Lattice-based Encryption Over Standard Lattices in Hardware

James Howe¹, Ciara Rafferty¹, Máire O’Neill¹, Francesco Regazzoni², Tim Günêysu³, and Kevin Beeden⁴

¹ Centre for Secure Information Technologies (CSIT), Queen’s University Belfast, UK. jhowe02@qub.ac.uk, c.m.rafferty@qub.ac.uk, m.oneill@ecit.qub.ac.uk.
² Advanced Learning and Research Institute, Università della Svizzera Italiana, Switzerland. regazzoni@alari.ch.
³ Horst Görtz Institute for IT-Security, Ruhr-University Bochum, Germany. gueneysu@crypto.rub.de.
⁴ Thales UK, Research and Technology, Reading, UK. kevin.beeden@uk.thalesgroup.com.

Abstract. Lattice-based cryptography has gained credence recently as a replacement for current public-key cryptosystems, due to its quantum-resilience, versatility, and relatively low key sizes. To date, encryption based on the learning with errors (LWE) problem has only been investigated from an ideal lattice standpoint, due to its computation and size efficiencies. However, a thorough investigation of standard lattices in practice has yet to be considered. Standard lattices may be preferred to ideal lattices due to their stronger security assumptions and less restrictive parameter selection process. In this paper, an area-optimised hardware architecture of a standard lattice-based cryptographic scheme is proposed. The design is implemented on a FPGA and it is found that both encryption and decryption fit comfortably on a Spartan-6 FPGA. This is the first hardware architecture for standard lattice-based cryptography reported in the literature to date, and thus is a benchmark for future implementations. Additionally, a revised discrete Gaussian sampler is proposed which is the fastest of its type to date, and also is the first to investigate the cost savings of implementing with λ/2-bits of precision. Performance results are promising compared to the hardware designs of the equivalent ring-LWE scheme, which in addition to providing stronger security proofs; generate 1272 encryptions per second and 4395 decryptions per second.

1 Introduction

A recent boom in post-quantum cryptography has seen dynamic growth in many diverse areas of cryptography, such as fully homomorphic encryption and identity based encryption. This has essentially been due to the threat of quantum
computers and quantum reduction algorithms, such as Shor’s [36] algorithm. Current cryptosystems, such as RSA, DSA, elliptic-curve cryptography (ECC), and ECDSA, that protect almost all Internet communications become completely insecure by virtue of quantum reductions. In January 2014, it was revealed that the NSA is funding a $79.7 million research program to build “a cryptologically useful quantum computer” [30]. Cryptography resilient to quantum reductions (or post-quantum cryptography) is generally split into four types:

- **Code-based cryptography** [23], such as the McEliece cryptosystem [21], is based on the hard problem of decoding a random linear code. That is, the secret code-generating matrix is hidden with permutating, scrambling, and noise operations, in which recovery is only possible with the original code-generating matrix. **Hash-based cryptography** [22] is based on the hardness of finding pre-images of cryptographic hash functions, the classical example being the Merkle signature scheme. **Multivariate-quadratic cryptography** [20] is based on the hard problem of finding the solution for a set of multivariate-quadratic systems over a finite field.

- **Lattice-based cryptography** [1, 28] is promising as it offers extended functionality whilst being, at the same time, more efficient for the basic primitives of public-key encryption and digital signature schemes. The hardness of schemes in lattice-based cryptography are usually shown to be as hard as finding the shortest (or closest) vector in a lattice, which is to date resilient to quantum reductions. Lattice-based schemes are now viable for practical instantiations (see e.g., FPGA implementation [25] of [10]); however only NTRU and ring-based schemes have currently been considered, due to the advantages in their theoretical performance.

