SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2016; **9**:1535–1546
Published online 21 February 2016 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.1442

RESEARCH ARTICLE

Efficient paillier cryptoprocessor for privacy-preserving data mining

Ismail San[1]*, Nuray At[1], Ibrahim Yakut[2] and Huseyin Polat[2]

[1] Department of Electrical and Electronics Engineering, Anadolu University, Eskisehir 26470, Turkey
[2] Department of Computer Engineering, Anadolu University, Eskisehir 26470, Turkey

ABSTRACT

Paillier cryptosystem is extensively utilized as a homomorphic encryption scheme to ensure privacy requirements in many privacy-preserving data mining schemes. However, overall performance of the applications employing Paillier cryptosystem intrinsically degrades because of modular multiplications and exponentiation operations performed by the cryptosystem. In this study, we investigate how to tackle with such performance degradation because of Paillier cryptosystem. We first exploit parallelism among the operations in the cryptosystem and interleaving among independent operations. Then, we develop hardware realization of our scheme using field-programmable gate arrays. As a case study, we evaluate our cryptoprocessor for a well-known privacy-preserving set intersection protocol. We demonstrate how the proposed cryptoprocessor responds promising performance for hard real-time privacy-preserving data mining applications. Copyright © 2016 John Wiley & Sons, Ltd.

KEYWORDS
cryptoprocessor; privacy; homomorphic encryption; data mining; FPGA

***Correspondence**
Ismail San, Department of Electrical and Electronics Engineering, Anadolu University, Eskisehir 26470, Turkey.
E-mail: isan@anadolu.edu.tr

1. INTRODUCTION

Privacy-preserving data mining (PPDM) algorithms have been proposed to respond increasing privacy awareness. In many cases, privacy of either individuals' or corporate data should be ensured to benefit from data mining methods. Without any protection, users might be subjected to profiling and faced with price discrimination. Moreover, they can receive disturbing amount of spam mails, messages, phone calls, or emails for unsolicited marketing. Because collected users' personal data are regarded as confidential and valuable assets, data collectors are obliged to protect such data [1].

In typical PPDM applications, privacy is a twofold problem: preserving privacy in *user*-to-*data holder* and *data holder*-to-*data holder* cases, as depicted in Figure 1. While the bottom part of the Figure 1 exhibits the framework for considering user-to-data holder, that is, individuals' privacy concerns, the top part shows data holder-to-data holder scenario considering corporate privacy in case of distributed data among data holders. The studies focusing on individual privacy framework concentrate on how user profiles can be collected and processed while ensuring privacy [2]. Corporate privacy is an issue, where two or more parties want to collaborate for input data to drive higher quality mining services. Collaboration is desirable for many parties to benefit from each other's data without divulging their privacy [3,4]. Either user-to-data holder or data holder-to-data holder case, the most important thing is to protect privacy.

To solve individual and corporate privacy protection problems, cryptographic techniques with homomorphic encryption property are widely used. In this sense, there are a quite number of PPDM algorithms exploiting Paillier homomorphic cryptosystem (PHC) to achieve privacy. Paillier [5] proposes an additive homomorphic encryption method with self-blinding property. By means of the specified method, ciphertexts can be processed to achieve encrypted version of their sums. From this addition property, multiplication operations can be performed between a ciphertext and a plaintext. Moreover, self-blinding property allows mapping a plaintext into possibly many different ciphertexts. By the way, the same plaintexts cannot be recognized from their ciphertexts.

While PHC provides verifiable correctness and trustworthiness without revealing information about the confidential data, its foremost shortcoming is computational cost because of the clumsy nature of cryptographic

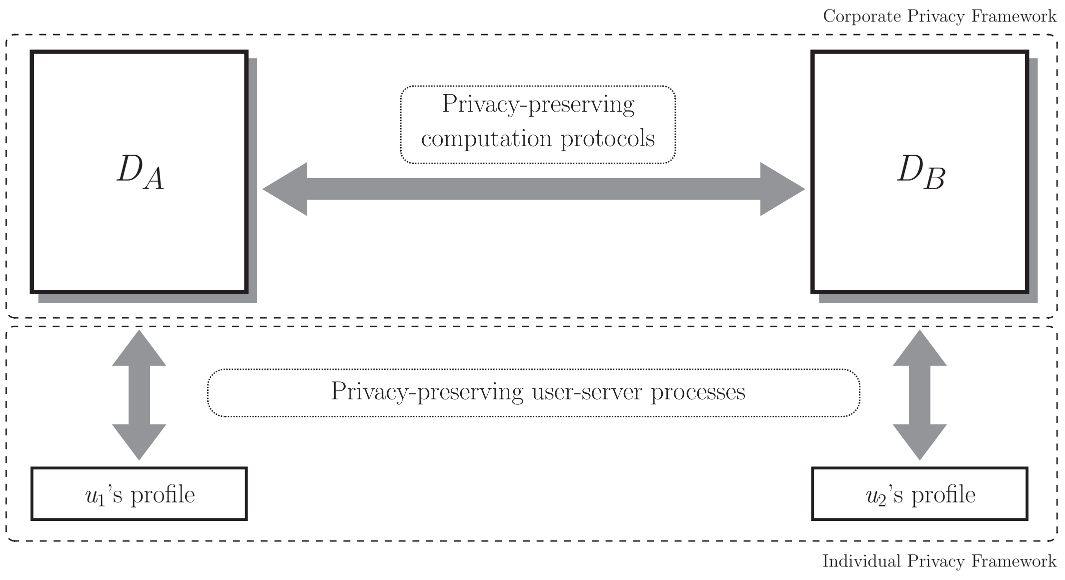
Corporate Privacy Framework
D_A
Privacy-preserving computation protocols
D_B
Privacy-preserving user-server processes
u_1's profile
u_2's profile
Individual Privacy Framework

Figure 1. Privacy frameworks for privacy-preserving data mining applications.

mechanisms [3,6]. This work is triggered with the need for the requirement of efficient PHC implementations for privacy protection [3,7]. Therefore, we develop a hardware-based solution to overcome the performance degradations because of Paillier encryption and propose a high-speed Paillier cryptoprocessor (PCP). We first take advantage of parallelism among the operations in Paillier cryptosystem to fully pipeline our datapath of the cryptoprocessor. Furthermore, we utilize interleaving among independent operations to enhance computations by avoiding data dependencies. We then exploit field-programmable gate arrays (FPGAs) for necessary hardware realization of our proposed high-speed cryptoprocessor. To validate the improved performance promised by our architecture, we conduct different sets of experiments. Empirical outcomes show that the proposed system is able to improve performance. As a case study, we also show how efficient the proposed cryptoprocessor is for a PPDM scheme called private matching proposed by Freedman *et al.* [8].

The remainder of the paper is organized as follows. In Section 2, we extensively study related works and highlight the significance of our topic with respect to the state of the art. We briefly present key preliminaries for the proposed cryptoprocessor in Section 3. After deeply discussing and illustrating our design philosophy and implementation details in Section 4, we explain our experiments and provide empirical outcomes together with a case study in Section 5. We finally conclude the study and give future directions in Section 6.

