IEICE TRANS. INF. & SYST., VOL.E106–D, NO.12 DECEMBER 2023

PAPER Special Section on Forefront Computing

FPGA-based Garbling Accelerator with Parallel Pipeline Processing

Rin OISHI[†a)], **Junichiro KADOMOTO**[†b)], ***Nonmembers***, **Hidetsugu IRIE**[†c)], ***Member***, ***and*** **Shuichi SAKAI**[†d)], ***Fellow***

SUMMARY As more and more programs handle personal information, the demand for secure handling of data is increasing. The protocol that satisfies this demand is called Secure function evaluation (SFE) and has attracted much attention from a privacy protection perspective. In two-party SFE, two mutually untrustworthy parties compute an arbitrary function on their respective secret inputs without disclosing any information other than the output of the function. For example, it is possible to execute a program while protecting private information, such as genomic information. The garbled circuit (GC)—a method of program obfuscation in which the program is divided into gates and the output is calculated using a symmetric key cipher for each gate—is an efficient method for this purpose. However, GC is computationally expensive and has a significant overhead even with an accelerator. We focus on hardware acceleration because of the nature of GC, which is limited to certain types of calculations, such as encryption and XOR. In this paper, we propose an architecture that accelerates garbling by running multiple garbling engines simultaneously based on the latest FPGA-based GC accelerator. In this architecture, managers are introduced to perform multiple rows of pipeline processing simultaneously. We also propose an optimized implementation of RAM for this FPGA accelerator. As a result, it achieves an average performance improvement of 26% in garbling the same set of programs, compared to the state-of-the-art (SOTA) garbling accelerator.

key words: *secure function evaluation, garbled circuit, FPGA*

1. Introduction

In recent years, there have been many cases of malicious extraction of personal data. Thus, privacy is emphasized as personal data is used more and more, and there is an increasing demand for the secure handling of data in many programs. For example, machine learning often uses data containing personal information. Secure function evaluation (SFE)—a protocol that allows two or more mutually untrustworthy people to compute an arbitrary function without disclosing their inputs and without disclosing any information other than the output of the function—can solve this problem.

One of the protocols for SFE is the garbled circuit (GC) proposed by Yao in 1986 [1]. It obfuscates the program by making the program a logic circuit and creating a ciphertext for each gate using a symmetric key cipher. GC research has been conducted from both software and hardware aspects: Bellare et al.'s JustGarble [2] is a software framework for GC, while Ebrahim et al.'s GarbledCPU [3] is an FPGA-based GC accelerator that can execute GC in hardware. It is advantageous in terms of ease of development, as programs can be designed in languages such as C and C++, and existing compilers can be used. On the other hand, it is difficult in terms of performance for practical use.

Fang proposed an FPGA overlay architecture with superior performance for software execution [4]. Based on this, Huang and Leeser show how to apply it to cloud services using AWS F1 instances [5], [6]. These use SHA-1 as the encryption function, which is no longer considered secure [7]. On the other hand, Hussain's FASE [8] solves this problem by using AES as the encryption function and outperforms [5], [6] in performance. FASE is an accelerator implemented using FPGAs running on a cloud server and is intended to be a model for serving programs to multiple clients in parallel. In this model, FASE creates an encrypted program that can be executed by clients from a "netlist", which is a set of gates of the program as a pre-processing step for GC, in order to distribute it to the clients. The creation of this encrypted program is called garbling.

FASE uses a pipeline to sequentially process the gates of the netlist and introduces an AES core for GC-optimized computation based on the fixed-key block cipher optimization presented at JustGarble. On the other hand, the pipeline can process only one gate per cycle. FASE, which has only one pipeline, can garble only one gate per cycle.

We propose an accelerator architecture for multiple engine operation. We introduce a new gate address and output destination management mechanism for multiple operations. We also provide sufficient ports to the RAM structures for multiple pipelines. The FPGA implementation of the multi-port RAM is conducted efficiently using a live value table (LVT). This implementation achieves an average performance improvement of 26% for garbling the same set of programs, compared to the state-of-the-art (SOTA) garbling accelerator.

The contributions of this paper are as follows:

- Based on the latest FPGA-based GC accelerator, an architecture to speed up garbling by parallel processing of the pipeline is proposed, implemented, and evaluated, resulting in an average performance improvement of 26% for garbling the same set of programs, com-

Manuscript received December 14, 2022.
Manuscript revised April 11, 2023.
Manuscript publicized August 2, 2023.

[†]The authors are with The University of Tokyo, Tokyo, 113–8656 Japan.
a) E-mail: oishi@mtl.t.u-tokyo.ac.jp
b) E-mail: kadomoto@mtl.t.u-tokyo.ac.jp
c) E-mail: irie@mtl.t.u-tokyo.ac.jp
d) E-mail: sakai@mtl.t.u-tokyo.ac.jp

DOI: 10.1587/transinf.2023PAP0002

pared to the SOTA garbling accelerator in the two parallel case.
- A new mechanism for the proper management of gate indexes and some RAM addresses is added to cope with changes in the processing order of gates due to pipeline parallelization.
- Stalls of garbling engines decreased by increasing the apparent number of ports by means of a RAM wrapper circuit using LVTs.

