applied sciences

MDPI

Article

Power-Balancing Software Implementation to Mitigate Side-Channel Attacks without Using Look-Up Tables

HanBit Kim [1], HeeSeok Kim [2] and Seokhie Hong [1,*]

[1] Graduate School of Information Security and Institute of Cyber Security & Privacy (ICSP), Korea University, Seoul 02841, Korea; luzdamoon@korea.ac.kr

[2] Department of Cyber Security, College of Science and Technology, Korea University, 2511 Sejong-Ro, Sejong 30019, Korea; 80khs@korea.ac.kr

* Correspondence: shhong@korea.ac.kr

Received: 17 February 2020; Accepted: 27 March 2020; Published: 2 April 2020

Abstract: With the increasing number of side-channel attacks, countermeasure designers continue to develop various implementations to address such threats. Power-balancing (PB) methods hold the number of 1s and/or transitions (i.e., Hamming weight/distance) of internal processes constant to ensure side-channel safety in an environment in which it is difficult to use random numbers. Most existing studies employed look-up tables (LUTs) to compute those operations, except for XOR and NOT operations. However, LUT-based schemes exhibit some side-channel issues in the address bits of LUTs. In this paper, we propose the application of AND and ADD operations to PB methods based on a rule that encodes 8-bit data into a 32-bit codeword without using LUTs. Unlike previous studies that employed LUTs, our proposals overcome side-channel vulnerabilities associated with the address bits and memory wastage. In addition, we evaluate the side-channel security ensured by the proposed method in comparison with that ensured by other methods. Finally, we apply our methods to SIMON/SPECK ciphers and analyze their performance by comparing them with older schemes.

Keywords: side-channel attacks; countermeasures; ARX block ciphers

1. Introduction

Side-channel attacks are techniques that extract secret values using physical leakages, such as power consumption, electromagnetic radiation, and running time, which occur during the execution of cryptographic algorithms on real devices [1–3]. Power analyses, one of the best-known side-channel attacks, are statistical techniques that extort a secret key based on the fact that total power consumed by a cryptographic module is strongly related to a computed value. To address these threats, countermeasure designers have attempted to break the link between computed data and physical measurements. However, it is very hard to completely break this relationship.

Another type of countermeasure is the use of masking methods to blind secret data by using random numbers, and as a consequence of such a scheme, power measurements can be randomly generated [4]. However, because masking methods unconditionally require random numbers, they are not suited for resource-constrained environments in which it may be difficult to employ a cryptographic pseudorandom number generator (PRNG) [5]. For instance, a dth-order side-channel attack that exploits the leakages related to d intermediate variables at the same time are theoretically countered by a dth-order masking method that d random values are involved per sensitive variable [6].

Power-balancing (PB) methods may ensure side-channel safety in an environment in which it is difficult to use PRNGs [7]. These methods encode data and construct operations to ensure

that the number of 1s and/or transitions of bits (i.e., Hamming weight/distance (HW/HD)) is constant, and therefore power leakages can be similarly generated regardless of the secret values [8]. Because the power consumption of a crypto-module applied to PB methods contains small amounts of information related to the computed data, these schemes do not ensure perfect security against power analyses. However, in environments where random numbers cannot be used, these methods exhibit advantages that reduce the impact of power analyses. Moreover, these methods can be combined with other countermeasures such as shuffling and dummy operation schemes to achieve a higher level of security [9].

The fundamental idea of PB methods was derived from dual-rail with precharge logic (DPL) styles in hardware [8]. In DPL styles, a logic bit a is represented by a 2-bit pair $(a, \overline{a})$, where $\overline{a}$ is the complement bit of a. Hoogvorst et al. was the first to attempt to create DPL styles of hardware into software [7]. Since then, several studies, such as constant XOR and look-up table-based (LUT-based) operations, have applied these schemes to symmetric ciphers [10–18]. Here, constant operators, such as constant XOR, are modified into operations that always hold the same HW/HD in internal process by using only assembly language. The most recent attempt is the study by Pour et al., which proposed constant ANDs by using only a small number of LUTs and applied these operations to the SIMON block cipher [19]. However, most studies that use LUTs exhibit vulnerabilities to power analyses targeted at address bits. Moreover, these implementations are inefficient in terms of memory usage [18]. In addition, both the data of LUTs and address bits should have the same HW as 0101, 0110, and 1001, whereas others can be unavailable.

In this paper, we first propose constant AND and ADD operations without using LUTs. Our operations are based on a rule that encodes 8-bit data into a 32-bit codeword. The 32-bit codewords are optimized for 32-bit microcontrollers such as ARM and AVR32, and our suggestions are applicable to ARX ciphers in environments where PRNG is not available. Moreover, we evaluate the side-channel security and analyze the performances of the proposed countermeasures. The main contributions of this paper are as follows.

1.1. New Constant Operations

We propose a new AND operation that can be applied to the PB method without using LUTs. This constant AND exhibits advantages in terms of performance and side-channel security compared with previous works using LUTs. In terms of performance, the proposed scheme requires the same running time regardless of the bit size of the computed data. Because previous LUT-based algorithms were designed on a divide and conquer strategy, they result in an increase in computing time with increase in the bit size of inputs. On the other hand, the proposed method computes a whole word at once; therefore, it takes the same amount of time regardless of the bit size of inputs. Moreover, our operation can reduce memory usage requirements. In terms of side-channel security, the proposed method is resilient against side-channel attacks that target leakages from the address bits of LUTs. Side-channel leakages in the process of loading LUTs involve not only information about the input and output values of LUTs, but also information about the address bits of memory cells in which LUTs are stored. Because our algorithm does not use LUTs themselves, it is immune to threats originating from address bits.

Next, we propose an addition operation applied to the PB method using the proposed method and existing constant operations. In order to apply PB methods to an addition operation, we optimized an SW-based kogge-stone adder for the multi-word environment. The algorithm operates very efficiently in a constrained environment where the bit length of codewords is too long to be handled all at once.

1.2. Security Evaluation

We evaluate the side-channel security ensured by the proposed method with those ensured by other methods. To measure the amount of information leakages in side-channel signals, we acquired power traces of the proposed, existing, and unprotected cases. It is found that the signal-to-noise

ratios (SNRs) and peaks of correlation power analysis (CPA) are reduced by up to approximately 2697% and 318% compared with the weak implementation against side-channel attacks, respectively. Through experiments, we determined the side-channel vulnerabilities of the address bits of LUTs. By not applying PB methods to address bits, information leakages that are similar to the unprotected case occur.

