Hardware/Software Cooperative Design Against Power Side-Channel Attacks on IoT Devices

Mingyu Yang, *Graduate Student Member, IEEE*, Tanvir Ahmed, *Member, IEEE*, Saya Inagaki, Kazuo Sakiyama, *Senior Member, IEEE*, Yang Li, *Member, IEEE*, and Yuko Hara-Azumi, *Member, IEEE*

***Abstract*—With the growth of Internet of Things (IoT) era, the protection of secret information on IoT devices is becoming increasingly important. For IoT devices, attacks that target information leakage through physical side-channels (e.g., a power side-channel) are a major threat in many use cases because IoT devices can be accessed easily by a hostile third party. However, securing resource-constrained IoT devices against side-channel attacks is a challenging issue. Generally, it is difficult to satisfy the requirements on side-channel protection while maintaining the low-power and real-time constrains of IoT devices. In this article, we propose a hardware/software cooperative design for cryptosystems that is suitable for resource-constrained IoT devices. Combining a security-oriented processor design (i.e., an instruction set architecture definition and its architectural structure) and careful implementations of masked software implementation for cipher algorithms can effectively improve the power–performance–area (PPA) while suppressing power side-channel leakage. In our evaluation, for three ciphers (Chaskey, Simon, and advanced encryption standard), we demonstrate that our work is superior to state-of-the-art works (two RISC-V processors and a small-scale low-power processor) in terms of both PPA and power side-channel protection.**

***Index Terms*—Constrained devices, embedded processor, hardware security, Internet of Things (IoT), side-channel attack.**

I. Introduction

WITH the ongoing development of Internet of Things (IoT) technologies, there is a growing need to protect an increasing amount of confidential data processed on IoT edge devices. To enable a variety of IoT devices (particularly low-end devices) to process cipher algorithms efficiently in real time, the hardware and software implementations of lightweight ciphers are important research areas. Lightweight cipher algorithms are designed to be simple using basic operations and/or small amounts of memory while proving their security level mathematically [1]. However, physical side-channel attacks could decipher secret information from IoT devices. Unlike a covert channel caused by the architectural design of high-end devices (e.g., branch speculation and complex memory hierarchy), a side-channel that emits physical information, such as power consumption and electromagnetic waves, presents significant risk threat even in low-end devices because physical access to the IoT devices could be realized easily by a hostile third party [2].

Masking is a provably secure countermeasure against such physical side-channel attacks [3]. To protect the secret data, masking divides them into multiple *shares* by introducing fresh randomness. A masking scheme using $d+1$ shares is theoretically safe against dth-order attacks. In addition, masking is applicable to software and hardware implementations [4], [5], both of which have advantages and disadvantages. Software masking can be applied more easily than hardware masking, which requires architectural changes. However, software masking suffers from longer latency by introducing more shares. Furthermore, hardware resources are typically shared between different operations to suppress circuit area. Thus, $d+1$ shares may be insufficient to protect against dth-order attacks [6]. In other words, to obtain effective protection against attacks, more shares are required, which results in increased latency overhead [7]. On the other hand, although hardware masking can reduce latency via parallel processing of computations on the shares, it incurs large circuit area and power overheads. Naturally, if the baseline implementation of the microarchitecture is larger and more complex, the masked implementation will have a more significant circuit footprint [8].

Conventionally, countermeasures against power side-channel attacks have exclusively been considered from either a software or hardware perspective. In contrast, a hardware/software cooperative approach is expected to holistically provide a more efficient solution to mitigate the incurred overheads. In addition, improvements to the cipher strength (e.g., increasing the key length or replacing the cipher algorithms) will be required in long-life IoT devices due to the development of new threats. Thus, to develop a cryptosystem on the IoT devices, embedded processors are expected to be preferable compared to cipher-dedicated circuits.

Motivated by these ideas, our work enables a novel cryptosystem design that combines hardware and software approaches in a way that is resistant to side-channel attacks,

Manuscript received 3 October 2023; revised 19 December 2023; accepted 15 January 2024. Date of publication 18 January 2024; date of current version 25 April 2024. This work was supported in part by the Japan Science and Technology (JST) Agency AIP Acceleration Research under Grant JPMJCR20U2; in part by the JST FOREST Program under Grant JPMJFR216P; and in part by the Japan Society for the Promotion of Science KAKENHI under Grant JP20H00590. *(Corresponding author: Mingyu Yang.)*

Mingyu Yang, Tanvir Ahmed, Saya Inagaki, and Yuko Hara-Azumi are with the School of Engineering, Tokyo Institute of Technology, Tokyo 152-8550, Japan (e-mail: mingyu@cad.ict.e.titech.ac.jp; tanvira@cad.ict.e.titech.ac.jp; saya@cad.ict.e.titech.ac.jp; hara@cad.ict.e.titech.ac.jp).

Kazuo Sakiyama and Yang Li are with the Graduate School of Informatics and Engineering, The University of Electro-Communications, Tokyo 182-8585, Japan (e-mail: sakiyama@uec.ac.jp; liyang@uec.ac.jp).

Digital Object Identifier 10.1109/JIOT.2024.3355417

especially power side-channel attacks. Specifically, to achieve an efficient power–performance–area (PPA) and resistance to power side-channel attacks, this work integrates masked software implementations and a security-oriented microprocessor whose instruction set architecture (ISA) and microarchitectural structure are defined to be lightweight cipher friendly. This enables protection of secret data against dth-order attacks using *only* $d+1$*-masked software* unlike previous methods [6]. In addition, our processor design does not employ a duplicated datapath circuit by masked and unmasked modules (i.e., arithmetic logic unit (ALU) and/or register file), differing from most existing hardened processors and cipher-dedicated co-processors [8], [9], [10], [11]. Thus, it enables low-power/energy processing of ciphers and other IoT applications such as eHealth monitoring [12], [13]. Another recent study proposed a masked hardware design for a bit-serial implementation of a RISC-V processor [14] to mitigate the circuit area overhead. In contrast to our hardware/software cooperative design approach, this is a hardware-specific approach that only focuses on ciphers. The most relevant work to our study is the RISC-V Ibex extension to be aware of masked software [15]. Although it is based on a small RISC-V core, our work targets further smaller, resource-limited edge devices for which top-down designs based on existing processors may not be suitable. Our evaluation on lightweight ciphers [Chaskey, Simon, and advanced encryption standard (AES)] demonstrates the effectiveness of our work against state-of-the-art works including [15] in terms of both PPA and power side-channel protection.

Our primary contributions are summarized as follows.

1) By implementing the hardware/software cooperative approach, we realize a cryptosystem design that simultaneously fulfills good PPA to be suitable for constrained IoT edge devices and resistance to power side-channel attacks.
2) From the software perspective, to prevent information leakage that previously masked software suffered due to unintended resource sharing in the underlying hardware architecture, we conduct hardware-aware software optimizations with only minimum shares in a provably secure masking countermeasure.
3) From the hardware perspective, unlike previous works that harden existing processors by introducing a duplicated datapath circuit or cipher-dedicated co-processor [8], [9], [10], [11], our work takes a bottom-up approach to define a minimum ISA and its architectural structure. Thus, it can not only achieve good PPA when processing masked software but also can suppress power side-channel leakage while maintaining efficiency for other IoT applications.

The remainder of this article is organized as follows. Section II briefly describes the motivation for this work. Section III describes the hardware/software cooperative design of tamper-proof cryptosystems for IoT edge devices. Section IV quantitatively evaluates our work compared to state-of-the-art works. Finally, this article is concluded in Section V.

II. Motivation

In the IoT era, with an increasing need to process confidential data (e.g., personal identifiable data) in real time, lightweight symmetric ciphers have been tailored to resource-constrained embedded systems. They essentially comprise of only a few basic instructions that are supported by even the ISA of low-end processors. In addition, they are parameterized (e.g., key length and rounds) to address emerging threats.