2. RELATED WORK

One way to achieve reasonable time for cryptographic algorithms is to apply an application-specific hardware design. Bhattacharjee *et al.* [9] describe a collaborative data mining and analysis solution for enterprisers having confidential data. Security of their scheme is based on IBM PCIXCC secure coprocessors (Armonk, NY, USA) as main processing units on encrypted data belong to a number of parties. Li [10] similarly proposes solutions to join various autonomous databases in a privacy-preserving manner via secure coprocessors. The proposed scheme allows a trusted third party to access secure coprocessor operations, and the trust notion is consolidated through a secure hardware usage. Smith and Safford [11] discuss the feasibility of privacy protection with secure coprocessors and describe a model consisting of a single server with encrypted records and a number of coprocessors enabling private database queries.

Hardware-based solutions like application-specific integrated circuits (ASICs) or FPGAs can be applied to efficiently implement computationally intensive algorithms. While ASICs have predetermined functionality to perform a particular task, FPGAs are able to perform an arbitrary task by combining logic gates, flip-flops, and memory units [12]. The ASIC implementation is faster and more secure than the software implementation [13]. However, the ASIC implementation is not flexible enough, and its longer design cycle and higher development cost make it less attractive and not preferable choice to build a cryptosystem in low volume designs [14]. The software-oriented implementation is flexible; however, its computational complexity is higher compared with its hardware equivalent. Moreover, storing the private key in the computer memory may endanger security. FPGAs offer more flexibility in highly parallel data processing, low latencies, and high-throughput rates [15]. Teubner *et al.* [15] show that FPGAs consume significantly lower power than conventional Personal

Computer (PC). The use of hard blocks available in FPGAs such as multipliers, block memories, and Input/Output (I/O) features allows to significantly reduce the area gap between ASIC and FPGA [16]. FPGAs having such prominent features can be solutions to fulfill performance implications of PHC.

There are various studies on efficient implementations of modular multiplication, inversion, and exponentiation [17–19]. Montgomery multiplication is one of the most popular algorithms [19]. It still constitutes a base for newly proposed schemes and operates with long integers to be represented by a radix (generally power of two) [20]. Moreover, high-radix Montgomery reduction method is well suited to the FPGA-based hardware environments because of the embedded building multiplier modules available in Xilinx (San Jose, CA, USA) FPGAs [17]. Various high-radix Montgomery modular exponentiation implementations have been proposed for improving performance [17,21]. However, as the radix increases, design complexity and length of clock cycle also increase dramatically because of the use of larger digit multipliers. These high-radix designs generally consume huge amounts of hardware area. Thus, low-radix designs are more attractive for hardware implementations. However, implementing larger than 17-bit pipelined multipliers is also possible. Analysis of the design tradeoffs for high-radix modular multipliers can be found in [22].

Unlike the previous solutions utilizing commercially available secure coprocessors [9–11], we design a high-speed coprocessor architecture to perform computationally intensive homomorphic operations. We also employ high-radix Montgomery algorithm incorporating the pipelined 32-bit multiplier modules to use hard blocks in FPGAs. Moreover, our scheme's privacy is based on the security achieved by Homomorphic Cryptosystem (HCs) rather than secure coprocessors. We also propose a new hardware architecture to increase efficiency of PPDM methods. There are two different studies [23,24] on FPGA implementation of multiplier and modular reduction tasks for Gentry's fully homomorphic encryption [25]. These are not fully homomorphic encryption processors, because they only implement primitives necessary for Gentry's scheme. Unlike this scheme, we focus on additive homomorphic method applied by many PPDM schemes. Our proposed hardware architectures are able to perform Paillier encryption and decryption, which provides additive homomorphism for PPDM. Similar to the schemes in [15,26], we focus on improving the performance of source exploiting tasks; however, our scheme additionally concentrates on how efficient solutions can be provided with privacy as in [27]. It also differs from the work [27] because we focus on cryptographic mechanisms rather than anonymity-based approaches. Our goal is to achieve a high-speed and high-throughput design of modular multiplication and exponentiation required by PHC.

3. PRELIMINARIES

3.1. Paillier homomorphic cryptosystem

Paillier [5] proposes a public key encryption scheme on composite residuosity classes. The scheme is based on the problem of deciding if the number is an nth residue modulo n^2. The degree of classes is set to a hard-to-factor number $n = pq$, where p and q are two large prime numbers. The system consists of key generation, encryption, and decryption steps, which utilize public and private keys and some precomputed values. Encryption and decryption operations consist of modular exponentiations, multiplication, and some specific operations.

Paillier encryption uses an encryption key $n = pq$, where n^2 is precomputed for encryption and decryption. Carmichael's function $\lambda(n)$ is denoted by λ. Moreover, a random value r is computed from the range $[1, n-1]$ such that $gcd(r, n) = 1$. Then, a base $g \in \mathcal{B}$ is selected and checked by $gcd\left(L\left(g^{\lambda} \mod n^2\right), n\right) = 1$. Finally, (n, g) pairs represent public keys, and the pair (p, q) is kept private. Encryption ($\mathcal{E}$) of PHC requires a modular exponentiation of base g and r. The computation is significantly enhanced by an acceptable choice of g [5].

The value $g^m \mod n^2$ is precomputed, and the result is multiplied with $r^n \mod n^2$. The same computation time can be achieved using two modular exponentiation modules. However, precomputation does not affect the security of the system, provided that the chosen value g fulfills the requirement $g \in \mathcal{B}$ imposed by the setting of Paillier scheme. A plaintext m can be encrypted, and the ciphertext c can be obtained as follows, where $m, r < n$:

$$c = g^m r^n \mod n^2 \tag{1}$$

Decryption ($\mathcal{D}$) of PHC requires a modular exponentiation of base c and g. The computation can be significantly improved by means of precalculating the modular inverse of $L(g^{\lambda} \mod n^2)$ because g and λ are unique values for one key. A ciphertext c can be decrypted, and m can be obtained as follows, where $c < n^2$:

$$m = \frac{L(c^{\lambda} \mod n^2)}{L(g^{\lambda} \mod n^2)} \mod n \tag{2}$$

in which

$$L(u) = \frac{u-1}{n} \qquad \forall u \in S_n \tag{3}$$

The set $S_n = \left\{u < n^2 | u = 1 \mod n\right\}$ is a multiplicative subgroup of integers modulo n^2 [5].

3.2. Modular exponentiation

Paillier homomorphic cryptosystem is heavily based on modular exponentiation operations. It mainly involves modular arithmetic operations and integer arithmetic including multiplication and exponentiation modulo n^2 for the encryption and decryption processes as seen from Equations 1 and 2. Modular exponentiation and multiplication with a large exponent and modulus (generally longer than 512 bits) are two of the most important arithmetic operations in several modern cryptographic algorithms. Efficient computation of modular exponentiation is very important for PHC. Modular exponentiation can be realized by performing a series of modular squaring and multiplication operations. It is time-consuming in the case of large operand sizes. Our high-throughput design's goal is to decrease execution time of each modular squaring and multiplication operations.

