A Programmable SoC Implementation of the DGK Cryptosystem for Privacy-Enhancing Technologies

Milad Bahadori and Kimmo Järvinen
University of Helsinki, Department of Computer Science, Helsinki, Finland
{milad.bahadori, kimmo.u.jarvinen}@helsinki.fi

Abstract—Additively homomorphic encryption has many applications in privacy-enhancing technologies because it allows a cloud service provider to perform simple computations with users’ data without learning the contents. The performance overhead of additively homomorphic encryption is a major obstacle for practical adaptation. Hardware accelerators could reduce this overhead substantially. In this paper, we present an implementation of the DGK cryptosystem for programmable systems-on-chip and evaluate it in real hardware. We demonstrate its efficiency for accelerating privacy-enhancing technologies by using it for computing squared Euclidean distances between a user’s input and a server’s database. We also provide comparisons with a recent implementation of Paillier cryptosystem and show that DGK offers major speedups. This work represents the first implementation of the DGK cryptosystem that uses hardware acceleration and demonstrates that the DGK benefits greatly from the hardware/software codesign approach.

Index Terms—HW/SW codesign, FPGA, homomorphic encryption, DGK cryptosystem, privacy-enhancing technologies.

I. INTRODUCTION

Users’ data is increasingly stored and processed in the cloud. While this trend has many obvious benefits, it comes with a heavy price on privacy: cloud service providers hold much of users’ (sensitive) data and can exploit it, e.g., for marketing purposes. Encrypting the stored data with normal encryption would solve the privacy problems but prevent all processing and nullify most of the benefits of cloud services as a consequence. Privacy-Enhancing Technologies (PETs) are methods to protect users’ privacy without sacrificing the functionality of services. Many PETs are based on homomorphic encryption, a type of encryption that allows computations with encrypted data without revealing its contents.

Additively Homomorphic Encryption (AHE) allows additions in the encrypted domain and makes it possible to implement practical PETs, particularly, when combined with certain other multi-party computation technologies. HW implementations of cryptography offer both speed and security improvements compared to software (see, e.g., [1]). AHE introduces significant performance overheads and, hence, improving its performance with HW acceleration may substantially improve the efficiency of PETs and help their practical adaptation.

So far, only few works have considered HW acceleration of AHE schemes and PETs; notable exceptions are [2], [3] that study acceleration of Paillier cryptosystem [4] using Field Programmable Gate Arrays (FPGAs).

In this paper, we focus on the Damgård-Geisler-Krøigaard (DGK) cryptosystem [5], [6]. It is an AHE scheme that provides fast encryption and decryption combined with small ciphertext sizes, but provides a smaller plaintext space compared to many other AHE schemes, most notably to Paillier cryptosystem [4]. From the computational point-of-view, DGK requires modular arithmetic with large integers similarly as Paillier cryptosystem but with smaller operand sizes. We propose an implementation of DGK in a Xilinx Zynq-7020 programmable System-on-Chip (SoC) using a multi-core design for large integer arithmetic reported in [3] and earlier used for Paillier cryptosystem. We show the suitability of the DGK implementation for PETs by studying privacy-preserving computations of Squared Euclidean Distances (SEDs) between a user’s input and entries of a server’s database.

We provide the following contributions:

- We present an efficient implementation of DGK in the HW/SW codesign originally introduced in [3] for Paillier cryptosystem. To the best of our knowledge, this is the first published hardware implementation of DGK.
- We show that DGK and our implementation can be efficiently used for privacy-preserving computation of SEDs, which are commonly used in PETs.
- We compare DGK and Paillier AHEs and conclude that DGK provides significantly faster encryption, decryption, and SED computations.

The remainder of this paper is organized as follows. Section II presents the preliminaries on DGK cryptosystem and privacy-preserving SEDs. Section III introduces the architecture of the HW/SW codesign and our implementation of DGK and SEDs. Section IV presents the results and analysis of the implementations, and finally, we end the paper by drawing conclusions in Section V.

II. PRELIMINARIES

Since the focus of this paper is on the implementation of the DGK cryptosystem for accelerating PETs (particularly, SEDs), we first briefly describe relevant details of the DGK cryptosystem and then discuss algorithmic aspects of privacy-preserving computation of certain distance metrics for PETs.