Initial lattice-based cryptoschemes by Ajtai [1] and Regev [28] were based on the hardness of standard lattice problems and later by Micciancio and Lyubashevsky [19] introduced the concept of ideal lattices. Inherently, the major difference between standard and ideal lattices is their use of keys, meaning that a standard-LWE matrix-vector computation with large dimensions, can be simplified in ring-LWE to a FFT/NTT calculation, which have the potential of turning a quadratic calculation into a quasi-linear one (see discussions in Section 2.1). Although the efficiencies made by the standard-to-ideal transition may significantly alleviate computational resources, ideal lattices may be more susceptible to attacks than standard, due to the use of structured matrices. For instance, the additional cyclic structure of ideal lattices is a potential exploit for an attacker. Indeed, recent research has begun to investigate the security implications of weak instances and algebraic attacks of ring-LWE [8, 12, 4, 9] as well as improved sieving algorithms for finding the shortest vector in an ideal lattice [35, 15, 5, 3]. For example, the Gauss sieve algorithm of Ishiguro et al. [15] reports a 600x speed-up when finding the shortest vector in an ideal lattice in comparison to a random, standard lattice of dimension 128.

Critically, if the practical intention is that of cryptographic long-term security, serious considerations would be needed to evaluate if the benefits of ideal lattice-based security outweigh the weakened confidence in security. For
instance, fixed long-term hardware security such as those on satellites, where re-programming in the field is generally not possible, would benefit from the use of standard lattice-based cryptography as it provides stronger long-term security assurances. Indeed, a scheme with stronger security assumptions is much more appropriate for these types of applications, especially considering that any compromise of the key material and/or algorithms would have devastating consequences to the security of data and/or the satellite itself.

The goal of this paper is to investigate the potential of computing over standard lattices in practice. For this purpose, a hardware architecture of a standard-LWE encryption scheme is proposed. The target platform is a Spartan-6 FPGA; FPGAs are highly suitable for use in cryptography as they are reconfigurable and allow fast prototyping. Additionally, prior ring-LWE hardware designs [14, 26, 32, 27] have either targeted Spartan-6 or Virtex-6 FPGAs and thus targeting a similar platform allows for fair performance comparison. Moreover, the proposed design is targeted to trade off between area and speed, to ensure practicality in real time applications.

More specifically, the main contributions are outlined as follows: the first hardware design of a standard lattice-based cryptoscheme is presented; implementing standard-LWE encryption on a Spartan-6 FPGA, which is designed to balance area and performance. Also, a suitably fast discrete Gaussian sampler is incorporated such that samples can be readily produced in parallel and without delay to the critical path. Additionally, the precision of the discrete Gaussian sampler is halved following the recent work of Saarinen [34], with comparisons made on the implications of adopting said precision or classical precision.

The structure of this paper is as follows: firstly, the LWE problem is introduced as well as the LWE encryption scheme by Lindner and Peikert [17]. Next, an assessment on adopting standard or ring-LWE cryptography is presented. Following this, a hardware architecture of the standard-LWE encryption scheme is presented. Finally, the implementation results are presented and a discussion on the performance of the proposed standard-LWE design in comparison to alternative ring-LWE hardware designs is given.

2 Learning With Errors

The learning with errors (LWE) problem, introduced by Regev [28], is to find the secret-key \( s \), given access to \((A, b) \in \mathbb{Z}_q^{n \times n} \times \mathbb{Z}_q^n\), where the public-key is generated uniformly given the lattice security parameter \( n \) and modulus \( q \), and \( b \equiv As + e \mod q \) where \( e \) is noise added from an error distribution \( \chi \). Usually, \( \chi \) is defined as the discrete Gaussian distribution \( D_{\mathbb{Z}, \sigma} \), which is defined with standard deviation \( \sigma \in \mathbb{R} \), where a value \( x \in \mathbb{Z} \), sampled from \( D_{\sigma} \), is output with probability proportional to \( \rho_{\sigma}(x)/\rho_{\sigma}(Z) \) where \( \rho_{\sigma}(x) = \exp(-\frac{x^2}{2\sigma^2}) \) and \( \rho_{\sigma}(Z) = \sum_{i=-\infty}^{\infty} \rho_{\sigma}(i) \).
2.1 LWE Encryption

Algorithm 1 defines the key generation, encryption, and decryption steps in the LWE encryption scheme by Lindner and Peikert [17].