Cryptosystems built on top of a microprocessor are preferable in terms of flexibility to the update of parameters and/or cipher algorithms compared to hardwired cipher circuits. To develop a microprocessor that can be employed on resource-constrained devices, a bottom-up approach in defining an ISA and its architectural structure should be considered. For the first step, here we examine the following symmetric ciphers that adopt different designs for permutation or substitution. These ciphers are used as benchmarks in our evaluation. These ciphers are selected because of their different operation used during computation. In our future work, more lightweight ciphers will be covered, e.g., Speck, LED, and Ascon.

1) *Chaskey* [16] is an addition–rotation–XOR (ARX)-based message authentication code (MAC) algorithm designed for 32-bit embedded processors. ARX ciphers are composed of only three types of operations (i.e., modular addition,[1] rotation, and bitwise XOR). The basic computation block is defined as the permutation π shown in Fig. 1(a). As a variant of Chaskey, Chaskey-12 [17], which contains 12 rounds of permutation, was standardized by the International Organization for Standardization as a lightweight MAC algorithm.
2) *Simon* [18] is a lightweight block cipher released by the U.S. National Security Agency. A single round of the Simon permutation is illustrated in Fig. 1(b). Simon is another type of ARX cipher. However, unlike Chaskey, addition is replaced by bitwise AND. In addition, Simon supports different block size, key length, and rounds combinations. In this work, Simon64, which adopts the smallest processing unit of 32-bit with a 128-bit key length, was used for our implementation and evaluation.
3) *AES* [19] is a block cipher standardized by the American National Institute of Standards and Technology. AES supports different settings in terms of key length, which defines the number of rounds. The nonlinear substitution step (i.e., SubBytes or S-box) is shown in Fig. 2. Through the S-box, the input values a on the left side are transformed into the output values b on the right side. Technically, although AES is not considered as a lightweight cipher, we utilize AES because it is a representative symmetric cipher that employs the S-box and has been widely evaluated in the literature. In our evaluation, similar to most existing works on side-channel attacks and countermeasures, we focus on the S-box, which is the most critical component in AES.

Table I shows the details of the cipher benchmarks in our evaluation. These ciphers support different block/key lengths, here we selected those variants to have the same

[1]Hereafter, we refer to modular addition as "addition" for brevity.

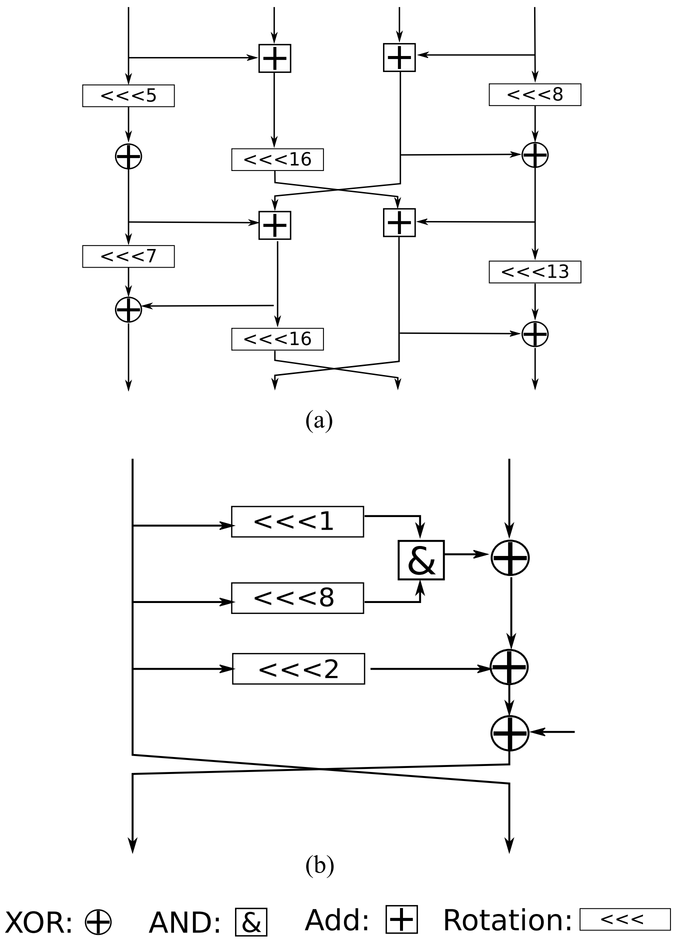
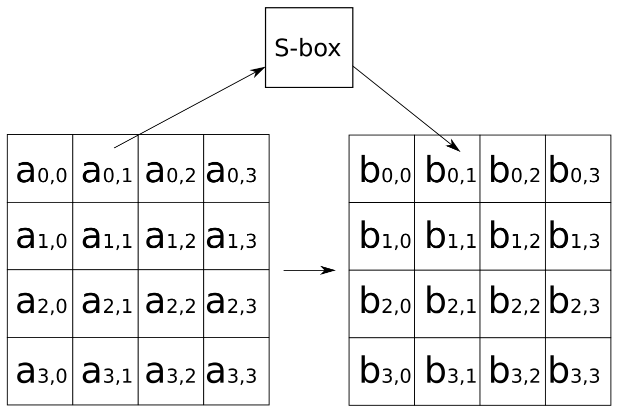


Fig. 1. One round of permutation. (a) Chaskey. (b) Simon.



S-box
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Fig. 2. AES S-box.

TABLE I
DETAILS OF THE CIPHERS

Cipher	Block Size	Key Size	Rounds	Masking
Chaskey	128	128	12	Boolean/Arithmetic
Simon	64	128	44	Boolean
AES	128	128	10	Boolean

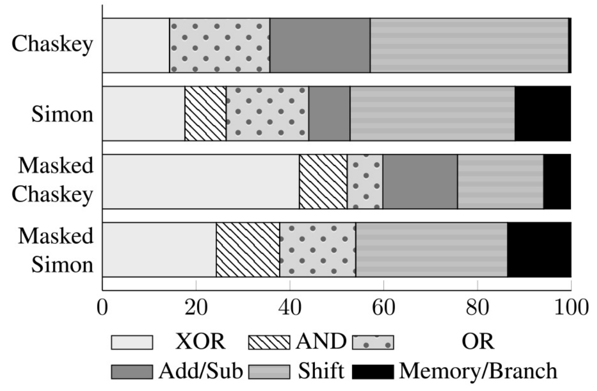


Fig. 3. Breakdown of operations in unmasked and masked software implementations of representative lightweight ciphers.

key length. The above cipher algorithms are mathematically proven to be secure. However, a side-channel that emits physical information, such as power consumption, can leak the secret information. To prevent side-channel leakage, a masking countermeasure is applied. This masking countermeasure divides secret information into multiple *shares* according to an XOR-ing scheme. Thus, in these cipher algorithms, the original (i.e., the unmasked) implementation and the masked implementation do not require complex operations. Specifically, as summarized in Fig. 3, only several types of simple instructions are used in both the masked and unmasked implementations of Chaskey and Simon (when compiled for the RISC-V RV32IMC ISA). The AES S-box uses fewer types of operations by utilizing the table-based S-box (i.e., primarily load/store operations). In contrast, most existing processors support many more types of instructions (e.g., 47 to 190 even in small-scale embedded processors according to a survey reported in [13]), which means that the most of them are unused to process lightweight ciphers. Even worse, in a previous study [20], even unused hardware resources [e.g., floating-point arithmetic unit (FPU)] were identified to become a side-channel leakage source due to glitch propagation.

Generally, the circuit area of microprocessors increases with the complexity of the ISA and functionalities. Thus, protecting such processors against side-channel attacks primarily by introducing a duplicated datapath circuit or cipher-dedicated co-processor [8], [9], [10], [11] makes it more difficult to develop a cryptosystem that is suitable for constrained devices.