1.3. Performance Analysis

We analyze the performances of the proposed schemes in comparison with those of the other schemes. We verify the number of operators in the proposed schemes as well as those in the existing studies, and we compare the number of clocks in an ARM simulator provided by IAR Embedded Workbench. We apply the proposed algorithm to SIMON and SPECK block ciphers and analyze their performances in comparison with existing studies. As a result, it is found that the SIMON cipher using constant ANDs and the SPECK cipher using constant ADDs exhibit performance improvements of approximately 33% and 37%, respectively.

The remainder of this paper is organized as follows. In Section 2, we introduce PB methods by employing dual-rail-based encoding schemes. Section 3 is the core of the paper, in which we present new dual-rail-based PB methods without the application of LUTs. In Section 4, we evaluate the security against side-channel attacks. In Section 5, we present the performance metrics of our proposed method as well as those of existing methods. Finally, we present the conclusions in Section 6.

2. Existing Works on Power-Balancing Countermeasures by Employing Dual-Rail-Based Encoding Schemes

PB methods are associated with hiding countermeasures that apply DPL styles of hardware to software. In hardware, DPL operates in pairs of data bits and complement bits. This circuit consumes the same amount of power, irrespective of the amount of data. Because hardware can be flexibly employed in circuit design, it is easy to implement operations of the DPL styles. On the other hand, software operated on 8/16/32-bit microcontrollers exhibit difficulties when DPL styles are applied. PB methods define operations that satisfy constant HW/HD using operators supported by microcontrollers. These methods are divided into (x, y)-code-based schemes and dual-rail-based schemes according to an encoding rule of constant HW/HD.

(x, y)-code-based schemes encode data using a codeword in which the HW is denoted by x and the size of the bits is denoted by y. For instance, if we encode 4-bit nibble data into a $(3, 6)$-code, we only use ${}_6C_3$ codewords, which are of 6-bit sizes with HW$=3$. In 2014, Servant el al. proposed constant HW codes using LUTs [14]. Rauzy et al. developed a tool to apply (x, y)-code-based countermeasures to arbitrary cryptographic algorithms [15]. Maghrebi et al. proposed a methodology for selecting optimal codewords on (x, y)-code [16]. This method was improved by Bhasin et al. [17]. Petrvalsky et al. applied the (x, y)-code-based countermeasure to AES and embedded it on an ARM Cortex-M3 [18]. One advantage of (x, y)-code-based schemes is that memories may be constructed compactly, but these schemes can only define operations using LUTs.

Dual-rail-based schemes operate on codewords that double or quadruple the size of the data. For instance, data $a_1a_0 (a_i \in F_2)$ is encoded into a codeword $\overline{a_1}a_1\overline{a_0}a_0$. In 2011, Hoogvorst et al. first tried to bring dual-rail-based schemes of hardware into software [7]. Han et al. proposed a constant XOR operation that does not require LUTs based on their encoding rule [10]. Chen et al. applied Han's encoding rule with constant HW/HD to the PRINCE block cipher [11]. Won et al. conducted further experiments with Chen's method [12,13]. The most recent balancing countermeasure based on a Han-like encoding rule is the method proposed by Pour et al. [19]. They first proposed a constant AND with only a small number of LUTs and applied these operations to the SIMON block cipher. One disadvantage of dual-rail-based schemes is that they have large codewords, but it is easy to define the operations because the encoding rules exhibit regularity.

In this section, we describe a 1-to-4 encoding rule, which encodes 8-bit data into a 32-bit codeword based on the method proposed by Pour et al. We denote constant XOR, NOT, and AND operations based on 1-to-4 encoding [19]. At the end of this section, we discuss the drawbacks of existing research and how to overcome it.

2.1. Balanced Encoding Rule

There are many ways of presenting 1-to-4 encoding. In this paper, we define constant operations based on three types of encoding rules as shown in Table 1. $\overline{x}$ is a complement value of x.

Table 1. Three types of proposed encoding rules.

Data Bit	$X_{\in E_1}(= E_1(x))$	$X_{\in E_2}(= E_2(x))$	$X_{\in E_3}(= E_3(x))$
x	$\overline{x}x\overline{x}x$	$\overline{xx}xx$	$x\overline{xx}x$
0	1010	1100	0110
1	0101	0011	1001

We denote the intermediate values of the algorithms as $x_{\in E_1}$, $X_{\in E_1}$, or $Y_{\in E_2}$. This means that the registers x and X are encoded in type 1, and the register Y is encoded in type 2. Some notations with no information of the encoding type indicate that the codeword satisfies constant HW/HD. However, the encoding uses a different format compared to the three defined encodings.

This encoding rule exhibits the following properties for any bit x, y.

- $\mathrm{HW}(E_1(x)) = \mathrm{HW}(E_2(x)) = \mathrm{HW}(E_3(x)) = 2$
- $\mathrm{HD}(E_1(x), E_2(y)) = \mathrm{HD}(E_1(x), E_3(y)) = \mathrm{HD}(E_2(x), E_3(y)) = 2$
- $E_1(x) = E_2(x) \oplus (\texttt{0b 0110}) = E_3(x) \oplus (\texttt{0b 1100})$

These properties are the key principles for designing constant operations. The first property is that all the HWs of codewords are twice the bit size of data. Second, the HDs between different types of codewords are twice the bit size of data. In other words, an XOR result of different types of codewords leaks constant power. Finally, the conversion processes of the encoding types can be simply computed by XORing a constant.

2.2. Constant XOR/NOT Operation

As can be observed intuitively from the properties described above, we can achieve constant XOR/NOT by converting the encoding type appropriately.

Constant XORs are summarized as follows.

$$E_1(x) \oplus E_2(y) = E_3(x \oplus y)$$
$$E_1(x) \oplus E_3(y) = E_2(x \oplus y)$$
$$E_2(x) \oplus E_3(y) = E_1(x \oplus y)$$

When different types of codewords are XORed, the result is denoted as another type of codeword. This result can be converted into other encoding types by XORing the constant described in the encoding rule.

The principles of these operations state that because a true pair's XOR result is true, a false pair's XOR result is true, and a cross pair's XOR result is false. In the encoding rule, the codeword bits of different encoding types are arranged in two cross pairs and two true/false pairs.

Similarly, constant NOTs can be easily defined. Constant NOTs include the following.

$$E_1(\overline{x}) = \overline{E_1(x)} = E_1(x) \oplus (\texttt{0b 1111})$$

$$E_2(\overline{x}) = \overline{E_2(x)} = E_2(x) \oplus (\texttt{0b 1111})$$
$$E_3(\overline{x}) = \overline{E_3(x)} = E_3(x) \oplus (\texttt{0b 1111})$$

These processes also satisfy constant HW/HD.

2.3. Constant AND Operation

Pour et al. proposed two methods of constant AND that use only small LUTs. Algorithm 1 and Table 2 present algorithms for constant AND and LUTs, respectively.

