible (set_h). The DSP block described in more detail in Figure 2 performs the main arithmetic of multiplying coefficients by values sampled from a Gaussian distribution. In order to save a block RAM we store the private key in a 256 \( \times \) 7-bit ROM which is realized using memory LUTs. In case a key update is necessary it is also possible to use a RAM with low overhead. As the decryption circuit requests the ciphertext coefficient by coefficient only one dual-port BRAM9 is necessary.

### B. Gaussian Sampling

When dealing with discrete Gaussians three parameters are important. The standard deviation \( \sigma = \frac{s}{\sqrt{2\pi}} \) (set to \( \sigma = 3.33 \) for \( s = 8.35 \)) describes the shape of the Gaussian distribution. The tail-cut factor \( \tau \) determines the size of excluded intervals at the boundaries that are ignored by the sampler. Thus a Gaussian sampler with tail-cut \( \tau \) will only output integers \( x \in \{ -\tau\sigma, ..., \tau\sigma \} \). The parameter \( \lambda \) determines the precision of the sampler, usually fixed between 80 to 100 bits [4].

Implementing discrete Gaussian sampling with such high precision is challenging. Until very recently, the only approaches being considered were the inversion method (requiring a big precomputed table of \( \tau\lambda\sigma \) bits) and rejection sampling using costly floating-point arithmetic. Lately Roy et al. [17] provided an implementation using the Knuth-Yao method which allows halving the size of the precomputed table. In order to further reduce the need for precomputation, we consider a proposal by Ducas et al. [4]. They propose a simple method to sample according to the Bernoulli distribution \( B_{\exp(-x/f)} \) with very low memory overhead using \( \log_2 (\max (x)) \) precomputed entries with \( \lambda \)-bit precision (see Algorithm 4). The sampling with constant and precomputed biases in Line 4 can be performed by sampling the binary expansion of a uniform number \( r \in [0,1) \) with \( \lambda \) bits of precision and returning one if and only if \( r < c \). To save random bits we evaluate the comparison bit-by-bit instead of sampling a \( \lambda \)-bit \( r \) at once. The general idea of rejection sampling is to choose a uniformly random \( u \in \{-\tau\sigma, ..., \tau\sigma\} \) which is then accepted with a probability proportional to \( \exp(-x^2/2\sigma^2) \). In Figure 5 a sampling algorithm is given which picks a uniformly random \( u \in \{0, ..., \tau\sigma\} \) and uses Algorithm 4 to accept proportional to \( \exp(-u^2/2\sigma^2) \). In order to sample in the range \( \{-\tau\sigma, ..., \tau\sigma\} \) we then reject the output zero \( (u = 0) \) with probability \( \frac{1}{2} \) and sample a sign bit. The number of required table entries is \( \log_2 \tau^2\sigma^2 \) and for \( \sigma = 3.33 \), \( \tau = 12 \) and \( \lambda = 33 \) we get a table size of 880 bits implemented in memory LUTs. Although we have eliminated the need for high precision evaluation of the \( \exp() \) function, we still need \( 2\tau/\sqrt{2\pi} \approx 10 \) trials and thus a high number of uniformly random bits. This is mitigated by using an implementation of the resource efficient Trivium stream cipher as a PRNG that can generate a sufficient amount of randomness to match the speed of the polynomial arithmetic. In order to ensure a constant run-time of the encryption we have implemented a small buffer FIFO (see Figure 3) which holds up to 16 sampled values. They are saved in the range \( [-\tau\sigma, \tau\sigma] \) and then brought into the range \( [0, q - 1] \) at the output of the FIFO.