A. DGK Cryptosystem

The DGK cryptosystem was introduced by Damgård, Geisler, and Krøigaard in [5] and later corrected in [6]. It comprises of the following three algorithms:

- **Key Generation.** Given a security parameter $\kappa$ and parameters $t$ and $\ell$ such that $\kappa > t > \ell$, choose two $t$-bit
Primes \( v_p \) and \( v_q \) and an \( \ell \)-bit prime \( u \). Then, choose two \( \kappa \)-bit primes \( p \) and \( q \) so that \( v_p \mid p-1 \) and \( v_q \mid q-1 \) as well as \( u \mid p-1 \) and \( u \mid q-1 \). Then, compute \( N = p \cdot q \). Choose \( g \in \mathbb{Z}_N^* \) of order \( v_p v_q \) and \( h \in \mathbb{Z}_N^* \) of order \( v_p v_q \). The public key is \( pk = (N, g, h, u) \) and the secret key is \( sk = (p, q, v_p, v_q) \). The parameter sizes should be chosen so that factoring a \( \kappa \)-bit \( N \) is hard, \( t \) is chosen based on the logarithm of the size of the subgroup of \( \mathbb{Z}_N^* \), and \( \ell \) defines the plaintext space. We use \( \kappa = 2048 \), \( t = 224 \), and \( \ell \in [16, 22] \) in this paper, as we target the 112-bit security level.

- **Encryption.** Take a message \( m \in \mathbb{Z}_u \) and a public key \( pk = (N, g, h, u) \) as inputs and select a 2.5t-bit random \( \tau \), as instructed in [6]. Then, compute and return the ciphertext:

\[
c = \text{Enc}(pk, m) = g^m \cdot h^\tau \mod N.
\]  

(1)

- **Decryption.** Take the ciphertext \( c \in \mathbb{Z}_N^* \) and the secret key \( sk = (p, q, v_p, v_q) \) as inputs and compute:

\[
m' = c^{v_p} \mod p.
\]  

(2)

If \( m' = 1 \), then \( m = 0 \). To decrypt other values, keep computing \( g^{v_p} \) for \( i \in \mathbb{Z}_u \) until \( g^{v_p} = m' \), then \( m = i \). This works because \( g^\tau \) has order \( u \) and there is a one-to-one correspondence between values of \( m \) and \( g^\tau \mod p \) [5] and it is computationally feasible since \( \mathbb{Z}_u \) is small. This process is denoted by \( m = \text{Dec}(sk, c) \).

Decryption can be carried out by using a precomputed Decryption Look-Up Table (DLUT) \( D = (\delta_0, \delta_1, \ldots, \delta_{u-1}) \) where \( \delta_i = g^{v_p} \mod p \). In fact, it suffices to store truncated \( \delta'_i = \text{lsb}_\lambda(\delta_i) \) where \( \text{lsb}_\lambda(\delta_i) \) denotes the \( \lambda \) Least Significant Bits (LSBs) of \( \delta_i \) and \( \lambda \) is selected as the smallest value ensuring that all \( \delta'_i \) are unique. In our case, we use \( D \) where \( \lambda \) is a multiple of 8 (a byte) and entries \( (\delta'_i, i) \) are sorted by \( \delta'_i \) to ensure fast searches. We consider two decryption variants:

- a fast variant that computes \( m' \) with (2) and finds \( (\delta'_i, i) \) such that \( \delta_i = \text{lsb}_\lambda(m') \) and directly returns \( m = i \).

- a safe variant that before returning \( m = i \) computes \( m'' = g^{v_p m} \mod p \) and verifies that \( m'' = m' \). This removes the possibility of a corrupted ciphertext \( c' \) returning a valid decryption result due to the truncation.

Let \( \langle m \rangle \) denote the DGK encryption of a plaintext \( m \). The DGK cryptosystem is additively homomorphic under multiplication modulo \( N \) and, therefore

\[
\text{Dec}(sk, \langle m_1 \rangle \cdot \langle m_2 \rangle \mod N) = m_1 + m_2 \mod u.
\]  

(3)

It directly follows from (3) that homomorphic multiplications by a scalar \( k \) can be computed via exponentiation:

\[
\text{Dec}(sk, \langle m \rangle^k \mod N) = k \cdot m \mod u.
\]  

(4)

DGK has the advantage that encryption, decryption, and homomorphic operations are faster than for Paillier because operations are carried out in \( \mathbb{Z}_N \) instead of \( \mathbb{Z}_{N^2} \). Another advantage is that ciphertexts are smaller by 50\% (\( \mathbb{Z}_N \) vs. \( \mathbb{Z}_{N^2} \)). The fact that the plaintext space is significantly smaller (\( \mathbb{Z}_u \) vs. \( \mathbb{Z}_N \)) can be a disadvantage as will be discussed later.