The rest of this paper is organized as follows: Sect. 2 describes GC and FASE, Sect. 3 presents the motivation and idea of the proposal, Sect. 5 describes the proposed architecture in detail, Sect. 6 compares the evaluation to existing methods, and Sect. 7 summarizes the conclusions.

2. Background

2.1 GC

GC, one of the SFE protocols invented by Yao in 1986 [1], can be applied to privacy preserving program execution and software rental.

In GC, Alice has input a, and Bob has input b, and they compute $f(a, b)$ without revealing their inputs to each other. The protocol obfuscates the circuit to perform the computation.

1. Let the program to be executed be the circuit C.
2. For each wire of the circuit C, labels corresponding to the values 0 and 1 are created, respectively. The labels are often set to 128 bits.
3. The ciphertext for a gate g (Fig. 1) whose gate logic is $z = g(x, y)$ is created, as shown in Table 1. For each gate, the labels of the output wire z (hereafter output labels)K_z^0, K_z^1 are encrypted with the labels of the two input wires x, y (hereafter input labels)K_x^i, K_y^j using the encryption function $E_{(k_0,k_1)}$ shared beforehand between Alice and Bob. (Hereafter, $E_{(k_0,k_1)}(n)$ means n encrypted with the common key (k_0, k_1)). Since ciphertexts are created for all input patterns, four ciphertexts are created per gate. These four ciphertexts are collectively called the garbled table of gates.
4. Alice sends a garbled table to Bob and sends both input labels corresponding to 0 and 1 on the input wire to Bob in a 1-out-of-2 OT. Bob receives only the label corresponding to his input. A 1-out-of-2 OT is a communication in which the sender appears to send two pieces of data, but the receiver receives only one of the two pieces. Alice uses this to send only the label corresponding to Bob's input without knowing Bob's input.
5. Bob decrypts the output label from the input labels and garbled table for each gate. If the input of the gate is already the output of a computed gate, the output label is used as the input label.
6. Alice shares a map of output labels and truth values with Bob, and Bob shares output labels with Alice; Alice and Bob obtain output values for circuit C.

Steps 2 and 3 are garbling by Alice and steps 5 and 6 are evaluation by Bob.

Various optimization techniques have been proposed for GC; FASE and the implementation proposed in this paper introduce the following:.

- Point and Permute [9]:The addition of random mask bits can reduce the size of the output map.
- Free XOR [10]:The wire label K^1 corresponding to 1 on the wire is made from XOR of the label K^0 corresponding to 0 and $(R\|1)$, which combines the value R defined in common throughout the circuit and 1 $(K^1 = K^0 \oplus (R\|1))$, resulting in XOR, XNOR, and NOT gate garbling being no longer necessary.
- Row Reduction [11]:Ciphertext is reduced by creating output labels in different ways
- Half Gate [12]:The gate is split to make it a half gate and is combined with Row Reduction to reduce the size of the garbled table to two rows.
- Garbling with a Fixed-key Block [13]: Efficient garbling with a fixed-key AES.

2.2 FASE

FASE, the latest FPGA accelerator proposed by Hussain et al., is used in conjunction with a host CPU. The host CPU prepares a netlist, which is the data of a Boolean circuit created from the program and contains the input and output wires and logic information of the gates that make up the circuit. The netlist is optimally scheduled so that the gates can be efficiently garbled, and FASE creates and outputs the common key, input labels, mask bits, and garbled table for the circuit from the netlist. The host CPU (corresponding to Alice in the previous section) receives the output data and sends the data to the client (corresponding to Bob in the previous section) in a 1-out-of-2 OT. This enables the client to execute programs using the received data.

FASE is capable of fast garbling; it manages RAM read/write cycles, improving throughput by at least two orders of magnitude, compared to earlier GC accelerators.

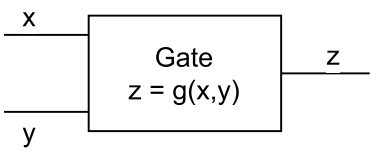


Fig. 1 gate g

Table 1 Truth Table for logic gate g

Input x key	Input y key	Output z key	Ciphertext
K_x^0	K_y^0	$K_z^{g(0,0)}$	$E_{(K_x^0,K_y^0)}(K_z^{g(0,0)})$
K_x^0	K_y^1	$K_z^{g(0,1)}$	$E_{(K_x^0,K_y^1)}(K_z^{g(0,1)})$
K_x^1	K_y^0	$K_z^{g(1,0)}$	$E_{(K_x^1,K_y^0)}(K_z^{g(1,0)})$
K_x^1	K_y^1	$K_z^{g(1,1)}$	$E_{(K_x^1,K_y^1)}(K_z^{g(1,1)})$

The cryptographic core uses 17% fewer resources than most modern GC accelerators.

The garbling performed by FASE is as follows. The netlist, which is the information on the gates of the circuit, is read one by one from the beginning, and labels are randomly created when new labels are required; the XOR circuit obtains output labels by XOR for XOR gates and others that do not require garbling. The other gates are those that encrypt the input label as plaintext using the AES function and the fixed AES key that is common to the entire circuit through a 10-stage pipeline process in the garbling engine. The final output is the key of the entire circuit (R of Free XOR and the fixed key of AES), the input labels, the garbled table and the mask bits.