Focusing on this bottleneck in existing works, we take a bottom-up approach in a hardware/software cooperative manner. In this work, we define a minimum ISA and its architectural structure that can process a masked software implementation of lightweight symmetric ciphers efficiently. By handling power side-channel attacks cooperatively in terms of both hardware and software, PPA can be improved while providing side-channel protection.

III. Hardware/Software Cooperative Approach Against Power Side-Channel Attacks

As discussed in [7], there is a gap between proven security in theory and safe implementation of cryptosystems in practice. This work addresses this gap on constrained IoT devices while considering the PPA of the cryptosystems. Specifically, by adopting the hardware/software cooperative approach, which combines a security-oriented hardware design

TABLE II
ISA DEFINITION FOR OUR PROCESSOR (R_A AND R_B: THE INPUT REGISTERS, I: AN IMMEDIATE VALUE, AND R_D: THE OUTPUT REGISTER. "/" INDICATES THAT EITHER OF TWO OPTIONS IS SELECTED, AND "<=>" INDICATES THAT ASSIGNMENT IS IN EITHER DIRECTION)

Instruction	Operation	Processing
SUB R_A/I, R_B, R_D	Subtraction	R_D=R_B - R_A/I (*)
LOGIC R_A/I, R_B, R_D	Logic	R_D=R_B $\wedge/\oplus$ R_A/I
MEM I, R_B, R_D	Memory	R_D <=> M[R_B-I]
SHIFT R_A, R_B/I, R_D	Shift	R_D = R_A <</>> R_B/I
(* An optional branch is used based on the computed result.)		

and a hardware-aware software implementation, this work attempts to develop a cryptosystem that is resistant to power side-channel attacks and suppresses PPA overheads. Note that this work assumes first-order attacks as well as the work [15] and apply two-share masking as a starting point.[2] The hardware design takes a bottom-up approach to define the ISA and architecture of a microprocessor to develop tamper-proof cryptosystems. The software implementation is carefully realized to avoid unintended leakage through hardware resource sharing such that only two-share making is sufficient to provide protection against first-order attacks.

In the following, we elaborate on the proposed approach relative to hardware and software perspectives.

A. Security-Oriented Hardware Design

To process masked lightweight ciphers while also providing a desired level of security under limited computing resources, we developed a uniquely defined ISA that supports a minimum set of instructions and a simple architecture according to the preliminary analysis shown in Fig. 3. In this work, we referred to a low-power 32-bit embedded processor SubRISC+ [12], [13] which was recently proposed for lightweight embedded applications (e.g., eHealth monitoring) as a baseline design of our microprocessor due to its simple ISA and small circuit footprint. SubRISC+ supports four types of instructions (i.e., subtraction, bitwise AND, shift with only predefined values, and memory access). In this work, we implemented support for subtraction, bitwise AND/XOR, shift with arbitrary values, and memory accesses such that different types of masked ciphers can be processed efficiently while suppressing both circuit area and power overheads. The ISA definition and instruction format of our processor are summarized in Table II and Fig. 4, respectively. As shown in Fig. 4, instructions are basically in a 16-bit format, and an optional 16-bit block is utilized to handle an immediate value or branch. Here, "Function" is a 1-bit flag signal that specifies different options in each instruction. It is used to determine whether the optional block will be used in `SUB`, to select between bitwise AND/XOR in `LOGIC`, to specify the assignment direction between the register file and memory in `MEM`, and to select the shift direction in `SHIFT`. Operations that are not directly supported by this ISA can be computed by combining the instructions listed in Table II. Interested readers are referred to [12] and [13] for additional details about the

[2]Resistance against higher order attacks will be explored in future work.

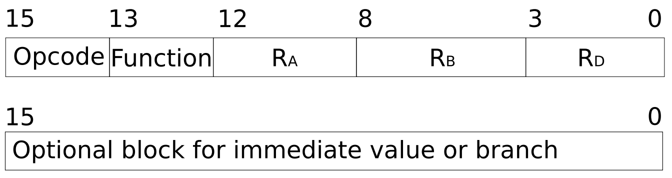


Fig. 4. Instruction format for our processor (short forms refer to Table II).

SubRISC+ instruction format and architecture. Unless stated otherwise, we still follow the original format definition of the SubRISC+ ISA.

To develop our security-oriented processor, based on our decision on the ISA and microarchitectural source of leakage [20], the following three extensions are applied to the baseline SubRISC+ architecture. An architectural overview of the proposed processor design is shown in Fig. 5.

1) As mentioned previously, we introduce *bitwise* XOR and *shift with arbitrary values*. Accordingly, the ALU implementation and decoder as well as the instruction format were extended to support these two instructions.
2) We employ a gating scheme that disables unnecessary switching on inactive operations in the ALU module. As pointed out in [20], even unused resources (e.g., FPU) can play an important role in leakage due to glitch powers caused by unnecessary switching. Considering this issue, conventional gating schemes to reduce power consumption can be an effective solution. This observation motivated us to cut off undesired switching activities in the ALU module, which is one of the busiest modules in terms of glitch generation. To implement the gated ALU, opcodes are used to determine the current instruction to be executed such that only the parts required for processing its operation are enabled (i.e., low-power input gating [21]).
3) Finally, as fresh randomness is required for masking (Section III-B), we introduce a pseudo-random number generator (PRNG) module that generates 32-bit random numbers. Here, the random values are generated on-the-fly and stored from the PRNG module to the register file. In this work, we used an XORSHIFT-ADD-based PRNG [22], which is a variant of Xorshift PRNG. In this work, an assumption on attack scenario that the attackers can not predict the output of the PRNG module is made. Since other types of PRNGs can be applied, we will explore a more suitable PRNG for our processor in the future work.

B. Software Implementation: Masking

Masking is an effective countermeasure to protect secret information in ciphers from side-channel attacks. Secret data are divided into multiple shares on which the computation for encryption is performed with fresh randomness to prevent from analysis of the physical characteristics. Because permutation or an S-box is critical for encryption, they must be masked using masking schemes that correspond to their operations. For example, Boolean masking is applied to operations where no carry occurs during computation (e.g., logic operations,

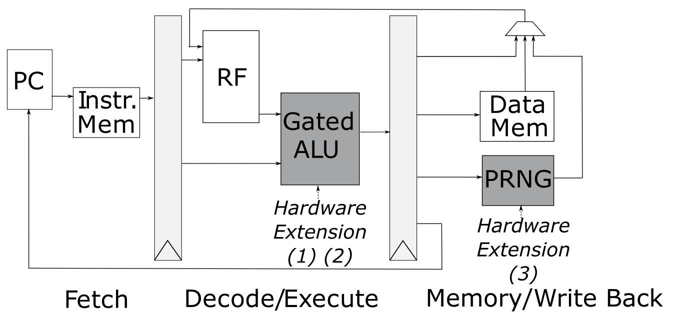


Fig. 5. Architectural overview of our proposed design (PC: program counter, ALU: arithmetic logic unit, RF: register file, and PRNG: pseudo-random number generator).

Algorithm 1 B2A Conversion

Input: (X', r), such that $X' = X \oplus r$, random value R
Output: A, such that $x = A+r$

```
1: $T <= X' \oplus R$ // $T$ is an intermediate value
2: $T <= T - R$
3: $T <= T \oplus X'$
4: $R <= R \oplus r$
5: $A <= X' \oplus R$
6: $A <= A - R$
7: $A <= A \oplus T$
```

rotation, and table references). On the other hand, operations that may involve carries between bits depending on the value (e.g., addition) must be masked using the arithmetic masking.