**B. Privacy-Preserving Distances for PETs**

Let \( x = (x_0, x_1, \ldots, x_{m-1}) \) be a query vector and \( Y \) be a database consisting of \( n \) vectors of length \( m \):

\( Y = \{y_0, y_1, \ldots, y_{n-1}\} \), where \( y_i = (y_{i0}, y_{i1}, \ldots, y_{im-1}) \).

We assume that \( x_j, y_{ij} \in \mathbb{Z}_2^\tau \) and use \( \tau = 4 \) and \( \tau = 8 \) as examples in this paper. Fig. 1 shows a use case where \( Y \) is in the possession of a server and \( x \) is a query sent by a user. The user wants to find out which \( y_i \) is closest to \( x \) under some distance metric. Both parties want to keep their inputs secret from the other party.

To ensure the secrecy of \( x \), the user encrypts \( x_j \) with his/her public key \( pk \) before sending them to the server: \( \langle x \rangle = (\langle x_0 \rangle, \langle x_1 \rangle, \ldots, \langle x_{m-1} \rangle) = (\text{Enc}(pk, x_0), \text{Enc}(pk, x_1), \ldots, \text{Enc}(pk, x_{m-1})) \). The server calculates encrypted distances \( d_{ij} \) between \( \langle x \rangle \) and each vector \( y_i \) in \( Y \) by utilizing the homomorphic properties of the encryption scheme. Typical applications do not use the distances as such, but require the indices of the \( k \) smallest distances, i.e., the \( k \) nearest neighbors (kNN). The indices cannot be retrieved directly with AHE. The kNN search is implemented, e.g., so that the server masks the encrypted distances with random masks and Yao’s garbled circuits [7] are used for removing the masks and finding the
### Algorithm 1: A straightforward algorithm for computing the encrypted middle-terms \( \langle \Delta_{1,2} \rangle \) for squared Euclidean distances (SEDs) [3].

**Input:** Database \( Y \), where \( y_{i,j} \in \mathbb{Z}_2^t \) with \( i \in [0, n - 1] \) and \( j \in [0, m - 1] \), and encrypted query \( \langle \langle -2x_0 \rangle, \langle -2x_1 \rangle, \ldots, \langle -2x_{m-1} \rangle \rangle \)

**Output:** Encrypted middle-terms for squared Euclidean distances \( \langle \langle \Delta_{0,2} \rangle, \langle \Delta_{1,2} \rangle, \ldots, \langle \Delta_{n-1,2} \rangle \rangle \)

1. \((t_0, t_1, \ldots, t_{n-1}) \leftarrow (1, 1, \ldots, 1)\)
2. **for** \( i = 0 \) to \( m - 1 \) **do**
   3. **for** \( j = 0 \) to \( m - 1 \) **do**
   4. \( s \leftarrow \text{ME}(\langle -2x_j \rangle, y_{i,j}, N)\)
   5. \( t_i \leftarrow \text{MM}(t_i, s, N)\)
6. \( \langle \Delta_{0,2} \rangle = t_0, \langle \Delta_{1,2} \rangle = t_1, \ldots, \langle \Delta_{n-1,2} \rangle = t_{n-1}\)
7. **return** \( \langle \langle \Delta_{0,2} \rangle, \langle \Delta_{1,2} \rangle, \ldots, \langle \Delta_{n-1,2} \rangle \rangle \)

### Algorithm 2: An optimized algorithm for computing the encrypted middle-terms \( \langle \Delta_{1,2} \rangle \) for squared Euclidean distances (SEDs) [3].