Algorithm 1: Pour's Constant AND Operation

Require: $x_{\in E_1}, y_{\in E_2}$
Ensure: $z_{\in E_3} (= E_3(x \wedge y))$
1: $T \leftarrow 0$
2: $x \leftarrow x \wedge (\texttt{0b 1001...})$
3: $y \leftarrow y \wedge (\texttt{0b 0110...})$
4: $T \leftarrow x \oplus y$
▶ Method-1:
5: **return** $LUT_1(T)$
▶ Method-2:
6: **return** $LUT_2(T) \oplus LUT_3(T)$

Table 2. Entries of LUT_1, LUT_2, and LUT_3.

$x_{\in E_1}$	$y_{\in E_2}$	T	$LUT_2(T)$	$LUT_3(T)$	$LUT_2(T) \oplus LUT_3(T) = LUT_1(T) = E_3(x \wedge y)$
[1010]	[1100]	[1100]	[1101]	[1011]	[0110]
[1010]	[0011]	[1010]	[1011]	[1101]	[0110]
[0101]	[1100]	[0101]	[1101]	[1011]	[0110]
[0101]	[0011]	[0011]	[0111]	[1110]	[1001]

The algorithm and the LUTs satisfy a constant HW. $\mathrm{HD}(T, LUT_1(T))$ of method-1 is constant at 2, and $\mathrm{HD}(T, LUT_2(T))$ and $\mathrm{HD}(T, LUT_3(T))$ of method-2 are constant at 1. In method-2, the HD of the output $LUT_2(T) \oplus LUT_3(T)$ is also constant.

However, as mentioned in the paper by Pour et al., even if few LUTs are employed, the address bits of LUTs leak side-channel information. The authors checked the hex file after a compiling phase and applied their methods to the address bits. However, if a system with cryptographic modules is large, it can be difficult to practically use special memory addresses.

To overcome these limitations, we propose constant AND and ADD that completely exclude LUTs. The algorithms generate data-independent leakages, which can be used for cryptographic modules without SBoxes such as ARX block ciphers.

3. A New Dual-Rail-Based Power-Balancing Countermeasure without Look-Up Tables

In this section, we propose a constant AND and a constant ADD based on dual-rail encoding schemes without LUTs. These algorithms are based on the 1-to-4 encoding rule. Unlike previous studies in which LUTs were always used, our proposals overcome side-channel vulnerabilities in address bits and memory usage. Our constant ADD is an extended version of the methods employed by prior studies [20,21] (Appendix A). The proposed algorithm is a software-optimized Kogge–Stone adder (KS adder), and it can calculate multi-word codewords. Briefly, the KS adder in the software proposed by Coron et al. did not consider carry bits for multi-word processing [20]. Won et al. proposed an algorithm that calculates carry bits, but the algorithm is inefficient because it deals with

carry bits as a decimal point [21]. We propose an optimized algorithm based on the fact that a carry bit in lower words does not exceed 1. We also propose an addition algorithm that can be applied to PB methods.

3.1. Constant AND Operation

Table 3 illustrates the processes of our constant AND. c is a result of $a \wedge b$.

Table 3. Operation process of the constant AND.

$\wedge$	$\overline{a}$ $\overline{b}$	a $\overline{b}$	$\overline{a}$ b	a b	$\cdots\cdots(1)$
$\oplus$	$c \oplus a \oplus b \oplus 1$ $\overline{a}$	$c \oplus a$ a	$c \oplus b$ 1	c 1	$\cdots\cdots(2)$
$\oplus$	$(c \oplus b)$ $\overline{b}$	c 0	$(c \oplus b) \oplus 1$ b	$c \oplus 1$ 1	$\cdots\cdots(3)$
	$c \oplus 1$	c	$c \oplus 1$	c	

The design principles include the following.

(1) Even if data is encoded into the dual-rail scheme, at least one bit must include an AND result. Based on this idea, we calculated an AND operation of different types of codewords. Step (1) in Table 3 is an AND result of the codeword a encoded with E_1 and the codeword b encoded with E_2. We verified that this AND operation satisfies constant HW/HD. Consequently, the AND operation between different types of codewords satisfy constant HW/HD. Tables 4 and 5 present an HW($E_1(a) \wedge E_2(b)$) and an HD($E_1(a), E_1(a) \wedge E_2(b)$), respectively. As can be observed in the tables, the HWs and the HDs of the AND result are 1.

Table 4. HW($E_1(a) \wedge E_2(b)$) of Step. (1) in Table 3.

a	b	$c \oplus a \oplus b \oplus 1$	$c \oplus a$	$c \oplus b$	c	**HW**
0	0	1	0	0	0	1
0	1	0	0	1	0	1
1	0	0	1	0	0	1
1	1	0	0	0	1	1

Table 5. HD($E_1(a), E_1(a) \wedge E_2(b)$) of Step. (1) in Table 3.

a	b	$c \oplus b$	c	$c \oplus a \oplus b \oplus 1$	$c \oplus a$	**HD(A,C)**
0	0	0	0	1	0	1
0	1	1	0	0	0	1
1	0	0	0	0	1	1
1	1	0	1	0	0	1

(2) We must remove a and b from each bit of the AND result while satisfying constant HW/HD. Here, we consider the removal of the a-terms. A simple method of removal is to XOR $aa00$. However, because the operation causes strong side-channel leakages related to a, a and $\overline{a}$ must be applied simultaneously to remove the a-terms. We removed the a-term with an OR result of the initial input $E_1(a)$ and 0011. The reason why the form is $\overline{a}a11$ is to maintain constant HW/HD.

(3) The b-terms have also been removed, similar to that in (2). To convert $E_2(b)$ into $\overline{b}0b1$, we computed ANDing 1010 and ORing 0001. All the processes satisfied constant HW/HD.

Algorithm 2 shows the algorithm described above. The initial process of T to 0 can be initialized to x or y.

