# D5.3 Final Report on Hardware-Optimized Schemes

| Document Identification | | | |
|---|---|---|---|
| **Status** | Final | **Due Date** | 30/04/2020 |
| **Version** | 1.0 | **Submission Date** | 29/04/2020 |

| | | | |
|---|---|---|---|
| **Related WP** | WP5 | **Document Reference** | D5.3 |
| **Related Deliverable(s)** | D5.1, D5.2, D5.5, D5.7, D5.9 | **Dissemination Level(*)** | PU |
| **Lead Participant** | UH | **Lead Author** | Kimmo Järvinen (UH) |
| **Contributors** | KU Leuven, UH, Kudelski | **Reviewers** | Marco Lewandowsky (FUAS)<br>Miha Stopar (XLAB) |


| Keywords: |
|---|
| Functional encryption, hardware, implementation, FPGA |



This document is issued within the frame and for the purpose of the FENTEC project. This project has received funding from the European Union's Horizon2020 under Grant Agreement No. 780108. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.




(*) Dissemination level.-PU: Public, fully open, e.g. web; CO: Confidential, restricted under conditions set out in Model Grant Agreement; CI: Classified, Int = Internal Working Document, information as referred to in Commission Decision 2001/844/EC.

# Document Information

| List of Contributors | |
|---|---|
| Name | Partner |
| Kimmo Järvinen | UH |
| Milad Bahadori | UH |
| Angshuman Karmakar | KU Leuven |
| Jose Maria Bermudo Mera | KU Leuven |
| Josep Balasch | KU Leuven |

| Document History | | | |
|---|---|---|---|
| Version | Date | Change editors | Changes |
| 0.1 | 25/03/2020 | Kimmo Järvinen (UH) | First version |
| 0.2 | 07/04/2020 | Kimmo Järvinen (UH), Angshuman Karmakar (KU Leuven) | Added text |
| 0.3 | 14/04/2020 | Kimmo Järvinen (UH), Milad Bahadori (UH) | Added text |
| 0.4 | 16/04/2020 | Kimmo Järvinen (UH) | Many small changes |
| 0.5 | 23/04/2020 | Jose Maria Bermudo Mera, Angshuman Karmakar, Josep Balasch (KU Leuven) | Updated various parts |
| 0.9 | 23/04/2020 | Kimmo Järvinen (UH) | Review version |
| 1.0 | 29/04/2020 | Kimmo Järvinen (UH) | Final version |

| Quality Control | | |
|---|---|---|
| Role | Who (Partner short name) | Approval Date |
| Deliverable Leader | Kimmo Järvinen (UH) | 29/04/2020 |
| Technical Manager | Michel Abdalla (ENS) | 29/04/2020 |
| Quality Manager | Diego Esteban (ATOS) | 29/04/2020 |
| Project Coordinator | Francisco Gala (ATOS) | 29/04/2020 |

# Table of Contents

# List of Figures

# List of Acronyms

| Abbreviation / acronym | Description |
|---|---|
| AXI | Advanced Extensible Interface |
| BN | Barreto-Naehrig |
| CLB | Configurable Logic Block |
| CP | Cryptoprocessor |
| DDH | Decisional Diffie–Hellman assumption |
| DDR3 | Double Data Rate 3 Synchronous Dynamic Random-Access Memory |
| DMEM | Data Memory |
| DSP | Digital Signal Processing |
| FE | Functional Encryption |
| FPGA | Field Programmable Gate Array |
| FSM | Finite State Machine |
| HW | Hardware |
| IMEM | Instruction Memory |
| I/O | Input/Output |
| LUT | Look-Up Table |
| LWE | Learning With Errors |
| MIFE | Multi-Input Functional Encryption |
| NTT | Number-Theoretic Transform |
| PCP | Pairing Cryptography Processor |
| RLWE | Ring Learning With Errors |
| RSA | Rivest–Shamir–Adleman public-key cryptosystem |
| SoC | System-on-Chip |
| SW | Software |

# Executive Summary

In this deliverable D5.3 “Final Report on Hardware-Optimized Schemes”, we give a description of work that has been done for developing hardware-optimized schemes for functional encryption (FE). D5.3 extends D5.2 “Preliminary Report on Hardware-Optimized Schemes” from May 2019. This deliverable is specifically about the work in Task 5.2. As defined in D5.1 “Security and Trust Models”, we assume full trust for the hardware-optimized schemes. This means that the computing platform is not susceptible to physical attacks because either adversaries do not have an access to it (e.g., is located in a secure environment) or it is certified to provide protection against physical attacks (e.g. a smart card or secure element). Consequently, the focus is in maximizing the efficiency of implementations which in this case means primarily accelerating the computations so that FE schemes can be computed as fast as possible. The work has focused on two different types of FE schemes: (1) schemes using large integer modular arithmetic and (2) schemes based on lattices. For the former, we describe an FPGA-based multi-core architecture for accelerating multi-input FE for inner product computations based on Paillier encryption and provide the final results for it. For the latter, we describe an instantiation of a post-quantum secure single-input FE scheme based on RLWE, detail its current status, and provide research directions towards an efficient hardware-optimized implementation. We describe a potentially new polynomial multiplication strategy to speed up RLWE based FE schemes. Additionally, we provide results for cryptographic pairings which are central building blocks for FE schemes with more expressive functions and additional features.

# 1 Introduction

## 1.1 Purpose of the document

This deliverable D5.3 "Final Report on Hardware-Optimized Schemes" gives a description of work in WP5 of FENTEC and in particular in Task 5.2 of WP5. The deliverable provides details about development of hardware-optimized schemes for functional encryption (FE). This deliverable extends the preliminary report D5.2 from the spring 2019.

As discussed in D5.1, Task 5.2 focuses on the case where the entire computing platform is trusted and emphasis is on the efficiency of implementations. Fig. 1 shows the trust models of WP5 as a recap from D5.1. This deliverable focuses on hardware-optimized schemes shown on the left.

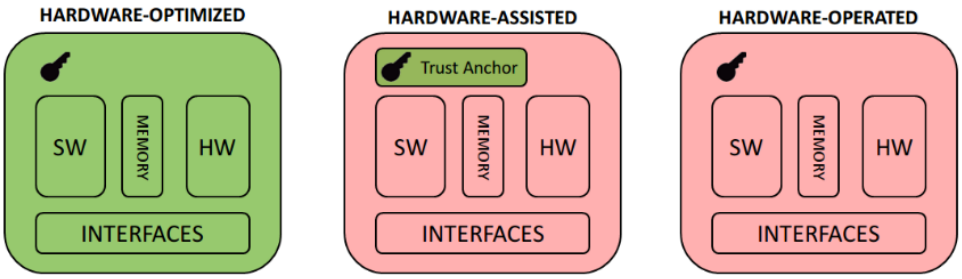


**Figure 1: Underlying security models for the different tasks in WP5. Green (trusted environment), red (untrusted environment).**

This deliverable focuses on two specific FE schemes:

- a multi-input FE scheme based on Paillier encryption proposed in [2, 4] and
- a single-input FE scheme based on learning with errors (LWE) introduced in [1].

For the former, we analyze the scheme, present a multi-core FPGA-based architecture for its efficient computation, and provide performance and area requirement results. For the latter, we analyze the scheme and presents specific plans for its efficient implementation. Additionally, we discuss computation of cryptographic pairings which are central building blocks for FE schemes for more complex functions and extended functionalities. Particularly, we focus on optimal ate pairings on Barreto-Naehrig curves which represent the state of the art of efficient pairings.

## 1.2 Structure of the document

This deliverable is structured as follows. Section 2 provides the preliminaries of the FE schemes and pairings studied in this deliverable and analyzes them from the point of view of how to efficiently compute them with the selected computing platform (e.g., FPGA). Section 3 describes the computing architectures of the current versions of implementations of the FE schemes and pairings discussed in Section 2. The final results for our implementations are given in Section 4. Section 5 ends this deliverable with conclusions and certain plans for the future work.

# 2 Analysis of FE schemes

In this section, we shall briefly introduce the FE schemes that are studied in this deliverable. The details of the schemes will be provided up to the level that is required to understand the hardware related aspects discussed in the following sections; readers interested in further details should consult the other FENTEC deliverables or the original publications (e.g., [2, 4, 1]).

## 2.1 Preliminaries

FE allows a key authority (the owner of a master secret key $msk$) to derive a decryption key $sk_f$ that permits the holder of $sk_f$ and $\mathrm{Enc}(x)$, the encryption of $x$, to learn the value of $f(x)$, but nothing else about $x$. E.g., FE allows calculating certain statistics over an encrypted data set without revealing the actual data.

Single-input FE permits a single user to encrypt a vector of values $\mathbf{x} = (x_0, x_1, \ldots, x_{m-1})$ so that an evaluator, who has $sk_f$, may compute $f(\mathbf{x})$. Multi-input FE [27, 26] contains $n$ slots and allows $n$ different users to encrypt their respective plaintexts $\mathbf{x}_i = (x_{i,0}, x_{i,1}, \ldots, x_{i,m-1})$. The decryption key $sk_f$ permits to compute the value $f(\mathbf{x})$ for $\mathbf{x} = (\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1})$. This makes the scheme very useful for many practical applications such as privacy-preserving data mining and delegated data processing because data to a function can be collected from different users.

## 2.2 Multi-input FE for inner products

While FE constructions for arbitrary polynomial-sized circuits exist (e.g., [48, 23]), they are mostly theoretical constructions that are far from being practical. In this deliverable, we focus on FE schemes that have been designed with efficiency in mind for a limited, but still practically relevant, functionality of computing inner products $\langle \mathbf{x}, \mathbf{y} \rangle$ from encryptions of $\mathbf{x}$. In the context of inner products, a multi-input FE scheme allows the holder of $sk_f$ to compute

$$f_{\mathbf{y}}(\mathbf{x}) = \sum_{i=0}^{n-1} \langle \mathbf{x}_i, \mathbf{y}_i \rangle = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} x_{i,j} y_{i,j}. \tag{1}$$

During the recent years, many schemes for (multi-input) FE for inner products have been presented based on various cryptographic assumptions and features [2, 3, 4, 15, 18, 20]. In this deliverable, we focus on the multi-input FE scheme introduced by Abdalla et al. in [2] that allows building a multi-input FE scheme from any single-input FE scheme without pairings. We consider two instantiations. The first one is based on [2] and uses the single-input FE based on Paillier encryption introduced by Agrawal et al. in [4]. For the sake of brevity, we skip many of the details here and refer interested readers to [2, 4] for details, but we provide the relevant algorithms below in Section 2.3. The second one uses the method described in [1] to instantiate a FE scheme based on LWE. We provide the relevant algorithms and the current status of FENTEC research on this topic in Section 2.4.

**Input**: The encryption key $sk_i = (N_i, g_i, \mathbf{h}_i, \mathbf{u}_i)$ for input $i$ consisting of the composite modulus $N_i$, a generator $g_i \in \mathbb{Z}^*_{N_i^2}$, and two vectors $\mathbf{h}_i = (h_{i,0}, h_{i,1}, \ldots, h_{i,m-1})$ with $h_{i,j} \in \mathbb{Z}_{N_i^2}$ and $\mathbf{u}_i = (u_{i,0}, u_{i,1}, \ldots, u_{i,m-1})$ with $u_{i,j} \in \mathbb{Z}_L$; The plaintext vector $\mathbf{x}_i = (x_{i,0}, x_{i,1}, \ldots, x_{i,m-1})$ with $x_{i,j} \in \mathbb{Z}_\ell$

**Output**: Ciphertext $\mathbf{c}_i = (c_{i,0}, c_{i,1}, \ldots, c_{i,m})$ where $c_{i,j} \in \mathbb{Z}_{N_i^2}$

1 $\mathbf{w} = (w_0, w_1, \ldots, w_{m-1}) \leftarrow (x_{i,0} + u_{i,0}, x_{i,1} + u_{i,1}, \ldots, x_{i,m-1} + u_{i,m-1}) \pmod{L}$

2 $r \leftarrow_R \{0, 1, \ldots, \lfloor N_i/4 \rfloor\}$

3 $c_{i,0} \leftarrow g_i^r$

4 **for** $j = 0$ **to** $m - 1$ **do**

5 $\quad c_{i,j+1} \leftarrow (w_j N_i + 1) h_{i,j}^r$

6 **return** $\mathbf{c}_i = (c_{i,0}, c_{i,1}, \ldots, c_{i,m})$

**Figure 2: Encryption for the input $i$ of the multi-input FE for inner products based on the Paillier encryption [2, adapted from Figs. 1, 3 and 9]**

## 2.3 Multi-input FE for inner products from Paillier encryption

In the following, we will shortly review the encryption and decryption (inner product computation) operations of the multi-input FE scheme for inner products based on Paillier encryption that was proposed by Abdalla et al. in [2]. The scheme is using the single-input FE based on Paillier encryption introduced by Agrawal et al. in [4] in a generic framework for multi-input FE. The major benefit from implementation point-of-view is that the schemes can be implemented without the use of cryptographic pairings which are computationally expensive and cumbersome to implement as will be discussed with more details later in this deliverable.