**Input:** Database \( Y \), where \( y_{i,j} \in \mathbb{Z}_2^t \) with \( i \in [0, n - 1] \) and \( j \in [0, m - 1] \), and encrypted query \( \langle \langle -2x_0 \rangle, \langle -2x_1 \rangle, \ldots, \langle -2x_{m-1} \rangle \rangle \)

**Output:** Encrypted middle-terms for squared Euclidean distances \( \langle \langle \Delta_{0,2} \rangle, \langle \Delta_{1,2} \rangle, \ldots, \langle \Delta_{n-1,2} \rangle \rangle \)

1. \((t_0, t_1, \ldots, t_{n-1}) \leftarrow (1, 1, \ldots, 1)\)
2. **for** \( j = 0 \) to \( m - 1 \) **do**
3. \( s \leftarrow 1\)
4. **for** \( k = 1 \) to \( 2^t - 1 \) **do**
5. \( s \leftarrow \text{MM}(s, \langle -2x_j \rangle, N)\)
6. **for** \( \{i \mid y_{i,j} = k\} \) **do** \(/* i \in [0, n - 1] */\)
7. \( t_i \leftarrow \text{MM}(t_i, s, N)\)
8. \( \langle \Delta_{0,2} \rangle = t_0, \langle \Delta_{1,2} \rangle = t_1, \ldots, \langle \Delta_{n-1,2} \rangle = t_{n-1}\)
9. **return** \( \langle \langle \Delta_{0,2} \rangle, \langle \Delta_{1,2} \rangle, \ldots, \langle \Delta_{n-1,2} \rangle \rangle \)

### III. Architecture and Implementation

#### A. High-Level HW/SW Codesign

This section shortly revisits the HW/SW codesign accelerator presented in [3]; more details are available in [3]. The overall architecture is shown in Fig. 2. It divides into HW and SW sides so that the computationally heavy long integer modular arithmetic is performed by the HW side (FPGA) and controlling of the HW side and auxiliary operations are performed by the SW side. The architecture is general and suits for different programmable SoCs with minor modifications. In this paper, we use a Xilinx Zynq-7020 [15] in ZedBoard [16] as the implementation platform.

Typically AHE computations in PETs (i.e. SEDs computations) include a lot of inherent parallelism. For this reason, the HW side is a multi-core architecture that includes multiple parallel and programmable Cryptography Processor (CP) cores that are designed to have a good balance between performance and area requirements. The CP cores are arranged into \( M \) clusters, each including \( N \) CP cores. All blocks in the HW side are connected in an Advanced Extensible Interface (AXI)-based structure. Each CP core can be individually programmed via microcode updates.
The HW/SW codesign includes a multi-level memory structure to overcome the bottleneck of data communication between the CP cores and the SW side. The three-level Data Memory (DMEM) is divided as follows. The HW side has Level-1 DMEM (L1-DMEM) in each CP core and Level-2 DMEM (L2-DMEM) that is shared for all CP cores. The SW side includes Level-3 Memory (L3-MEM) that consists of both on-chip and off-chip memory (i.e., DDR3 in ZedBoard). Each CP core also includes Level-3 Memory (L3-MEM) that consists of both on-chip and off-chip memory (i.e., DDR3 in ZedBoard). Each CP core also includes Instruction Memory (IMEM) for storing microcodes loaded into the CP core from L3-MEM by the SW side. Data communication between memory levels uses High Performance (HP) AXI interfaces. Also, the General Purpose (GP) AXI interfaces are applied for transferring commands and status between the SW and HW sides (see Fig. 2). 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 specific computations. These controlling operations include sending and receiving data and microcodes to/from the CPs, issuing commands to the CPs, offline and online programming of the CP cores (by the microcodes) and other modules in the HW side, receiving the status of the CPs and other modules from the HW side, and making control decisions based on the received status.

B. Cryptography Processor

The CP core is an efficient programmable processor for large integer modular arithmetic optimized for the resources of modern FPGAs (e.g., for DSP slices, BRAMs, etc.). The CP core is based on a micro-programming architecture which provides both flexibility (for parameter sizes) and programmability (for different algorithms) combined with a small area footprint. The architecture of the CP core is shown in Fig. 3.