Algorithm 2: `ConstAND` , AND Operator Applied to Power-balancing Countermeasure

Require: $x_{\in E_1}, y_{\in E_2}$
Ensure: $z_{\in E_1} (= E_1(x \wedge y))$
1: $T \leftarrow 0$
2: $T \leftarrow x \wedge y$ {HW : 8, HD : 8}
3: $x \leftarrow x \vee$ (`0b 0011...`) {HW : 24, HD : 8}
4: $y \leftarrow y \wedge$ (`0b 1010...`) {HW : 8, HD : 8}
5: $y \leftarrow y \vee$ (`0b 0001...`) {HW : 16, HD : 8}
6: $T \leftarrow T \oplus x$ {HW : 16, HD : 24}
7: $T \leftarrow T \oplus y$ {HW : 16, HD : 16}
8: **return** T

3.2. Constant ADD Operation

In this section, we present a description of the manner in which the PB methods can be used for addition operations involving constant Boolean operations. We were inspired by the Kogge–Stone adder (KS adder) in achieving our goal. A concept of the KS adder was developed by Peter M. Kogge and Harold S. Stone [22]. The KS adder, which can be calculated using only Boolean operators, is known as a very fast addition algorithm. Coron et al. expanded the KS adder that was designed using operators that are supported by general software such as XOR, AND, and SHIFT [20]. Won et al. developed the KS adder to handle multi-word data [21]. Each algorithm is summarized in Appendix A. Additionally, the papers of Coron et al. and Won et al. are mainly about implementing the KS-adder in software, not related to PB methods. However, in order to compare the performance to the multi-word operation, the algorithms were set as a comparison group.

To apply PB methods to the KS adder, several problems must be addressed. For instance, assume that we apply our encoding rule to 8-bit based block ciphers in a 32-bit microcontrollers. In this case, we simply apply the constant XOR and AND operations to Coron's algorithm. In other words, we replace 8-bit data with a 32-bit codeword and put it in a 32-bit register. However, most ARX block ciphers require 32-bit additions. These cryptosystems represent 32-bit data using four 32-bit codewords in our encoding rule. This means that a carry bit generated in a lower word should be updated in an upper word, as in the case of arbitrary-precision arithmetic. Won's algorithm addressed this problem; however, the algorithm is not efficient because it computes the carry bit of a lower word as a decimal point. For instance, suppose that 32-bit data is calculated as four 8-bit words. Won's algorithm requires five 8-bit KS adders.

We designed an algorithm that directly applies a carry bit of a lower word to the least significant bit (LSB) of an upper word. Algorithm 3 shows our KS adder. Algorithm 4 is an addition algorithm with multi-words comprising our KS adder and Coron's KS adder.

Here, we present a comparison of the differences between our Algorithm 3, Coron's Algorithm A1, and Won's algorithms Algorithms A3 and A5. Coron's algorithm Algorithm A1 did not consider carry bits. According to the design principles of the KS adder, a carry bit of an upper word is the most significant bit of the last state of G. That is, the carry bit of an upper word can be easily solved. A problem arises when handling a carry bit in a lower word.

Won's Algorithms A3 and A5 overcame this problem based on the idea of a decimal point. Assume that the words to be added are (`0b 0101`) and (`0b 0011`), and a carry bit is generated in a lower word. Won's algorithm performs computations by modifying the two words into (`0b 0101.1`) and (`0b 0011.1`). This method intuitively handles the carry bit of a lower word; however, it is inefficient. The algorithm requires the initial and final processes for converting words between a normal format and a decimal format, and the algorithm again computes the KS adder.

Algorithm 3: $\texttt{KSAdder}_{\texttt{Ours}}$, Our Kogge–Stone Adder

Require: $x, y \in \{0,1\}^k$, Carry $c \in \{0,1\}$, and $l = max(\lceil \log_2(k-1) \rceil, 1)$
Ensure: Carry $c \in \{0,1\}$, $z(= x + y \mod 2^k)$
1: $P \leftarrow x \oplus y \oplus c$
2: $P' \leftarrow P$
3: $G \leftarrow ((c \oplus x) \wedge (c \oplus y)) \oplus c$ $\{(x \wedge c) \oplus (x \wedge y) \oplus (y \wedge c)\}$
4: **for** $i = 0$ to $l - 2$ **do**
5: $\quad G \leftarrow (G \wedge (G \ll (1 \ll i))) \oplus G$
6: $\quad P \leftarrow (P \wedge (P \ll (1 \ll i)))$
7: **end for**
8: $G \leftarrow (G \wedge (G \ll (1 \ll (l-1)))) \oplus G$
9: $c||z \leftarrow P' \oplus (2G)$ {Carry bit is MSB of G}
10: **return** (c, z)

Algorithm 4: Our Multi-bit Adder Combining Kogge–Stone Adders

Require: $X(= x_{n-1}||...||x_0), Y(= y_{n-1}||...||y_0)$ $\{x_i, y_i \in \{0,1\}^k\}$
Ensure: $Z(= z_{n-1}||...||z_0)$
1: $C \leftarrow 0$
2: $(C, z_0) \leftarrow \texttt{KSAdder}_{\texttt{Coron}}(x_0, y_0)$
3: **for** $i = 1$ to $n - 1$ **do**
4: $\quad (C, z_i) \leftarrow \texttt{KSAdder}_{\texttt{Ours}}(x_i, y_i, C)$
5: **end for**
6: **return** $(z_{n-1}||...||z_0)$

Before describing our algorithm, consider the following example; "A carry bit is generated in a lower word, and both LSBs of the words are 1." Even in this case, a carry bit, which is generated in both LSBs and the lower carry bit, does not exceed 1. We redesigned the software-based KS adder based on this fact. Lines 1, 3 in Algorithm 3 are the keys for handling a carry bit. We assume that the KS algorithm is computed in 1-bit units. The algorithm is of the form in which for loops after Line 4 are removed. Here, P' denotes the result of the addition, and G denotes a carry bit. Now, suppose that we increase the size of the bits. However, even if the word size increases, the LSB of a result $P' \oplus 2G$ does not change. Based on these ideas, we initialized P' to an XOR result of a lower carry bit and two input words (Line 1). In the case of G, the LSB of G denotes the carry bit that is generated by a lower carry bit and the LSBs of inputs. We initialized G using the majority rule (Line 3). However, because the well-known majority rule includes three AND operators, we adjusted the formula with one AND operator.

We further designed Algorithms 5 and 6 with PB methods. Algorithms A2, A4, and A6 in Appendix A show the algorithms of Coron et al. and Won et al. with PB methods.