Algorithm 1 The LWE Encryption scheme [17]

1: procedure Key Generation(A, 1^n)
2: A ← Z_q^{n×n}
3: R_1, R_2 ← D_{2^n}^{t×t}
4: P ≡ R_1 - A · R_2 mod q
5: end procedure

6: procedure Encryption(A, P, m ∈ Σ^t)
7: e_1, e_2, e_3 ← D_{2^n}^{n×σ} · D_{2^n}^{σ×σ} · D_{σ}^{ℓ×σ}
8: ŷ = encode(m)
9: c_1 ≡ e_1^T A + e_2^T mod q
10: c_2 ≡ e_1^T P + e_3^T + ŷ^T mod q
11: end procedure

12: procedure Decryption(c_1, c_2, R_2)
13: m = decode(c_1^T R_2 + c_2)
14: end procedure

The medium parameter set (n, q, σ) = (256, 4093, 3.33) is also taken from the encryption scheme, where the work of Brakerski et al. [6] is used to simplify the modulus q = 4093 to 2^{12} = 4096. This simplification allows for a more efficient implementation, since a modular reduction component is simplified to taking the log_2(4096) = 12 least significant bits. Hence, as stated in [17] and using the aforementioned parameters, the encryption scheme is believed to be at least as secure as AES-128.

2.2 The Pros and Cons of Ideal Lattice-based Cryptography

Ideal lattices are a subset of standard lattices, with the computational property of being related to polynomials via matrices of a certain form. That is, instead of having a matrix A ∈ Z_q^{n×n} that is independently and identically distributed, the matrix is structured in such a way that one column a_i ∈ Z_q^n is chosen, with the other n − 1 columns derived as the coefficient representation of the polynomial a_i x^i in the ring Z_q[x] / ⟨f⟩, for some univariate polynomial [18]. With this construction of matrices, the matrix-vector multiplication A · x corresponds to polynomial multiplications and additions in the ring Z_q^n. This means that the n-dimensional vector-matrix product costs O(n log n) arithmetic operations instead of O(n^2), wherein optimised components such as NTTs can be used.

From this, the ring-LWE problem is defined much like the LWE problem. Given a prime modulus q ≡ 1 mod 2n, random polynomials s, a_1, a_2, ..., a_n, b_1, b_2, ..., b_n ∈ Z_q[x] / ⟨x^n + 1⟩, where b_i ≡ a_i s + e_i mod q, with e_i following
some small error distribution, the goal of the ring-LWE problem is to find \( s \), given access to pairs \( (a_i, b_i) \). Observe here the constraint on the prime modulus \( q \), which is dependent on the lattice dimension \( n \), causing significant restrictiveness when choosing these parameters.

For lattice-based cryptography in general, it can be seen that the shift to ideal lattices is desirable for efficient implementations. However, this move does not allow for a fine-grained parameter selection; essentially this is due to the restriction that the monic polynomial \( f \in \mathbb{Z} \) in the quotient ring \( \mathbb{Z}_q[\mathbf{x}]/\langle f \rangle \) needs to be irreducible, for instance \( f = (\mathbf{x}^n + 1) \) if and only if \( n \) is a power of two. As a consequence, many of the proposed parameter sets for ideal lattices are actually derived from its analogous problem in standard lattices (for instance, parameters in [27] taken from [17]) and some of the sets provide more security than required, which have adverse effects on performance. Parameter selection for standard lattices is better understood and, although there are practical advantages to adopting an ideal lattice-based scheme, there are several reasons for choosing a scheme based on standard lattice problems, such as avoiding the assumption that the security level associated with ideal lattices is equivalent to that of standard lattices. These are demonstrated in the recent examples of weak ring-LWE instances [11, 12, 8] and improved cryptanalysis on ideal lattices [5, 3, 9, 15, 35].

2.3 Standard-LWE Encryption In Practice

Despite the recent research into both software and hardware designs for ideal lattice-based cryptography, there has not yet been an investigation into the performance of standard lattice-based cryptography in hardware. In Section 3, the optimised hardware design of the LWE encryption scheme of Lindner and Peikert [17] is presented, which gains its main efficiencies from the nature of the encryption scheme, its parameters, and targeting the FPGA platform.