Let $n$ be the number of inputs (users) and $m$ the length of each user's input $\mathbf{x}_i = (x_{i,0}, \ldots, x_{i,m-1})$ as defined above. Let $\ell$ and $L$ be integers such that the operands of the inner product $x_{i,j}, y_{i,j} \in \mathbb{Z}_\ell$ and the result of the inner product $p \in \mathbb{Z}_L$; hence, it must hold that $L > nm(\ell - 1)^2$ in order to ensure that $p$ is not reduced modulo $L$.

The key authority generates an encryption key $sk_i$ for each user $i$, for $i = 0, \ldots, n-1$, such that it comprises an RSA-like modulus $N_i$, a generator $g_i$, $\mathbf{u}_i = (u_{i,0}, \ldots, u_{i,m-1})$ with $u_{i,j} \in_R \mathbb{Z}_L$, and $h_{i,j} = g^{s_{i,j}}$ for $j = 0, \ldots, m-1$, where $s_{i,j}$ are random values derived from discrete Gaussian distribution. The decryption key for $\mathbf{y}$ is derived for the evaluator such that $sk_{\mathbf{y}} = (z, d_1, \ldots, d_{n-1}, \mathbf{y})$ where $z = \sum_{i=0}^{n-1} \langle \mathbf{u}_i, \mathbf{y}_i \rangle$ and $d_i = \sum_{j=0}^{m-1} y_{i,j} s_{i,j}$. The idea of the multi-input FE scheme from [2] is that, during encryption, the values of the single-input FE schemes are masked with $\mathbf{u}_i$ and, during decryption, this mask is then removed from the overall result by using $z$. The details of the other parameters can be found in [2, 4].

Fig. 2 shows an algorithm that allows the user $i$ to encrypt its input values $\mathbf{x}_i$. The algorithm follows almost directly the encryption process from [4] (also given in [2, Fig. 9]) with the exception that in line 1, the user first masks the values with the masks vector $\mathbf{u}_i$ as described in [2, Figs. 1 and 3]. Because the masked values $\mathbf{w}$ are encrypted instead of $\mathbf{x}_i$, the evaluator can compute the inner product independently for each user input without learning their real values.

Fig. 3 shows an algorithm that allows the evaluator, who possesses the decryption key $sk_{\mathbf{y}}$, to compute the inner product $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=0}^{n-1} \langle \mathbf{x}_i, \mathbf{y}_i \rangle$ from the users' encrypted inputs $\mathbf{c}_i$ for $i = 0, \ldots, n-1$ and the respective weight vectors $\mathbf{y}_i$. One iteration of the for loop (in lines 1–6

**Input**: The decryption key $sk_{\mathbf{y}} = (\mathbf{N}, \mathbf{y}, \mathbf{d}, z)$ for inner product $\langle \mathbf{x}, \mathbf{y} \rangle$ consisting of the composite moduli $\mathbf{N} = (N_0, N_1, \ldots, N_{n-1})$, the weight vectors $\mathbf{y} = (\mathbf{y}_0, \mathbf{y}_1, \ldots, \mathbf{y}_{n-1})$ with $\mathbf{y}_i = (y_{i,0}, y_{i,1}, \ldots, y_{i,m-1})$ and $y_{i,j} \in \mathbb{Z}_\ell$, the decryption keys for individual inputs $\mathbf{d} = (d_0, d_1, \ldots, d_{n-1})$ with $d_i \in \mathbb{Z}$, and the decryption key for inner product $z \in \mathbb{Z}_L$; The ciphertexts $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{n-1})$ where $\mathbf{c}_i = (c_{i,0}, c_{i,1}, \ldots, c_{i,m})$ and $c_{i,j} \in \mathbb{Z}_{N_i^2}$

**Output**: inner product $p \in \mathbb{Z}_L$

1 **for** $i = 0$ **to** $n - 1$ **do**
2 $\quad p_i \leftarrow c_{i,0}^{-1}$
3 $\quad p_i \leftarrow p_i^{d_i}$
4 $\quad$ **for** $j = 0$ **to** $m - 1$ **do**
5 $\quad\quad p_i \leftarrow p_i \cdot c_{i,j+1}^{y_{i,j}}$
6 $\quad p_i \leftarrow \frac{p_i - 1 \bmod N_i^2}{N_i}$
7 $p \leftarrow \sum_{i=0}^{n-1} p_i - z \bmod L$
8 **return** $p$

**Figure 3: Decryption (inner product computation) of the multi-input functional encryption for inner products based on the Paillier encryption [2, adapted from Figs. 1, 3 and 9]**

of Fig. 3) computes the inner product of the user $i$ homomorphically and decrypts the masked result. This directly uses the decryption process of the single-input FE from Paillier encryption from [4] and relies on the fact that Paillier encryption is additively homomorphic which permits computing the encryption of the product $x_{i,j}y_{i,j}$ by raising the encryption of $x_{i,j}$ to power $y_{i,j}$ (in line 5 of Fig. 3). The real value of the inner product of all users' masked inner products can be recovered in the end (in line 7 of Fig. 3) by subtracting the decryption key $z$ from their sum.

## 2.4 Single-input FE for inner products from RLWE

Jointly with XLAB and ENS, Paris we propose a scheme which is inspired by the FE LWE schemes from [1, 4], but based on the ring-LWE assumption from [37]. It allows to encrypt vectors from the $l$-dimensional space with absolute value of coefficients smaller than $B$ and decrypting the inner product of them with a vector with the same bounds.

**Construction:**

- **Setup:** For $n$ a power of two we fix a prime $q$ and ring $R = \mathbb{Z}[x]/(x^n+1)$, i.e. the elements of $R$ are polynomials of degree at most $n-1$ over $\mathbb{Z}$. Denote with $R_q = R/(q\mathbb{Z})$. Fix $\sigma$ such that the ring-LWE decision problem is hard over $R_q$, where the errors and secret are sampled from $D_\sigma$. We uniformly at random sample $a \in R_q$ (sampling coefficient independently) and elements $\{s_i \mid i \in \{1, \ldots, l\}\}, \{e_i \mid i \in \{1, \ldots, l\}\}$ from $R$, by sampling coefficients of $s_i, e_i$ from the discrete Gaussian distribution with standard deviation $\sigma_1 = \sqrt{2l}B\sigma$, denoted by $D_{\sigma_1}$. Then $\{s_i \mid i \in \{1, \ldots, l\}\}$ is a set of secret keys and the public key is $(a, \{pk_i \mid i \in \{1, \ldots, l\}\})$, where $pk_i = as_i + e_i \in R_q$. We also fix $p$ to be a number bigger

than $2B^2l$.

- **Encrypt:** To encrypt a vector $\vec{m} = (m_1, \ldots, m_l) \in \mathbb{Z}^l$ with coefficients from $(-B, B)$ we sample $r$ and $f_0$ by sampling its coefficients independently from $D_{\sigma_2}$ where $\sigma_2 = \sqrt{2}\sigma$, and $\{f_i \mid i \in \{1, \ldots, l\}\}$ by sampling their coefficients independently from $D_{\sigma_3}$ where $\sigma_3 = (\sqrt{2(nl\sigma_2^2 \log(1/\epsilon) + 1)}\sigma_1$. We fix $1_{R_q}$ to be the identity element of $R_q$ and calculate:

$$c_0 = ar + f_0 \in R_q,$$

$$c_i = pk_i r + f_i + (\lfloor q/p \rfloor m_i)1_R \in R_q.$$

  Then $(c_0, \{c_i \mid i \in \{1, \ldots, l\}\})$ is the encryption of $\vec{m}$.

- **KeyGen:** To generate a key that decrypts $\vec{m} \cdot \vec{y}$ for $\vec{y} = (y_1, \ldots, y_n) \in \mathbb{Z}^l$ we calculate

$$s_{\vec{y}} = \sum_{i=1}^{l} y_i s_i \in R.$$

- **Decryption:** To decrypt $(c_0, \{c_i \mid i \in \{1, \ldots, l\}\})$ using $s_{\vec{y}}$ and $\vec{y}$ we calculate

$$d = (\sum_{i=1}^{l} y_i c_i) - c_0 s_y \in R_q.$$

  Then $d$ should be close to $\frac{(\vec{m} \cdot \vec{y})q}{p} 1_R$ (a bit perturbed coefficients) and we can extract $\vec{m} \cdot \vec{y}$.

The above RLWE based inner product scheme has been updated from what was reported in deliverable 5.2. We have changed the construction to make it provably secure. At the time of writing this deliverable we are finalizing the proof of security. We plan to do the following in the coming months.

1. Once the security proofs are finalized we will find suitable parameters for $q, \sigma$, and $n$. Since we changed the construction the old parameters are not valid anymore.
2. A generic C implementation and optimized vectorized implementation using Intel AVX instructions for different levels of security and different values of $B$ and $l$ will be provided.
3. A research paper will be sent to a conference with submission date around August-September, 2020.

## 2.5 Cryptographic pairings and their use in FE

While FE for linear functions (namely, inner products) can be realized from standard assumptions without cryptographic pairings, extending FE to support more complex functions (e.g., quadratic functions) or certain additional features (e.g., function hiding, decentralization, etc.) can be implemented with the use of bilinear maps through cryptographic pairings. Examples of such FE constructions include [2, 3, 12, 13, 18].

A cryptographic pairing is a bilinear map $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3$ where $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive groups and $\mathbb{G}_3$ is a multiplicative group. Many types of cryptographic pairings and pairing-friendly elliptic curves have been proposed in the literature. Because of this, the implementation that will be described in Section 3.3 is designed so that it can be programmed to implement different types

**Input**: $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$.
**Output**: $a_{\text{opt}}(Q, P) = f$, where $f \in \mathbb{F}_{p^{12}}$.
**Constant**: $t \in \mathbb{Z}$ so that $p = 36t^4 + 36t^3 + 24t^2 + 6t + 1$ and $r = 36t^4 + 36t^3 + 18t^2 + 6t + 1$ are primes; and $s = 6t + 2 = \sum_{i=0}^{L-1} s_i 2^i$, where $s_i \in \{-1, 0, +1\}$.

1 $T \leftarrow Q$, $f \leftarrow 1$
2 **for** $i = L - 2$ **to** $0$ **do**
3 $\quad f \leftarrow f^2 \cdot l_{T,T}(P)$; $T \leftarrow 2T$
4 $\quad$ **if** $s_i \neq 0$ **then**
5 $\quad\quad f \leftarrow f \cdot l_{T,s_iQ}(P)$; $T \leftarrow T + s_iQ$
6 $Q_1 \leftarrow \pi_p(Q)$; $Q_2 \leftarrow -\pi_{p^2}(Q)$
7 $f \leftarrow f \cdot l_{T,Q_1}(P)$; $T \leftarrow T + Q_1$
8 $f \leftarrow f \cdot l_{T,Q_2}(P)$; $T \leftarrow T + Q_2$
9 $f \leftarrow f^{(p^{12}-1)/r}$
10 **return** $f$

**Figure 4: Optimal ate pairing over BN curves.**

of pairings and parameters. Despite this, we will focus on implementing optimal ate pairings on a BN curve to keep the discussion simple and focused. In the context of optimal ate pairings on BN curves, $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive groups of points on elliptic curves $E(\mathbb{F}_p)$ and $E(\mathbb{F}_{p^k})$ and $\mathbb{G}_3$ is the multiplicative group of $\mathbb{F}_{p^k}$. The parameters must be chosen so that discrete logarithms in all three groups are infeasible; e.g., for approximately 128-bit security level, we need a 256-bit prime $p$ and $k = 12$.

The algorithm for computing an optimal ate pairing over BN curves is given in Fig. 4. The two main operations in the algorithm are the Miller loop in lines 2–5 and the final exponentiation in line 9. The former consists of elliptic curve arithmetic in $E(\mathbb{F}_{p^2})$ and line evaluations in $\mathbb{F}_{p^{12}}$ that can be interleaved. The latter is an exponentiation in $\mathbb{F}_{p^{12}}$ that can be decomposed into $f^{(p^6-1)(p^2+1)(p^4-p^2+1)/r}$, of which the two first terms can be efficiently computed with Frobenius operators and conjugations. The last term is computationally the most demanding part and is called the hard part.