Algorithm 5: $\texttt{BKSAdder}_{\texttt{Ours}}$, Our KS Adder Applied to Power-balancing Methods

Require: $x_{\in E_1}, y_{\in E_2}$, Carry $c_{\in E_2}$, and $l = max(\lceil \log_2(k-1) \rceil, 1)$
Ensure: Carry $c_{\in E_2}$, $z_{\in E_3} (= E_3(x + y + c \mod 2^k))$

$P, P', G \leftarrow 0$
$P_{\in E_3} \leftarrow x_{\in E_1} \oplus y_{\in E_2}$
$P_{\in E_1} \leftarrow P_{\in E_3} \oplus c_{\in E_2}$
$P'_{\in E_1} \leftarrow P_{\in E_1}$
$c_{\in E_3} \leftarrow c_{\in E_2} \oplus (\texttt{0b 1010...})$
$x_{\in E_2} \leftarrow x_{\in E_1} \oplus c_{\in E_3}$
$y_{\in E_1} \leftarrow y_{\in E_2} \oplus c_{\in E_3}$
$G_{\in E_1} \leftarrow \texttt{ConstAND}(y_{\in E_1}, x_{\in E_2})$
$G_{\in E_2} \leftarrow G_{\in E_1} \oplus c_{\in E_3}$
for $i = 0$ to $l - 2$ **do**
    $T \leftarrow 0$; $T \leftarrow G_{\in E_2} \ll (4 \ll i)$
    $T_{\in E_2} \leftarrow T \oplus (\texttt{0b ...0000 1100...})$
    $T_{\in E_1} \leftarrow \texttt{ConstAND}(P_{\in E_1}, T_{\in E_2})$
    $G_{\in E_3} \leftarrow T_{\in E_1} \oplus G_{\in E_2}$
    $G_{\in E_2} \leftarrow G_{\in E_3} \oplus (\texttt{0b 1010...})$
    $T \leftarrow 0$; $T \leftarrow P_{\in E_1} \ll (4 \ll i)$
    $T_{\in E_1} \leftarrow T \oplus (\texttt{0b ...0000 1010...})$
    $P_{\in E_2} \leftarrow P_{\in E_1} \oplus (\texttt{0b 0110...})$
    $P_{\in E_1} \leftarrow \texttt{ConstAND}(T_{\in E_1}, P_{\in E_2})$
end for
$T \leftarrow 0$; $T \leftarrow G_{\in E_2} \ll (4 \ll (l-1))$
$T_{\in E_2} \leftarrow T \oplus (\texttt{0b ...0000 1100...})$
$T_{\in E_1} \leftarrow \texttt{ConstAND}(P_{\in E_1}, T_{\in E_2})$
$G_{\in E_3} \leftarrow T_{\in E_1} \oplus G_{\in E_2}$
$c \leftarrow 0$; $c \leftarrow \texttt{MSCW of } G_{\in E_3}$ {MSCW : Most Significant Codeword}
$c_{\in E_2} \leftarrow c \oplus (\texttt{0b ...1100 1010})$
$T \leftarrow 0$; $T \leftarrow G_{\in E_3} \ll 4$
$T_{\in E_2} \leftarrow T \oplus (\texttt{0b ...1010 1100})$
$z_{\in E_2} \leftarrow P'_{\in E_1} \oplus T_{\in E_2}$
return $(c_{\in E_2}, z_{\in E_3})$

Algorithm 6: Our Multi-bit Adder Combining Kogge–Stone Adders Applied to Power-balancing Methods

Require: $X_{\in E_1} (= x_{n-1}||...||x_0)$, $Y_{\in E_2} (= y_{n-1}||...||y_0)$
Ensure: $Z_{\in E_3} (= z_{n-1}||...||z_0)$

$C \leftarrow 0$
$(C, z_0) \leftarrow \texttt{BKSAdder}_{\texttt{Coron}}(x_0, y_0)$
for $i = 1$ to $n - 1$ **do**
    $(C, z_i) \leftarrow \texttt{BKSAdder}_{\texttt{Ours}}(x_i, y_i, C)$
end for
return $(z_{n-1}||...||z_0)$

4. Side-Channel Security Evaluation

In this section, we evaluate the side-channel security ensured by the proposed method and those ensured by existing methods. We used the SCARF ARM (ARM920T) evaluation board, which has an operating frequency of approximately 100 MHz [23]. Power signals were collected using Wave Runner 204Xi-A oscilloscope from LeCroy at a sample rate of 1GS/s. We set two comparative groups, which include an 8-bit AND operation (u8AND) without any countermeasures and a constant AND operation with Pour's method-1. We evaluated the address bits of Pour's method-1 to analyze the side-channel vulnerabilities in LUTs. The implementations verified (at the assembly level) that the registers were used as intended by the designer.

Figures 1–3 present the evaluation results of the u8AND operation, our `ConstAND`, and Pour's method-1, respectively. In the figures, the upper left side is a graph of 10 traces. The x-axis denotes the samples, and the y-axis denotes the relative magnitude. The upper right side shows an SNR graph of [24] according to HW. The x-axis denotes the samples, and the y-axis denotes the SNR. The lower left side shows a graph of CPA [25] results with 10,000 traces. The x-axis denotes the samples and the y-axis denotes the correlation coefficient. The lower right side is a graph of the highest CPA peaks according to the number of traces. The x-axis denotes the number of traces and the y-axis denotes the correlation coefficient. The SNR formula is as follows [24].

Definition 1 (Signal-to-Noise Ratio (SNR) [24])**.** *The signal-to-noise Ratio of a leakage is denoted by a random variable L, which depends on the informative part denoted by I, as follows.*

$$SNR[L, I] = \frac{Var[\mathbb{E}[L|I]]}{\mathbb{E}[Var[L|I]]} \tag{1}$$

Figure 1 presents the results of the normal AND operation. These results illustrate the physical characteristics of the evaluation board. As observed from the CPA peaks, this board exhibits ideal side-channel leakages as in an HW model. The SNR is also approximately $5 \sim 8$, and this board exhibits strong leakages in power consumption like the register B.