In this work, we employ a conventional masking countermeasure[3] [23] for software implementations of Chaskey, Simon, and AES, which were selected for their different properties on masking. As described in the following, Chaskey involves both arithmetic masking and Boolean masking on the permutation, Simon involves only Boolean masking on the permutation, and the AES S-box involves only Boolean masking on the table-based substitution.

Masked Chaskey Implementation: As discussed in Section II, Chaskey is an ARX cipher where both arithmetic and logic operations are used in the permutation [Fig. 1(a)]. Thus, Boolean masking is applied to both XOR and rotation, and arithmetic masking is applied to addition. When mixing these masking schemes, the correctness of the final results under masking must be ensured carefully. To realize this, we implement the Goubin conversion [24] between these schemes in two directions [i.e., from Boolean to arithmetic (B2A) and from arithmetic to Boolean (A2B)]. The pseudocode for these schemes is described in Algorithms 1 and 2, respectively.

While the B2A algorithm is relatively lightweight in taking only several operations, the A2B algorithm involves a loop that is iterated $K-1$ times where K is the number of bits of the input ($K = 32$ in our evaluation). Obviously, this loop becomes a bottleneck. Thus, to mask Chaskey while suppressing the latency overhead, the A2B and B2A conversions must be inserted into the appropriate positions along with both masking

[3]Other masking countermeasures (e.g., threshold implementation and domain-oriented masking) can be similarly applied.

Algorithm 2 A2B Conversion

Input: (A, r), such that $X = A + r$, random value R
Output: X', such that $X = X' \oplus r$

```
1: $T <= 2R$ // $T$ is an intermediate value
2: $X' <= R \oplus r$
3: $O <= R \wedge X'$ // $O$ is an intermediate value
4: $X' <= T \oplus A$
5: $R <= R \oplus X'$
6: $R <= R \wedge r$
7: $O <= O \oplus R$
8: $R <= T \wedge A$
9: $O <= O \oplus R$
10: for $k = 1 \ldots K-1$ do
11:     $R <= T \wedge r$
12:     $R <= R \oplus O$
13:     $T <= T \wedge A$
14:     $R <= R \oplus T$
15:     $T <= 2R$
16: end for
17: $X' <= X' \oplus T$
```

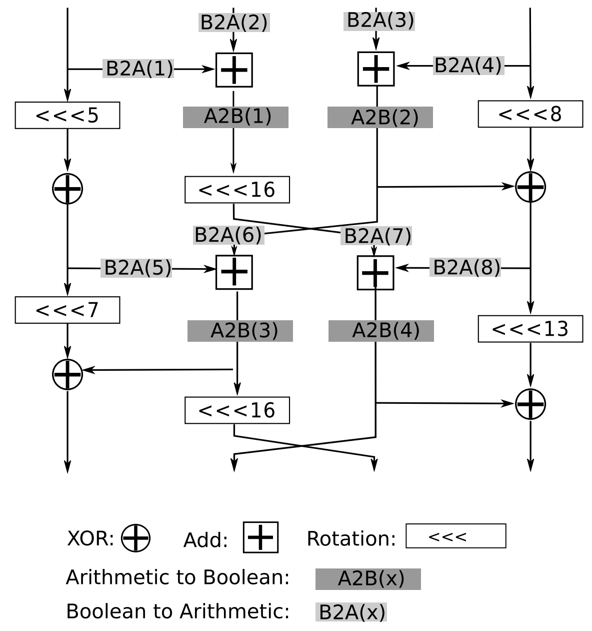


Fig. 6. Masked Chaskey with A2B and B2A conversions.

schemes. Fig. 6 shows our implementation. The number specified in each A2B or B2A conversion indicates an index for simple comprehension. In this implementation, conversions are inserted immediately before and after the addition operations. To minimize the latency overhead caused by these conversions, “B2A(2)” and “B2A(6)” can be omitted if the precedent results are passed directly to the additions. However, in this case, both the converted and unconverted results are required, thereby resulting in register file spilling.

A recent work [25] pointed out a potential leakage source in the above A2B conversion algorithm (i.e., when the intermediate variable R is overwritten as appeared on lines 6 and 8 of Algorithm 2). We examined how the intermediate variable R is used and found that R does not have to be written to the same variable (on line 8 of Algorithm 2). Consequently, we resolve this potential leakage point by assigning R to another variable.

Masked Simon Implementation: The Simon permutation does not involve operations with carries. Thus, only Boolean masking is used (i.e., no conversion is required, unlike Chaskey). An existing masking method [26] for the Simon permutation is explained as follows:

$$\mathrm{rout}[a] = l[a] \tag{1}$$
$$\mathrm{rout}[b] = l[b] \tag{2}$$
$$\begin{aligned}\mathrm{lout}[a] = {} & r[a] \oplus l[a]^2 \oplus \left(l[a]^1 \wedge l[a]^8\right)\\ & \oplus\left(l[a]^1 \wedge l[b]^8\right) \oplus k[a]\end{aligned} \tag{3}$$
$$\begin{aligned}\mathrm{lout}[b] = {} & r[b] \oplus l[b]^2 \oplus \left(l[b]^1 \wedge l[b]^8\right)\\ & \oplus\left(l[b]^1 \wedge l[a]^8\right) \oplus k[b]\end{aligned} \tag{4}$$

where r and l are the inputs on the right and left sides of the round function, *rout* and *lout* are the outputs on the right and left sides, and k is a round key. Two shares of each input or key are represented by a and b. The exponent operations mean left rotation with the specified number of bits (e.g., $l[a]^2$ means left rotation by 2 bits). In this masking method, the key point is to handle the AND operation as follows:

$$l^1 \wedge l^8 => \left(l[a]^1 \oplus l[b]^1\right) \wedge \left(l[a]^8 \oplus l[b]^8\right).$$

By extending the right-hand side of this equation, the following format is obtained:

$$\left(l[a]^1 \wedge l[a]^8\right) \oplus \left(l[a]^1 \wedge l[b]^8\right) \oplus \left(l[b]^1 \wedge l[b]^8\right) \oplus \left(l[b]^1 \wedge l[a]^8\right).$$

This equation is then divided into two shares in (3) and (4) to ensure correctness under the masking scheme. In addition, before each round, the round key k is preprocessed to be divided into $k[a]$ and $k[b]$ such that $k = k[a] \oplus k[b]$.

Masked AES Implementation: According to convention, we implemented the AES S-box in a table look-up manner. Then, for masking, we followed a principle in [27] to apply Boolean masking to the table S-box and indexing of the S-box. This process ensures that the original values are not processed during encryption and masked values are processed alternatively with a masked index.

C. Software Implementation: Resource Allocation and Instruction Scheduling

We then carefully consider hardware resource allocation and scheduling of instructions for the masked software implementations. The aforementioned masking implementations using $d + 1$ shares and conversions between different masking schemes should theoretically protect secret data from dth-order side-channel attacks. However, the protection effect may be subject to implementation of the underlying embedded processors. As discussed in [6], 1) transitions between two shares of the same secret data at the same memory location or entry may lead to secret leakage. In addition, 2) shares of the same secret data should not be accessed within two successive instructions [15]. In other words, failing to satisfy these two constraints can lead to disclosure of Hamming distance between two shares.

To satisfy these constraints and avoid unintentional resource sharing of the same secret data, we carefully implemented the software to be aware of our microprocessor (Section III-A) in terms of the register file and memory allocation [for constraint 1)] and instruction scheduling [for constraint 2)].

First, we addressed constraint 1) using dedicated register file entries or memory locations for each share to explicitly avoid transitions between shares. Note that this approach is potentially register hungry, particularly for embedded processors with limited register file entries (e.g., 16 or fewer). Thus, if the number of dedicated register file entries is insufficient for all intermediate variables, dedicated memory locations are also allocated. When implementing masked software, constraint 1) can be satisfied by specifying these register/memory allocations in the assembly code explicitly or via proper usage of global variables in the C code.