In this deliverable and, particularly, in the implementation discussed in Section 3.3, we assume that optimal ate algorithm of Fig. 4 is computed by using the subalgorithms from [14]. They used $t = 2^{62} - 2^{54} + 2^{44}$ that enables efficient computation of the Miller loop and the hard part of the final exponentiation while providing 126-bit security level. In [14], $\mathbb{F}_{p^{12}}$ is represented as a tower extension field with the following irreducible binomials:

$$\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2 - \beta), \text{where } \beta = -5 \tag{2}$$
$$\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \xi), \text{where } \xi = u \tag{3}$$
$$\mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[w]/(w^2 - v). \tag{4}$$

Consequently, arithmetic operations in the above fields are computed with series of operations in $\mathbb{F}_p$ and can utilize Karatsuba-like constructions to build the tower field arithmetic.

## 2.6 Requirements for hardware acceleration

As can be seen in Section 2.4, in the case of single-input FE based on RLWE, both encryption and decryption operations require polynomial arithmetic in the ring $R_q$. The polynomial multiplication in $R_q$ is defined as a convolution between two vectors and will be the most expensive operation while addition and rounding are operations performed coefficient-wise and therefore with linear complexity.

During encryption, $(m + 1)$ convolutions of $N$-dimensional vectors must be performed. This will largely determine the performance of encryption. However, all these multiplications can be computed in parallel so they will benefit from a hardware accelerator with multiple arithmetic cores.

During decryption, also $(m + 1)$ convolutions of $N$-dimensional vectors must be computed, although this time the results are not independent but have to be combined to retrieve the output message. Nonetheless, as for encryption, decryption will also benefit from multiple polynomial arithmetic cores running in parallel.

The approach for accelerating lattice-based cryptography with hardware can follow a HW/SW paradigm where only the expensive arithmetic operations, namely the convolution and modular reductions, are performed in the HW side, and the rest of operations as well as the control of the special hardware is carried out in a processor. We have already proven this approach as successful for accelerating lattice-based encryption in our recent publication available at [39] and similar principles can be applied for lattice-based FE. Regarding the polynomial multiplier, given the large dimension of $N$ when compared to lattice-based public key encryption or digital signatures, the architecture shall be based on the NTT rather than other approaches such as Karatsuba or Toom-Cook based multipliers.

Next, we show a survey of the computational requirements for different parties in the multi-input FE setting. In particular, we have studied which parties could benefit from dedicated hardware accelerators.

Fig. 5 shows a pictorial presentation of the setting for multi-input FE for inner products. The key authority generates and distributes the encryption keys to the $n$ users and provides the evaluator with the decryption key $sk_{\mathbf{y}}$. This is a one-time effort and does not require hardware acceleration. Hence, we do not consider it after this point and focus on the computations of the users and the evaluator (e.g., server) from now on in this deliverable.

As can be seen from Fig. 2 and Fig. 3, in the case of multi-input FE based on Paillier encryption, both encryption and decryption operations are computationally involved operations requiring modular multiplications and exponentiations with a large modulus $N^2$.

Encryption (Fig. 2) needs to be computed for all $n$ users, which equals to $(m + 1) \cdot n$ modular exponentiations in total (and several modular multiplications). However, as shown in Fig. 5, they are distributed for different users and, thus, benefit from inherent parallelism. The computational cost of a single user grows linearly with $m$, the length of the user's input vector. Hardware acceleration may be required if $m$ is large and/or data needs to be sent to the evaluator frequently.

Decryption, on the other hand, is computed solely by the evaluator and becomes a significant computational burden, especially, when $n$, the number of users, is large: the evaluator needs to compute exponentiations with large exponents $d_i$ for each user (line 3 of Fig. 3). However, it

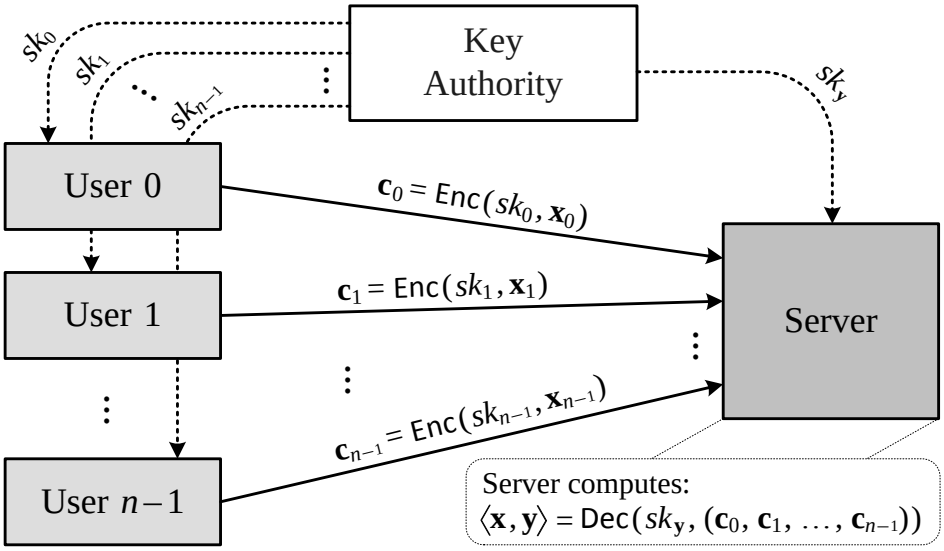


**Figure 5: Multi-input FE for inner products. The grey-colored parties benefit from hardware accelerators.**

is reasonable to assume that the modular exponentiations that are computed with each $y_{i,j}$ (in line 5 of Fig. 3) are a lot cheaper because $y_{i,j}$ are typically small ($y_{i,j} \in \mathbb{Z}_\ell$). Nevertheless, we estimate that the evaluator is the party in Fig. 5 that has the most urgent need for hardware acceleration. Fig. 3 also includes a lot of inherent parallelism because each users' inputs can be processed in parallel before combining them in line 6 of Fig. 3. This motivated us to design a multi-core architecture for accelerating the algorithms of Fig. 2 and Fig. 3 as will be discussed in more details in Section 3.

Pairings are complicated cryptographic computations that are often considered good targets for HW acceleration. There are, however, certain aspects in pairing computatations that hinder designing efficient HW implementations. Arguably, the most important difficulty is the complexity of pairing computations in terms of the number of required algorithms (e.g., [14] utilizes 31 algorithms), which leads to complicated control logic in hardware implementations. This overhead can be mitigated by adapting the HW/SW codesign paradigm where control is taken care of by the SW side and pure computation is done efficiently in the HW side. In optimal ate pairings, the actual computations are based on $\mathbb{F}_p$ arithmetic due to the tower field arithmetic. Hence, the HW side in a HW/SW codesign is based on an efficient $\mathbb{F}_p$ processor that is then controlled by the SW side. It should be noted that similar HW/SW codesign paradigm suits well also for (multi-input) FE computations and, hence, a HW/SW codesign for pairing computations can be integrated to a HW/SW codesign for FE computations in a relatively straightforward way. In the following sections, we will consider FE computation and pairing computation designs mostly in an isolated manner for the sake of clarity. However, in reality, they would most likely be integrated into a single HW/SW codesign to support FE computations that require both large integer modular arithmetic and cryptographic pairings.

# 3 Architectures for the FE schemes

This section details the architectures of the current versions of the prototype implementations of FE schemes discussed in Section 2, that is, based on Paillier encryption and based on RLWE. For the former we describe a complete FPGA-based accelerator architecture, while for the latter we identify its main bottlenecks and sketch how to overcome them. We also describe the efficient, complex and programmable pairing architecture.

## 3.1 FE schemes based on large integer modular arithmetic

We have designed an FPGA-based architecture for accelerating the multi-input FE scheme based on Paillier encryption that was described in Section 2.3. The overall architecture is a HW/SW codesign implemented on a reconfigurable system-on-chip (SoC) that consists of processor cores and a reconfigurable FPGA fabric. In the following, we consider the Xilinx Zynq-7000 all programmable SoCs and, especially, the Xilinx Zynq-7020 xc7z020clg484-1 in the Avnet ZedBoard development kit as the main implementation platform, but the architecture is generic in the sense that it can be used in other reconfigurable SoCs from Xilinx, or even from other manufacturers (e.g., Intel), with minor modifications. In Zynq-7000, the SW side consists of software running on a dual-core ARM Cortex-A9 and the HW side is an Artix-7 FPGA.

Fig. 6 shows the high-level architectural diagram of the accelerator architecture. In the architecture, speed-critical operations (i.e. large integer modular arithmetic) are computed in the HW side while SW side takes care of controlling the HW side and all external peripherals (i.e., the I/O and DDR3 memory) as well as cryptographic tasks that are not speed or security critical. In this deliverable, we will focus mostly on the accelerator architecture that is implemented in the HW side whereas the HW/SW codesign features are discussed more closely in D5.4. We discuss here how the modular multiplications and exponentiations of the algorithms in Fig. 2 and Fig. 3 are implemented on the FPGA. One of the key design requirements for designing the accelerator architecture was to make it flexible in the sense that it can be easily modified to run different FE schemes with different parameters (i.e., either the FE scheme discussed in Section 2.3 with various parameter sizes or even a completely different FE scheme based on large integer modular arithmetic). To achieve this, it was chosen to implement it with a microcode-based architecture where the SW side can configure the HW side to implement different functionalities just by providing new microcodes (i.e., without the need to reprogram the FPGA).

The FPGA side of our architecture is built around a multi-core structure that consists of $M$ clusters each including $N$ cryptoprocessor (CP) cores. In the prototype implementation of our architecture on the ZedBoard, we have selected $M = 6$ and $N = 2$. A CP core is the main element of our architecture that performs the actual large integer arithmetic. The rest of the architecture is, in principle, for controlling the CP cores and providing data to them efficiently. To achieve this, the HW side connects to the SW side through four parallel AXI-ports for data transfers and general-purpose ports for control signals. To reduce the need of data transfers between the HW and SW sides, the architecture has a three-level data memory hierarchy: (1) Local memory in each CP core (i.e., L1), (2) Shared memory for all CP cores in the FPGA (i.e., L2), and (3) SW side memory consisting of an off-chip DDR3 memory and on-chip caches (i.e., L3). The SW side transfers operands from L3 memory into the L1 memory of a CP core before operations and

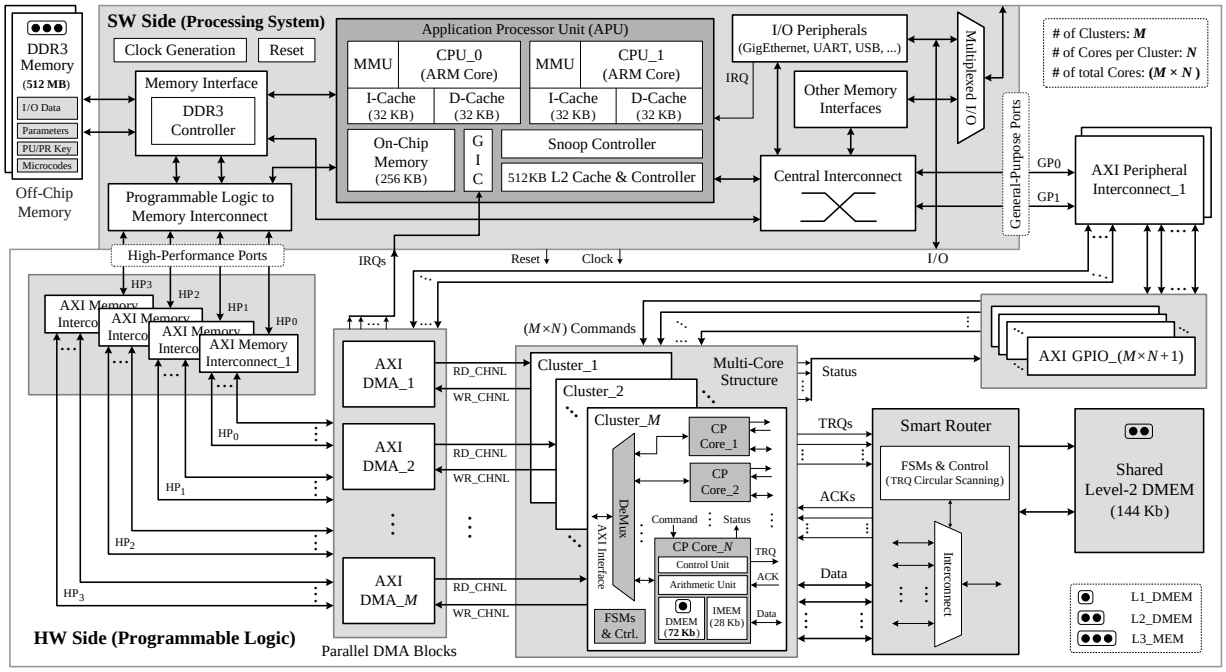


**Figure 6: High-level architecture of HW/SW codesign of our multi-core architecture for multi-input FE from Paillier encryption**

reads the results back in the end. L2 memory enables faster data sharing between the CPs during computations by removing the need to move data to and from the SW side.