A feature of FASE is that it uses pipelining to perform garbling, which increases resource utilization and makes garbling more efficient. Furthermore, the read/write cycles of the RAM that stores the output labels are managed to increase port utilization and improve performance.

2.3 Related Work

Based on the FPGA overlay architecture [4] proposed by Fang, studies have been proposed using F1 instances of AWS [5], [6]. These methods are superior to software execution in terms of performance. One of the major differences from FASE is the difference in encryption functions. These methods use SHA-1 as the encryption function, while FASE uses AES. The security of SHA-1 has been questioned [7], and AES is more secure than SHA-1. The length of the label is 128 bits for FASE, while it is 80 bits for these methods. There is also a difference in the netlist handled. These methods use FlexSC [14] to generate netlists, while FASE uses TinyGarble [15]. This difference results in different circuit structures for the same program. The latter tends to have fewer gates. As a result, FASE performs better than them for garbling programs of similar size.

2.4 LVT [16]

LVT, is a technique to realize multi-port RAM by using additional block RAMs, enables the RAM to behave as true multi-port RAM by directing reads to the appropriate memory bank based on the memory bank holding the latest write value.

In this method of implementing multi-port RAM, the hardware consists of a matrix of memory banks, a table called the LVT, and multiple selectors. The memory banks are the memory entities. When writing, values are written to all memory banks connected to the write port. At the same time, the number of the write port is written to the LVT. When reading, the value is read from all memory banks corresponding to the read port and input to the multi-selector, which selects the latest value based on the write port number read from the LVT.

3. Parallel Garbling Accelerator

3.1 Key Issues for Performance Improvement

The garbling speed is the bottleneck for speeding up GC. Therefore, speeding up FASE with architectural techniques is useful to make SFEs more common.

FASE has a single garbling engine, and currently, only one gate can be garbled at a time. To speed up garbling, we propose parallelizing the pipeline that processes the gates. If the pipeline is parallelized n-fold, the throughput is expected to increase up to n-fold. The throughput is increased if there is a possibility that other gates can be processed simultaneously while a gate is stalled due to gate dependencies. On the other hand, if the netlist is too complex and the gates have complex dependencies on each other, the effect of the parallelization is small.

In a single-column pipeline process, the gates stored in the netlist are processed in order, but due to the parallelization of the pipeline, while one side is waiting for processing, the other side may move on to the next gate. To cope with this situation, which could not have occurred in the original FASE implementation, a newly developed mechanism is introduced the manager of the gate index and the RAM address.

FASE may read and write output labels at the same time. A maximum of two read and two write requests can occur simultaneously, which exceeds the number that can be processed in one cycle, because the RAM for the output labels is a dual-port RAM in the FASE implementation. This means that there are not enough ports of RAM for the output labels, at which point a stall occurs. This stall occurs on average in about 13% of all cycles in the benchmarks and is a bottleneck in performance. Parallelization further increases the demand on the number of ports. To solve this problem, LVT is used.

3.2 Parallelization

3.2.1 Duplication

The number of gate processing pipelines is increased from one in the original implementation to several. This means modifying the architecture to allow multiple gates to be processed simultaneously. Here, those that can be simply duplicated are duplicated. Specifically, the garbling engine and the XOR circuit are duplicated.

3.2.2 Multiport RAM

There are parts of FASE that cannot be simply duplicated. These are the RAM that stores gate information, labels, and garbled table. Since this information must be shared by multiple pipelines, the number of read/write ports is the only thing that increases, apparently without leaving a single RAM. However, due to the original lack of ports, many stalls

occur if the conventional RAM wrapper circuit is used. As multi-port RAM is difficult to use on FPGAs, pseudo multi-port RAM is realized using the LVT technique.

3.2.3 Gate Index

The gate index is a number assigned to the gates in the netlist in the order in which they are processed; FASE processes the gates one by one, so the gates in the netlist should be processed in that order. Therefore, the gate indexes should simply be incremented one by one as soon as the processing is finished. As the proposed method processes multiple gates simultaneously, the processing of the other gate may be completed while one of the gates is being processed. In this situation, it is not simple enough to allow the next gate to be processed, so a new mechanism is introduced to manage the indexes correctly.

3.2.4 GT Address

The GT address identifies which gate has the garbled table, except for XOR, XNOR, NOT gates and FF. In other words, not all gates have a garbled table, so the GT address is managed separately from the gate index.

3.2.5 OM Address (Mask Address)

The mask bit is the last digit of the output label corresponding to 0 used in point and permute [9]. This mask bit tells the evaluator whether the output label corresponds to 0 or 1. The OM address is the address that defines the order of the mask bits (i.e., which output label the mask bit corresponds to). The OM address should not be incremented as each gate is processed. The OM address is incremented only when a gate whose output is the output of the entire circuit is processed. As not all gates are outputs of the circuit, the mask address is also managed separately from the gate index.

3.3 Overall Flow of Garbling Computation

Hereafter, the cycle of the GC circuit is called a netlist cycle; the clock cycle of the FPGA accelerator is simply called a cycle.