### IV. Results

Performance results and resource consumption were obtained post-place and route with Xilinx ISE 14.6 and a small Spartan-6 LX9 (speed grade -2) as target device. Synthesis and place-and-route options were optimized for small area and we also utilize the memory LUT capabilities of the device (LUTM). In Table I we give the overall resource consumption of the encryption and decryption cores. Our

---

3 The Bernoulli distribution \( B_c \) outputs one (acceptance) with probability \( c \in [0,1) \) and zero (rejection) otherwise.
Rejection sampling: \( \text{SAMPLE}(\tau, \sigma) \)
1. \( u \leftarrow \text{UNIFORM}(0, \lfloor \tau \sigma \rfloor) \)
2. if \( b = \exp(-u^2/(2\sigma^2)) = 0 \) then restart
3. if \( u = 0 \) then
4. \( b \leftarrow \text{UNIFORM}(0, 1) \)
5. if \( b = 0 \) then restart
6. end if
7. \( b \leftarrow \text{UNIFORM}(0, 1) \)
8. return \((-1)^b\)

Fig. 5: Rejection sampling using Algorithm 4. The \text{UNIFORM}(x, y) algorithm outputs a uniformly random integer \( u \in \{x, \ldots, y\} \).

results demonstrate the significant advantage in applying the result of Brakerski et al. [2] to the design. This is especially striking for the decryption core since the core for \( q = 4096 \) needs only 63 percent of the slices compared to the core for \( q = 4093 \). The size of the encryption core is dominated by the resource consumption of the sampler. Still the Enc-4096 core is 19 slices smaller than the Enc-4093 core. The achieved clock frequencies of 128/144 MHz and 179/189 MHz match the requirements of typical lightweight scenarios and result in 934/1057 and 2700/2849 encryption and decryption operations per second, respectively (each handling \( n \) bits of plaintext). Compared with the speed optimized implementation given in [8] this design is about ten times smaller but still achieves a reasonably high throughput matching the requirements for many applications. In comparison, the costly decryption of the code-based McEliece cryptosystem implemented on a more powerful Xilinx Virtex-5 device [5] is more than ten times larger than our implementation but still slower. An implementation of elliptic curve cryptography (ECC-P233) [9] is equally efficient in terms of area but achieves less throughput. We have also evaluated the Gaussian sampler separately which requires 132 LUTs, 40 flip-flops and 37 slices. The supported frequency for the stand-alone instantiation is 136 MHz. The core requires on average 96 random bits and 144 cycles per sampled value \( x \in \{-\tau\sigma, \ldots, \tau\sigma\} \) \( (\tau = 12, \sigma = 3.33 \text{ and } \lambda = 80) \). The advantage of using the Bernoulli approach for rejection sampling is that it is possible to obtain a design which is 10 slices smaller than recent work by Roy et al. using the Knuth-Yao method [17] at the cost of slightly reduced performance and increased consumption of random bits.

## TABLE I: Resource consumption and performance of our implementation compared to other proposals

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>LUT/LUTM/FFS/LICE</th>
<th>BRAM/ DSP</th>
<th>MHz</th>
<th>Cycles</th>
<th>OPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enc-4093 (S6)</td>
<td>360/36290/114</td>
<td>2/1</td>
<td>128</td>
<td>136986</td>
<td>934</td>
</tr>
<tr>
<td>Enc-4096 (S6)</td>
<td>282/35238/95</td>
<td>2/1</td>
<td>144</td>
<td>136212</td>
<td>1057</td>
</tr>
<tr>
<td>Dec-4093 (S6)</td>
<td>162/18136/51</td>
<td>1/1</td>
<td>179</td>
<td>66304</td>
<td>2700</td>
</tr>
<tr>
<td>Dec-4096 (S6)</td>
<td>94/18937/32</td>
<td>1/1</td>
<td>189</td>
<td>66338</td>
<td>2849</td>
</tr>
<tr>
<td>Enc-7681 (S6) [8]</td>
<td>4121/3513/1434</td>
<td>14/1</td>
<td>160</td>
<td>6861</td>
<td>23320</td>
</tr>
<tr>
<td>Dec-7681 (S6) [8]</td>
<td>4121/3513/1434</td>
<td>14/1</td>
<td>160</td>
<td>4404</td>
<td>36350</td>
</tr>
<tr>
<td>ECC-P233 (S3) [9]</td>
<td>578/244452</td>
<td>&lt;</td>
<td>81</td>
<td>15976331</td>
<td>5</td>
</tr>
</tbody>
</table>

REFERENCES