One of the most efficient algorithms for performing modular exponentiation in hardware is binary modular exponentiation algorithm, which is known as square-and-multiply algorithm. The algorithm requires $2n$ modular multiplication operations in the worst case. The right-to-left binary modular exponentiation algorithm (Algorithm 1) is adopted because there is no data dependency between modular multiplication operations. Hence, it is possible to exploit the parallelism between squaring and multiplication operations by means of implementing two parallel modular multiplier modules in the exponentiation. Because of parallel execution of such operations, modular exponentiation is completed in the order of $O(n\tau_{mm})$ time, where τ_{mm} is the time required to complete one modular multiplication operation.

Algorithm 1 Right-to-left binary modular exponentiation

```
Input: Positive integers $B, E, M, -M^{-1}, R^2$
    $E = \sum_{i=0}^{E_b-1} E_i 2^i$, $E_i \in \{0,1\}$, $R^2 = 2^{64d} \bmod M$
Output: $P = B^E \bmod M = \mathsf{ModExp}(B, E, M)$
1.  $S_0 \leftarrow \mathsf{ModMult}(R^2, 1, M)$;            (Initial phase)
2.  $Z_0 \leftarrow \mathsf{ModMult}(R^2, B, M)$;
3.  for $i \leftarrow 0$ to $n-1$ do
4.    if $E_i = 1$ then
5.      $S_{i+1} \leftarrow \mathsf{ModMult}(S_i, Z_i, M)$;      (Multiply)
6.    else
7.      $S_{i+1} \leftarrow S_i$;
8.    end if
9.    $Z_{i+1} \leftarrow \mathsf{ModMult}(Z_i, Z_i, M)$;          (Square)
10. end for
11. $S_{n+1} \leftarrow \mathsf{ModMult}(S_n, 1, M)$;            (Final phase)
12. return $P \leftarrow S_{n+1}$;
```

3.3. Exploring montgomery modular multiplication's high-radix architecture

One efficient algorithm for performing modular multiplication in hardware is Montgomery algorithm [19]. The radix-2 Montgomery modular multiplication requires less area compared with high-radix versions. However, its computational performance is worse. High-radix brings area cost because of the need for high-radix multipliers. The use of high-radix values in Montgomery reduction allows one to use hard-wired multiplier blocks in FPGAs.

Because modular exponentiation consists of series of modular multiplications, performance of the exponentiation modulo n^2 operation relies on performance of the multiplication operation modulo n^2. Hence, we accommodate the high-speed pipelined coprocessors in [28], where the authors take advantage of high-radix Montgomery modular multiplication and Karatsuba algorithm because of their suitability to Xilinx 7 series of FPGAs.

Algorithm 2 Montgomery multiplication on PCP (adapted from [28] for 2^{32} radix size)

```
Input: radix-$2^{32}$, $d = \lceil \frac{n}{32} \rceil$, $32 \cdot d \geq n + 3$,
    $X, Y, M, S_i \in \{0, 1, \ldots, 2^n - 1\}$,
    $X = \sum_{i=0}^{d-1} 2^{32i} X_i, X_d = 0$,    $Y = \sum_{i=0}^{d-1} 2^{32i} Y_i$,
    $M = \sum_{i=0}^{d-1} 2^{32i} M_i, M_d = 0$,
    $S_i = \sum_{j=0}^{d-1} 2^{32j} S_{(i,j)}, S_d = 0$,
    $-M^{-1}, X_i, Y_i, M_i, S_{(i,j)} \in \{0, 1, \ldots, 2^{32} - 1\}$,
    $\delta_0, \delta_1, \delta_2 \in \{0, 1, \ldots, 2^{47} - 1\}$,
    $\delta_{i,high} = \delta_{i,(47:32)}, \delta_{i,low} = \delta_{i,(31:0)}$
Output: $S_d \leftarrow XY2^{-32d} \bmod M =$
    $\mathsf{ModMult}(X, Y, M)$,
1.  $S_0 \leftarrow 0$;                                  (Initialization)
2.  for $i \leftarrow 0$ to $d-1$ do
3.    // On-the-fly $q_i$ computation
4.    $\alpha \leftarrow X_0 \cdot Y_i$;
5.    $\beta \leftarrow \alpha + S_{i,0}$;
6.    $q_i \leftarrow \beta \cdot (-M^{-1}) \bmod 2^{32}$;      ($q_i$ computation)
7.    // Pipelined computation of next value of S
8.    $\delta_1 \leftarrow [X_0 \cdot Y_i]_{(31:0)} + [q_i \cdot M_0]_{(31:0)}$;
9.    $\delta_2 \leftarrow \delta_1 + S_{i,0}$;
10.   $S_{(i+1,0)} \leftarrow \delta_{2,low}$;              (First word of $S_{i+1}$)
11.   $C \leftarrow \delta_{2,high}$;
12.   for $j \leftarrow 1$ to $d$ do
13.     $\delta_0 \leftarrow [X_{j-1} \cdot Y_i]_{(63:32)} + [q_i \cdot M_{j-1}]_{(63:32)}$;
14.     $\delta_1 \leftarrow \delta_0 + [X_j \cdot Y_i]_{(31:0)} + [q_i \cdot M_j]_{(31:0)}$;
15.     $\delta_2 \leftarrow \delta_1 + S_{i,j} + C$;
16.     $S_{(i+1,j-1)} \leftarrow \delta_{2,low}$;           (Remaining words)
17.     $C \leftarrow \delta_{2,high}$;
18.   end for
19. end for
20. if $S_d \geq M$ then
21.   $S_{d+1} \leftarrow S_d - M$;                        (Last reduction)
22. end if
```

Algorithm 2 presents the radix-2^{32} version of high-radix Montgomery modular multiplication on Virtex-7 FPGA. The architecture of modular multiplication component has a great impact on performance of the cryptoprocessor because the proposed PCP consists of a series of modular multiplications. We adapted the high-speed modular coprocessors from [28]. Interleaving q_i computation (Algorithm 2, lines 3 and 6) and computation of the next value of S_{i+1} (Algorithm 2, lines 12 and 18) described in [28] improves the PCP's throughput. This approach ensures to generate the q_i values on the fly. It is necessary to compute the first q_0 content before the first computation of S_1 because S_1 depends on q_0. This operation is performed during loading the input data to the input shift registers. After initial computing of q_0 content, the q_i computation is synchronized with the S_{i+1} computation.

4. PAILLIER CRYPTOPROCESSOR DESIGN

There are several design approaches that can be applied to cryptographic algorithms for efficient hardware implementations such as a low-area coprocessor design approach [29], a compact coprocessor design approach [30], a high-speed architecture design approach [31], and so on. In fact, high speed and high throughput are the most important optimization goals in designing PCP. In this study, we exploit the parallelism in the operations and utilize pipelining as a digital design strategy to improve overall throughput.

4.1. Design philosophy

We scrutinize how to design a high-speed architecture so that the coprocessor meets the requirements of PPDM schemes. To achieve such task, we utilize a design philosophy, which consists of the following four fundamental design methods.