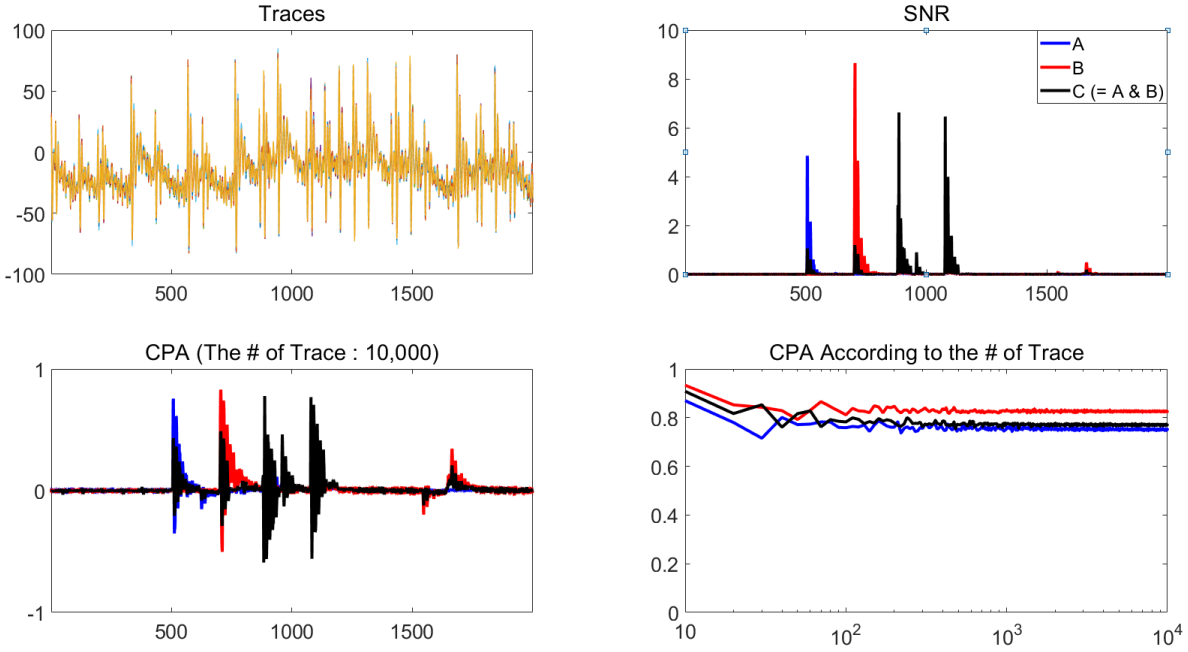

Figure 1. Evaluation Results of the u8AND.

Figure 2 presents the results of our `ConstAND`. PB methods are known to reduce SNRs and decrease CPA peaks. In this experiment, the decreasing rates of SNR were 429% to 2697%. The decreasing rates of CPA peaks were 48% to 318%. The reason for the occurrence of strong side-channel leakages at approximately 1450 is that we assume the register B to have no ideal HW model. However, our countermeasure exhibits decreased SNRs and CPA peaks compared to the u8AND experiments. The proposed method exhibits sufficient performance characteristics as PB methods.

Figure 3 presents the results of Pour's method-1. We implemented a constant AND based on Pour's method-1 using the divide and conquer method on four 8-bit codewords. In other words, we implemented a constant AND consisting of four $\mathrm{LUT}_{i(=1,\ldots,4)}$ operations in 8-bit units. We did not post the results of experiments related to the data, and the results showed low leakages as in our `ConstAND` experiments. We focused on demonstrating the side-channel vulnerabilities of the address bits of LUTs. As the authors had noted in their paper, special editing of the hex file is required to prevent side-channel leakages of the address bits. However, it is practically difficult to manipulate the hex file of cryptographic modules in a large system. We experimented with an assumption that the hex file cannot be manipulated. We restored the address bit of the target LUT using brute force attacks focusing on the starting address from `0x00` to `0xFF`. The experimental results of the address bits are almost the same as in the case of the normal AND operation. The reason why the SNR is small compared with the CPA peaks is that the codewords of the LUT input includes only 2-bit information.

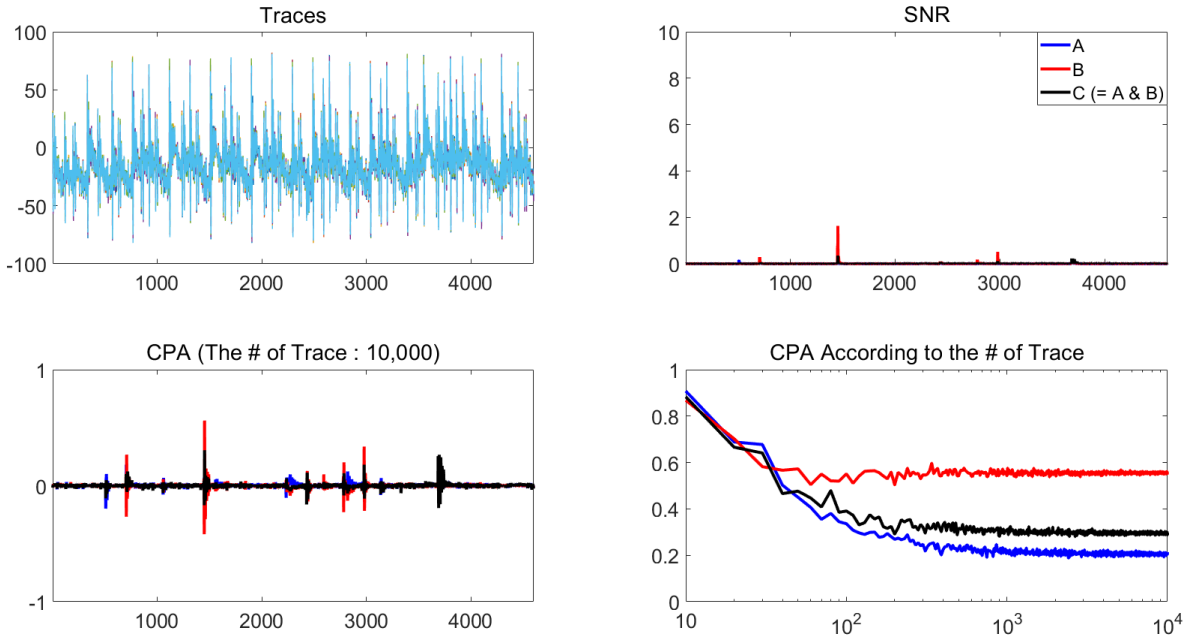

Figure 2. Evaluation results of our algorithm.

We evaluated the side-channel security ensured by our algorithm. The SNR and the CPA peak were decreased by up to 2697% and 318%, respectively, compared to the normal operations. Compared with existing methods that use LUTs, the proposed method is incapable of attacking the address bits itself.