Fig. 7 shows the details of the CP core. One CP core consists of an arithmetic unit, a data memory (DMEM) unit (L1), an instruction memory (IMEM) unit, an address generation and control unit, and external interfaces (AXI blocks in Fig. 7). The main component is the arithmetic unit that performs the modular integer arithmetic with (1) a modular multiply-add accumulator unit, which is implemented by using hardwired multipliers in the DSP slices, (2) a modular adder/subtractor unit, and (3) registers and multiplexers. The width of the datapath was chosen to be 72 bits in order to efficiently map it into the DSP slices. DMEM and IMEM were implemented with embedded memory blocks configured as simple or true dual-port RAMs.

The CP core performs modular arithmetic in the Montgomery domain [41] in order to efficiently support multiple moduli. The values of Montgomery reduction that depend on the modulus are precomputed in the SW side and transferred into all CP cores. The main operations needed for multi-input FE computations are modular multiplications and modular exponentiations, and are performed with radix-$2^k$ Montgomery modular multiplication and left-to-right square-and-multiply modular exponentiation, respectively.

Encryptions are performed so that each user computes the algorithm of Fig. 2 independently of other users as shown in Fig. 5. The best way to utilize the parallelism offered by the multi-core architecture is to distribute the $m+1$ modular exponentiations of Fig. 2 to different CP cores. No interaction between the CP cores is needed here because all $m+1$ elements in the ciphertext vector $\mathbf{c}_i$ are computed independently of each other.

Decryptions are executed with the algorithm of Fig. 3 by the evaluator alone. The parallelism of the multi-core architecture could be utilized mainly in two ways: (1) each individual input

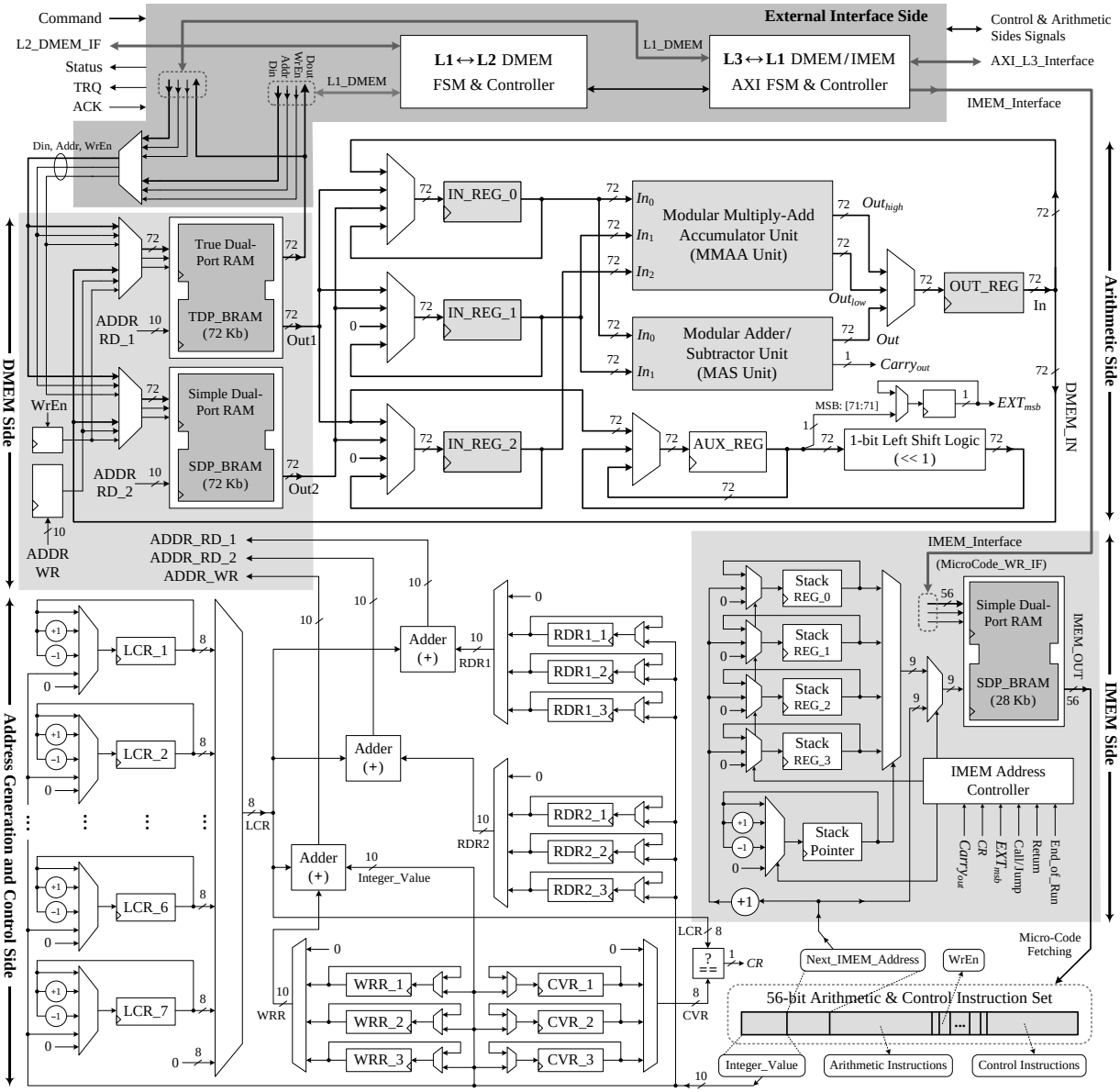


Figure 7: The architectural diagram of the CP core

from the users, which requires the computation of a single-input FE decryption, is distributed to different CP cores or (2) the $m+1$ modular exponentiations required in the single-input FE decryption of each individual input are distributed to different CP cores so that one CP core computes all exponentiations for a specific $j$ for all users $i = 0, \ldots, n-1$. The former is both simpler and more efficient. The latter suffers from the fact that one of the exponentiations (i.e., $p_i^{d_i}$ in line 3 of Fig. 3 that relates to $c_{i,0}$) is the dominant one because the rest (i.e., $c_{i,j+1}^{y_{i,j}}$ for $j = 1, \ldots, m$ in line 5 of Fig. 3) are considerably simpler in practice, as $y_{i,j}$ are likely to be small; we consider 8-bit $y_{i,j}$ for evaluating our prototype in Section 4. Yet another way to utilize parallelism would be to devote different decryptions to different cores, but this is straightforward and beneficial only if there are very large numbers decryptions that must be evaluated simultaneously, and therefore we do not consider it further in this deliverable.

We provide both resource requirements and performance results of this accelerator architecture for computing the multi-input FE scheme from [2] in Section 4.

## 3.2 Lattice-based FE schemes

Lattice-based cryptography builds upon polynomial rings. Therefore, the two critical components for the implementation of lattice-based FE schemes are noise sampling and polynomial arithmetic. We elaborate on these components below.

### 3.2.1 Noise sampling

Sampling from a particular noise distribution is crucial in lattice based cryptography. Although in earlier works this noise used to be sampled from a discrete Gaussian distribution, the current trend is to sample from a centered binomial distribution. Due to their simple operations, binomial samplers are both fast and easy to protect against side-channel attacks. For most cases, changing from Gaussian distributions to binomial distributions does not reduce the security of the cryptosystem.

Unfortunately, as the current constructions of FE schemes are very complex and it is hard to prove their security using binomial distributions, we are bound to use discrete Gaussian distribution. It should also be noted that the standard deviations $\sigma$ of discrete Gaussian distributions used in inner product FE schemes are very high and increases with the length of the vectors and the bound of each element. Some recent works show that it is possible to sample from Gaussian distributions with smaller $\sigma$ [34, 36, 30], but these methods do not scale very well with increasing $\sigma$.

One possible solution to overcome the problem of sampling with large $\sigma$ in constant-time is to use Gaussian samplers with small $\sigma$ as a base sampler and then combine samples to generate samples from discrete Gaussian distributions with larger $\sigma$ [42, 40]. There are some methods available on combining samples from smaller distributions to generate samples from distributions with larger $\sigma$. One such method is using Kullback-Liebler divergence, which is Rényi divergence of order 1 [10] as shown by Pöppelman et al [42]. To generate a sample $x \leftarrow \sigma$, two samples $x_1, x_2 \leftarrow \sigma_1$ are generated, and combined as $x_1 + k_1 x_2$. The $\sigma, \sigma_1$ and $k_1$ are related as $\sigma_1 = \frac{\sigma}{\sqrt{1+k_1^2}}$. The cryptosystem can be proven secure if the Kullback-Leibler divergence of the sampled data created in this way from the actual distribution is $\leq 2^{-S}$ where $S$ is a security parameter.

This splitting can be further extended to one more level. We split $\sigma_1$ such that $\sigma_2 = \frac{\sigma_1}{\sqrt{1+k_2^2}}$. Hence, a sample $x \leftarrow \sigma$ can be generated by 4 samples $x_1,\ x_2,\ x_3,\ x_4\ \leftarrow \sigma_2$ and combined as $x\ = (x_1 + k_2 x_2) + k_1(x_3 + k_2 x_4)$. Such splittings can go on until the final $\sigma$ is small enough to be generated in constant-time as described in [34, 36, 30].

However, over the course of this work some methods have been discovered [51, 31, 52] which can generate samples from large standard deviation and arbitrary centers in constant-time. However, the exact samplers to be used in our samplers can be decided after the parameters have been deduced as mentioned in Sec.2.4.

### 3.2.2 Polynomial arithmetic

The elements of lattice-based cryptosystems are operated with modular arithmetic, typically using a prime modulus, which adds an extra complexity to the implementation. Even for simpler primitives such as key encapsulation mechanisms and digital signatures, the polynomial multiplication has proven to be the bottleneck for the performance of lattice-based cryptography. To achieve a desired level of security (e.g., 128 bits), FE schemes require polynomials of even higher degree as well as larger modulus when compared to simpler cryptographic constructions. This renders polynomial multiplication a critical component in our implementations.

Most commonly, a polynomial is represented by its coefficients so that it can be considered as a vector of length $N$, where $N-1$ is the degree of the polynomial. This representation allows for certain operations such as addition of two polynomials, evaluation of a polynomial at a point or inner product of two polynomials to be performed in linear time $\mathcal{O}(N)$. However, the time complexity of polynomial multiplication, which is the core operation for lattice-based cryptosystems, is quadratic in the number of coefficients $\mathcal{O}(N^2)$ with this representation.

Approaches to improve the time complexity of this operation are built upon representing the polynomial as a set of point-value pairs [19]. The product of two polynomials in the point-value domain is computed as a point-wise product with linear complexity in the number of points $\mathcal{O}(N)$. Therefore, the overall complexity of this method will derive from the complexity of performing the evaluation of the polynomial, or its inverse operation, the interpolation. The most straightforward method is Karatsuba's algorithm [33], where the polynomials are split into two and the final result can be obtained from three multiplications of polynomials with halved length. This algorithm can be applied recursively to get a time complexity $\mathcal{O}(N^{log_2(3)})$. Toom-Cook $k$-way is a generalization of Karatsuba's algorithm where the polynomial is evaluated in $k$ points instead of only two. Finally, if the evaluation points are chosen carefully, a variant of the Fourier transform which works on integer arithmetic, namely NTT, can be utilized to achieve a quasilinear time complexity $\mathcal{O}(N \log N)$. Furthermore, NTT has proven to be particularly efficient when implemented on hardware [43].

The other implementation issue has to do with the modular arithmetic. In order to achieve the desired level of security the modulus $q$ becomes a large prime number. In software, this translates into the need for multi-precision modular arithmetic, which penalizes the performance. In hardware, custom data-width is available, however, the throughput of the system would suffer a deterioration because large values have longer delay chains. To mitigate this issue, we plan to use a Residue Number System for carrying out the modular arithmetic. The basic idea is to select the modulus $q$ as a product of smaller prime moduli $q_i$ and split the computations using the Chinese reminder theorem. A similar idea has already been applied successfully, e.g., to RSA

and fully homomorphic encryption to speed up the non-linear operations [11]. This approach is beneficial on software platforms since it avoids the costly multi-precision arithmetic, but it is even more advantageous on hardware where the computations for each of the smaller moduli can also be parallelized.

Our investigations show that the polynomials used in the RLWE based FE scheme described in Sec. 2.4 can be of length upto 2048. Multiplying such large polynomials is a challenging task. As evident from the above discussion the obvious choice for multiplying such polynomials is NTT polynomial multiplication due to its asymptotical faster running time. The *butterfly* operations of NTT accesses the coefficients of polynomials non-consecutively such as $0-16-32-48-\ldots-2048$ or $0-256-512-768-\ldots-2048$ etc. The impact of such memory accesses are not captured in the asymptotical runtime calculations and often nullifies the computational advantage of NTT as shown in [35].