(1) *Pipelining*: To maximize the clock frequency and the throughput, pipelining is employed in the datapath. To compute the main operations of PCP, a set of arithmetic operation modules connected in series executes in parallel by inserting buffers between elements. It allows us to break up long strings of data and shorten critical paths. Necessary computations are separated into a set of equal stages. We balance each stage by utilizing DSP48E1 blocks for high frequency values.

(2) *Parallel Execution*: Each stage in the overall datapath performs one part of the algorithm. Our pipelined datapath allows parallel execution of independent tasks of the algorithm at the same time. Hence, the stages of our pipelined datapath operate simultaneously using different resources, which increases the system's throughput.

(3) *Interleaving*: We interleave independent operations performing modular multiplication and exponentiation utilized in this study. This allows us to obtain a tight scheduling, which substantially increases the throughput.

(4) *Resource Sharing*: We reuse a modular multiplication (ModMult) component for a series of modular multiplication operations, which cannot be parallelized, in PCP to enhance resource utilization.

This design philosophy that we follow in this study allows us to develop a scalable and high-speed processor for encryption and decryption procedures of PHC. All these design methods can significantly improve the PHC's performance.

We display the required sizes for PHC and its inner components in Table I. As seen from Algorithm 2, there are two loops, which are controlled by i and j indices. Moreover, one can see how these indices change compared to size of the key, as shown in Table I. Note that the modular multiplication size is two times greater than the Paillier size. PHC involves $2n$-bit modular multiplication, n-bit modular exponentiation, and $L(u)$ operation. We apply the following design strategies to efficiently compute these operations to build the high-speed PCP.

(1) *Modular Multiplication*: We employ high-radix Montgomery modular multiplication algorithm (Algorithm 2) for modular multiplication operations. Specifically, we adapt the modular multiplication coprocessor presented in [28]. We select radix 2^{32} version to efficiently utilize the DSP48E1 blocks because these blocks are able to add three 48-bit operands in just one clock cycle. The 2^{32} radix Montgomery multiplication consists of 32-bit multiplication operations. However, the DSP48E1 blocks are not able to perform 32-bit multiplication. Hence, we first implement 32-bit 6 cycle pipelined multiplication module utilizing 4 DSP48E1 blocks.

(2) *Modular Exponentiation*: We use right-to-left binary modular exponentiation algorithm (Algorithm 1), which has repeated modular multiplication operations depending on the exponent value. It is in fact simply square and multiply algorithm according to the exponent. We mainly adapt the modular exponentiation coprocessor presented in [28].

(3) *$L(u)$ Operation:* There are two different ways to implement this operation. One of them is realized by means of a multiplication with a precomputed value.

Table I. Fundamental sizes of Paillier-n and its inner components.

Key size n [bits]	ModMult size $2n$ [bits]	# of rounds j	# of pipeline step i for each round j
512	1024	64	65
1024	2048	128	129
2048	4096	256	257

It is possible to decrement u by one with one of the DSP48E1 blocks. Then, (n^{-1} mod $2^{|n|}$) or (n^{-1} mod $n + 1$) is multiplied with the computed $u - 1$ in order to perform $L(u) = \frac{u-1}{n}$. The other way mainly consists of a integer division of very long numbers. There is no need to apply a decrement by one operation required by the $L(u)$ operation because quotient of the division by n always conveys the correct result of $\frac{u-1}{n}$ operation. The division takes varying numbers of clock cycles to complete the process of division depending on the values of the inputs. We accommodate this method in our architecture. A long integer division is computed with a specific pipelined division accelerator. In this approach, a precomputation of the multiplication input is needed.

The algorithm proposed in [32] is adapted for the long integer division in this study. The main feature of the algorithm depends on the quotient selection with the difference of the most-significant non-zero bit positions of the divisor and dividend. Hence, it dramatically reduces the total number of long subtractions required in the division. This helps us improve the computational performance of the division operation. The result of the division operation is calculated in varying number of steps according to the the most-significant non-zero bit positions of the divisor and dividend. In this study, we use this algorithm and implement it in digit-serial fashion.

4.2. Hardware implementation

Our architecture consists of shift registers, an Arithmetic Logic Unit (ALU), and a control unit. Shift registers are needed to store the input and the output operands. Our pipelined ALU utilizes the DSP48E1 blocks to perform the required 2^{32} radix multiplication operations. Our control unit is responsible for providing necessary control signals for the datapath to compute the Paillier autonomously. We assume that our PCP is provided with the required input values including input operands, modulo number, and its 32-bit inverse value ($-M^{-1}$).

4.2.1. Shift registers.

Our proposed high-speed PCP operates on 32-bit blocks of data in a sequential manner. We take advantage of this to keep the data in shift registers, which can be implemented in SRL32 blocks existing in 7-family of Xilinx FPGAs. Using SRL32 blocks consumes less resources than storing the data in the slice registers of FPGA. Shift registers are organized into 32-bit words to store necessary operands.

4.2.2. Arithmetic and logic unit.

Paillier homomorphic cryptosystem mainly consists of encryption and decryption. We describe the computation flow of encryption $\mathcal{E}$ and decryption $\mathcal{D}$ of PHC, which includes modular multiplication and exponentiation

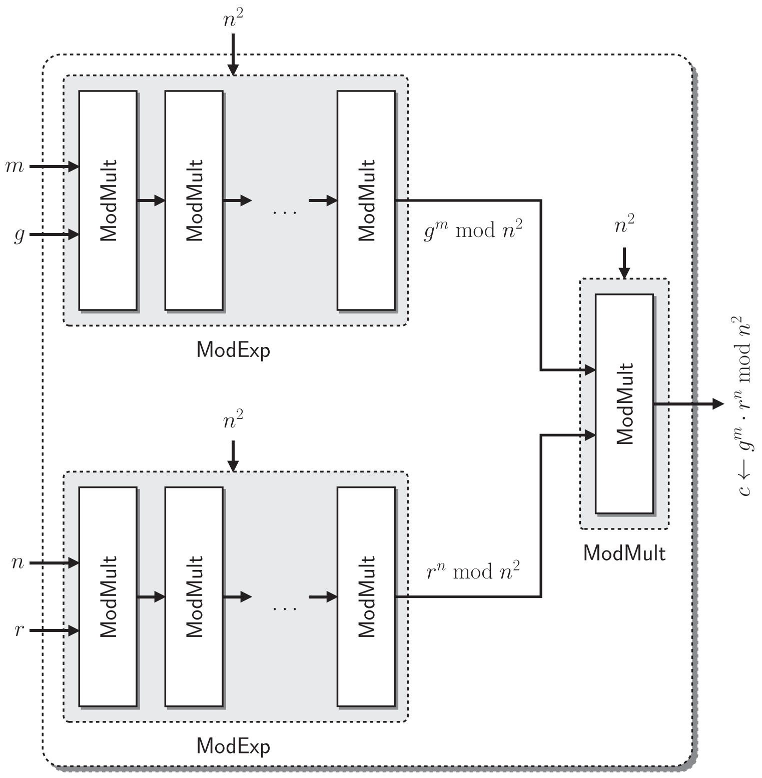

Figure 2. Architecture of Paillier encryption procedure of Paillier homomorphic cryptosystem.