The basic process flow is the same as that in FASE. The difference is that two gates are processed at the same time, so that if one gate is fully processed in one netlist cycle but the other gate is not, it will wait until the other gate is processed. It is not possible to process gates in different netlist cycles at the same time.

Before starting the procedure, the FPGA accelerator receives the netlist from the host CPU and stores it internally.

1. The R and AES keys to be used in Free XOR [10] described in 2.1 are created and sent to the host CPU.
2. Labels corresponding to the constants 0 and 1 are created for the initial value of the D flip-flop (DFF) and sent to the host CPU. This is because garbling a circuit with flip-flops requires the initial value of the flip-flop in the first netlist cycle.
3. For each netlist cycle
   a. For each DFF,
      i. If it is the first netlist cycle, the output labels are either constant labels or circuit input labels. If it is a circuit input label, it is created and sent to the host CPU.
      ii. For any other netlist cycle, the output labels are copies of the input labels.
   b. For each gate,
      i. If the gate is connected to an input of the circuit, an input label is created and sent to the host CPU.
      ii. Output labels and a garbled table are created. In the case of XOR, XNOR, and NOT gates, only the output labels are created.
      iii. If the output of the gate is the output of the circuit, the mask bits used by point and permute are stored internally.
   c. At the end of the netlist cycle, all mask bits are sent to the host CPU.
4. The host CPU communicates with the client and computes outputs while using OT.

3.4 Gate Processing Pipeline

Each entry in the netlist describes a single logic gate or flip-flop in the target garbled circuit. Each pipeline reads the netlist one entry at a time and performs garbling over three stages in the pipeline. A single gate or flip-flop is processed in the pipeline. Three stages are called the fetch and decode stage, the execution stage, and the write-back stage, respectively.

3.4.1 Fetch and Decode Stage (F&D)

Fetch the gate data from the netlist; since netlist RAM always reads data corresponding to one and two gate indices ahead, the gate data for the gate index to be read can be read instantly. Therefore, in this stage, the gate data is read and stored in registers as the input label address, logic (in the case of a gate), and *is_output*.

3.4.2 Execution Stage (E)

The output labels and ciphertexts of the Garbled Table (for non-XOR gates) are calculated. For non-XOR gates, 11 cycles are required for encryption; for XOR gates and FF, the computation can be done in one cycle.

3.4.3 Write-Back Stage (W)

Write back the output labels and Garbled Table (for non-XOR gates) from the Garbled Engine or XOR.

4. Parallel Processing

The accelerator fetches two gates or FF data from the netlist. If they are in the same netlist cycle, they are sent to the next stage; otherwise, the one with the larger netlist cycle is stalled. It is not possible to process gates in different netlist cycles at the same time. In the F&D stage, label data is loaded. If the flag of the label data to be read is not ready, the stall occurs because the labels have not yet been generated.

As an example, consider the circuit shown in Fig. 3, where the left gate in Fig. 2 is represented as shown on the right.

The blue arrows represent the inputs of the entire circuit and the red arrows represent the outputs of the entire circuit. This circuit is an adder circuit, and when n-cycle clock inputs are made using FF and inputs are made to wires 0 and 1 simultaneously, one bit at a time, the result of the addition of the n bits of the inputs is obtained from the output one bit at a time. Garbling is performed based on this circuit.

The timeline of the 2-row parallel process is shown in Fig. 4. Since the restriction on the number of ports is removed, only hazards caused by dependencies are generated. The hazard at gate 3 is due to the dependency to use the output label of FF 2 as input, and the hazard at gate 7 is due to the dependency to use the output label of gate 5 as input.

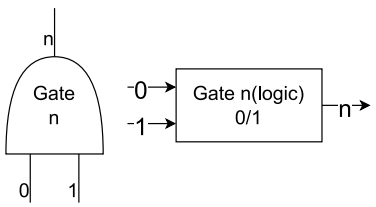


Fig. 2 Gate example

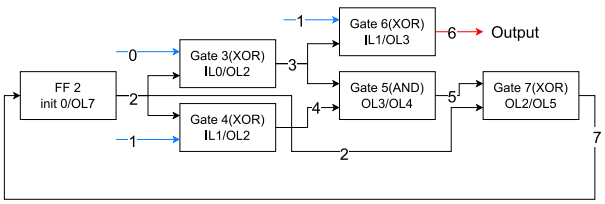


Fig. 3 Circuit example of adder circuit

5. Architecture

5.1 Overview

A two-parallel pipeline version is described here. Figure 5 shows our proposed architecture. The colored areas in the diagram are our extensions. We have replicated the garbling engine and XOR circuits (in blue) and created a new address manager (in purple). The netlist, in orange, has more ports, and the key generator creates two labels at the same time. The red part is RAM, where the number of ports is extended using LVT.

The two gates are read from the netlist module simultaneously. The two input labels corresponding to each gate are read out, and if they have not been created, they are created. If both input labels for a gate are not ready, it waits until they are ready. When the input labels are ready, the output labels and the garbled table are created through the garbling engine or the XOR circuit. After the creation process, the gate index is incremented, and the next gate is read from the netlist. In this case, the incrementing method is devised so that processing is carried out from the smallest gate index.