Our approach to overcome this problem is using Toom-Cook or Karatsuba based methods with NTT. As discussed before the former methods are good at replacing a large polynomial multiplication with multiple smaller polynomial multiplications, *e.g.* a polynomial multiplication of length 2048 can be performed by 3 polynomial multiplications of length 512 or 7 polynomial multiplications of length 256 using Karatsuba and Toom-Cook 4-way respectively with addition of some extra overhead. Recently, in our work [38] we proposed different techniques to improve the Toom-Cook or Karatsuba based polynomial multiplications by reducing different overheads to a great extent. Using these techniques the integration of Toom-Cook and NTT based polynomial multiplication can be made more seamless. We hope that a new multiplication strategy combining Toom-Cook and NTT can be helpful in speeding up RLWE based FE schemes significantly.

## 3.3 Cryptographic Pairings

The main objective in designing the architecture for pairings is to have a compact and programmable (support for many types of pairings and parameters) yet high-performance core for cryptographic pairing computations. Our architecture is constructed as a generic HW/SW codesign and can be instantiated in various programmable SoC with minor modifications. However, in this deliverable, we consider mainly instantiations in Xilinx all-programmable SoC because we use Avnet ZedBoard and Xilinx ZCU102 evaluation kits for prototyping. We will refer to the specific features of those programmable SoC whenever such a distinction is required. Also, to provide programmability and to decrease resource utilization, the HW part of our architecture uses a microprogramming appraoch instead of implementing hardwired FSM for the specific algorithms of pairing computations. It is also noteworthy that this HW/SW codesign is very similar to the architecture for multi-input FE schemes based on large integer modular arithmetic discussed in Section 3.1 and, thus, the pairing implementation can be easily integrated together with it as we shortly discuss later in Section 3.4.

Fig. 8 illustrates the high-level architecture of the HW/SW codesign which is divided into two main parts including SW and HW sides (called PS and PL in Xilinx terminology, respectively). The SW side consists of ARM core(s), on-chip and off-chip (i.e., DDR3) memories, and other interconnection and control. The HW side consists of PCP and supporting modules. The data and control communications between the SW and HW sides are based on the capabilities of the specific programmable SoC, and we use the AXI HP and GP interfaces of Xilinx SoC. The HP interface is employed for high-performance transfer of data and microcodes, and the GP

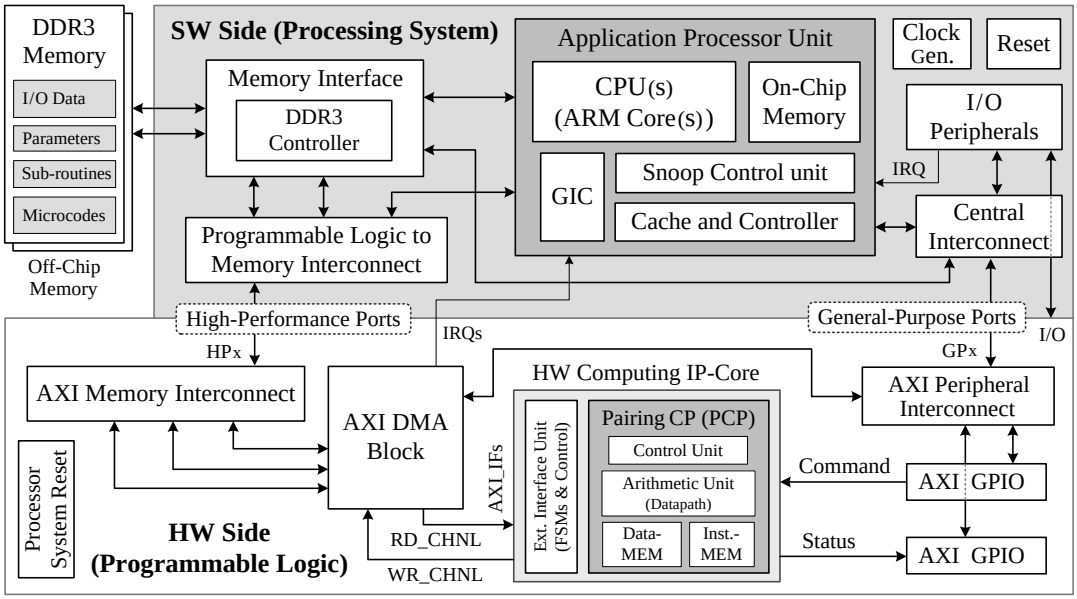


**Figure 8: High level architecture of the HW/SW codesign for the pairing**

interface is used for transferring commands and status (see Fig. 8). The SW side is responsible for controlling the HW side and external peripherals. Specifically, the SW side performs the high-level control and managing of the execution-flow of the pairing computation.

### 3.3.1 Pairing Cryptography Processor (PCP)

The cost of a pairing computation is generally expressed by the total number of required field operations (i.e., multiplications, additions/subtractions, constant-multiplications, and inversions). Moreover, the efficiencies of the architecture and the scheduling technique of field operations are the main factors that determine the overall performance of a pairing implementation [25]. The main objective in designing the PCP is to achieve a good trade-off between programmability, speed, and area requirements and to efficiently utilize the resources of modern FPGA (e.g., DSPs and BRAMs) in implementing base field arithmetic (i.e., arithmetic in $\mathbb{F}_p$). Because the tower extension field arithmetic is ultimately based on $\mathbb{F}_p$ arithmetic, this allows us to efficiently implement different arithmetic operations in $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^4}$, $\mathbb{F}_{p^6}$, and $\mathbb{F}_{p^{12}}$ (tower field arithmetic).

Fig. 9 depicts the architecture of the PCP, which contains external interface, arithmetic (datapath), control, DMEM, and IMEM units. The external interface unit is used for command, status, data, and microcode communication with the external modules. The arithmetic unit performs the arithmetic operations in $\mathbb{F}_p$ with a datapath that consists of three parallel multiplier and two adder/subtractor units. This datapath allows efficient computation of tower field arithmetic. The computations can be arranged so that only multiplications are in the critical path (additions/subtractions are computed simultaneously with multiplications). The IMEM stores microcodes for algorithm(s) that are run in the PCP. The control unit generates addresses for DMEM and makes decisions for loop iterations and conditional statements. The inputs and outputs of the arithmetic unit are connected to DMEM, which stores data that is required during an algorithm run. DMEM is a duplicated $1024 \times 256$-bit true dual-port RAM with two independent read and write ports and supports "4-read", "2-write", or "2-read and 1-write" operations

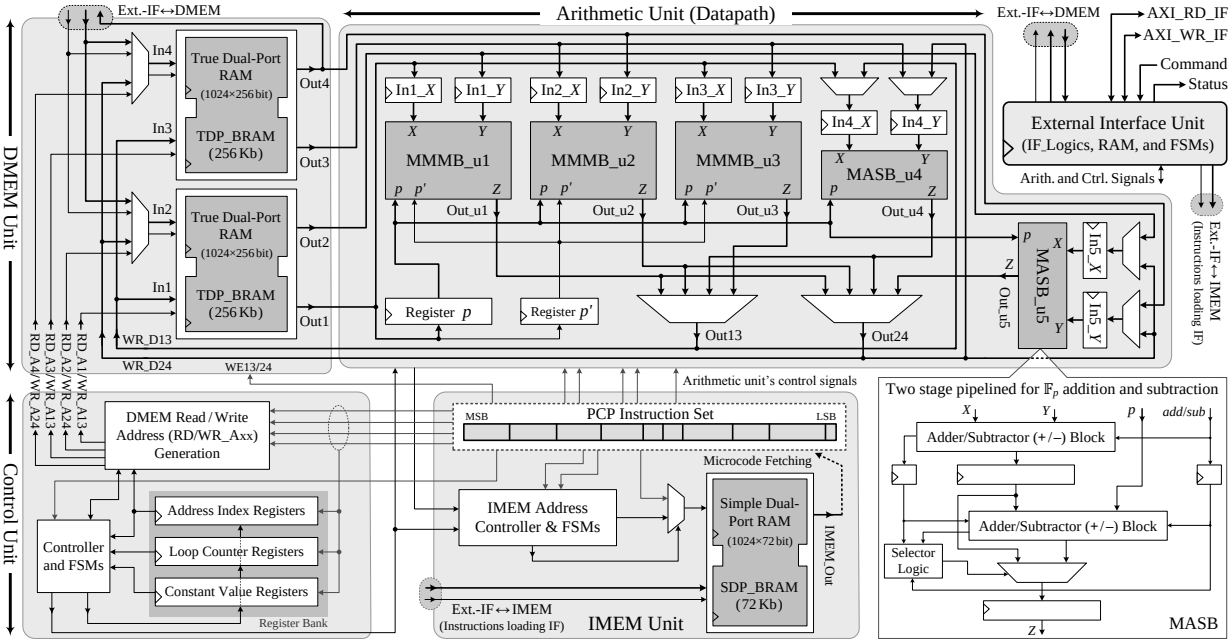


**Figure 9: Architecture of the pairing cryptography processor PCP**

from/to DMEM. This facilitates efficient scheduling and parallelization of $\mathbb{F}_p$ arithmetic. DMEM is also interfaced with the external interface unit for communicating data with the SW side.

## 3.4 Integrating Multi-CP and PCP Cores in a HW/SW Codesign

In this deliverable, we integrated the previous multi-CP core architecture from Section 3.1 and PCP core from Section 3.3 in the same HW/SW codesign to be able to evaluate, implement, and accelerate advanced FE schemes that require both large integer modular arithmetic and cryptographic pairings. The integrated architecture is also organized as a generic HW/SW codesign and can be instantiated in various programmable SoC with minor modifications. However, we consider mainly instantiation in Xilinx all-programmable SoC because we use a Xilinx ZCU102 evaluation kit for prototyping. This evaluation kit includes a Xilinx Zynq UltraScale+ MPSoC ZU9EG chip (i.e., xczu9eg-2ffvb1156e) featuring a quad-core ARM Cortex-A53 processor running up to 1.5GHz in the SW side and a 16nm FinFET+ based FPGA in the HW side. Such a programmable SoC allows a significantly more powerful instance of our HW/SW codesign to be implemented in a single chip. In our first prototype, we considered to separate clock domain for CP cores (i.e., clock domain_1) and PCP core (i.e., clock domain_2) in the HW side. Also, we implemented 16 CP cores (8 clusters with 2 CPs in each cluster) and a PCP core into the FPGA (i.e., HW side). Total percentages of the resource utilization of the HW side (i.e., FPGA) in Xilinx ZU9EG chip are about 26 % of LUTs, 16 % of flip-flops, 42 % of CLBs (each CLB contains one slice and each slice provides eight LUTs and sixteen flip-flops), 14 % of BRAMs, and 12 % of DSPs. Also, the maximum clock frequency of the domain 1 and 2 of the HW side are 184 and 230 MHz, respectively.