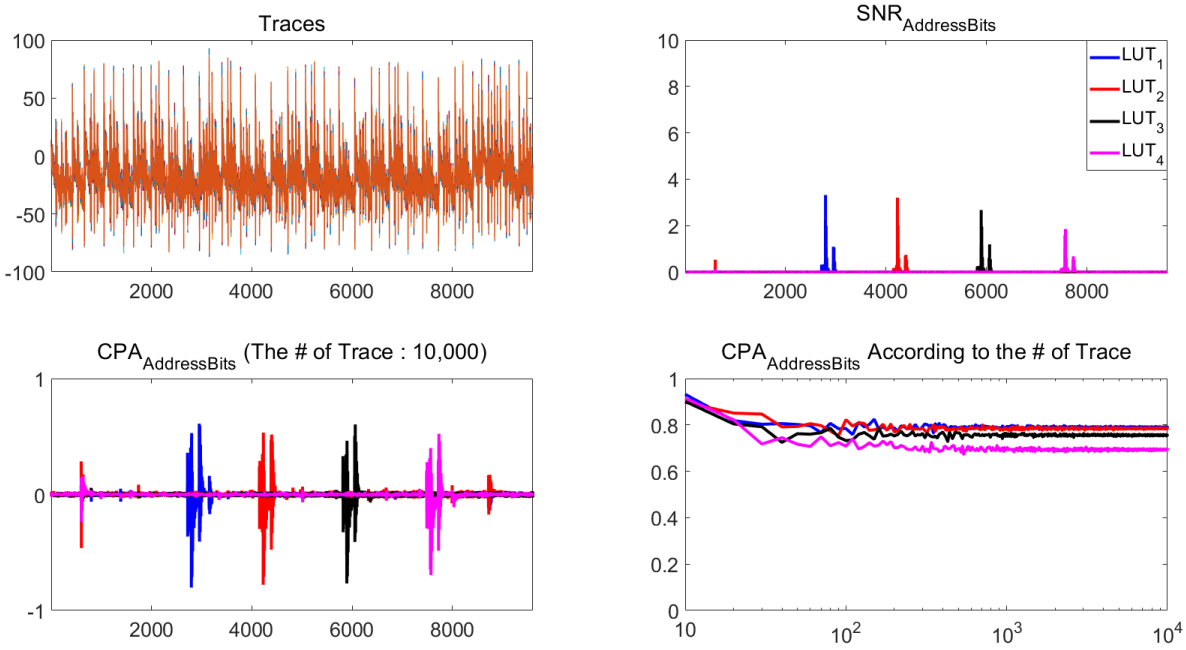

Figure 3. Evaluation results of Pour's method-1.

5. Performance Analysis and Case Study

In this section, we measure the performance of our method and those of existing studies. We analyzed the number of operators in the proposed method as well as those in the comparison groups (Coron's algorithm and Won's algorithm) and compared the number of operating clocks using the ARM simulator (Cortex-M0) provided by IAR Embedded Workbench. We applied PB methods to SIMON and SPECK block ciphers and compared their performances with previous studies. The reason why SIMON and SPECK block ciphers were selected as case studies is that each cipher consists of AND operators and ADD operators as main operations, so that the additional cost of the proposed countermeasures can be precisely analyzed. In these comparisons, the SIMON block cipher compared the performance of the proposed method with unprotected case and Pour's countermeasure, and the SPECK block cipher compared the performance of the proposed method with unprotected case and Won's technique. However, these comparisons may raise the question of whether the results may differ for other block ciphers. The answers to the question are as follows. When applying side-channel countermeasures to block ciphers, the main part of overhead is the nonlinear function. In other words, the additional cost of applying PB methods to nonlinear functions has a major impact on overall performance. As can be seen from the number of operators in the algorithms, it can be inferred that similar results will occur in other block ciphers.

5.1. Performance Analysis

The number of operators is presented in Table 6. The number of simulation clocks is shown in Table 7. Here, n is the number of words, l is the $max(\lceil \log_2(k-1) \rceil, 1)$, and k is the size of the "data" bits of a word. For instance, 32-bit data is transformed into four 32-bit codewords with 1-to-4 encoding. Here, n is 4, k is 8, and l is 3 ($= \lceil \log_2 7 \rceil$). We counted the output operation of the KS algorithms (i.e., the operation to split into a carry bit and an addition result) using one AND and one SHIFT.

Table 6. Comparison of the total number of operations.

Operator	Ours	Coron's	Won's
ine [ConstAND]			
XOR	2	N/A	N/A
AND	2	N/A	N/A
OR	2	N/A	N/A
ine [KS algorithm]			
XOR	$l+6$	$l+2$	$l+3$
AND	$2l+1$	$2l+1$	$2l+1$
SHIFT	$4l$	$4l$	$4l$
ine [PB KS algorithm]			
XOR	$5l+6$	$5l+3$	$5l+4$
ConstAND(# of operator)	$2l$ $(12l)$	$2l$ $(12l)$	$2l$ $(12l)$
SHIFT	$4l$	$4l$	$4l$
ine [Multi-bit adder]			
Coron's KS adder	1	N/A	1
Ours KS adder	$n-1$	N/A	-
Won's KS adder	-	N/A	n
e.t.c.	-	N/A	Line 3, 4, 9, 11
ine [PB Multi-bit adder]			
XOR	$5nl+6n-3$	N/A	$5nl+4n+5l+3$
ConstAND(# of operator)	$2nl$ $(12nl)$	N/A	$2nl+2l$ $(12nl+12l)$
SHIFT	$4nl$	N/A	$4l$

Table 7. Comparison of the total number of operations 2.

Algorithm	Ours	Coron's	Won's
KS alg. $(k=8)$	76	68	72
PB KS alg. $(k=8)$	277	270	274
Multi-bit adder $(k=8, n=4)$	340	N/A	485
PB Multi-bit adder $(k=8, n=4)$	1,133	N/A	1,603

As can observed in Tables 6 and 7, our `ConstAND` can operate with only six operators. In the KS algorithms, our method exhibits additional operations in the initial process of calculating P and G. However, the total number of operations is almost the same as that in the other methods because the operations inside the for loops are the same. By applying the PB methods to the KS adder with `ConstAND`, the total number of operations of the three algorithms are almost the same. However, our method exhibits advantages in the multi-bit KS adder. Coron's KS adder cannot handle the carry bit of a lower word, and it was used only in a least significant word. To process n words, Won's multi-bit adder required a total of $(n+1)$ KS adders, where the number of Coron's KS adders is 1 and the number of Won's KS adders is n. On the other hand, our method required a total of n KS adders, where the number of Coron's KS adders is 1 and the number of our KS adders is $(n-1)$. Furthermore, Won's multi-bit adder requires conversion of the input and output between normal and decimal representations.

5.2. SIMON/SPECK Block Cipher

The SIMON and SPECK families were developed for content security on constrained devices by a group of researchers at the US National Security Agency's Research Directorate [26]. The SIMON block cipher consists of only AND, ROTATION, and XOR operations. The SPECK block cipher consists of an ARX (ADD, ROTATION, and XOR) structure. We implemented SIMON and SPECK using the constant AND and ADD, and we compared the performance with the ARM simulator.

Table 8 presents the ARM simulation results. The 32-bit coding is implemented using 32-bit `long` type registers, and the 8-bit coding is implemented using 8-bit `char` type registers. Because the

SPECK block cipher required 32-bit additions, these additions also used 32-bit additions even with the 8-bit coding.

Table 8. Comparison of the total number of operations 3.

64/96 SIMON	# of Clocks	64/96 SPECK	# of Clocks
32-bit Coding	1406	32-bit Coding	877
8-bit Coding	3603	8-bit Coding*(w/o Addition)*	2321
Ours	11,794	Ours	33,149
Pour's Method-1	15,658	Won's Method	45,369

The nonlinear functions of the SIMON block cipher include AND operators. We implemented the protected SIMON cipher with Pour's method-1 and our `ConstAND`. As a result, it was found that our method demonstrated a performance improvement of 33% compared to Pour's method-1. This difference in performance was because Pour's method-1 handles a codeword in small LUT units, whereas our method processes the entire codeword.