Next, instruction scheduling was adjusted to satisfy constraint 2), which allows us to schedule instructions on values that do not share the same secret data in a row. In addition, swapping the operands of commutative operations also helps realize efficient scheduling. Note that these adjustments can be certainly done in the assembly level. However, if the C code implementation is used, designers must ensure that the instruction scheduling in the compiled code is as intended.

In summary, these combined hardware-aware optimizations follow the principle that unintentional resource sharing of the same secret data during encryption is avoided [6].

IV. Evaluation

In this section, we first describe our experimental setup. Then, we demonstrate the effectiveness of our work against state-of-the-art works in terms of PPA and security.

A. Experimental Setup

In this evaluation, we used the SAKURA-X board [28], which was designed for side-channel evaluation through special power measurement connectors. This board comprises two field-programmable gate arrays (FPGAs): 1) a Xilinx Kintex-7 XC7K160T FPGA and 2) a Xilinx Spartan-6 XC6SLX45 FPGA. The Kintex-7 FPGA is used to implement the cryptosystem, and the Xilinx Spartan-6 XC6SLX45 FPGA is used to implement a controller for the Kintex-7 FPGA (e.g., clock generator). For the Kintex-7 FPGA, the clock frequency was set to 12 MHz.

For the masked implementations of Chaskey, Simon, and AES S-box, the following processors were implemented on the Kintex-7 FPGA to evaluate the effects of our method comprehensively.

1) *Ibex*[4]*:* The Ibex is one of the smallest RISC-V cores whose ISA is defined as RV32IMC. The number of register file entries is 32.
2) *SubRISC+:* The SubRISC+ is one of the smallest microprocessors based on which our processor was newly

[4]github.com/lowRISC/ibex

TABLE III
COMPARISON OF PPA (MASKED SOFTWARE IMPLEMENTATIONS)

	Performance (cycles)			Time (us)			Dynamic power (mW)			Energy (nJ)			Area		
	Chaskey	Simon	AES S-box	Chaskey	Simon	AES S-box	Chaskey	Simon	AES S-box	Chaskey	Simon	AES S-box	LUTs	FFs	DSPs
Ibex	1,060	46	40	88.33	3.83	3.33	15.22	9.46	10.67	1344.38	36.23	35.53	2,612	926	1
SubRISC+ [12]	3,619	126	49	301.58	10.50	4.08	1.11	0.88	0.95	334.75	9.24	3.88	832	626	0
coco-Ibex [15]	1,060	46	40	88.33	3.83	3.33	13.24	8.35	10.28	1169.49	31.98	34.23	3,758	1,946	1
SSP	1,240	47	45	103.33	3.92	3.75	1.21	0.96	1.02	125.03	3.76	3.83	856	628	0

developed. In addition, its ISA was uniquely defined for small-scale IoT devices [12]. The number of register file entries is 16.

3) *coco-Ibex:* coco-Ibex [15] is the most relevant work to our study in that it was developed by partially refining Ibex to be aware of masked software implementations. A verification tool was used to identify side-channel leakage sources in the Ibex netlist. Then, to mitigate secret-dependent glitches, a gating scheme was applied to modules that had leakage (e.g., register file, ALU, and load/store unit). Because we found some critical bugs in its open-sourced RTL code from which the bitstream could not be generated, we reproduced coco-Ibex from the original Ibex code.
4) *Small and Secure Processor (SSP):* This is the proposed processor (Section III-A). In the following, we refer to our processor as the SSP.

These processors were evaluated in terms of implementation results (i.e., the PPA). We used Xilinx Vivado 2020.2 to evaluate the circuit area (resource utilization) and power consumption as well as the FPGA implementation. Performance (in terms of cycle counts per round of the Chaskey/Simon permutation and the AES S-box) was measured using a cycle accurate simulator for each processor. For software compilation and assembling, we developed an in-house toolchain for SSP and used an open-source toolchain for each of the compared processors. Results on execution time are obtained using a clock frequency of 12 MHz. This frequency is used in subsequent simulations and evaluations. In addition, coco-Ibex and SSP were evaluated in terms of security (i.e., the resistance to first-order power side-channel attacks). In this security evaluation, Welch's t-test [29] was conducted to evaluate first-order side-channel leakage on the measured power traces. Here, a Keysight MSOX3104T oscilloscope was used for data acquisition. We conducted a nonspecific leakage assessment, where random and fixed plaintexts were fed to the processors to compute t-values. If the peak of t-value was greater that the threshold of $|4.5|$, the null hypothesis is rejected with a confidence of more than 99.999%. In other words, the power traces for random and fixed plaintexts can be distinguished statistically and thus may leak secret information. To understand the security of our work holistically, the security evaluation was performed at the instruction and cipher levels, where sampling rates of the oscilloscope were set to 5 GSa/s and 313 MSa/s, respectively.

B. Results

PPA Evaluation: The four processors are compared in terms of PPA in Table III. Here, to facilitate a fair comparison, data/instruction memory and the PRNG module were excluded from all processors. From the cycle count results, we see that even though the proposed SSP supports much fewer instructions than Ibex and coco-Ibex, it successfully achieves comparable cycle counts as well as execution time with all ciphers. Considering that SubRISC+ and SSP demonstrate similar ISAs, the security-oriented ISA definition in SSP (particularly the bitwise XOR and shift with arbitrary values) improved the efficiency of processing ciphers. In addition, as discussed in Section II, most types of instructions on Ibex and coco-Ibex were unused even though their ISA is relatively compact among RISC-V ISAs.

Next, we compare the results in terms of power consumption and circuit area (i.e., resource utilization) together. As expected, we found that the Ibex has a larger circuit area because it supports more instructions and functionalities. In addition, owing to the netlist refinements to mitigate side-channel leakage, coco-Ibex incurred nonnegligible area overhead. However, interestingly, coco-Ibex exhibited lower power consumption than the original Ibex due to the gating scheme in coco-Ibex. Compared to Ibex and coco-Ibex, SubRISC+ and the proposed SSP achieved an order of magnitude less power consumption and area due to their simpler microarchitectural structures. Although the proposed SSP demonstrated a slightly higher overhead in both metrics than SubRISC+, the advantage in terms of cycle reduction is much greater.[5] The effect of the cycle count reduction was particularly large for both Chaskey and Simon, where many XOR and rotation operations appear. These operations can be performed efficiently by the ISA of the proposed SSP. Thanks to the reduction on cycle count of the proposed SSP that leads to less execution time, the energy consumption of the proposed SSP outperforms not only the RISC-V processors but also SubRISC+. Overall, we found that the proposed SSP achieved better PPA than the compared state-of-the-art processors for different types of cipher algorithms.

Although in this work we focused on cipher applications and side-channel protection, it is also important to see the whole picture when considering noncipher applications. Evaluations are done to examine the PPA differences between the proposed SSP with a compact ISA and RISC-V Ibex with a generally used embedded ISA. Such that the potential performance losses when executing noncipher applications due to SSP's compact ISA are verified. We evaluated a set of applications that often performed on IoT edge devices for data processing and analysis. The evaluation on noncipher applications that

[5]The ISA of the proposed SSP includes the ISA of SubRISC+ and the proposed SSP has less than 2.9% LUTs overhead. Thus, the proposed SSP can still efficiently process other applications (e.g., eHealth applications [13]).