Fig. 10 describes the high-level architecture of the integrated HW/SW codesign system. This figure illustrates the block diagram of the HW/SW codesign which is designed in Xilinx Vivado

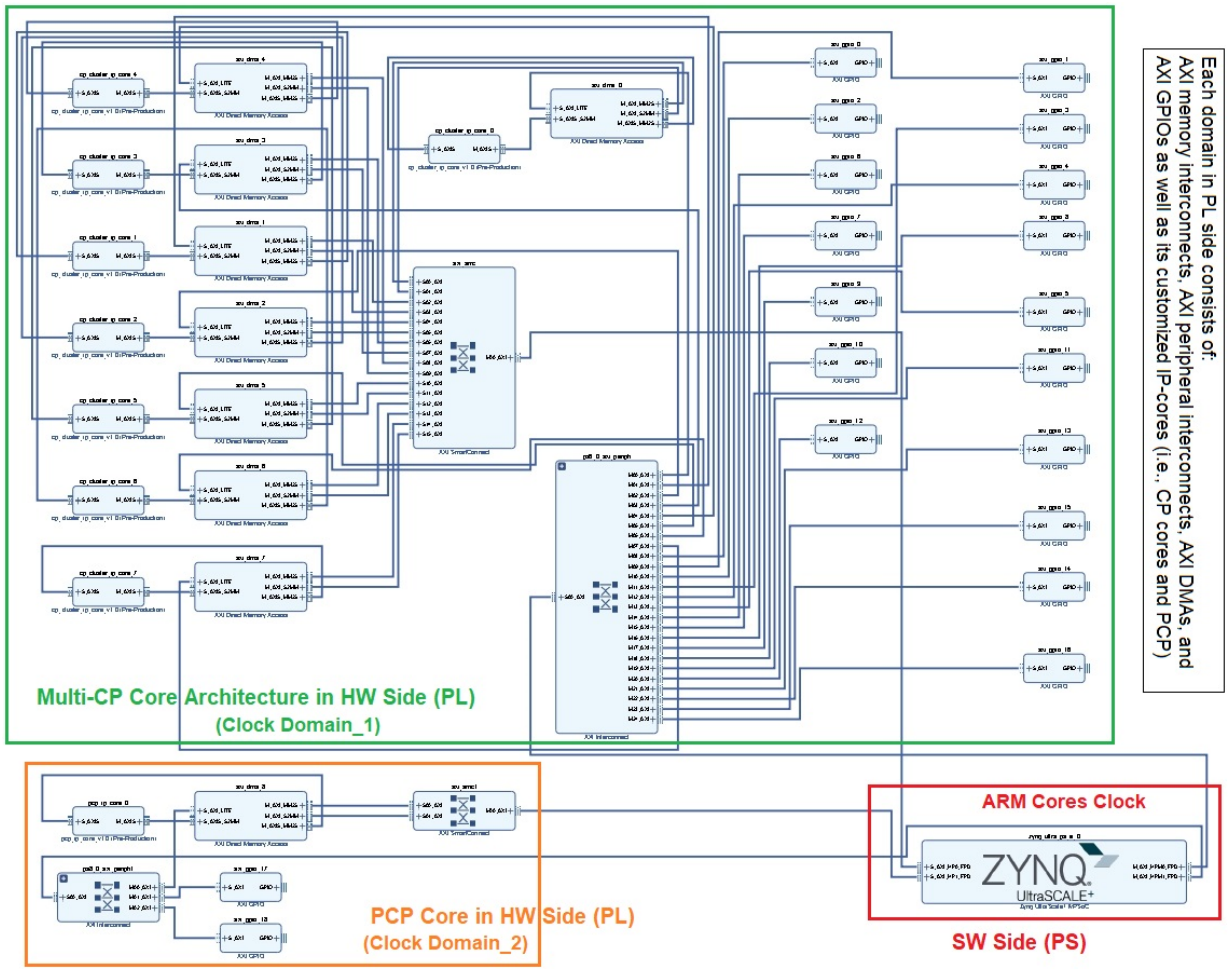


**Figure 10: High-level block diagram of the integrated multi-CP core architecture and PCP core in a HW/SW codesign (high-level blocks and interconnects are shown)**

2019.1 tool. As mentioned before, the HW/SW codesign of the PCP is very similar to the architecture for multi-input FE schemes based on large integer modular arithmetic (i.e., multi-CP core architecture) discussed in Section 3.1 and, thus, the pairing implementation can be easily integrated together with it in a HW/SW codesign. As shown in Fig. 10, the green area is related to the multi-CP core architecture and the orange area is related to the PCP core architecture and both of them are located in HW side (i.e., FPGA). Furthermore, the red area shows the processing system side (i.e., SW side). The described HW/SW codesign is implemented and then, extracted in the Xilinx software development kit (SDK), and in the next step, we are going to develop and extend the SW side programs as well as real-time operating system (RTOS) for a quad-core ARM processors (e.g., preparing hardware platform codes, board support packages, first start boot loader, application routines, and etc.). Finally, we will be able to evaluate and realize the high-level FE schemes in our system.

# 4 Results

This section provides the results of the architectures discussed in Section 3.1 and Section 3.3 for (multi-input) FE using large integer modular arithmetic (in particular, Paillier encryption) and cryptographic pairings, respectively.

## 4.1 Results for multi-input FE from Paillier encryption

The architecture of the multi-input FE scheme based on Paillier encryption that was introduced in Section 3 was compiled for the Xilinx Zynq-7020 xc7z020clg484-1 SoC that is on the selected Avnet ZedBoard development kit. The FPGA area requirements of the architecture are presented in Table 1 both for a single CP core and for the whole architecture with the selected parameters $M = 6$ and $N = 2$. These parameters were chosen because they fully utilize the available resources. They show that the critical resource is the number of available slices as almost all of them (98.47 %) are used by the current architecture. Also, the DSP utilization is high, which was expected because one of the design goals of the architecture was to efficiently utilize them for efficient modular multiplication.

The architecture is generic in the sense that it can be used for many cryptosystems that utilize modular arithmetic with large integers (e.g., RSA). In this deliverable, we focus solely on using it for (multi-input) FE. Table 2 collects performance characteristics of the architecture for different basic modular operations with different modulus sizes.

Table 3 collects the performance results computing the multi-input FE scheme discussed in Section 2.3. We provide the results for three parameter sets: small ($n = 4$, $m = 16$), medium ($n = 16$, $m = 32$), and large ($n = 64$, $m = 64$). For all sets, the security parameter was chosen to be $\kappa = 2048$ and the inner product operands and result sizes to be $\ell = 2^8$ and $L = 2^{32}$, respectively. Performance values are provided for both encryption and decryption (inner product computation). Notice that encryption values are for a single user encrypting his/her input of length $m$ whereas decryption values are for the whole inner product computation with inputs from all $n$ users.

Comparable SW results for the same functionality are not available which complicates performance comparisons between our multi-core FPGA-based architecture and SW implementations. We wrote Python (v. 2.7.10) implementations of the same scheme using standard Python functions (e.g., pow for modular exponentiations) for test vector generation purposes and, in that implementation, the encryption and decryption algorithms provided in Fig. 2 and Fig. 3, respectively, take several seconds to complete with the same parameter sizes. These numbers are not directly comparable because the Python code has not been optimized for performance with similar effort levels as our FPGA implementation. Nevertheless, they imply that our FPGA-based accelerator will be able to deliver major performance advantages for multi-input FE computations.

| Component (#) | | | LUTs (53200) | Registers (106400) | Slices (13300) | BRAMs (140) | DSPs (220) |
|---|---|---|---|---|---|---|---|
| Single cp-core design | | | | | | | |
| External interface unit | | | 73 | 54 | 28 | 0 | 0 |
| Arithmetic unit | | MMAA | 690 | 1295 | 314 | 0 | 16 |
| | | MAS | 152 | 1 | 58 | 0 | 0 |
| | | Other | 178 | 387 | 75 | 0 | 0 |
| Data memory unit | | | 164 | 0 | 86 | 4 | 0 |
| Control unit | | | 327 | 196 | 111 | 0 | 0 |
| Instruction memory unit | | | 197 | 0 | 91 | 1 | 0 |
| Total resource usage | | | 1781<br>3.35 % | 1933<br>1.82 % | 763<br>5.74 % | 5<br>3.57 % | 16<br>7.27 % |
| Multi-cp-core design ($M = 6$, $N = 2$, and # of total cp-cores = 12) | | | | | | | |
| AXI memory interconnects (3) | | Min. | 1969 | 1932 | 755 | 0 | 0 |
| | | Max. | 1976 | 1932 | 822 | 0 | 0 |
| AXI DMAs (6) | | Min. | 2105 | 3490 | 1101 | 3 | 0 |
| | | Max. | 2120 | 3490 | 928 | 3 | 0 |
| Clusters (6) | Min. | CP-core-1 | 1779 | 1933 | 735 | 5 | 16 |
| | | CP-core-2 | 1783 | 1933 | 772 | 5 | 16 |
| | | FSMs | 138 | 109 | 89 | 0 | 0 |
| | Max. | CP-core-1 | 1787 | 1933 | 761 | 5 | 16 |
| | | CP-core-2 | 1775 | 1933 | 758 | 5 | 16 |
| | | FSMs | 143 | 109 | 95 | 0 | 0 |
| Smart router and L2-DMEM | | | 377 | 4 | 200 | 4 | 0 |
| AXI peripheral intercon. (1) | | | 1092 | 1020 | 568 | 0 | 0 |
| AXI GPIOs (13) | | Min. | 55 | 121 | 35 | 0 | 0 |
| | | Max. | 62 | 237 | 71 | 0 | 0 |
| Processor system reset | | | 17 | 19 | 13 | 0 | 0 |
| Total resource usage | | | 42871<br>80.58 % | 53328<br>50.12 % | 13096<br>98.47 % | 82<br>58.57 % | 192<br>87.27 % |

**Table 1: Summary of resource requirements in Xilinx Zynq-7020 xc7z020clg484-1.**

| Operation | Length $\lambda$ (bit) | Latency (# of FPGA side clocks) | Execution Time (single cp-core) FPGA: 1×@122 MHz ARM: @ 666.67 MHz | Max. Throughput (ops) (multi-cp-core) FPGA: 12×@122 MHz ARM: @ 666.67 MHz |
|---|---|---|---|---|
| MA/MS | 1024 | 38 | 0.311 $\mu s$ | 38526312 |
| | 2048 | 66 | 0.541 $\mu s$ | 22181808 |
| | 4096 | 122 | 1.000 $\mu s$ | 12000000 |
| | 8192 | 236 | 1.934 $\mu s$ | 6203388 |
| MR | 1024 | 530 | 4.344 $\mu s$ | 2762256 |
| | 2048 | 1426 | 11.689 $\mu s$ | 1026636 |
| | 4096 | 4394 | 36.016 $\mu s$ | 333180 |
| | 8192 | 15281 | 125.254 $\mu s$ | 95796 |
| MM | 1024 | 617 | 5.057 $\mu s$ | 2372771 |
| | 2048 | 2003 | 16.418 $\mu s$ | 730903 |
| | 4096 | 7127 | 54.418 $\mu s$ | 205416 |
| | 8192 | 27248 | 223.344 $\mu s$ | 53728 |
| ME-1 [a] | 1024 | 950810 | 7.794 $ms$ | 1539 |
| | 2048 | 6159400 | 50.487 $ms$ | 237 |
| | 4096 | 43800644 | 359.022 $ms$ | 33 |
| | 8192 | 334848125 | 2744.657 $ms$ | 4 |
| ME-2 [a] | 1024 | 478231 | 3.920 $ms$ | 3061 |
| | 2048 | 3086904 | 25.302 $ms$ | 474 |
| | 4096 | 21917877 | 179.655 $ms$ | 66 |
| | 8192 | 167456672 | 1372.596 $ms$ | 8 |
| ME-3 [a] | 1024 | 7455 | 61.107 $\mu s$ | 196376 |
| | 2048 | 24081 | 197.385 $\mu s$ | 60794 |
| | 4096 | 85573 | 701.418 $\mu s$ | 17108 |
| | 8192 | 327015 | 2680.451 $\mu s$ | 4476 |

[a] Modular Exponentiation-1, 2, and 3 (i.e., ME-1, ME-2, and ME-3) with exponent length $t = \lambda,\ \lambda/2,$ and 8-bit and Hamming weight of the exponent $HW\text{-}Exp = \lambda/2,\ \lambda/4,$ and 4, respectively, where $\lambda \in \{1024, 2048, 4096, 8192\}$. For all cases, modulus length is $\lambda$ bits.

**Table 2: Performance characteristics of the designs in Xilinx Zynq-7020 xc7z020clg484-1 FPGA, including both single CP-core and multi-CP-core HW/SW co-designed systems.**

| Multi-Input FE Operation (Security parameter $\kappa = 2048$) | | HW/SW Multi-Core System (This work) | | |
|---|---|---|---|---|
| | | Latency (# of clocks) | | Total Execution Time ($ms$) |
| | | FPGA (HW) | ARM (SW) | |
| MIFE Encryption | Small ($m = 16$) | 43863516 | 270504 | 359.943 |
| | Medium ($m = 32$) | 65795274 | 399126 | 539.904 |
| | Large ($m = 64$) | 131590548 | 784992 | 1079.788 |
| MIFE Decryption | Small ($n = 4$, $m = 16$) | 45298890 | 77269998 | 487.207 |
| | Medium ($n = 16$, $m = 32$) | 93706204 | 310868766 | 1234.384 |
| | Large ($n = 64$, $m = 64$) | 299424756 | 1249220622 | 4328.123 |

**Table 3: Performance characterization of the multi-input FE scheme based on the Paillier cryptosystem on the proposed HW/SW multi-core accelerator system.**

| Component | LUTs | Registers | Slices | DSPs | BRAMs |
|---|---|---|---|---|---|
| PCP Core | 8516 | 9641 | 3178 | 36 | 17 |
| MASB | 460 | 474 | 180 | 0 | 0 |
| MMMB | 1941 | 2292 | 822 | 12 | 0 |
| External Interface | 68 | 45 | 21 | 0 | 1 |
| Xilinx IP-Cores | 3987 | 5839 | 1725 | 0 | 3 |
| Total Usage % | 23.6% | 14.5% | 37.0% | 16.4% | 15.0% |

**Table 4: HW Side Resource Utilization of the PCP HW/SW codesign in Zynq-7020 SoC Prototype**

## 4.2 Results for pairing computations

To evaluate the performance of the HW/SW codesign, we implemented it on real hardware using Avnet Zedboard with a low-cost Xilinx Zynq-7020. The target chip includes a dual-core ARM Cortex A9 and an Artix-7 FPGA. For the SW side, we used C++ and Xilinx software development kit for developing software for a real-time operating system (RTOS). For the HW side, we used Verilog (HDL) and Vivado for implementing the design to the FPGA. The resource requirements are summarized in Table 4. The maximum clock frequencies for the FPGA and ARM are 105 and 667 MHz, respectively. Based on Vivado, the total power consumption of the chip is about 1.9W. All results are final post-place&route results and validated with real hardware, unless mentioned otherwise. Table 5 gives the number of clock cycles to compute different parts of Fig. 4 over the $\mathrm{BN}_{126}$ curve from [14] in the FPGA.