The key generation stage is computed off-line with the keys being stored on-device in BRAM, where key storage is optimised such that the public-key is expressed as \( \mathbf{A} = \mathbf{A}_0 \parallel \mathbf{A}_1 \), which means all matrices \( (\mathbf{A}_0, \mathbf{A}_1, \mathbf{P}) \) are exactly the same size, allowing for repetitive matrix-vector computations on-device. The global constant \( \mathbf{A} \leftarrow \mathbb{Z}_{q}^{n \times n} \) is uniformly generated using the PyCrypto package \texttt{Crypto.Random.random}, a cryptographically strong version of Python’s standard random module. The off-line Gaussian noise needed for matrices \( \mathbf{R}_1, \mathbf{R}_2 \leftarrow D_\sigma \) is generated using Sage’s discrete Gaussian sampler [2], where the matrices are used to calculate the public-key, \( \mathbf{P} \equiv \mathbf{R}_1 - \mathbf{A} \cdot \mathbf{R}_2 \mod q \), with the secret-key, \( \mathbf{R}_2 \), being stored for use during decryption.

3 Hardware Optimised LWE Encryption Over Standard Lattices

In this section, the components within the hardware architecture are detailed. The optimised LWE encryption scheme is described in Algorithm 2, where the
SUM operations, as in lines 10 & 18, are the multiply and accumulate (MAC) components. Additionally, Fig. 1 illustrates the high-level architecture of the LWE encryption scheme. A one-time initialisation stage is required to read in keys, as well as generating 256 Gaussian samples for encryption. The core components of the hardware encryption unit are the discrete Gaussian sampling and the arithmetic unit, and are described in Section 3.1 and 3.3 respectively. In decryption, the hardware architecture is similar to encryption, with one exception, that it does not require a discrete Gaussian sampler module, and thus a hardware decryption diagram is omitted. Encoding and decoding, outlined in Section 3.2, are relatively low cost operations in hardware, performed using simple bit shifting and comparison respectively, and therefore are not included in the high level architecture.

\begin{algorithm}
\caption{Encryption($A_0, A_1, P, m \in \{0, 1\}^\ell$)}
\begin{algorithmic}[1]
\State for $i = 0$ to $n - 1$ do
\State $e_1(i) \leftarrow D_n\sigma \quad \leftarrow$ Computed on-the-fly after first encryption.
\EndFor
\State for $k = 0$ to $2$ do \Comment i.e., $A_0$, $A_1$, or $P$. \Comment $SUM = 0$
\If {$k \in \{0, 1\}$} \Comment $SUM = 0$
\For {$j = 0$ to $\ell - 1$} \Comment Computed in parallel to MAC operations.
$e \leftarrow D_n$
\EndFor
\State $c_j = SUM + e \mod q$
\Else \Comment $SUM = 0$
\For {$j = 0$ to $\ell - 1$} \Comment Computed in parallel to MAC operations.
$e \leftarrow D_n$
\EndFor
\State $c_j = SUM + e + \text{\`m}(j) \mod q$
\EndIf
\EndFor
\end{algorithmic}
\end{algorithm}

3.1 Discrete Gaussian Sampling

There are a number of various techniques to consider for discrete Gaussian sample generation, such as the CDT technique [24], the Knuth-Yao algorithm [16], the discrete Zigurrat [7], and Bernoulli sampling [10]. In this research, the technique adopted to generate discrete Gaussian samples in hardware is based on
the Bernoulli sampler of Ducas et al. [10, 25], due to its simplicity and adaptability, and without the use of large pre-computations and long-integer arithmetic. The approach combines the inversion method [24] and the original technique of rejection sampling [13]. The calculation of transcendental functions, seen in the rejection sampling technique, is bypassed by the use of Bernoulli variables. This is then combined with the inversion method, to sample from the so-called binary discrete Gaussian distribution $D_{\sigma_{\text{bin}}}$, where $\sigma_{\text{bin}} = \sqrt{\frac{1}{2 \ln(2)}}$, whose cumulative probabilities are of a special form in binary, meaning that they can be computed on-the-fly whilst also reducing the probability of rejection.