The CP core contains an external interface unit, an arithmetic unit, a data memory unit (L1-DMEM), a control unit, and an instruction memory unit (IMEM). The arithmetic unit contains Modular Multiply-Add Accumulator (MMAA) and Modular Adder/Subtractor (MAS) blocks for computing mod-
Fig. 3. Architectural diagram of the cryptography processor (CP) core. Structure details of the datapath is depicted in the right side of this figure.

ular arithmetic. The inputs and output of the arithmetic unit are connected to L1-DMEM, which stores data that is required during an algorithm run. Two words can be read and one word written simultaneously from and to L1-DMEM to facilitate efficient modular arithmetic. The microcodes stored in IMEM are sequences of instructions for the units of the CP core. Each instruction consists of different fields such as arithmetic, control, next IMEM address, DMEM address values, DMEM, and IMEM fields. These fields apply all required controlling signals for the units for a working cycle of the CP core. The microcodes are generated by hand through a customized platform and scripts. The external interface unit is the top module and a wrapper for the other units in the CP core architecture. The main tasks of this unit are receiving/sending command/status from/to external module(s), supporting AXI-based read and write interfaces with the SW side, supporting read and write interfaces with the shared L2-DMEM, and controlling the other units of the CP core.

C. Implementation of Target Applications

In this section, we describe details of implementing the target applications using the HW/SW codesign. These implementations are carried out with software updates for the SW side and microcode updates for the HW side, and do not require any modifications to the HW/SW codesign described in [3], consequently, showing the power of its flexibility.

1) DGK Encryption and Decryption: DGK encryption is computed completely in the HW side (i.e., FPGA side) with consecutive Modular Multiplications (MMs) and Modular Exponentiations (MEs). The SW side controls the overall computation process as well as data and microcodes transfers between the SW and HW sides.

DGK decryption consists of two phases: (1) precomputation and (2) main computations. The precomputation is performed only once for each key and it constructs the DLUT and then sorts and stores it in L3-MEM. The main computations consists of two or three steps for the fast and safe variants, respectively. First, an ME is performed in the HW side for calculating $m'$ using (2). Second, a binary search with $\text{lsb}_i(m')$ is performed from the DLUT in the SW side to obtain $m$. Third, for the safe variant, the validity of $m$ is verified with an extra ME in the HW side.

2) Squared Euclidean Distance: The SED computation consists of two parts: first, the middle-terms of (6) using Alg. 1 or Alg. 2 and, second, the final distances of (7).

a) Alg. 1: A distance between the user’s input $x$ and a vector $y_i$ in the server’s database is computed in a single CP core. Each CP core computes consecutive MEs and MMs (i.e., lines 3–5 of Alg. 1) for a specific $i$ and then two MMs to compute (7). I.e., the computation proceeds in a row-wise manner where $Y$ is seen as a matrix with $n$ rows (vectors) and $m$ columns (vector elements). All computation happens in the HW side and the SW side performs simple control and data transfer tasks.

b) Alg. 2: The CP cores operate in a mixed column-and-row-wise manner. First, they operate column-wise so that columns (i.e., different iterations of the for-loop in line 2) are assigned to different CP cores. Each CP core computes MMs (i.e., lines 3–7) for a specific column $j$. According to Alg. 2, different CP cores (different $j$) must contribute to the same $t_i$. This is implemented so that each CP core has a local copy of $t_i$ until the end of the column-wise processing, after which they are combined with MMs in a row-wise manner. Second, the CP cores compute (7) in a row-wise manner similarly to Alg. 1. Obviously, the SW side now performs more control and data transfer tasks but the HW side performs fewer computations.

IV. RESULTS AND ANALYSIS

To evaluate the performance of DGK and to demonstrate the efficiency of the implementation for acceleration of PETs, we implemented it on real hardware. We targeted Xilinx Zynq programmable SoCs and specifically Avnet ZedBoard Zynq...
Evaluation and Development Kit [16] that includes a low-cost Xilinx Zynq-7020 xc7z020clg484-1 [15]. The target chip includes a dual-core ARM Cortex A9 (the SW side) and an Artix-7 FPGA (the HW side). For the SW side, we used C+1 and Xilinx SDK for developing software for a Real-Time Operating System (RTOS). For the HW side, we used Verilog and Vivado 16.3. The resource requirements of the HW side in Xilinx Zynq-7020 device are summarized in Table I. The clock frequencies for the FPGA and ARM are 122 and 667 MHz, respectively. Based on Vivado, the total power consumption is about 3.2W. The following results are final post-place&route results and validated with real hardware.