5.2 Key Generator

The key generator is a module that creates keys and labels at the required time. It first creates the AES key for the entire circuit and the Free XOR key R. It also creates the constant labels for the initial values of the FFs. When it enters the gate processing stage, it creates the input labels for the gates if no labels have been created for the inputs of the entire circuit.

Due to the possibility of creating all input labels for two gates at the same time, we increased the number of labels that the label generator can create at once from two to four.

5.3 Garbling Engine

The garbling engine handles gates other than FF, XOR, XNOR, and NOT gates. This module creates a garbled table and an output label corresponding to an input value of 0 from input labels corresponding to two input values of 0. It has four AES cores with 10-stage pipeline processing and a

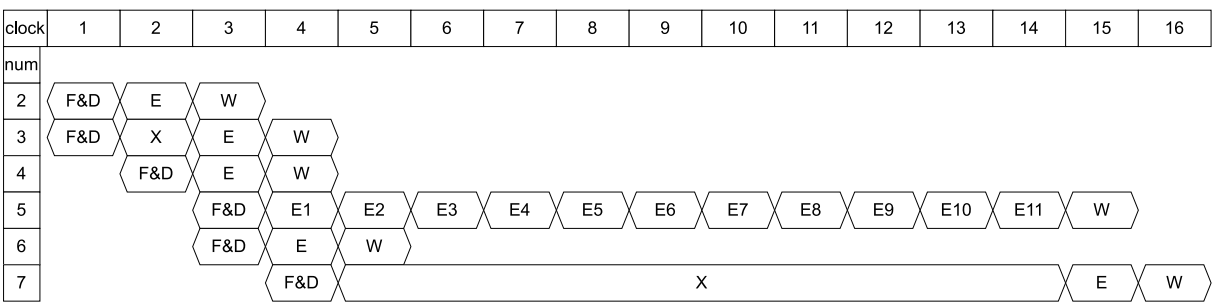


Fig. 4 Timeline of parallelized (2-row) pipeline processing

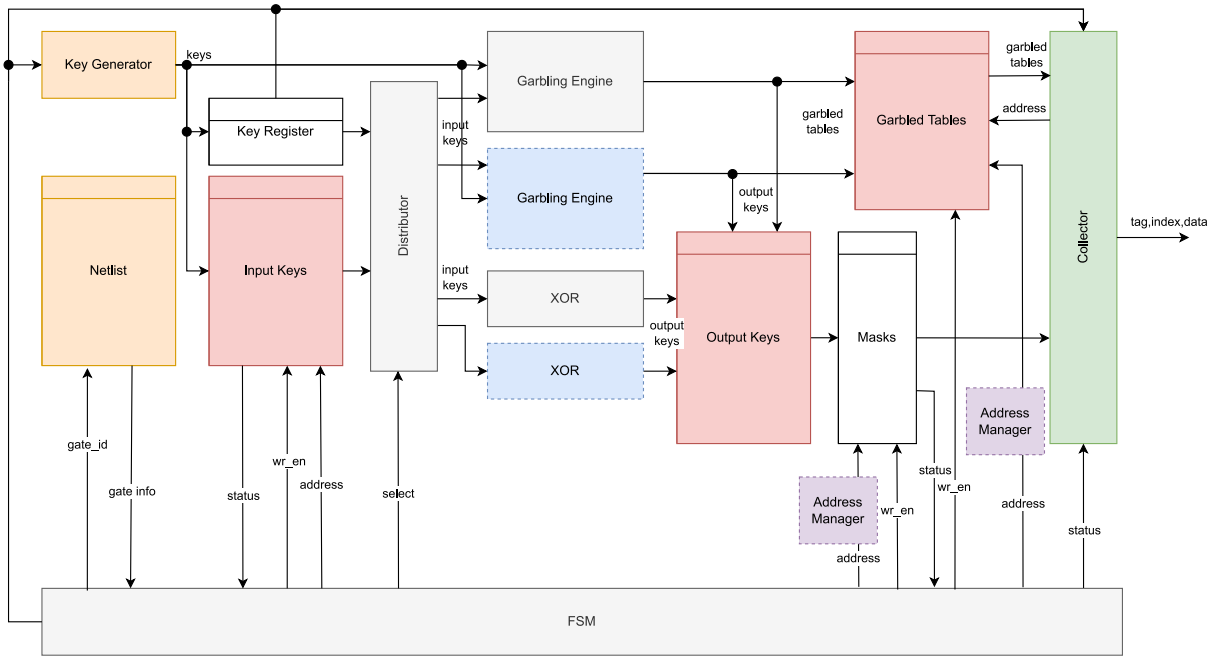


Fig. 5 Architecture of proposed accelerator

10-stage FIFO circuit.

In the simultaneous processing of the two gates, these encryption processes are not dependent on each other. Therefore, we simply duplicate the garbling engine.

5.4 Control Logic

The control logic determines the state of the FSM based on the state of the FPGA as a whole, which determines whether to finish processing a gate and start processing the next gate, and whether to write the created labels, garbled table and mask bits into the RAM.

The FSM has the initial state of IDLE, GETKEYS for R and AESkey creation, CONSTLABELS for constant key creation, WAIT for the wait state, DFF for FF processing, GARBLE for gate processing, MASKS for waiting for mask bit transmission completion, and RSTCOUNTERS to reset the netlist cycle. Those states exist in FASE. We added the DFFANDGARBLE state, in which a gate that is DFF on one side and non-FF on the other side is processed. This state is caused by processing two gates simultaneously.