Due to the nature of the discrete Gaussian distribution, that is, in theory, its infinitely long tails and infinitely high precision, there are a number of compromises that need to be made in order to produce a practical (yet secure) implementation. Two parameters ($\lambda, \tau$) are added to the other main parameter $\sigma$, to make a tuple of Gaussian parameters for a practical implementation of a discrete Gaussian sampler:

- The standard deviation $\sigma$ is conventional when using the Gaussian distribution in general, and it governs the distribution’s shape. Since the sampling technique used adopts the Bernoulli approach, the standard deviation is defined such that $\sigma = k\sigma_{\text{bin}}$, hence with $k = 4$, $\sigma = 3.39$.
- The precision parameter $\lambda$ determines the level of precision required for an implementation, requiring the statistical distance between the “perfect” theoretical distribution and the one chosen for practice to be no greater than $2^{-\lambda}$.
The tail-cut parameter $\tau$ is used in calculating how much of the less-heavy tails can be excluded from the practical implementation, for a given security level. That is, with a target security level of $\lambda$-bits, the portion of the tails that are undesired is less than $2^{-\lambda}$. Therefore, the tail-cut parameter can be defined as $\tau = 13.4$, for target security level $\lambda = 128$.

Saarinen [34] recommends that for a target security level of $\lambda$-bits, requirements need no greater precision than $\lambda/2$, arguing that there exists no algorithm that can distinguish between a “perfect” sampler and one with statistical distance $2^{-\lambda/2}$. Accordingly, to demonstrate the resources saved after this adjustment, two sets of results are provided in Table 1 for encryption and decryption, as well as separate implementation results for the discrete Gaussian sampler in Section 4, for $\lambda = 128$ and $\lambda = 64$.

The discrete Gaussian sampler, in this case adopting the Bernoulli technique, is therefore devised to also consider the three practical Gaussian parameters $(\sigma, \lambda, \tau)$. The Bernoulli technique is described in Algorithm 3, which, when sampling according to the Bernoulli distribution $B_{\exp(-x/f)}$ using a small pre-computed table of $\log_2([\max x])$ entries, is calculated with $\lambda$-bits of precision.

**Algorithm 3** The Bernoulli sampling technique of Ducas et al. [10], that is, $B_{\exp(-x/f)}$ for $x \in [0, 2^f)$.

```
1: procedure Bernoulli Sampling
2: Precomp: $c_i = \exp(-2^i/f)$ for $0 \leq i < \ell$
3: for $i = 0$ to $\ell - 1$ do
4:   if $x_i = 1$ then
5:     sample $A_i \leftarrow B_{c_i}$
6:     if $A_i = 0$ then return 0
7:   end if
8: end if
9: end for return 1
10: end procedure
```

The aspect of the Bernoulli technique that requires sampling with a constant bias uses a Trivium instantiation as a PRNG, and samples a random $r \in [0, 1)$ with up to $\lambda$-bits of precision; if $r < c$ returns 1, else 0. Rejection sampling is then used with Algorithm 3 to produce values $x \leftarrow D_{\sigma}$, where the value $x \in \{-\tau\sigma, \ldots, \tau\sigma\}$, derived from another Trivium instantiation, is accepted with probability proportional to $\rho_{\sigma}(x) = \exp(-x^2/(2\sigma^2))$. Conventionally, the uniform value is sampled over the positive and negative axis; however due to the symmetrical property of the distribution, only the positive is required, and therefore $x \in \{0, \ldots, \tau\sigma\}$, with the probability $x = 0$ being halved (due to it being counted twice), where a sign-bit $b \in \{0, 1\}$ is sampled to recover back the negative axis. The table of pre-computed values requires $\lambda \log_2(2.4\tau\sigma^2)$ bits, which, with $\lambda = 128$ consumes 1092 bits and with $\lambda = 64$ consumes 546 bits, in LUTs.
Additionally, and further explained in Section 3.3, the number of cycles per sample performance of the discrete Gaussian sampler is key to ensuring reductions in latency. More specifically, to guarantee the final MAC products are not delayed (< 256 clock cycles) when adding the Gaussian noise, the sampler is chosen such that two Trivium PRNGs are used in parallel, which for one Trivium as in [27] would take $2 \times 144$ clock cycles, whereas here one sample takes on average 12 clock cycles. For a slight increase in area cost, this allows faster sampling and is designed to match the performance of the arithmetic unit, detailed in Section 3.3.