A. Performance of Target Applications

Table II show the encryption and decryption timings. Encryption that requires operations modulo $N$ ($\kappa = 2048$ bits) is expectedly slower than decryption that operates with smaller modulus $p$ ($\kappa/2 = 1024$ bits). As most of the delay for encryption is in the HW side, multiple encryptions can be performed in parallel by utilizing the $N \cdot M$ CP cores of the multi-core architecture. However, the SW side delays dominate encryptions making parallel processing more difficult. Table II presents decryption results for a 16-bit $u$. As will be seen below, this precision is too small for certain SAT computations that require up to 22-bit precision. The effect of increased precision to timings is small: encryption and decryption times increase only about 1% and 12%, respectively, for $\ell = 22$ (and $\lambda = 48$) compared to Table II. The most significant disadvantage of increased precision is the size of the sorted DLUT: up to 384 KB for $\ell = 16$ and $\lambda = 32$, 8192 KB for $\ell = 20$ and $\lambda = 40$, 16384 KB for $\ell = 21$ and $\lambda = 40$, and 36864 KB for $\ell = 22$ and $\lambda = 48$. The exact size of a sorted DLUT is $u \cdot (\lceil \lambda/8 \rceil + \lceil \ell/8 \rceil)$ bytes.

Similarly to [3], Table III collects the timings of SED computations with two database densities of 25% and 100% (the portion of nonzero values) and three database sizes (small (S): $n = 32$, $m = 16$, medium (M): $n = 128$, $m = 32$, and large (L): $n = 512$, $m = 64$). Additionally, we provide results with two database precisions $\tau = 4$ and $\tau = 8$ for $y_{ij} \in \mathbb{Z}_2^\kappa$; [3] considered only $\tau = 8$ but this precision requires an up to 22-bit precision ($\ell = 22$) for the distances $d_i$. Hence, we provide results also for the smaller precision $\tau = 4$ which allows using a 16-bit plaintext space for all database variants and also has practical relevance as shown, e.g., in [13].

### Table I

| Component (#) | LUTs | REGs | SLICES | BRAMs | DSPs |
|---------------|------|------|--------|-------|------|
| Single CP core design | | | | | |
| External interface | 73 | 54 | 28 | 0 | 0 |
| Data-path | | | | | |
| MMX | 699 | 1295 | 314 | 0 | 16 |
| MAS | 152 | 1 | 58 | 0 | 1 |
| Other | 178 | 387 | 75 | 0 | 0 |
| DMM | 164 | 0 | 86 | 0 | 4 |
| Control unit | 327 | 196 | 111 | 0 | 0 |
| IMEM | 197 | 0 | 91 | 0 | 1 |
| Tot. resource usage | 1781 | 1933 | 763 | 5 | 16 |

### Table II

| | HW/SW codesign system (single CP core design) | | | |
| --- | ------------------------------------- | | | |
| Operation ($\kappa = 2048$) | Latency ($\#$ of cycles) | Time (ms) | PMU (KB) |
| Enc | 1759184 | 1976 | 14.45 | 771 |
| Dec (Fast) | 210621 | 2197845 | 3.30 | 6/0 |
| Dec (Fast, CT $h$) | 280762 | 2217072 | 3.33 | 6/0 |
| Dec (Safe) | 436272 | 3546337 | 5.32 | 7/0 |
| Dec (Safe, CT $h$) | 581564 | 3584791 | 5.38 | 7/0 |

*Peak Memory Usages (PMUs) of the HW (1xL1, L2) and SW (L3) sides.

**DLUT searches may have small timing variation; otherwise fully CT.**
may not fully reflect the speedups in real applications. On the other hand, packing is a computationally intensive operation that increases the load of the sender (the server) [3]. Finally, we emphasize that the precision offered by DGK (e.g., 22 bits) is enough for many practical applications and in those cases it offers major benefits.

### C. Security Model

Because PETs are the primary target applications, we consider a security model where the adversary is the other party of the protocol. I.e., an adversarial user aims to find out the server’s database $Y$ and an adversarial server aims to find out either (a) the user’s secret key $sk$ or (b) the user’s input $x$ (note that (a) implies (b), but not vice versa). We assume that DGK and the protocol are secure and focus on information leakage from the implementation. Furthermore, we assume that the adversary lacks physical access to the other party’s computation platform (also via malware) and limit our analysis to remote timing side-channel attacks. Thus, Table II includes results also for Constant Time (CT) variants. The basic modular arithmetic (multiplication, addition, and subtraction) is CT by default, but MEs have two versions: square-and-multiply (non-CT) and square-and-multiply-always (CT).