In addition to the above primary platform, we implemented the architecture also on other platforms. To demonstrate the generality of our HW/SW codesign and its efficiency on a modern programmable SoC, we implemented it also on a Xilinx Zynq UltraScale+ MPSoC ZU9EG chip featuring a quad-core ARM Cortex-A53 processor running up to 1.5GHz in the SW side and a 16nm FinFET+ based FPGA in the HW side. To enable fair comparisons with other pairing designs, we implemented the HW side on a Xilinx Virtex-6 FPGA device. Table 6 reports the performance characteristics of implementation in the above platforms. The results are for optimal ate pairing implementation over $\mathrm{BN}_{126}$ curve. Also, Table 7 shows a comparative analysis of hardware and software results of optimal ate pairings over BN curves with 126–128-bit security levels. It shows that our design, which occupies only 3072 slices, 36 DSPs, and 18 BRAMs in

| SW/HW | Miller Loop (Lines 2–5) and Lines 7–8 | | | | | Line 6 | Final Exponentiation (Line 9) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| IMEM LD-S/F† | Sqr: $f^2$ $\in \mathbb{F}_{p^{12}}$ | $2T$, $l_{T,T}(P)$ | $T+Q$, $l_{T,Q}(P)$ | Mult: $f \cdot$ $l_{T,X}(P)$ | Miller-Loop‡ | $\pi_p(Q)$, $\pi_{p^2}(Q)$ | $a^{-1} \in$ $\mathbb{F}_p / \mathbb{F}_{p^{12}}$ | $f^t \in \mathbb{F}_{p^{12}} /$ $\mathbb{G}_{\phi_6}(\mathbb{F}_{p^2})$ | Fb: $f^{p^i}$ $\in \mathbb{F}_{p^{12}}$ | $f \cdot g$ $\in \mathbb{F}_{p^{12}}$ | Final Exp.‡ |
| 46 / 1344 | 475 | 502 | 619 | 513 | 101286 | 86 | 11938 / 13511 | 34599 / 22707 | 160 | 703 | 103064 |

† Loading a Segment or Full microcode pack (2.25 or 72 Kbit) into IMEM (IMEM LD-S/F).
‡ Cycle counts of IMEM LD-S/F operations are considered.

**Table 5: Cycle Counts of the HW Side (FPGA) for Different Steps of Optimal Ate Pairing Algorithm (Optimal Ate Algorithm of Fig. 4 over $BN_{126}$ Curve)**

| Design Platform (SoC / FPGA Device) | SW/HW Interface | # IMEM S/F Packs† | PCP Core in HW Side (FPGA) | | | | | SW Side | | Time† (ms) |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | $F_{max}$ MHz | Slices /CLBs | DSPs | RAM | # of Cycles† | $F_{max}$ MHz | # of Cycles† | |
| Zynq-7000 SoC (xc7z020clg484-1) | 64 bit / AXI HP0 | S: 24/F: 8 | 105 | 3199‡ 24% | 36 16.4% | 18 12.8% | 208146 | 667 | 132066 | 2.18 |
| Virtex-6 FPGA (xc6vlx240tff1759-3) | 64 bit / HW cfg. | S: 24/F: 8 | 156 | 3072‡ 8.2% | 36 4.7% | 18 4.3% | 208146 | — | — | 1.33* |
| Zynq UltraScale+ SoC (xczu9eg-ffvb1156-2-e) | 128 bit / AXI HP0 | S: 24/F: 8 | 230 | 1873§ 5.5% | 36 1.4% | 18 2.0% | 202098 | 1200 | 87465 | 0.95 |

† For implementing optimal ate pairing over $BN_{126}$ curve.
‡ Slices: each slice consists of four LUTs and eight flip-flops.
§ CLBs: each CLB contains one slice and each slice provides eight LUTs and sixteen flip-flops.
* Total computation time of the PCP in the HW (FPGA) side without SW side time overhead.

**Table 6: Performance Characteristics of the Programmable HW/SW Codesign System in Different Platforms for Pairing Implementation**

Virtex-6, compares favorably to other designs in respect to flexibility, programmability, and area and still offers comparable speed.

It is common and effective to analyze computational costs of pairing algorithms by expressing the costs of different parts of algorithms with the numbers of $\mathbb{F}_p$ and/or $\mathbb{F}_{p^2}$ arithmetic operations. The costs of additions/subtractions are hidden in our datapath. Furthermore, $\mathbb{F}_{p^2}$ multiplication and squaring have the same cost due to the pipeline scheme. Hence, we can estimate the costs of different steps using only the numbers of $\mathbb{F}_{p^2}$ multiplications/squarings. Because each $\mathbb{F}_{p^2}$ multiplication/squaring contains three parallel $\mathbb{F}_p$ multiplications/squarings (in our datapath) and the design computes an $\mathbb{F}_{p^2}$ multiplication/squaring with a latency $L_M = 38$, we estimate that the total number of clock cycles of a pairing algorithm is as follows:

$$T = \lceil C \times ( \lceil M_p / 3 \rceil \times L_M + I_p \times L_I ) \rceil, \tag{5}$$

where $M_p$ and $I_p$ are the numbers of multiplications and inversions in $\mathbb{F}_p$, $L_I$ is the latency of an inversion in $\mathbb{F}_p$, and $C$ is the overhead of loading microcode packs. Based on our experiments, $C \approx 1.1$. For the optimal ate pairing from [14] considered in this work, we have $M_p = 14300$, $I_p = 1$, and $L_I = 11938$ and (5) gives $T = 212393$. The measurements from real hardware show that real number is 208146 clock cycles and, hence, the estimate given by (5) has an error of

| Reference | Platform | Flexibility | Resource or Area (Slices / DSPs / BRAMs) | $F_{max}$ (MHz) | # of Cycles ($\times 10^3$) | Tot. Time (ms) |
|---|---|---|---|---|---|---|
| This work | Virtex-6 | Yes | 3072 / 36 / 18 | 156 | 208 | 1.33 |
| [25] | Virtex-6 | No | 5163 / 144 / 21 | 166 | 62 | 0.38 |
| [17] | Virtex-6 | No | 7032 / 32 / 22 | 250 | 143 | 0.57 |
| [22] | Virtex-6 | No | 4014 / 42 / 5 | 210 | 245 | 1.17 |
| [24] | Virtex-4 | Yes | 52K / – / – | 50 | 821 | 16.4 |
| [45] | Virtex-6 | No | 9476 / – / – | 145 | 80 | 0.56 |
| [16] | Virtex-6 | No | 17560 / 11 / – | 225 | 407 | 1.80 |
| [47] | Virtex-6 | No | 45K / 128 / – | 103 | 395 | 3.83 |
| [49] | Virtex-6 | No | 7032 / 32 / 48 | 250 | 166 | 0.66 |
| [50] | Virtex-6 | No | 5237 / 64 / 41 | 210 | 78 | 0.34 |
| [46] | Virtex-6 | No | 5570 / 30 / – | 225 | 80 | 0.35 |
| [29] | Virtex-6 | No | 4380 / 131 / – | 125 | 283 | 2.26 |
| [44] | Zynq 7020 SoC | Yes | 598 / – / – | 324 | — | 134 |
| [28] | ASIC | No | 323K Gates | 633 | 330 | 0.52 |
| [21] | ASIC | No | 183K Gates | 204 | 593 | 2.91 |
| [32] | ASIC | Yes | 97K Gates | 338 | 5,340 | 15.8 |
| [14] | Core i7 Intel | Yes | — | 2,800 | 2,330 | 0.83 |
| [5] | Phenom II AMD | Yes | — | 3,000 | 1,562 | 0.52 |
| [6] | Opteron II AMD | Yes | — | 2,400 | 1,500 | 0.62 |
| | ARM Cortex A15 | Yes | — | 1,700 | 6,089 | 3.58 |

**Table 7: Performance Comparison of HW and SW Implementations of Optimal Ate Pairing over BN curves (126-128 bit Security)**

about 2 %. Table 8 collects estimates of computational costs of different pairing algorithms in our HW/SW codesign by using (5) and the $\mathbb{F}_p$ operation counts available in [32].

| Pairing over $BN_{128}$ curve | # of Mult./Squr. $\in \mathbb{F}_p$ | # of Add./Sub. $\in \mathbb{F}_p$ | # of Inversion $\in \mathbb{F}_p$ | # of FPGA Cycles | Total Time† (ms) |
|---|---|---|---|---|---|
| Optimal ate pairing | 17,913 | 84,956 | 3 | 288,984 | 1.852 |
| Ate pairing | 25,870 | 121,168 | 2 | 386,747 | 2.479 |
| $\eta$ pairing | 32,155 | 142,772 | 2 | 474,318 | 3.040 |
| Tate pairing | 39,764 | 174,974 | 2 | 580,323 | 3.720 |
| Compressed $\eta$ | 75,568 | 155,234 | 0 | 1,052,942 | 6.750 |
| Compressed Tate | 94,693 | 193,496 | 0 | 1,319,417 | 8.458 |

† Total computation time is estimated based on the FPGA cycle counts in Virtex-6 device.

**Table 8: Total Calculation Time of Various Pairings on BN Curves (from [32])**

# 5 Conclusions and future work

In this deliverable and in WP5 of FENTEC, the work has focused on FE schemes based on both large integer modular arithmetic as well as on cryptographic pairings that are used in many advanced FE schemes that support more complicated functionalities (e.g., quadratic functions) or additional features. The most significant amount of work in Task 5.2, which is the main focus of this deliverable, was devoted to multi-input FE schemes based on large integer arithmetic (particularly, Paillier encryption) and to cryptographic pairings.

We have developed an FPGA-based multi-core architecture for large integer modular arithmetic described in Section 3 and, as we showed in Section 4, that it can be used for efficient computation of a multi-input FE scheme based on Paillier encryption proposed by Abdalla et al. in [2]. The results given in Section 4 are promising and indicate that the multi-core architecture can offer significant speedups for users encrypting their inputs as well as evaluators computing the inner products. For the latter scheme, we have described our current efforts in instantiating a single-input FE scheme based on RLWE. We have enumerated the challenges we are currently facing and presented research directions towards achieving an efficient implementation. We also described an architecture for efficient computations of cryptographic pairings. It is compact and supports multiple types of pairings and parameters and still achieves performance levels comparable to larger and less programmable cores available in the literature. We also showcased how the architecture for FE schemes based on large integer modular arithmetic and the architecture for cryptographic pairings can be integrated into a single HW/SW codesign in order to support FE schemes that require both large integer modular arithmetic and pairings.

The next steps to be taken with the multi-core architecture for large integer modular arithmetic described in Section 3.1 is to combine it with the pairing cryptography processor described in Section 3.3 and to use it for implementing FE schemes with more expressive functionalities and/or advanced features. The potential of using the resulting hardware accelerated FE implementation for speeding up selected FENTEC use cases will be surveyed (e.g., how much speed-up can be expected). Furthermore, it will be considered whether it makes sense to integrate the FE implementation into the prototype(s) of the use case(s) for demonstration purposes. The work related to multi-core architecture for large integer modular arithmetic and pairings has resulted in a number of manuscripts for scientific articles discussing the use of the architectures described in this deliverable to different multi-input FE schemes or schemes for certain privacy enhancing technologies [7, 8, 9]. These manuscripts have been submitted to different venues.

In this period, we have been working together with XLAB and ENS on a new semantically secure RLWE based FE scheme. Also, as a first approach towards hardware acceleration and to build know-how, we have implemented an arithmetic co-processor for lattice-based key encapsulation [39]. Also, we have very efficient methods for Toom-Cook multiplications [38] that can be used to break multiplications of large polynomials into multiplications of smaller polynomial which are both more efficient and easier to implement on both hardware and software platforms.

In the next steps, we will deduce efficient parameters for this FE scheme. Now we are working on selecting efficient multiplication and noise sampling algorithms that will be used in the RLWE based FE scheme. In near future we plan to create an efficient multi-core vectorized implementation for the scheme, with support for different levels of security, different lengths of vectors with different bounds. These implementations will be lastly ported to dedicated hardware libraries which can utilize massively parallel hardware such as GPUs and FPGAs.