### 3.2 Encoding/Decoding

This stage of the encryption scheme encodes the message $m \in \{0, 1\}^n$. Encoding the message is necessary due to the small noise terms, $e_1^R_1 + e_2^R_2 + e_3^R$, being present after decryption. The encoding of the message is defined such that, for each bit $m$ of the message $m$, encode($m$) = $\bar{m} := m \lfloor \frac{q}{2} \rfloor \in \mathbb{Z}_{q^\ell}$. Decoding is adapted from [17] to work with unsigned integers, hence in this research decode($\bar{m}$) := 1 if $\frac{1}{4}q \leq \bar{m} < \frac{3}{4}q$, and 0 otherwise.

### 3.3 Arithmetic

The computations needed for the scheme are computed in the arithmetic module, which calculates the vector-matrix product as well as the addition of the error vectors and message. Each entry in the ciphertext is computed individually, using two clocked counters corresponding to the row and column addresses of the matrices $A_0$, $A_1$, and $P$. For the multiply-accumulate (MAC) operations, the dedicated DSP48A1 unit is exploited, which on the Spartan 6 FPGA family contains an $18 \times 18$-bit multiplier, an adder, and an accumulator. Additionally, the structural decomposition of the two vector-matrix products into three identical operations allows for ease of repetition, meaning that the component can be used exactly the same way for computing $e_1^R A_0$, $e_1^R A_1$, or $e_1^R P$, since the matrices are exactly the same size $A_0, A_1, P \in \mathbb{Z}_{n \times \ell}^q$. Moreover, since the arithmetic is the bottleneck of the encryption scheme, the discrete Gaussian random number generator and message encoding can be implemented in parallel to the MAC operations.

More specifically, as seen in lines 10 & 18 of Algorithm 2, 256 clock cycles are required for each accumulation computation per index of $j$, during which two Gaussian samples are required: one for the addition of the Gaussian noise, and the other for the error-vector store $e_1$ for the next encryption. For this, a double-buffered store (sometimes called the page-flip method or ping-pong buffering) for the Gaussian noise vector $e_1$ is implemented. The first buffer is used to provide values for the current encryption, where the second is used to accumulate new values, the buffers are then swapped at the end of the encryption cycle. This means that an initial generation of $e_1 \leftarrow D_n^\sigma$ is only needed once, instead of per encryption, saving more than 3000 cycles per encryption. Implementing
the lowest-cost Bernoulli, seen in [27], would require roughly 144 clock cycles per Gaussian sample, which does not satisfy this design architecture, hence a Bernoulli sampler with two Trivium PRNGs is instantiated which improves the sampler performance enough to satisfy the requirement of generating two Gaussian samples per 256 clock cycles. Additionally, since the modulus $q = 2^{12}$ is chosen for use in this research, a modular reduction component is not needed since this operation is equivalent to selecting the 12 least significant bits from the result.

4 Results

Table 1 shows the performance of the proposed hardware design of standard-LWE encryption and decryption. Results are also provided for prior ring-LWE hardware designs [14, 26] and low-area ring-LWE designs [32, 27] for comparison. This design is optimised for practical use, a reasonably low-area design is proposed, balanced with the inclusion of a larger Gaussian sampler to maximise performance. This can be seen by the fact that only one DSP of the FPGA is utilised, but a double-buffered store for the error-vector is used as well as two PRNGs for the discrete Gaussian sampler which significantly decreases latency.