TABLE IV
EVALUATION ON NONCIPHER APPLICATIONS

	Performance (cycles)		Time (us)		Dynamic Power (mW)		Energy (nJ)	
	Ibex	SSP	Ibex	SSP	Ibex	SSP	Ibex	SSP
Sort	912,889	588,241	76,074.08	49,020.08	10.40	4.09	791,170.43	200,492.13
Motion	85,466	72,006	7,122.17	6,000.50	10.27	4.84	73,144.69	29,042.42
Edge	148,730	212,919	12,394.17	17,743.25	11.36	4.00	140,797.77	70,973.00
Histogram	27,348	30,987	2,279.00	2,582.25	9.78	4.60	22,288.62	11,878.35
DTW	716,318	615,163	59,693.17	51,263.58	9.91	4.11	591,559.31	210,693.31

TABLE V
DETAILS OF THE NONCIPHER APPLICATIONS

Applications	Description
Sort	Sort a set of integers using quick sort algorithm.
Motion	Detect unmatched blocks from two images.
Edge	Apply a Laplacian filter to a image.
Histogram	Construct a histogram of pixels from a image
DTW	Execute the Dynamic Time Wraping (DTW) algorithm.

compares the proposed SSP and RISC-V Ibex is shown in Table IV. The evaluated noncipher applications are briefly introduced in Table V. From the results on performance and execution time, we can observe that SSP and Ibex have comparable cycle counts and execution time. With a much compact ISA design, SSP even outperforms Ibex in several applications. The reason on this is memory and branch instructions that takes multiple cycles on Ibex reduced the relative performance compared to SSP. For Edge and Histogram, when applications have less multiple cycle instructions, Ibex shows better performance than SSP. For the comparisons on power consumption, SSP shows better power efficiency than Ibex due to its architectural design. Thanks to SSP's power efficiency, SSP shows better energy performance than Ibex even in the cases that SSP requires a longer execution time. The results on area are consistent with the results in Table III because there is no change on the circuit design. Overall, these results show the proposed SSP achieved better PPA than the compared state-of-the-art processors in noncipher IoT applications.

Security Evaluation at the Instruction Level: Next, to clearly examine the potential information leakage at a fine granularity, we performed evaluations of instruction-level masking. Comparisons between unmasked and masked instructions on the proposed SSP are shown in Figs. 7 and 8, respectively. Auxiliary horizontal lines are drawn at |4.5| in these figures. The four most critical operations involved in the benchmarks (i.e., XOR, AND, addition, and shift operations) were evaluated by the nonspecific *t*-test using 50 000 traces for random/fixed plaintexts. These operations were masked in software by the two-share masking scheme. Here, NOP instructions were inserted before and after the target instruction(s) to remove interference from other instructions. As shown in the corresponding figures, the region between vertical dotted lines indicates the period to evaluate the target instruction(s). As can be seen, by applying the two-share masking, the proposed SSP can successfully protect secret information against first-order attacks.

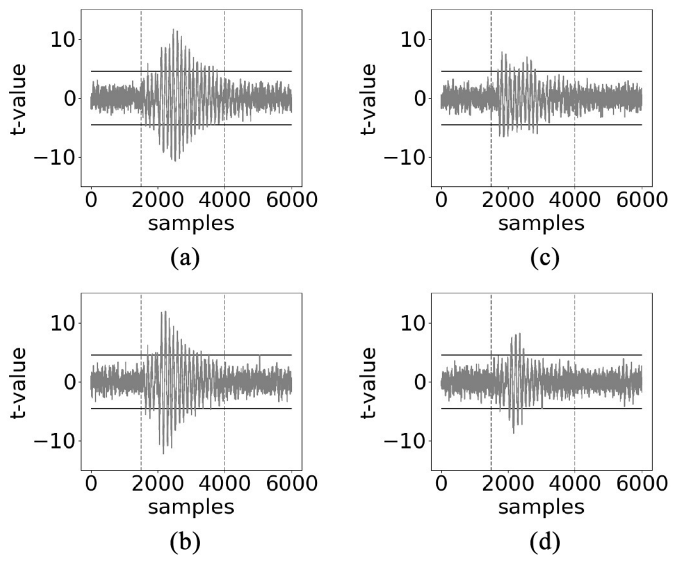


Fig. 7. Unmasked instructions on SSP. (a) XOR. (b) AND. (c) Add. (d) Shift.

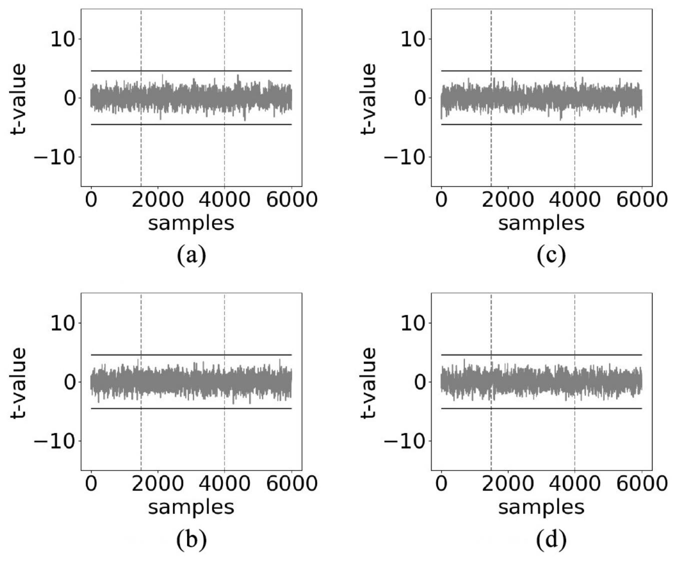


Fig. 8. Masked instructions on SSP. (a) XOR. (b) AND. (c) Add. (d) Shift.

Security Evaluation at the Cipher Level: Finally, security evaluations were performed at the cipher level. Similar to the previous instruction-level evaluation, here, both unmasked and masked ciphers were implemented on the proposed SSP and evaluated using a nonspecific *t*-test (Figs. 9 and 10, respectively). In addition, we evaluated the masked ciphers on coco-Ibex Fig. 11. For the masked ciphers on both coco-Ibex

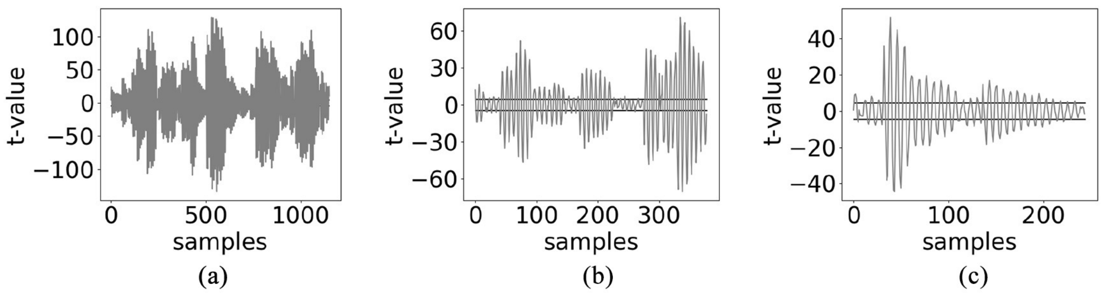


Fig. 9. Unmasked ciphers on SSP (20 000 traces). (a) Chaskey. (b) Simon. (c) AES.

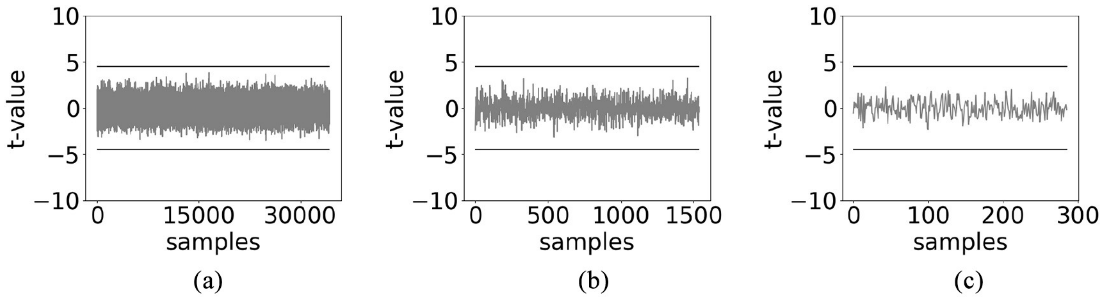


Fig. 10. Masked ciphers on SSP (500 000 traces). (a) Chaskey. (b) Simon. (c) AES.