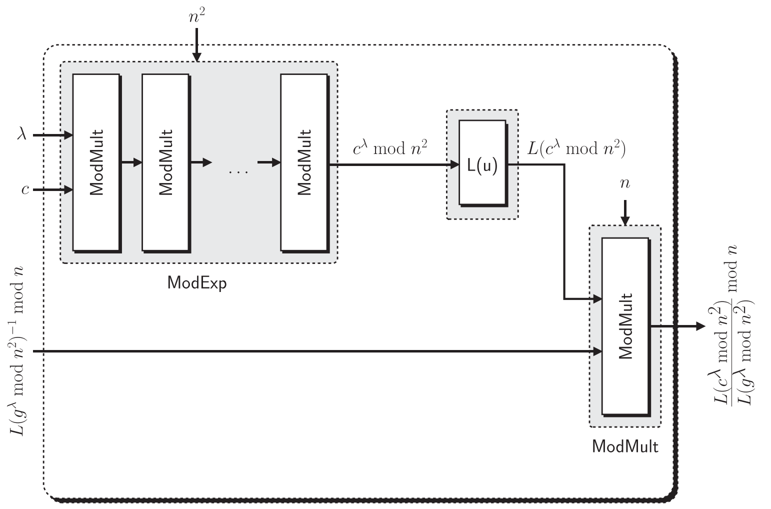

Figure 3. Architecture of Paillier decryption procedure of Paillier homomorphic cryptosystem.

operations, in Figures 2 and 3, respectively. Encryption function requires two ModExp components. Each ModExp component mainly consists of two ModMult components. These ModMult modules can perform a series of modular multiplications required in Algorithm 1 in parallel because they operate using different input resources. Finally, to obtain the final result of the encryption, outputs of two modular exponentiations are multiplied by one of the existing ModMult modules. One of the ModMult modules is reused for the last modular multiplication operation required in the $\mathcal{E}$. Furthermore, one of the modular exponentiation $g^m \bmod n^2$ components can be precomputed once for all, which is given in [5]. When this optimization is applied, there is no need to have the second ModExp; thus, this optimization brings resource saving.

Figure 3 illustrates the computation flow of decryption, which includes modular multiplication, exponentiation, and $L(u)$ operation. Decryption needs a ModExp component. The operation of the ModExp component is the same as in $\mathcal{E}$. The result of the modular exponentiation is fed into $L(u)$ module to perform $\frac{u-1}{n}$ operation. The precomputed value of $L(g^\lambda \bmod n^2)^{-1} \bmod n$ and the result of the $L(u)$ operation are multiplied with the existing ModMult module to obtain the final outcome of the decryption.

Our architecture is built around 32- and 48-bit datapath. The main datapath performs all operations required by PHC. The datapath allows us to realize the encryption and the decryption of PHC with high performance because it has very high operating frequency and the number of clock cycles needed for the Paillier $\mathcal{E}$ and $\mathcal{D}$ are small because of our design rationale. For modular multiplication operation ModMult, we select 2^{32} radix size version of modular multiplier coprocessor proposed in [28].

The high-radix modular multiplication operation is regarded as the most critical part of our proposed PCP. It requires to design high-radix multipliers in the datapath. On FPGAs, the best design strategy consists of implementing this multiplier by means of DSP48E1 blocks. The control signals allow us to efficiently perform the operations defined in Algorithm 2. The datapath has several pipeline units to improve the performance of the computation of modular multiplication operation. We manage to keep these pipeline units continuously busy without any pipeline stall. Therefore, we substantially improve the performance of the Paillier computation.

4.2.3. Control unit.

The control unit consists of a counter mechanism and shift registers. The counter mechanism simply counts the number of iterations required for the size of Paillier scheme. It has three counters. Two of them are counting the inner and the outer loop of Algorithm 2 while the last one is required to count the exponent size. Shift registers in the control unit are responsible for managing the control signals for the datapath to cope with the pipeline delays. The user starts the Paillier encryption or decryption by loading the input operands into input shift registers and applying a pulse for start control signal. Then, fully autonomous PCP computes the required output value and asserts a ready signal to indicate that the output of Paillier encryption or decryption is ready.

5. EMPIRICAL OUTCOMES AND DISCUSSION

In this section, we present our empirical results. We first explain our methodology for performance evaluation. After the metrics used to evaluate the performance of our design are defined, empirical results are displayed and discussed.

Table II. Number of clock cycles required for Paillier cryptoprocessor with varying n sizes.

Paillier	Modulo size n	Modulo size n^2	# of words $[d]$ for n^2	# of ModMult with n^2	# of ModMult with n	# of $L(u)$ cycles	Total # of cycles
Encryption	512	1024	32	516	0	0	569 148
Decryption				514	2	$\sim$ 15 800 *	583 338
Encryption	1024	2048	64	1028	0	0	4 357 692
Decryption				1026	2	$\sim$ 63 000 *	4 414 420
Encryption	2048	4096	128	2052	0	0	34 176 060
Decryption				2050	2	$\sim$ 245 000 *	34 396 228

∗The number of cycles required to perform $L(u)$ operation depends on the values of u and n.
Average numbers of cycles among several experiments are considered for $L(u)$.

5.1. Methodology

Our proposed high-speed PCP's implementations and the presented design strategies target Xilinx FPGAs. Therefore, the results of this study are valuable, where FPGA technology comes into play. We mainly used Virtex-7 family of FPGAs to prototype our architecture. The Virtex 7-family of FPGAs provides architectural elements designed for maximum performance and higher integration making a good choice for the high-speed architectures. We captured our PCP in the VHDL language and evaluated the performance of a fully autonomous implementation of our architecture on 7vx330t-3 FPGA. We synthesized our hardware architecture with Xilinx ISE 14. We also simulated our design with Xilinx ISim simulator to verify the correctness of the proposed architecture. PHC was implemented by using Java language in general purpose Central Processing Unit (CPU) to confirm that the proposed PCP works properly. We compared the performances of software implementation with our hardware-based design. We provided the performance evaluation results with detailed discussions and compared our performance results with the figures of Java implementation.

5.2. Evaluation metrics

We used three performance metrics to evaluate our proposed hardware design as follows:

(1) *Circuit Size*: Required hardware area is very important in efficient hardware designs, especially for the applications in which area also matters. We measured the size of the architecture implemented in FPGA with respect to the slices and embedded building blocks.
(2) *Latency*: Latency measures the time (we considered the worst-case scenario) to complete the encryption and decryption of PCP in terms of clock cycles. Our high-performance design achieves significant reduction in the latency.
(3) *Throughput*: The throughput represents the amount of data encrypted or decrypted per second. Because our proposed high-speed PCP allows concurrent processing of multiple blocks of data, throughput significantly improves.