Although $\text{Enc}(pk, m)$ does not use the secret key $sk$, it should still be CT to prevent information leakage about $m$ via a timing channel. A CT encryption follows directly from CT MEs. It also suffices to use CT MEs to protect $\text{Dec}(sk, c)$ from leaking $sk$. However, timings of DLUT searches may leak information about $m$. Table lookups are notoriously difficult to make fully CT in SW, while it is easy to make a search (in our case binary search) with a constant number of iterations, memory hierarchy of modern processors (with caches) may incur data-dependent timing variations. Arguably, this leakage is not significant enough in our setting to compromise $x$ for a number of reasons. First, the server may measure the timing between sending the ciphertext (see Fig. 1) and the beginning of the following phase (typically the garbled circuit phase). This timing, therefore, sums $n$ decryptions and prevents from collecting information about individual $x_j$. Second, the ciphertexts contain distances that include only indirect information about $x$. Third, the server cannot cheat by using fabricated database entries or incorrect calculations to increase the dependencies between $d_i$ and $x$ because the user would immediately notice it as a poor quality of the service, and so on. Nevertheless, timings of DLUT lookups should be carefully considered for each particular application.

An adversarial user wants to find out $Y$ by exploiting the timing of computing SEDs. Again, the user can measure only the overall timing of computing all distances and, hence, can make estimates, e.g., on the density of $Y$ and the sum of Hamming weights of all $y_{i,j}$. It is evident that this leakage is not enough to compromise $Y$ or to construct $Y' \approx Y$ which functions similarly. Even this small leakage can be prevented by using a CT-variant of Alg. 1 by using CT MEs for computations with $y_{i,j}$. The cost depends particularly on the density of $Y$ but also on how $y_{i,j}$ are distributed in $\mathbb{Z}_{2^n}$.

### V. Conclusions

We presented an efficient implementation of DGK on a programmable SoC and demonstrated its feasibility for accelerating privacy-preserving computation of SEDs, an important operation for many PETs. To the best of our knowledge, this was the first reported implementation of DGK that uses HW acceleration. We also provided comparisons between two AHE schemes: DGK [5, 6] and Paillier [4] cryptosystems.

DGK benefits greatly from the HW/SW codesign paradigm. The HW side can efficiently accelerate modular arithmetic (especially, exponentiations) and privacy-preserving SED computations can efficiently utilize the parallel processing capabilities of the multi-core architecture. Implementing DGK

### TABLE III

| Parameters setting | Multi-core HW/SW codesign system (12 CP cores design, $M = 6$ and $N = 2$) |
|--------------------|---------------------------------------------------------------------------|
|                    | DGK#1 (Straightforward) (Alg. 1 + (7)) | DGK#2 (Optimized) (Alg. 2 + (7)) |
|                    | S: $(n = 32, m = 16)$ | M: $(n = 128, m = 32)$ | L: $(n = 512, m = 64)$ |
| Total Time (ms)    | PMU+ (KB) | PMU+ (KB) | PMU+ (KB) | PMU+ (KB) | PMU+ (KB) |
|--------------------|-----------|-----------|-----------|-----------|-----------|
| **Database density=25%** | S | M | L | S | M | L |
| $\tau = 4$ | 2.11 | 13.48 | 101.84 | 3.39 | 22.35 | 172.79 |
| $(\lambda = 32, \ell = 16)$ | 97 | 97 | 328 | 97 | 97 | 328 |
| **Database density=75%** | S | M | L | S | M | L |
| $\tau = 8$ | 5.25 | 32.45 | 10.84 | 9.96 | 18.44 | 83.14 |
| $(\lambda \in \{40, 48\}$, $\ell \in \{20, 21, 22\}$) | 89 | 89 | 2412 | 89 | 89 | 2412 |
| $\tau = 4$ | 5.63 | 34.44 | 11.36 | 97 | 97 | 97 |
| $(\lambda = 32, \ell = 16)$ | 44.73 | 147.40 | 79.38 | 97 | 97 | 97 |
| **Database density=100%** | S | M | L | S | M | L |
| $\tau = 8$ | 6.63 | 44.73 | 172.79 | 3.39 | 22.35 | 172.79 |
| $(\lambda \in \{40, 48\}$, $\ell \in \{20, 21, 22\}$) | 97 | 97 | 328 | 97 | 97 | 328 |

* Peak Memory Usages (PMUs) of the HW side (i.e., includes 12 x L1+L2; and 168 KB in total), and the SW side (i.e., L3 includes 256 KB on-chip memory (OCM) and 512 MB off-chip memory (DDR3)).
decryption efficiently only in HW would be difficult because it requires large DLUTs or, otherwise, brute-force searches through the plaintext space must be done. Hence, efficient decryption requires interplay between the HW and SW sides.