Currently there exists no other hardware designs of standard-LWE in the literature. Thus, the results are compared to existing results achieved for hardware designs of ring-LWE encryption and decryption. The comparison of ring-LWE and standard-LWE is used as a benchmark to measure the practicality of the proposed hardware designs for standard-LWE. It must be noted that a direct comparison of ring-LWE and standard-LWE is somewhat incongruous; ring-LWE inherently requires much smaller key sizes and therefore is expected to consume less hardware resources than a hardware design for standard-LWE. For instance,

Table 1: Post-place and route results of standard-LWE (LWE) and ring-LWE (RLWE) encryption/decryption, using the main parameter set (256, 4096, 3.39), except [14, 26, 32] where $q = 4093$. 

<table>
<thead>
<tr>
<th>Operation &amp; Algorithm</th>
<th>Device</th>
<th>LUT/FF/SLICE</th>
<th>BRAM/DSP</th>
<th>MHz</th>
<th>Cycles</th>
<th>Ops/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>LWE Encrypt ($\lambda = 128$)</td>
<td>S6LX45</td>
<td>6152/4804/1866</td>
<td>73/1</td>
<td>125</td>
<td>98304</td>
<td>1272</td>
</tr>
<tr>
<td>LWE Encrypt ($\lambda = 64$)</td>
<td>S6LX45</td>
<td>6078/4670/1811</td>
<td>73/1</td>
<td>125</td>
<td>98304</td>
<td>1272</td>
</tr>
<tr>
<td>LWE Decrypt</td>
<td>S6LX45</td>
<td>63/58/32</td>
<td>13/1</td>
<td>144</td>
<td>32768</td>
<td>4395</td>
</tr>
<tr>
<td>RLWE Encrypt [14]</td>
<td>V6LX240T</td>
<td>2980/16/144396</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>RLWE Decrypt [14]</td>
<td>V6LX240T</td>
<td>124158/65174</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>RLWE Encrypt [26]</td>
<td>S6LX16</td>
<td>4121/3513/-</td>
<td>14/1</td>
<td>160</td>
<td>6861</td>
<td>23321</td>
</tr>
<tr>
<td>RLWE Decrypt [26]</td>
<td>S6LX16</td>
<td>4121/3513/-</td>
<td>14/1</td>
<td>160</td>
<td>4404</td>
<td>36331</td>
</tr>
<tr>
<td>RLWE Encrypt [26]</td>
<td>V6LX75T</td>
<td>4549/3624/1506</td>
<td>12/1</td>
<td>262</td>
<td>6861</td>
<td>38187</td>
</tr>
<tr>
<td>RLWE Decrypt [26]</td>
<td>V6LX75T</td>
<td>4549/3624/1506</td>
<td>12/1</td>
<td>262</td>
<td>4404</td>
<td>59492</td>
</tr>
<tr>
<td>RLWE Encrypt [27]</td>
<td>S6LX9</td>
<td>282/238/95</td>
<td>2/1</td>
<td>144</td>
<td>136212</td>
<td>1057</td>
</tr>
<tr>
<td>RLWE Decrypt [27]</td>
<td>S6LX9</td>
<td>94/87/32</td>
<td>1/1</td>
<td>189</td>
<td>66338</td>
<td>2849</td>
</tr>
<tr>
<td>RLWE Encrypt [32]</td>
<td>V6LX75T</td>
<td>1349/860/-</td>
<td>2/1</td>
<td>313</td>
<td>6300</td>
<td>49751</td>
</tr>
<tr>
<td>RLWE Decrypt [32]</td>
<td>V6LX75T</td>
<td>1349/860/-</td>
<td>2/1</td>
<td>313</td>
<td>2800</td>
<td>109890</td>
</tr>
</tbody>
</table>
73 Block RAMs are used in this design, which includes key storage (being x128 larger than ring-LWE) message storage, and ciphertext storage. Consequently, larger key sizes also have an effect on the LUTs, FFs, and slices used. However, the results either compete with, or better, many of the ring-LWE implementations. The area resources used in the proposed LWE encryption with $\lambda = 64$ significantly better the implementation of [14], and closely compete with [26]. Also, the ring-LWE designs on the Spartan 6 family of FGPA's closely coincide with the 125 MHz achieved in this implementation. For cycle count and operations per second results, it should be considered that in general x128 more calculations are needed. Despite this, a cycle count of 98304 is only around 14x more for [26, 32] whilst bettering [27], and achieving 1272 operations per second is at most around x40 less than [26, 32] whilst bettering [27].