5.5 Memory Management

The memory that stores the FASE input labels, output labels, and garbled table is a module that manages the dual-port RAM with a wrapper circuit. The order of reading and writing is adjusted to suppress stalls, but stalls still occur in dual-port RAMs due to the small number of ports. The proposed method handles two gates simultaneously, doubling the number of ports. Therefore, if many write and read requests are made at the same time, a large number of stalls occur, but by using LVT, stalls are eliminated.

LVT takes one cycle between writing and reading certain data, so a forwarding mechanism was added by having a cache in the RAM module that uses LVT. Specifically, the data to be written is always written to the cache as well, and when the write signal is set to 1 and the write address and read address match, the data is read directly from the cache without referring to the LVT. This enables data to be read out in the next cycle after the write in the shortest possible time.

The netlist is a RAM that stores information on the gates of a circuit. To read out the information of two gates at the same time, a dual-port RAM was used instead of a single-port RAM.

The input label memory was solved by implementing a four-read four-write memory using LVT. Sixteen dual-port RAMs with one read and one write are used as memory banks.

The output label memory was also solved by implementing a four-read four-write memory using LVT. This also uses 16 dual-port RAMs with one read and one write as memory banks. This also eliminated a stall cycle that was 13%.

The garbled table memory implements a one-read, two-write memory using LVT. There are two memory banks.

5.6 Gate Index Management

The gate index is the index of the gate being processed, and there are a total of two gate indexes as identification numbers for each gate being processed. The incrementing rule for the gate indexes is as follows: the increment signal is always 1 for FF and 1 for the other gates only when all gate input labels are ready and 0 otherwise.

- The initial indexes are set to 0 and 1, respectively.
- If both increment signals are 1 and neither gate index overflows, each new index shall be one after the larger of the original two indexes and two after the index of the larger of the original two indexes.
- If the one-way increment signal is 1 and the corresponding gate index does not overflow, the new index of the one to be incremented shall be one after the larger of the original two indexes.
- Otherwise, the index remains unchanged.

By incrementing with this rule, gates can be processed in parallel sequentially in descending order of indexes.

5.7 GT Address Management

The GT address is the address to write the garbled table of the gate corresponding to the gate index 10 cycles ago; the GTbeg address is the GT address 10 cycles ago corresponding to the gate index currently being handled (before the gate has been read from the netlist and processed by the garbled engine). There are two GT addresses and two GTbeg addresses, which is the number of gates.

The GTbeg address works according to the following rules.

- Prepare an initial signal to determine the initial state.
- In the initial state
  - If the gate being processed is not an AND gate, the address is set to -1.
  - If the gate being processed is an AND gate, the address is set to 0. If there are multiple gates, they are assigned addresses 0 and 1 in order of decreasing gate index. The initial signal is changed to get out of the initial state.
- If the FSM is not in the initial state, and the FSM state is other than WAIT, the gate index increment signal is 1 for the previous cycle, and the gate being processed is an AND gate, the latest address already assigned +1 is assigned as the new address. If there are multiple gates, the latest address +1 and +2 are allocated in order of decreasing gate index.
- The address does not change when the above is not the case.

This GTbeg address is output as a GT address after 10 cycles through the FIFO module and becomes the location to be written into GT memory.

A different address is prepared to determine which address can be read out. This address indicates that data has already been written to the previous address and is ready to be read.

5.8 OM Address Management

When a mask bit is written to RAM, it is when the gate whose output has become the output of the entire circuit one cycle earlier has finished processing. This gate is henceforth referred to as the output gate. The signal for whether or not to write the mask bits is "the gate index increment signal is 1 one cycle ago and the gate processed one cycle ago is the output gate". This signal is called the OM address increment signal (OMinc), and since two gates are processed simultaneously, there are two of these signals corresponding to the gates. When this signal is 1, a mask bit is written to the OM address.

The mask bit is used to identify the output key of the circuit, but not all gates are outputs of the circuit. Therefore, it is necessary to manage the OM address in addition to the gate index. The increment rule of the OM address is as follows.

- Prepare an initial signal to determine the initial state.
- In the initial state
  - If the gate being processed is not an AND gate, the address is set to -1.
  - If the gate being processed is the output gate, the address is set to 0. If there are multiple gates, they are assigned 0 and 1 in order of decreasing gate index. The initial signal is changed to get out of the initial state.
- If the FSM is not in the initial state, and the OM address increment signal of one cycle ago is 1 and the gate being processed is an output gate, the latest address +1 that has already been allocated is allocated as the new address. If there are multiple applicable gates, the previously allocated addresses +1 and +2 are allocated in order of decreasing gate index.
- The address does not change when the above is not the case.

5.9 Collector

The collector is configured to collect the necessary data and to send it to the host CPU. The number of output ports has been doubled to cope with the doubling of the FASE data output per cycle due to the two gates that process the data simultaneously.

6. Evaluation

6.1 Evaluation Setup

We implemented our proposed parallelized garbling accelerator in RTL. We used the public repository of FASE available on GitHub [17] as a base.