5.3. Empirical results

We first performed a set of experiments to show how our proposed PCP's latency changes with varying n sizes. Recall that we used Xilinx ISE 14 development tools to perform our place-and-route and timing analysis. Also note that the datapath width of the PCP is 32 bit. We considered the least favorable case for PCP in which the exponent has all 1 s, which means that the maximum number of modular multiplication operations is performed for each size. After calculating the related latencies in terms of number of cycles for varying n sizes from 512 to 2048, we displayed the outcomes in Table II. Notice that the values listed in the fourth and the fifth columns (d and number of ModMult) were obtained, as defined in Algorithm 3. The number of $L(u)$ cycles depends on the values of u and n. Hence, the number of cycles required to perform $L(u)$ operation changes with the values of u and n, and average numbers of cycles among several experiments are provided for $L(u)$ in Table II.

The number of cycles required to perform ModMult operation depends on d and the number of pipeline stages of the multiplier. In other words, *Latency of ModMult* = $(d+1)\cdot(d+1)+t_m$, where t_m denotes the necessary cycles required to compute the output of ModMult and equals to 9 for $d = 16$ and 14 for bigger d values. For $d = 16$, $t_m = 9$, and the number of pipeline stages of the multiplier should be small to compute q_i values (Algorithm 2) before next S_i computation. This causes to increase the critical path of the design. However, this is valid only for Paillier decryption when $n = 512$. For other cases, when d is equal to 32, 64, or 128, $t_m = 14$, and multiplier has enough pipeline stages to reduce the critical path of the design. Latency of $\mathcal{E}$ depends on the number of ModMult operations. It can be computed as *Latency of* $\mathcal{E}$ = $(n+4)\cdot$(*Latency of* ModMult). Latency of $\mathcal{D}$ depends on the number of ModMult operations and can be estimated as follows:

$$\begin{aligned} \textit{Latency of } \mathcal{D} =&(n+2)\cdot\left(\textit{Latency of } \mathsf{ModMult} \textit{ with } n^2\right) \\ &+2\cdot(\textit{Latency of } \mathsf{ModMult} \textit{ with } n) \\ &+\textit{Latency of } L(u) \end{aligned}$$

As expected and seen from Table II, latency improves with decreasing n sizes. However, security becomes worse

Table III. Place-and-route results for Paillier cryptoprocessor on Virtex-7 (7vx330t-3) field-programmable gate array.

Paillier	n	Area [slices]	DSP48E1 blocks	Frequency [MHz]	Throughput [Kbits/s]
Encryption	512	3178	45	386	347
	1024	3690	45	386	90
	2048	4497	45	325	19
Decryption	512	2830	27	232	203
	1024	3254	27	323	74
	2048	3868	27	312	18

The throughput of Paillier cryptoprocessor is computed using the clock cycles given in Table II.

with decreasing n sizes. Although a number of cycles spent for decryption are slightly more than the ones spent for encryption, they are very close to each other.

We then used place-and-route PCP encryption and decryption designs to determine the area utilization and the throughput of PCP. We calculated throughput values for varying n sizes and displayed them in Table III. The throughput values listed in the table were computed for one block message. We also presented area, DSP block numbers, and frequency values, which are the place-and-route results of Xilinx ISE 14. We calculated the throughput of PCP as follows:

$$Throughput = \frac{n \cdot f}{Latency\ of\ \mathcal{E}\ or\ \mathcal{D}}$$

where n represents the Paillier size and f denotes the operating frequency. When the throughput of Paillier encryption was calculated, latency of $\mathcal{E}$ was considered. For the throughput of Paillier decryption, latency of $\mathcal{D}$ was used in the equation of throughput calculation.

According to the outcomes presented in Table III, with increasing n sizes, throughput decreases; however, security enhances. As seen from Table III, our PCP for PHC requires 3178 slices and 45 DSP48E1 blocks. And it achieves competitive throughout of 347 Kbits/s for encryption of 512-bit message blocks. The FPGA device (7vx330t-3) selected for experiments has total 51 000 slices and 1120 DSP48E1 blocks. It is possible to instantiate more than once in such a reconfigurable platform to achieve more throughput values. Existing parallelism in PHC schemes allows one to employ multiple instantiations to attain higher performance. Our empirical results show that our PCP performs well especially for applications in which time and area constraints are important. This phenomenon is observed because our design takes advantage of pipelining, which allows us to use different stage of the datapath at a time as well as to operate at higher frequencies.

To compare our proposed cryptoprocessor with the software-based implementation of PHC in terms of throughput, we finally conducted another set of simulations. We performed our experiments on an Intel i3 1.8 GHz machine (Santa Clara, CA, USA) to evaluate the PHC's software-based implementation. The presonal computer has two cores, 1.8-GHz clock rate (which is much more higher than the frequency of our architecture) and 3-MB cache. The implementation was realized using Java platform. Throughput values were computed for one-block message. After measuring average throughput values for varying security levels (different n sizes), we presented them in Table IV.

The results presented in Table IV show that throughput improves with decreasing n sizes as it is similar in FPGA-based equivalents. If we compare our design's outcomes displayed in Table III with the results representing the software implementation of PHC shown in Table IV, one can figure out that the proposed architectures perform better than the software realizations with respect to throughput. For Paillier encryption, the speed-up is about three because it requires two modular exponentiation modulo n^2. To improve the speed-up of Paillier decryption, it can be instantiated multiple times thanks to its low-area utilization. Our design improves encryption throughput from 96 and 7.6 to about 347 and 19 for n being 512 and 2048, respectively. Hence, on average, speed-up due to our design is three for one-block message. Moreover, our design makes it possible to execute PHC in parallel. However, software implementation always runs in sequential manner. The speed-up achieved by PCP might be higher, where many Paillier encryptions are needed to be computed in parallel. Such a scenario always exists in many PPDM applications because they operate on high-dimensional matrices of data with privacy. Thus, in such applications, where parallel PHC computations are possible, our cryptoprocessor can perform more than one PHC computation in single latency of PCP. To explain it concretely, our PCP is instantiated more than 16 times in an average seven family of FPGAs. This means that 16 Paillier encryptions can be computed in parallel.

Table IV. Performance of the software implementation of Paillier homomorphic cryptosystem.

Paillier	Throughput [Kbits/s]		
	$n = 512$	$n = 1024$	$n = 2048$
Encryption	96	26	7.6
Decryption	196	54	13.6

The processor number of central processing unit is i3-3217U.