# References


[1] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *Public-Key Cryptography – PKC 2015*, pages 733–751, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg. (Pages 6, 7, and 9)

[2] Michel Abdalla, Dario Catalano, Dario Fiore, Romain Gay, and Bogdan Ursu. Multi-input functional encryption for inner products: Function-hiding realizations and constructions without pairings. In *Advances in Cryptology—CRYPTO*, volume 10991 of *LNCS*, pages 597–627. Springer, 2018. (Pages 3, 6, 7, 8, 9, 10, 17, and 30)

[3] Michel Abdalla, Romain Gay, Mariana Raykova, and Hoeteck Wee. Multi-input inner-product functional encryption from pairings. In *Advances in Cryptology—EUROCRYPT*, volume 10210 of *LNCS*, pages 601–626. Springer, 2017. (Pages 7 and 10)

[4] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In *Advances in Cryptology—CRYPTO*, volume 9816 of *LNCS*, pages 333–362. Springer, 2016. (Pages 6, 7, 8, and 9)

[5] Diego F Aranha, Koray Karabina, Patrick Longa, Catherine H Gebotys, and Julio López. Faster explicit formulas for computing pairings over ordinary curves. In *Advances in Cryptology — EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 48–68. Springer, 2011. (Page 28)

[6] Reza Azarderakhsh, Dieter Fishbein, Gurleen Grewal, Shi Hu, David Jao, Patrick Longa, and Rajeev Verma. Fast software implementations of bilinear pairings. *IEEE Transactions on Dependable and Secure Computing*, 14(6):605–619, 2015. (Page 28)

[7] Milad Bahadori and Kimmo Järvinen. Compact and programmable yet high-performance SoC architecture for cryptographic pairings. Submitted, 2020. (Page 30)

[8] Milad Bahadori and Kimmo Järvinen. A programmable SoC based accelerator for privacy enhancing technologies and functional encryption. Submitted, 2020. (Page 30)

[9] Milad Bahadori and Kimmo Järvinen. A programmable SoC implementation of the DGK cryptosystem for privacy-enhancing technologies. Submitted, 2020. (Page 30)

[10] Shi Bai, Adeline Langlois, Tancrède Lepoint, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: Using the rényi divergence rather than the statistical distance. In Tetsu Iwata and Jung Hee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015: 21st International Conference on the Theory and Application of Cryptology and Information Security,Auckland, New Zealand, November 29 – December 3, 2015, Proceedings, Part I*, pages 3–24. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. (Page 17)

[11] Jean-Claude Bajard, Julien Eynard, Anwar Hasan, and Vincent Zucca. A full rns variant of fv like somewhat homomorphic encryption schemes. Cryptology ePrint Archive, Report 2016/510, 2016. https://eprint.iacr.org/2016/510. (Page 19)

[12] Carmen Elisabetta Zaira Baltico, Dario Catalano, Dario Fiore, and Romain Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In *Advances in Cryptology — CRYPTO 2017*, volume 10401 of *LNCS*, pages 67–98. Springer, 2017. (Page 10)

[13] Manuel Barbosa, Dario Catalano, Azam Soleimanian, and Bogdan Warinschi. Efficient function-hiding functional encryption: From inner-products to orthogonality. In *Topics in Cryptology — CT-RSA 2019*, volume 11405 of *LNCS*, pages 127–148. Springer, 2019. (Page 10)

[14] Jean-Luc Beuchat, Jorge E González-Díaz, Shigeo Mitsunari, Eiji Okamoto, Francisco Rodríguez-Henríquez, and Tadanori Teruya. High-speed software implementation of the optimal ate pairing over Barreto–Naehrig curves. In *Pairing-Based Cryptography — Pairing 2010*, volume 6487 of *LNCS*, pages 21–39. Springer, 2010. full version: https://eprint.iacr.org/2010/354. (Pages 11, 13, 26, 27, and 28)

[15] Allison Bishop, Abhishek Jain, and Lucas Kowalczyk. Function-hiding inner product encryption. In *Advances in Cryptology—ASIACRYPT*, volume 9452 of *LNCS*, pages 470–491. Springer, 2015. (Page 7)

[16] Riadh Brinci, Walid Khmiri, Mefteh Mbarek, Abdellatif Ben Rabâa, and Ammar Bouallègue. Efficient hardware design for computing pairings using few FPGA in-built DSPs. Cryptology ePrint Archive, Report 2015/116, 2015. https://eprint.iacr.org/2015/116. (Page 28)

[17] Ray CC Cheung, Sylvain Duquesne, Junfeng Fan, Nicolas Guillermin, Ingrid Verbauwhede, and Gavin Xiaoxu Yao. FPGA implementation of pairings using residue number system and lazy reduction. In *Cryptographic Hardware and Embedded Systems — CHES 2011*, volume 6917 of *LNCS*, pages 421–441. Springer, 2011. (Page 28)

[18] Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Decentralized multi-client functional encryption for inner product. In *Advances in Cryptology—ASIACRYPT*, volume 11273 of *LNCS*, pages 703–732. Springer, 2018. (Pages 7 and 10)

[19] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*, chapter 30. MIT Press, 2009. (Page 18)

[20] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. Functional encryption for inner product with full function privacy. In *Public-Key Cryptography—PKC 2016*, volume 9614 of *LNCS*, pages 164–195. Springer, 2016. (Page 7)

[21] Junfeng Fan, Frederik Vercauteren, and Ingrid Verbauwhede. Faster $\mathbb{F}_p$-arithmetic for cryptographic pairings on Barreto-Naehrig curves. In *Cryptographic Hardware and Embedded Systems — CHES 2009*, volume 5747 of *LNCS*, pages 240–253. Springer, 2009. (Page 28)

[22] Junfeng Fan, Frederik Vercauteren, and Ingrid Verbauwhede. Efficient hardware implementation of fp-arithmetic for pairing-friendly curves. *IEEE Transactions on Computers*, 61(5):676–685, 2011. (Page 28)

[23] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929, 2016. (Page 7)

[24] Santosh Ghosh, Debdeep Mukhopadhyay, and Dipanwita Roychowdhury. High speed flexible pairing cryptoprocessor on FPGA platform. In *Pairing-Based Cryptography — Pairing 2010*, volume 6487 of *LNCS*, pages 450–466. Springer, 2010. (Page 28)

[25] Santosh Ghosh, Ingrid Verbauwhede, and Dipanwita Roychowdhury. Core based architecture to speed up optimal ate pairing on FPGA platform. In *International Conference on Pairing-Based Cryptography*, volume 7708 of *LNCS*, pages 141–159. Springer, 2012. (Pages 20 and 28)

[26] Shafi Goldwasser, S Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In *Advances in Cryptology—EUROCRYPT*, volume 8441 of *LNCS*, pages 578–602. Springer, 2014. (Page 7)

[27] S. Dov Gordon, Jonathan Katz, Feng-Hao Liu, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. Cryptology ePrint Archive, Report 2013/774, 2013. https://eprint.iacr.org/2013/774. (Page 7)

[28] Jun Han, Yang Li, Zhiyi Yu, and Xiaoyang Zeng. A 65 nm cryptographic processor for high speed pairing computation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(4):692–701, 2014. (Page 28)

[29] Zhongyuan Hao, Wei Guo, Jizeng Wei, and Dazhi Sun. Dual processing engine architecture to speed up optimal ate pairing on FPGA platform. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 584–589. IEEE, 2016. (Page 28)

[30] J. Howe, A. Khalid, C. Rafferty, F. Regazzoni, and M. O'Neill. On practical discrete gaussian samplers for lattice-based cryptography. *IEEE Transactions on Computers*, PP(99):1–1, 2016. (Pages 17 and 18)

[31] James Howe, Thomas Prest, Thomas Ricosset, and Mélissa Rossi. Isochronous gaussian sampling: From inception to implementation. Cryptology ePrint Archive, Report 2019/1411, 2019. https://eprint.iacr.org/2019/1411. (Page 18)

[32] David Kammler, Diandian Zhang, Peter Schwabe, Hanno Scharwaechter, Markus Langenberg, Dominik Auras, Gerd Ascheid, and Rudolf Mathar. Designing an ASIP for cryptographic pairings over Barreto-Naehrig curves. In *Cryptographic Hardware and Embedded Systems — CHES 2009*, volume 5747 of *LNCS*, pages 254–271. Springer, 2009. (Pages 28 and 29)

[33] A. Karatsuba and Yu. Ofman. Multiplication of many-digital numbers by automatic computers. *Proceedings of USSR Academy of Sciences*, 145(7):293–294, 1962. (Page 18)

[34] A. Karmakar, S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede. Constant-time discrete gaussian sampling. *IEEE Transactions on Computers*, 67(11):1561–1571, Nov 2018. (Pages 17 and 18)

[35] Angshuman Karmakar, Jose Maria Bermudo Mera, Sujoy Sinha Roy, and Ingrid Verbauwhede. Saber on ARM cca-secure module lattice-based key encapsulation on ARM. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):243–266, 2018. (Page 19)

[36] Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Pushing the speed limit of constant-time discrete gaussian sampling. a case study on falcon.

Cryptology ePrint Archive, Report 2019/267, 2019. https://eprint.iacr.org/2019/267, to appear in DAC 2019, Las Vegas, NV, USA. (Pages 17 and 18)

[37] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010. (Page 9)

[38] Jose Maria Bermudo Mera, Angshuman Karmakar, and Ingrid Verbauwhede. Time-memory trade-off in toom-cook multiplication: an application to module-lattice based cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):222–244, 2020. (Pages 19 and 30)

[39] Jose Maria Bermudo Mera, Furkan Turan, Angshuman Karmakar, Sujoy Sinha Roy, and Ingrid Verbauwhede. Compact domain-specific co-processor for accelerating module lattice-based key encapsulation mechanism. *IACR Cryptology ePrint Archive*, 2020:321, 2020. (Pages 12 and 30)

[40] Daniele Micciancio and Michael Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part II*, pages 455–485. Springer International Publishing, Cham, 2017. (Page 17)

[41] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985. (Page 15)

[42] Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems – CHES 2014: 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, pages 353–370. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. (Page 17)

[43] Sujoy Sinha Roy, Frederik Vercauteren, Nele Mentens, Donald Donglong Chen, and Ingrid Verbauwhede. Compact ring-lwe cryptoprocessor. In *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, pages 371–391, 2014. (Page 18)

[44] Ahmad Salman, William Diehl, and Jens-Peter Kaps. A light-weight hardware/software co-design for pairing-based cryptography with low power and energy consumption. In *2017 International Conference on Field Programmable Technology (ICFPT)*, pages 235–238. IEEE, 2017. (Page 28)

[45] Anissa Sghaier, Loubna Ghammam, Medyen Zeghid, Sylvain Duquesne, and Mohsen Machhout. Area-efficient hardware implementation of the optimal ate pairing over BN curves. Cryptology ePrint Archive, Report 2015/1100, 2015. https://eprint.iacr.org/2015/1100. (Page 28)

[46] Anissa Sghaier, Medien Zeghid, Loubna Ghammam, Sylvain Duquesne, Mohsen Machhout, and Hassan Yousif Ahmed. High speed and efficient area optimal ate pairing processor implementation over BN and BLS12 curves on FPGA. *Microprocessors and Microsystems*, 61:227–241, 2018. (Page 28)

[47] A Tengfei Wang, B Wei Guo, and C Jizeng Wei. Highly-parallel hardware implementation of optimal ate pairing over Barreto-Naehrig curves. *Integration*, 64:13–21, 2019. (Page 28)

[48] Brent Waters. A punctured programming approach to adaptively secure functional encryption. In *Advances in Cryptology—CRYPTO*, volume 9216 of *LNCS*, pages 678–697. Springer, 2015. (Page 7)

[49] Gavin Xiaoxu Yao, Junfeng Fan, Ray C.C. Cheung, and Ingrid Verbauwhede. A high speed pairing coprocessor using RNS and lazy reduction. Cryptology ePrint Archive, Report 2011/258, 2011. https://eprint.iacr.org/2011/258. (Page 28)

[50] Gavin Xiaoxu Yao, Junfeng Fan, Ray CC Cheung, and Ingrid Verbauwhede. Faster pairing coprocessor architecture. In *Pairing-Based Cryptography — Pairing 2012*, volume 7708 of *LNCS*, pages 160–176. Springer, 2012. (Page 28)

[51] Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. Facct: Fast, compact, and constant-time discrete gaussian sampler over integers. Cryptology ePrint Archive, Report 2018/1234, 2018. https://eprint.iacr.org/2018/1234. (Page 18)

[52] Raymond K. Zhao, Ron Steinfeld, and Amin Sakzad. Cosac: Compact and scalable arbitrary-centered discrete gaussian sampling over integers. Cryptology ePrint Archive, Report 2019/1011, 2019. https://eprint.iacr.org/2019/1011. (Page 18)