The circuit of the proposed method has all the garbling features of FASE. In this implementation, the maximum number of gates that can be supported is 2^{13}, the maximum input of the garbling program is 2^{10} bits, and the maximum output is 2^8 bits. These parameters correspond to the largest size of the benchmarks in Table 2. The larger the size of the benchmark function, the larger the number of gates and I/O

Table 2 Benchmark Functions

Benchmark	Function	Input bits	Netlist cycles	Gates	XORs
Mill_8_8	Millionaire's	8	8	4	3
Add_8_1	Addition	8	1	37	30
Add_8_8	Addition	8	8	5	2
Hamm_32_1	Hamming dist.	32	1	188	157
Hamm_32_32	Hamming dist.	32	32	13	8
Hamm_512_512	Hamming dist.	512	512	21	12
Mult_256_512	Multiplication	256	512	1,699	1,186
MAC_8_1	MAC	8	1	397	231
MAC_16_1	MAC	16	1	1,678	1,077
MAC_32_1	MAC	32	1	7,036	4,805
CORDIC_32_31	Trigonometric	32	31	2,464	1,544
AES_128_11	AES	128	11	4,662	3,225

Table 3 Performance

Benchmark	FASE (cycles)	Proposal (cycles)	Improvement
Mill_8_8	259	252	1.03
Add_8_1	139	133	1.05
Add_8_8	179	172	1.04
Hamm_32_1	354	262	1.35
Hamm_32_32	2,980	2,884	1.03
Hamm_512_512	78,340	75,780	1.03
Mult_256_512	1,681,412	901,124	1.87
MAC_8_1	716	588	1.22
MAC_16_1	2,889	2,241	1.29
MAC_32_1	11,031	8,133	1.36
CORDIC_32_31	129,925	97,003	1.34
AES_128_11	84,044	55,664	1.51

bits, and the larger the RAM size.

We evaluated the proposed accelerator using Xilinx Vivado 2019.2 with data from the Xilinx Virtex UltraScale VCU108 (XCVU095) evaluation kit. As a comparison, we used FASE, an RTL implementation from a public repository, which met the timing constraints at a maximum of 80 MHz in our evaluation environment described above.

Table 2 shows the benchmark functions, the number of input bits or netlist cycles each benchmark has, and the number of gates each benchmark contains. The evaluation includes the 128-bit AES encryption program. It is a practical and large-scale program.

6.2 Performance

Our implementation operates at the same operating frequency as FASE and the critical path is not extended. Therefore, it is simply better to have smaller cycles required for garbling and worse to have larger cycles. The cycles required for garbling are summarized in Table 3. The performance improvement rate is calculated as follows: cycles required for garbling of FASE/cycles required for garbling of the proposed method. The largest performance improvement among the benchmarks was achieved by the summation program (Mult_256_512), with a performance improvement of 87%. On average, the performance of all benchmarks was improved by 26%. When gates with independent wire inputs alternate in the netlist, they can be processed simultaneously, and parallelization improves performance.

Table 4 Resource Utilization of FASE and Proposal

Resource	FASE		Proposal		Ratio
	Num	%	Num	%	Proposal/FASE
LUT	56,111	10.44	158,902	29.56	2.83
LUTRAM	315	0.41	619	0.81	1.97
FF	15,415	1.43	44,497	4.14	2.89
BRAM	69	3.96	585	33.83	8.53
IO	334	40.14	618	74.28	1.85
BUFG	7	0.73	9	0.94	1.29

Table 5 Resource Utilization of modules

Resource		LUT	LUTRAM	FF	BRAM
Total	Num	56,111	315	15,415	69
(FASE)	%	10.44	0.41	1.43	3.96
Total	Num	158,902	619	44,501	585
(Proposal)	%	29.56	0.81	4.14	33.83
Garbling Engine	Num	26,575	285	5,445	0
(FASE)	%	4.94	0.37	0.51	0
Garbling Engine	Num	52,975	557	10,856	0
(Proposal)	%	9.85	0.73	1.01	0
Input Labels	Num	1,073	0	1,031	4
(FASE)	%	0.20	0	0.10	0.23
Input Labels	Num	4,965	0	3,094	64
(Proposal)	%	0.92	0	0.29	3.7
Output Labels	Num	22,012	0	8,596	28.5
(FASE)	%	4.09	0	0.80	1.65
Output Labels	Num	41,539	0	25,116	456
(Proposal)	%	7.73	0	2.34	26.39

However, when the circuit is small, the effect of wire dependence is significant and performance is not improved. Changing the order of gates in the netlist improves performance in some case.

6.3 Resource Utilization

Table 4 shows the resources used. BRAM usage is increased by 8.53 times due to the use of LVT. This is the largest increase in the resources. However, despite the increase in BRAM usage, it still consumes only 34% of the BRAM on the FPGA. The remaining resources increase by a factor of 1.5 to 3.

Table 5 shows the resources used in the internal modules of FASE and the proposal method. The garbling engine is approximately doubled in resources used due to the duplication. The input and output labels, in particular, increased the use of BRAM, and much of the increase in BRAM consumption for the system as a whole is due to these modules.