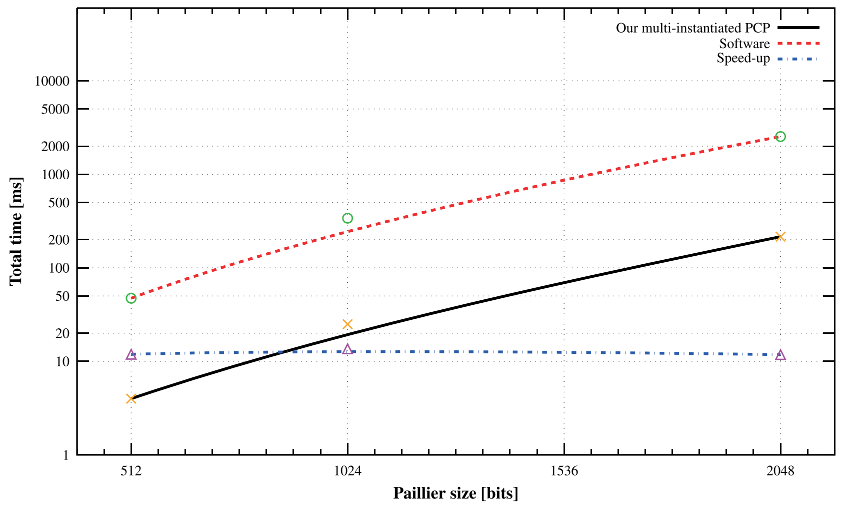

Figure 4. Comparing Paillier cryptoprocessor (PCP) with software implementation for Protocol 1.

5.4. Case study: private matching protocol with the Paillier cryptoprocessor

According to the empirical results, substantial performance improvement is attained because of our design. Furthermore, we evaluated our PCP based on a PPDM scheme to emphasize on the contribution of our proposed hardware architecture. Secure set intersection problem can be faced in many applications for online collaborating parties. The protocol [8] given later allows to perform a set-intersection operation between two sets of a given application with privacy.

We consider this secure set intersection protocol proposed by [8] using PCP as a case study. We selected the $k_{\mathcal{C}} = 4$ and $k_{\mathcal{S}} = 8$ for the set sizes of this protocol. Accordingly, $\mathcal{C}$ needs to deal with five parallel Paillier encryptions for providing privacy of the coefficients of the polynomial $P(y)$. The resulting set of Paillier encryptions of the coefficients is sent to $\mathcal{S}$. $\mathcal{S}$ homomorphically multiplies and adds each element of the set Y with the coefficients taken from $\mathcal{C}$ with privacy. The computed $\mathsf{Enc}(P(y))$ value is then multiplied with a random r value and added the element of the set Y with exploiting homomorphic property of PHC. The result of this operation is sent back to $\mathcal{C}$ to find whether it is an intersection of the sets or not. $\mathcal{C}$ finally decrypts the value and outputs the decrypted values for which there is a match with the sets.

The proposed PCP gives more speed-up in such a private matching set intersection protocol described in Protocol 1. We evaluated the speed-up using both our proposed PCP and the software-based platform with varying n sizes to show the improvements because of our design. We utilized the same methodology and hardware and software settings described previously. We considered necessary computations performed by client $\mathcal{C}$ in the protocol. Multiple instantiations of encryption and decryption architectures in the reconfigurable platform were taken into account. After estimating the times (in milliseconds) required to perform the operations of client $\mathcal{C}$ in the protocol for the parameters set earlier, we showed

Protocol 1 PM-Semi-Honest Intersection Protocol

Input: $\mathcal{C}$'s input is a set $X = \{x_1, \ldots, x_{k_\mathcal{C}}\}$, $\mathcal{S}$'s input is a set $Y = \{y_1, \ldots, y_{k_\mathcal{S}}\}$. The elements in the input sets are taken from a domain of size N.

1. $\mathcal{C}$ performs the following:
   - (a) She chooses the secret-key parameters for a semantically-secure homomorphic encryption scheme, and publishes its public keys and parameters. The plaintexts are in a field that contains representations of the N elements of the input domain, but is exponentially larger.
   - (b) She uses interpolation to compute the coefficients of the polynomial $P(y) = \sum_{u=0}^{k_\mathcal{C}} \alpha_u y^u$ of degree $k_\mathcal{C}$ with roots $\{x_1, \ldots, x_{k_\mathcal{C}}\}$.
   - (c) She encrypts each of the $(k_\mathcal{C} + 1)$ coefficients by the semantically-secure homomorphic encryption scheme and sends to $\mathcal{S}$ the resulting set of ciphertexts, $\{\mathsf{Enc}(\alpha_0), \ldots, \mathsf{Enc}(\alpha_{k_\mathcal{C}})\}$.
2. $\mathcal{S}$ performs the following for every $y \in Y$,
   - (a) He use the homomorphic properties to evaluate the encrypted polynomial at y. That is, he computes $\mathsf{Enc}(P(y)) = \mathsf{Enc}\left(\sum_{u=0}^{k_\mathcal{C}} \alpha_u y^u\right)$.
   - (b) He chooses a random value r and computes $\mathsf{Enc}(rP(y) + y)$.
     He randomly permutes this set of $k_\mathcal{S}$ ciphertexts and sends the result back to the client $\mathcal{C}$.
3. $\mathcal{C}$ decrypts all $k_\mathcal{S}$ ciphertexts received. She locally outputs all values $x \in X$ for which there is a corresponding decrypted value.

them for our hardware design and software implementation in Figure 4.

As seen from Figure 4, our proposed hardware PCP significantly performs better than the software implementation of PHC for all n. For instance, our proposed hardware design performs 13 times better than the software implementation when n is 1024. Similar speed-ups are observed for other n sizes. Hence, such speed-ups show linear behavior with respect to n. However, for increasing n, the time gap between software and PCP is getting more significant. Even the total time for PCP when $n = 2048$ is closer that for software when $n = 512$. These results show that PCP is able to perform Paillier computations efficiently in PPDM applications, where time constraints are stringent. The outcomes also verify that our design definitely performs better than software implementation.

6. CONCLUSIONS AND FUTURE WORK

Because privacy and efficiency are two clashing goals, there are some performance losses in data mining schemes where privacy concerns are taken into account. In this context, we looked solution of such performance problem through hardware-oriented solutions. We proposed a high-speed Paillier cryptoprocessor. The proposal can be utilized to overcome computational cumbersome brought by PHC. Moreover, it may gain insights for solving some computational challenges faced in such applications. Also note that our study is the first attempt on designing hardware architecture for Paillier-based homomorphic cryptosystem.

The design philosophy we proposed in this study led us to develop a high-performance cryptoprocessor at different levels of security for Paillier cryptosystem. Our hardware architecture is built around mostly 48-bit datapath, and it is able to compute the necessary operations for implementing encryption and decryption processes of PHC. Despite the various control signals required for the different steps of pipelined computation, our control unit remains compact. In other words, almost all control signals are generated by means of a counter, a start signal, and its delayed versions. We manage to implement the Paillier cryptosystem (including encryption and decryption) with only a few pipeline stall thanks to the descriptions of 2^{32} radix Montgomery modular multiplication and exponentiation coprocessors (adapted from [28]) and a careful organization of overall pipelined datapath. The key element of our approach to achieve high-throughput design is to take advantage of the parallelism of Paillier to

(1) deeply pipeline the ALU to achieve a high clock frequency;
(2) find parallelism between independent tasks;
(3) decrease the data dependencies and hazards to achieve more parallelism and exploit this parallelism for better design;
(4) reuse same resources for different executions of the algorithm enhancing resource sharing; and
(5) design pipelined datapath for the independent tasks so that each pipeline units can operate on different set of inputs at the same time.