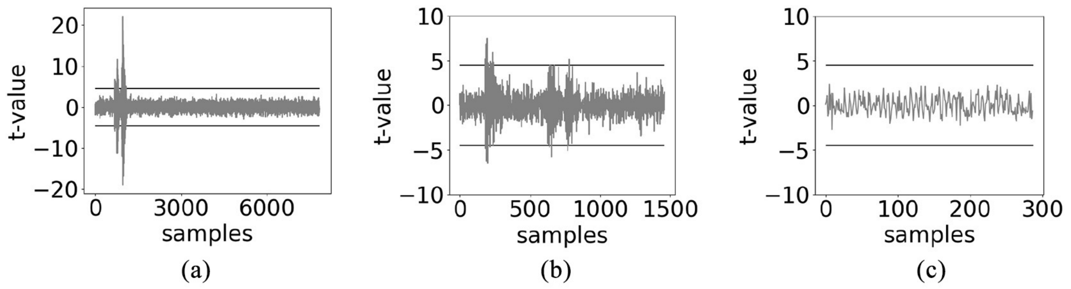


Fig. 11. Masked ciphers on coco-Ibex (100 000 traces). (a) Chaskey.[6] (b) Simon. (c) AES.

and the proposed SSP, we employed the two-share masking and optimizations described in Sections III-B and III-C. We set the largest number of traces for the evaluations of our work (the combination of masked ciphers and SSP; Fig. 10) to facilitate an in depth examination of the effects on security.

As expected from the results of unmasked instructions shown in Fig. 7, we clearly observe that the t-values are greater than |4.5| almost all over the ciphers in Fig. 9. Due to the interference between instructions, secret leakage was more severe than that in the instruction-level evaluations, which indicates the need for a hardware-aware software implementation. According to the results shown in Fig. 11, even though the ciphers were all masked, coco-Ibex suppressed t-values below |4.5| for only the AES S-box, which is consistent with the results reported in the work [15], but failed for both Chaskey and Simon. In the work [15], Gigerl et al. employed an application-specific integrated circuit (ASIC) verification tool to determine the leakage sources, and then evaluated coco-Ibex on an FPGA. However, the exact same behaviors are not guaranteed between ASIC and FPGA. This gap might fail to capture all leakage sources, and coco-Ibex still leaks in Chaskey and Simon. Thus, additional refinements to address with these leakage sources will incur further PPA overhead in coco-Ibex. In contrast, for the masked ciphers, the proposed SSP suppresses the t-values within |4.5| in all ciphers. These results demonstrate that the proposed SSP can eliminate the leakage sources for different ciphers using our security-oriented bottom-up microprocessor design, even without employing a costly datapath duplication [8], [9], [10] or refinements done in the work [15].

In summary, considering both the results on unmasked and masked versions of ciphers as well as the PPA results, the proposed SSP achieves a good tradeoff between PPA and side-channel protection cooperatively from both hardware (security-oriented processor) and software (microarchitecture-aware optimization) perspectives.

[6]The reason for much fewer samples than the samples in Fig. 10(a) is because only a part of the results is presented here to clearly show potential leakage on coco-Ibex. As shown in Table III, the total cycle counts are comparable between coco-Ibex and SSP.

V. Conclusion

In this article, we have proposed a hardware/software cooperative design for cryptosystems that are resistant to power side-channel attacks and suitable for resource-constrained IoT devices. Here, we took a bottom-up approach to define a security-oriented microprocessor and implement masked software to which hardware-aware optimizations were applied. From the cooperative design perspective, minimum ISA and its architectural structure design together with hardware-aware software optimizations are used to prevent from information leakage from masked cipher implementations. In comparative evaluations with state-of-the-art low-power (and security-aware) processors, we demonstrated that our work achieved the most efficient PPA for both cipher and noncipher applications and security against first-order power side-channel attacks for different types of lightweight ciphers.

In the future, further improvements to the PPA of the proposed processor on will be considered. Also, further development on the PRNG module and higher order protection on different ciphers will be included. In addition, we plan to perform an ASIC implementation and conduct further evaluations using the proposed processor.

References

[1] A. Bogdanov et al., “PRESENT: An ultra-lightweight block cipher,” in *Proc. Int. Workshop Cryptogr. Hardw. Embedded Syst.*, 2007, pp. 450–466.

[2] H. D. Tsague and B. Twala, “Practical techniques for securing the Internet of Things (IoT) against side channel attacks,” in *Internet of Things and Big Data Analytics toward Next-Generation Intelligence*. Cham, Switzerland: Springer, 2018, pp. 439–481.

[3] E. Prouff and M. Rivain, “Masking against side-channel attacks: A formal security proof,” in *Proc. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2013, pp. 142–159.

[4] M. Rivain, E. Prouff, and J. Doget, “Higher-order masking and shuffling for software implementations of block ciphers,” in *Proc. Int. Workshop Cryptogr. Hardw. Embedded Syst.*, 2009, pp. 171–188.

[5] J. D. Golic, “Techniques for random masking in hardware,” *IEEE Trans. Circuits Syst. I, Reg.*, vol. 54, no. 2, pp. 291–300, Feb. 2007.

[6] J.-S. Coron, C. Giraud, E. Prouff, S. Renner, and P. K. Vadnala “Conversion of security proofs from one leakage model to another: A new issue,” in *Proc. Constr. Side-Channel Anal. Secure Design*, 2012, pp. 69–81.

[7] L. De Meyer, E. De Mulder, and M. Tunstall, “On the effect of the (micro) architecture on the development of side-channel resistant software,” Cryptology ePrint Arch., IACR, Bellevue, WA, USA, Rep. 2020/1297, 2020.

[8] E. De Mulder, S. Gummalla, and M. Hutter, “Protecting RISC-V against side-channel attacks,” in *Proc. Design Autom. Conf.*, 2019, pp. 1–4.

[9] S. Gao et al., “An instruction set extension to support software-based masking,” *IACR Trans. Cryptogr. Hardw. Embedded Syst.*, vol. 2021, no. 4, pp. 283–325, Aug. 2021.

[10] W. Diehl, A. Abdulgadir, J.-P. Kaps, and K. Gaj, “Side-channel resistant soft core processor for lightweight block ciphers,” in *Proc. Int. Conf. ReConFig. Comput. FPGAs*, 2017, pp. 1–8.

[11] B. Marshall, D. Page, and T. Hung Pham, “A lightweight ISE for ChaCha on RISC-V,” in *Proc. Int. Conf. Appl.-Specific Syst., Archit. Processors*, 2021, pp. 25–32.

[12] K. Saso and Y. Hara-Azumi, “Revisiting simple and energy-efficient embedded processor designs towards the edge computing,” *IEEE Embedded Syst. Lett.*, vol. 12, no. 2, pp. 45–49, Jun. 2020.

[13] M. Yang and Y. Hara-Azumi, “Implementation of lightweight eHealth applications on a low-power embedded processor,” *IEEE Access*, vol. 8, pp. 121724–121732, 2020.

[14] K. Stangherlin and M. Sachdev, “Design and implementation of a secure RISC-V microprocessor,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 11, pp. 1705–1715, Nov. 2022.

[15] B. Gigerl, V. Hadzic, R. Primas, S. Mangard, R. Bloem, “Coco: Co-design and co-verification of masked software implementations on CPUs,” in *Proc. USENIX Secur. Symp.*, 2021, pp. 1469–1468.