6.4 Scalability

In the case of 3- or 4-row architectures, performance increases, but performance is limited by wire dependence. In this case, IO is the bottleneck in terms of resources used, which can be overcome by upgrading to a larger FPGA.

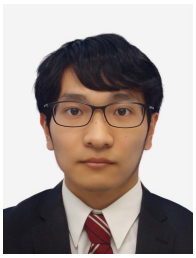
7. Conclusion

This paper proposed a parallelized garbling accelerator. The accelerator increased the garbling throughput by running

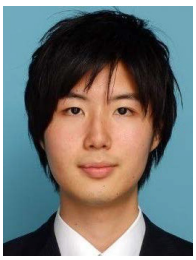
multiple pipelines simultaneously. The control logic to enable parallelized garbling was developed. An RTL implementation of the accelerator, which is capable of running two garbling engines in parallel, was realized. This accelerator can run as parallel as possible, taking into account netlist dependencies. Simulation-based performance comparisons showed an average performance improvement of 26%. Our accelerator can garble the 128-bit AES encryption program within 700μs.

References

[1] A.C.-C. Yao, "How to generate and exchange secrets," 27th Annual Symposium on Foundations of Computer Science (SFCS 1986), pp.162–167, 1986.
[2] M. Bellare, V.T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," 2013 IEEE Symposium on Security and Privacy, pp.478–492, 2013.
[3] E.M. Songhori, T. Schneider, S. Zeitouni, A.R. Sadeghi, G. Dessouky, and F. Koushanfar, "GarbledCPU: A MIPS processor for secure computation in hardware," 2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC), pp.1–6, 2016.
[4] X. Fang, S. Ioannidis, and M. Leeser, "Secure function evaluation using an fpga overlay architecture," Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '17, New York, NY, USA, p.257–266, Association for Computing Machinery, 2017.
[5] K. Huang, M. Gungor, X. Fang, S. Ioannidis, and M. Leeser, "Garbled circuits in the cloud using fpga enabled nodes," 2019 IEEE High Performance Extreme Computing Conference (HPEC), pp.1–6, 2019.
[6] M. Leeser, M. Gungor, K. Huang, and S. Ioannidis, "Accelerating large garbled circuits on an fpga-enabled cloud," 2019 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC), pp.19–25, 2019.
[7] P.K.C.A.A.A.G.Y.M.G.A.P.B.G.C.B.G. Marc Stevens (CWI Amsterdam), Elie Bursztein (Google), "Google online security blog: Announcing the first sha1 collision," 2017. https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html
[8] S.U. Hussain and F. Koushanfar, "FASE: FPGA acceleration of secure function evaluation," 2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp.280–288, 2019.
[9] D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols," Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, STOC '90, New York, NY, USA, pp.503–513, Association for Computing Machinery, 1990.
[10] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free XOR gates and applications," Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part II, ICALP '08, Berlin, Heidelberg, pp.486–498, Springer-Verlag, 2008.
[11] M. Naor, B. Pinkas, and R. Sumner, "Privacy preserving auctions and mechanism design," Proceedings of the 1st ACM Conference on Electronic Commerce, EC '99, New York, NY, USA, pp.129–139, Association for Computing Machinery, 1999.
[12] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole: Reducing data transfer in garbled circuits using half gates." Cryptology ePrint Archive, Report 2014/756, 2014.
[13] M. Bellare, V.T. Hoang, S. Keelveedhi, and P. Rogaway, "Efficient garbling from a fixed-key blockcipher," Proceedings of the 2013 IEEE Symposium on Security and Privacy, SP '13, USA, pp.478–492, IEEE Computer Society, 2013.
[14] X. Wang and K. Nayak, "Flexsc," 2014. https://github.com/wangxiao1254/FlexSC
[15] E.M. Songhori, S.U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar, "Tinygarble: Highly compressed and scalable sequential garbled circuits," 2015 IEEE Symposium on Security and Privacy, pp.411–428, 2015.
[16] C.E. LaForest and J.G. Steffan, "Efficient multi-ported memories for FPGAs," Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA '10, New York, NY, USA, p.41–50, Association for Computing Machinery, 2010.
[17] ACESLabUCSD, "FASE: FPGA acceleration of secure function evaluation," 2019. https://github.com/ACESLabUCSD/FASE



Rin Oishi is currently a student of the Department of Information and Communication Engineering, Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan. His research interests include secret computation using computer architecture.



Junichiro Kadomoto is currently an assistant professor with the Department of Information and Communication Engineering, Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan. His research interests include energy-efficient computer architectures. He received the Ph.D. degree in information science and technology from The University of Tokyo. He is a member of ACM, IEEE, and IPSJ.

Hidetsugu Irie is currently a professor with the Department of Information and Communication Engineering, Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan. His research interests include computer systems and human–computer interaction. He received the Ph.D. degree in information science and technology from The University of Tokyo. He is a member of ACM, IEEE, IEICE, and IPSJ.

Shuichi Sakai is currently a professor with the Department of Information and Communication Engineering, Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan. His research interests include computer systems and their applications. He received Dr. Eng. degree from The University of Tokyo in 1986. He is a senior member of IEEE, and a Fellow of IEICE and IPSJ. He was the recipient of the IEEE Outstanding Paper Award in 1995.