The comparison between DGK and Paillier cryptosystems showed that DGK is significantly faster: Encryption is more than 12 times faster, decryption speedups are 34–72 times, and SED computations are about 2 to 3.5 times faster. Although the numbers are clearly faster for DGK than for Paillier, the small plaintext space of DGK may become an issue in practice.

To summarize, we showed that DGK cryptosystem is suitable for efficient HW/SW codesign implementation. It provides a good alternative for the more traditional Paillier cryptosystem used in many PETs. If the application is such that the small plaintext space does not become an issue, then significant speedups can be achieved by using DGK instead of Paillier and accelerating it with our implementation.

ACKNOWLEDGMENT

This work has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 780108 (FENTEC).

REFERENCES

[1] N. Sklavos, R. Chaves, G. D. Natale, and F. Regazzoni, Hardware Security and Trust: Design and Deployment of Integrated Circuits in a Threatened Environment. Springer, 2017.

[2] I. San, N. At, I. Yakut, and H. Polat, “Efficient Paillier cryptoprocessor for privacy-preserving data mining,” Secur. Commun. Netw., vol. 9, no. 11, pp. 1535–1546, 2016.

[3] M. Bahadori and K. Järvinen, “A programmable SoC based accelerator for privacy-preserving applications and functional encryption,” Unpublished manuscript, 2020.

[4] P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” in EUROCRYPT 1999, ser. LNCS, vol. 1592. Springer, 1999, pp. 223–238.

[5] I. Damgård, M. Geisler, and M. Kroigaard, “Efficient and secure comparison for on-line auctions,” in ACISP 2007, ser. LNCS, vol. 4586. Springer, 2007, pp. 416–430.

[6] ——, “A correction to “efficient and secure comparison for on-line auctions”,” Cryptology ePrint Archive, Report 2008/321, 2008, https://eprint.iacr.org/2008/321.

[7] A. C.-C. Yao, “How to generate and exchange secrets,” in SFCS 1986. IEEE, 1986, pp. 162–167.

[8] A.-R. Sadeghi, T. Schneider, and I. Wehrenberg, “Efficient privacy-preserving face recognition.” in ICISC 2019, ser. LNCS, vol. 9209, Springer, 2014, pp. 229–244.

[9] D. Evans, Y. Huang, J. Katz, and L. Malka, “Efficient privacy-preserving biometric identification,” in NDSS 2011, 2011.

[10] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft, “Privacy-preserving face recognition,” in PETS 2009, ser. LNCS, vol. 5672. Springer, 2009, pp. 235–253.

[11] H. Li, L. Sun, H. Zhu, X. Lu, and X. Cheng, “Achieving privacy preservation in WiFi fingerprint-based localization,” in INFOCOM 2014. IEEE, 2014, pp. 2337–2345.

[12] Y. Yang and K. Järvinen, “The death and rebirth of privacy-preserving WiFi fingerprint localization with Paillier encryption,” in INFOCOM 2018. IEEE, 2018, pp. 1223–1231.

[13] K. Järvinen, H. Lepnikoski, E.-S. Lohan, P. Richter, T. Schneider, O. Tkachenko, and Z. Yang, “PILOT: practical privacy-preserving indoor localization using outsourcing,” in EuroS&P 2019. IEEE, 2019, pp. 448–463.

[14] R. Zhang, J. Zhang, Y. Zhang, J. Sun, and G. Yan, “Privacy-preserving profile matching for proximity-based mobile social networking,” IEEE J. Sel. Areas Commun., vol. 31, no. 9, pp. 656–668, 2013.

[15] Xilinx Inc., “Zynq-7000 all programmable SoC,” Technical Reference Manual (UG585), 2016, https://www.xilinx.com/support/documentation/userguides/ug585-Zynq7000-TRM.pdf.

[16] Avnet, “Zynq™ evaluation and development hardware user’s guide (ZedBoard),” User’s Guide, 2014, http://zedboard.org/sites/default/files/documentation/ZedBoard_HW_UG_v2_2.pdf.