[16] N. Mouha, B. Mennink, A. Van Herrewege, D. Watanabe, B. Preneel, and I. Verbauwhede, “Chaskey: An efficient MAC algorithm for 32-bit microcontrollers,” in *Proc. Int. Conf. Select. Areas Cryptogr.*, 2014, pp. 306–323.

[17] N. Mouha, “Chaskey: A MAC algorithm for microcontrollers–status update and proposal of Chaskey-12–,” Ph.D. dissertation, Dept. Electr. Eng., Inria Paris Rocquencourt, Paris, France, 2015.

[18] R. Beaulieu, S. Treatman-Clark, D. Shors, B. Weeks, J. Smith, and L. Wingers, “The SIMON and SPECK lightweight block ciphers,” in *Proc. Design Autom. Conf.*, 2015, pp. 1–6.

[19] M. Dworkin, “Recommendation for block cipher modes of operation. methods and techniques,” U.S. Dept. Commerce., National Inst. Stand. Technol., Gaithersburg, MD, USA, Rep. TR-800-38A , 2001.

[20] V. S. Bokharaie and A. Jahanian, “Power side-channel leakage assessment and locating the exact sources of leakage at the early stages of ASIC design process,” *J. Supercomput.*, vol. 78, pp. 2219–2244, Feb. 2022.

[21] H. Kapadia, L. Benini, and G. De Micheli, “Reducing switching activity on datapath buses with control-signal gating,” *IEEE J. Solid-State Circuits*, vol. 34, no. 3, pp. 405–414, Mar. 1999.

[22] D. Blackman and S. Vigna, “Scrambled linear pseudorandom number generators,” *ACM Trans. Math. Softw. (TOMS)*, vol. 47, no. 4, pp. 1–32, 2021.

[23] J.-S. Coron and L. Goubin, “On Boolean and arithmetic masking against differential power analysis,” in *Proc. Int. Workshop Cryptogr. Hardw. Embedded Syst.*, 2000, pp. 231–237.

[24] L. Goubin, “A sound method for switching between Boolean and arithmetic masking,” in *Proc. Int. Workshop Cryptogr. Hardw. Embedded Syst.*, 2001, pp. 3–15.

[25] B. Gigerl, R. Primas, and S. Mangard, “Formal verification of arithmetic masking in hardware and software,” Cryptology ePrint Arch., IACR, Bellevue, WA, USA, Rep. 2022/849, 2022.

[26] A. Shahverdi, M. Taha, and T. Eisenbarth, “Silent Simon: A threshold implementation under 100 slices,” in *Proc. Int. Symp. Hardw. Orient. Secur. Trust*, 2015, pp. 1–6.

[27] M. Nassar, Y. Souissi, S. Guilley, and J.-L. Danger, “RSM: A small and fast countermeasure for AES, secure against 1st and 2nd-order zero-offset SCAs,” in *Proc. Design, Autom. Test Eur. Conf.*, 2012, pp. 1173–1178.

[28] Y. Hori, T. Katashita, A. Sasaki, and A. Satoh, “SASEBO-GIII: A hardware security evaluation board equipped with a 28-nm FPGA,” in *Proc. Global Conf. Consum. Electron.*, 2012, pp. 657–660.

[29] T. Schneider and A. Moradi, “Leakage assessment methodology,” *J. Cryptogr. Eng.*, vol. 6, pp. 85–99, Feb. 2016.

Mingyu Yang (Graduate Student Member, IEEE) received the B.E. degree in computer engineering from Nanyang Technological University, Singapore, in 2017, and the M.E. degree in information and communications engineering from Tokyo Institute of Technology, Tokyo, Japan, in 2020, where he is currently pursuing the Ph.D. degree with the School of Engineering.

He worked on embedded systems design and development during internship with Panasonic R&D Center, Singapore, from 2016 to 2017. He was also a Research Staff with TUMCREATE, Singapore, and Nanyang Technological University from 2017 to 2018. In 2022, he worked as an Intern with Toyota Central R&D Labs, Nagakute, Japan. His research interests include hardware security and low-power embedded systems.

Mr. Yang is a Student Member of ACM.

Tanvir Ahmed (Member, IEEE) received the M.Sc. degree in electrical engineering from Linköping University, Linköping, Sweden, in 2011, and the Ph.D. degree in information science from Nara Institute of Science and Technology, Ikoma, Japan, in 2014.

He is currently a Researcher with Tokyo Institute of Technology, Tokyo, Japan. Before joining Tokyo Institute of Technology, he held research positions with Fujitsu Laboratories Ltd., Tokyo; Preferred Networks, Inc., Tokyo; and Edgecortix Inc., Tokyo, from 2014 to 2022 to design neuromorphic and machine learning architecture. His research interests are domain-specific accelerator design (e.g., machine learning), reconfigurable computing (both FPGA and CGRA), and machine learning for compiler and EDA optimization.

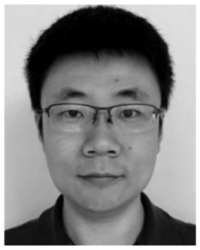
Saya Inagaki received the B.E. degree in information and communications engineering from Tokyo Institute of Technology, Tokyo, Japan, in 2021, where she is currently pursuing the master's degree with the School of Engineering.

She worked on a research internship with RWTH Aachen University, Aachen, Germany, in 2021. Her major research interest is high-level synthesis for hardware security.

Kazuo Sakiyama (Senior Member, IEEE) received the B.E. and M.E. degrees from Osaka University, Suita, Japan, in 1994 and 1996, respectively, the M.S. degree from The University of California at Los Angeles, Los Angeles, CA, USA, in 2003, and the Ph.D. degree in electrical engineering from Katholieke Universiteit Leuven (KU Leuven), Leuven, Belgium, in 2007.

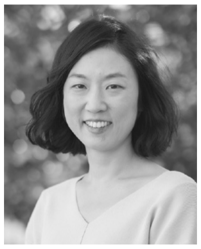
He is currently a Professor with The University of Electro-Communications (UEC), Tokyo, Japan. At UEC, he leads the hardware security research for embedded cryptosystem, cyber–physical system, and physical authentication. Before joining UEC in 2008, he was with Hitachi, Ltd., Tokyo (currently, Renesas Electronics) as a Digital Hardware Designer, and later with KU Leuven as a Ph.D. Research Assistant.

Dr. Sakiyama is a member of IACR and IEICE.



Yang Li (Member, IEEE) received the B.E. degree in electronic and information engineering from Harbin Engineering University, Harbin, China, in 2008, and the M.E. and Ph.D. degrees in information and communication engineering from the University of Electro-Communications, Chofu, Japan, in 2011 and 2012, respectively.

He is currently an Associate Professor with the Department of Informatics, University of Electro-Communications. His main research interests include security evaluation and improvement for cryptographic hardware and IoT devices.



Yuko Hara-Azumi (Member, IEEE) received the Ph.D. degree in information science from Nagoya University, Nagoya, Japan, in 2010.

She was a JSPS Postdoctoral Research Fellow with Ritsumeikan University, Kyoto, Japan, from 2010 to 2012, during which she was also a Visiting Scholar with the University of California at Irvine, Irvine, CA, USA, and Karlsruhe Institute of Technology, Karlsruhe, Germany. In 2012, she joined Nara Institute of Science and Technology, Ikoma, Japan, as an Assistant Professor. Since 2014, she has been with the Department of Information and Communications Engineering, School of Engineering, Tokyo Institute of Technology, Tokyo, Japan, where she is currently an Associate Professor. Her research interests include system-level design automation, especially on high-level and logic synthesis, microprocessor architecture, and hardware/software co-design for embedded/IoT systems.

Dr. Hara-Azumi has served as an Organizing and Program Committee Member for several premier conferences, including DAC, ICCAD, DATE, CASES, ASP-DAC, and FPL. She is a member of ACM.