We performed experiments and simulations to evaluate the proposed design with respect to latency, throughput, and time to complete encryptions and decryptions. Our empirical outcomes showed that the proposed architecture performs much more better than software implementations of the Paillier homomorphic cryptosystem.

Our results show that the PCP proposed in this study is an excellent solution, which remedy the performance problems raised by PPDM applications. In addition to PPDM applications, our design can also be considered for other applications utilizing PHC such as e-voting and e-auctions. We mainly design our architecture for PPDM mechanisms in information systems. Hence, it would be interesting to conduct side-channel and fault injection attacks in a future work.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous referees for their valuable comments. Additionally, the authors would like to acknowledge Xilinx and the Xilinx University Program for its generous donation of materials in terms of design tools.

REFERENCES

1. OECD. *Guidelines on the Protection of Privacy and Transborder flows of Personal Data*, Vol. 11: Paris, France, 1981.
2. Yakut I, Polat H. Privacy-preserving Eigentaste-based collaborative filtering. *Lecture Notes in Computer Science* 2007; **4752**: 169–184.
3. Yakut I, Polat H. Arbitrarily distributed data-based recommendations with privacy. *Data & Knowledge Engineering* 2012; **72**: 239–256.
4. Magkos E, Maragoudakis M, Chrissikopoulos V, Gritzalis S. Accurate and large-scale privacy-preserving data mining using the election paradigm. *Data & Knowledge Engineering* 2009; **68**(11): 1224–1236.
5. Paillier P. Public-key cryptosystems based on composite degree residuosity classes. *Lecture Notes in Computer Science* 1999; **1592**: 223–238.
6. Basu A, Vaidya J, Dimitrakos T, Kikuchi H. Feasibility of a privacy preserving collaborative filtering scheme on the Google App Engine: A performance case study, *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, Trento, Italy, 2012; 447–452.
7. Parkes DC, Rabin MO, Shieber SM, Thorpe CA. Practical secrecy-preserving, verifiably correct and

trustworthy auctions, *Proceedings of the 8th International Conference on Electronic Commerce*, Fredericton, New Brunswick, Canada, 2006; 70–81.

8. Freedman MJ, Nissim K, Pinkas B. Efficient private matching and set intersection. *Lecture Notes in Computer Science* 2004; **3027**: 1–19.
9. Bhattacharjee B, Abe N, Goldman K, Zadrozny B, Chillakuru VR, del Carpio M, Apte C. Using secure coprocessors for privacy preserving collaborative data mining and analysis, In *Proceedings of the 2nd International Workshop on Data Management on New Hardware (DaMoN '06)*, ACM, New York, NY, USA, 2006. Article 1 . DOI: http://dx.doi.org/10.1145/1140402.1140404.
10. Li Y. Privacy preserving joins on secure coprocessors. *Ph.D. Thesis*, EECS Department, University of California, Berkeley, 2008.
11. Smith SW, Safford D. Practical server privacy with secure coprocessors. *IBM Systems Journal* 2001; **40** (3): 683–695.
12. Ferlin EP, Lopes HS, Lima CRE, Perretto M. A FPGA-based reconfigurable parrallel architecture for high-performance numerical computation. *Journal of Circuits, Systems and Computers* 2011; **20** (05): 849–865.
13. Bayhan D, Ors SB, Saldamli G. Analyzing and comparing the Montgomery multiplication algorithms for their power consumption, *Proceedings of 2010 International Conference on Computer Engineering and Systems*, Cairo, Egypt, 2010; 257–261.
14. Murphy B, Churiwala S. SpyGlass application in an FPGA to ASIC conversion flow. White paper, Atrenta, 2009.
15. Teubner J, Muller R, Alonso G. Frequent item computation on a chip. *IEEE Transactions on Knowledge and Data Engineering* 2011; **23**(8): 1169–1181.
16. Kuon I, Rose J. Measuring the gap between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 2007; **26**(2): 203–215.
17. Blum T, Paar C. High-radix Montgomery modular exponentiation on reconfigurable hardware. *IEEE Transactions on Computers* 2001; **50**(7): 759–764.
18. Beuchat JL. Modular multiplication for FPGA implementation of the IDEA block cipher, *Proceedings of the 14th IEEE International Conference on Application-specific Systems, Architectures and Processors*, The Hague, The Netherlands, 2003; 412–422.
19. Montgomery PL. Modular multiplication without trial division. *Mathematics of Computation* 1985; **44** (170): 519–521.
20. Perin G, Mesquita DG, Martins JB. Montgomery modular multiplication on reconfigurable hardware: systolic versus multiplexed implementation. *International Journal of Reconfigurable Computing* 2011; **2011**: 6:1–6:10.
21. Bo S, Kawakami K, Nakano K, Ito Y. An RSA encryption hardware algorithm using a single DSP block and a single block RAM on the FPGA. *International Journal of Networking and Computing* 2011; **1** (2): 277–289.
22. Walter CD. Space/time trade-offs for higher radix modular multiplication using repeated addition. *IEEE Transactions on Computers* 1997; **46**(2): 139–141.
23. Wang W, Huang X. FPGA implementation of a large-number multiplier for fully homomorphic encryption, In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, 2013; 2589–2592. DOI: 10.1109/ISCAS.2013.6572408.
24. Cao X, Moore C, O'Neill M, O'Sullivan E, Hanley N. Accelerating fully homomorphic encryption over the integers with super-size hardware multiplier and modular reduction. *IACR Cryptology ePrint Archive* 2013; **2013**: 616.
25. Gentry C. Fully homomorphic encryption using ideal lattices, *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing,* STOC '09, ACM, New York, NY, USA, 2009; 169–178.
26. Wen YH, Huang JW, Chen MS. Hardware-enhanced association rule mining with hashing and pipelining. *IEEE Transactions on Knowledge and Data Engineering* 2008; **20**(6): 784–795.
27. Sawada J, Nishi H. Hardware acceleration and data-utility improvement for low-latency privacy preserving mechanism, *Proceedings of the 22nd International Conference on Field Programmable Logic and Applications*, Oslo, Norway, 2012; 499–502.
28. San I, At N. Improving the computational efficiency of modular operations for embedded systems. *Journal of Systems Architecture* 2013, DOI: 10.1016/j.sysarc.2013.10.013.
29. At N, Beuchat JL, Okamoto E, San I, Yamazaki T. A low-area unified hardware architecture for the AES and the cryptographic hash function Grøstl. Cryptology ePrint Archive, Report 2012/535, 2012.
30. San I, At N. Compact Keccak hardware architecture for data integrity and authentication on FPGAs. *Information Security Journal: A Global Perspective* 2012; **21**(5): 231–242.
31. Güneysu T. Utilizing hard cores of modern FPGA devices for high-performance cryptography. *Journal of Cryptographic Engineering* 2011; **1**(1): 37–55.
32. Hiasat AA, Abdel-Aty-Zohdy HS. A high-speed division algorithm for residue number system, *IEEE International Symposium on Circuits and Systems, ISCAS '95,* Vol. 3, Seattle, Washington, USA, 1995; 1996–1999.