Comparing results for standard-LWE decryption is difficult, since most implementations assimilate these with encryption results. However, the results compare very well to the ring-LWE decryption results of Pöppelmann and Güneysu [27], whilst significantly improving upon Göttert et al. [14].

Encryption results are also provided with two separate discrete Gaussian samplers, one with classical precision $\lambda = 128$ and the other, incorporating the result of Saarinen [34], with $\lambda = 64$. The impact on the discrete Gaussian sampler shows a saving of 11.1% in LUTs, 11.7% in FFs, and 7.8% in slice registers. The results demonstrate the significance of applying the result of Saarinen [34], although it is expected that the results become more significant for a CDT [24] discrete Gaussian sampler and smaller ring-LWE designs.

With regards to area consumption, the size of the encryption module is affected by the size of the public-keys, with $A_0, A_1, \text{ and } P$ being 1180 kilo-bits (kb) in total (compared to 6 kb for ring-LWE). The secret-key size is 196.6 kb in the decryption module, compared to 3 kb for ring-LWE; however since keys are 6x smaller, with decryption using 80-times less slices and computing more than 3 decryptions per encryption, it has less significance. The ciphertext size for standard-LWE is less than that of ring-LWE, with standard-LWE being 4.6 kb whereas a ring-LWE ciphertext is 6.1 kb.

Separate results are also given for the discrete Gaussian sampler as a stand-alone module, which overall takes 952 LUTs, 1058 FFs, and 322 slice registers for $\lambda = 128$ and 846 LUTs, 934 FFs, and 297 slice registers for $\lambda = 64$, and produces one sample per 12 clock cycles on average. Although this significantly exceeds the requirement of two samples per 256 clock cycles (as in Section 3.3), flexibility is desired to account for practical worst-case samples which can occur, for example, due to an abundance of rejections.

The design improves on Knuth-Yao samplers [33, 32] (16/17 clock cycles) and Bernoulli samplers [27] (144 clock cycles) used for lattice-based encryption schemes in hardware, for $\sigma = 3.33$. The optimised Knuth-Yao design of Roy et al. [31] is able to calculate one sample every 2.5 clock cycles on average, which with 8 bits of randomness (taking 4 clock cycles assuming two Trivium instantiations, as in this implementation), would take on average 6.5 clock cycles.
The area consumed by the proposed hardware architecture of standard-LWE encryption is greater than the other low-area ring-LWE hardware designs; however standard-LWE does not require the additional security assumptions associated with using structured ideal lattices and still fits on a lightweight device. Thus, there exists a trade-off in terms of security and performance when choosing between standard-LWE and ring-LWE schemes.

5 Conclusions

In this research, the first hardware design of a standard-LWE encryption scheme is proposed. Considering that ring-LWE requires only $O(n)$ elements of $\mathbb{Z}$, as opposed to $O(n^2)$, to represent $n$ vectors [29], the standard-LWE results compete with the implementations of [27] and [32]. And in addition to the security benefits associated with standard-LWE, the design fits on the lightweight Spartan-6 FPGA and offers somewhat comparable performance to existing ring-LWE schemes. Thus, due to potential security risks associated with ring-LWE schemes and the practical performance of hardware design of standard-LWE scheme illustrated in this research, standard-LWE schemes should be considered for applications requiring long term security assurances.

Acknowledgment

The authors would like to thank Thomas Pöppelmann and the anonymous reviewers, and wish to acknowledge this work was partially funded by the European Union H2020 SAFEcrypto project (grant no. 644729).

References