The nonlinear functions of the SPECK cipher include ADD operators. We implemented this cipher with four 32-bit codewords based on 1-to-4 encoding. As a result, our method demonstrated a performance improvement of 37% compared to Won's method. The biggest reason for the difference in performance is that our method includes 4 times the number of KS adders, whereas Won's method includes 5 times the number of KS adders for protected addition.

6. Conclusions

In this paper, we proposed power-balancing methods without LUTs. We designed new constant AND and ADD operations based on 1-to-4 encoding. We also evaluated the side-channel security on a real board and assessed the performance based on the number of operators and simulation results. The proposed methods could be optimized for 32-bit microcontrollers such as ARM, AVR32, etc. The biggest advantage of our methods is that these schemes do not use look-up tables at all. As a result, the proposed schemes do not lead to side-channel leakages from the address bits of memories. On the other hand, the proposed algorithms have a disadvantage in terms of running time compared with the implementation coded by using only LUTs. An environment that meets these conditions is one in which memory resources are sufficient and the implementation is coded with complete control over the side-channel leakages of address bits. However, from a practical perspective, it is very difficult to implement all the operations using LUTs and check the side-channel leakages of address bits for all of them.

Given that threats such as side-channel attacks are growing in number, customized countermeasures are required in various environments. Our proposals are particularly effective for an environment where random numbers cannot be used as in IoT environments.

Author Contributions: Conceptualization, H.K. (HanBit Kim); methodology, H.K. (HanBit Kim) and H.K. (HeeSeok Kim); software, H.K. (HanBit Kim); validation, H.K. (HanBit Kim) and S.H.; investigation, H.K. (HanBit Kim); writing—original draft preparation, H.K. (HanBit Kim); writing—review and editing, H.K. (HanBit Kim), H.K. (HeeSeok Kim) and S.H.; supervision, H.K. (HeeSeok Kim) and S.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: This work was supported as part of Military Crypto Research Center (UD170109ED) funded by Defense Acquisition Program Administration (DAPA) and Agency for Defense Development (ADD).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Existing Algorithms for Addition and Their Countermeasures

For completeness, we include the existing algorithms and protected algorithms that employ PB methods in the appendix. Algorithms A1–A6 show the algorithms of Coron et al. and Won et al. In addition, we show the examples of our algorithm and the existing software-based KS adder algorithms with Figure A1. Above from left, Figure A1 present examples of the software-based KS adders of us, Coron et al., and Won et al., respectively. As you can see in the example, Coron's algorithm did not consider carry bits. Therefore, it can not compute the addition of multi-word data by this algorithm alone. On the other hand, Won's algorithm performs computations by modifying the two words into ((0b 110.0) and (0b 111.0).) at $c = 0$ such as the first addition. If a carry bit is 1, the algorithm performs computations by modifying the two words into ((0b 101.1) and (0b 010.1).) such as the second addition. However, the algorithm requires the initial and final processes for switching words between a normal format and a decimal format. To overcome these cons, we closely dissected the principle of a carry-bit propagation in the software-based KS algorithm and proposed an optimized algorithm.

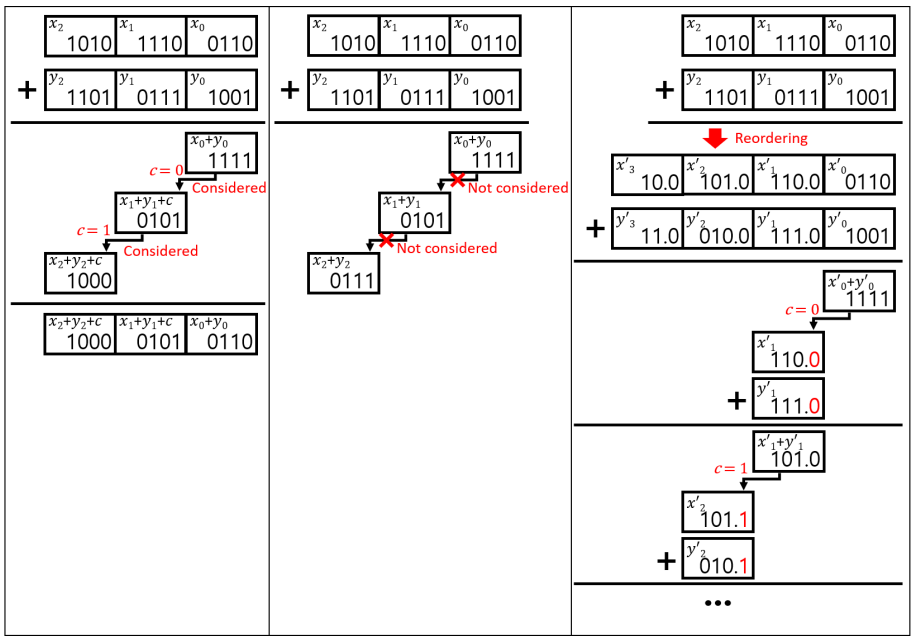

Figure A1. Concepts and examples of our, Coron's, and Won's Software-based Kogge–Stone Adder Algorithms.

Algorithm A1: $\texttt{KSAdder}_{\texttt{Coron}}$, Modified Coron's Kogge–Stone Adder

Require: $x, y \in \{0,1\}^k$, and $l = max(\lceil \log_2(k-1) \rceil, 1)$
Ensure: Carry $c \in \{0,1\}$, $z(= x + y \mod 2^k)$
1: $P \leftarrow x \oplus y$
2: $P' \leftarrow P$
3: $G \leftarrow x \wedge y$
4: **for** $i = 0$ to $l - 2$ **do**
5: $\quad G \leftarrow (G \wedge (G \ll (1 \ll i))) \oplus G$
6: $\quad P \leftarrow (P \wedge (P \ll (1 \ll i)))$
7: **end for**
8: $G \leftarrow (G \wedge (G \ll (1 \ll (l-1)))) \oplus G$
9: $c||z \leftarrow P' \oplus (2G)$ {Carry bit is MSB of G}
10: **return** (c, z)

Algorithm A2: $\texttt{BKSAdder}_{\texttt{Coron}}$, Coron's KS Adder Applied to Power-balancing Methods

Require: $x_{\in E_1}, y_{\in E_2}$, Carry $c_{\in E_2}$, and $l = max(\lceil \log_2(k-1) \rceil, 1)$
Ensure: Carry $c_{\in E_2}$, $z_{\in E_3} (= E_3(x + y \mod 2^k))$

$P, P', G \leftarrow 0$
$P_{\in E_3} \leftarrow x_{\in E_1} \oplus y_{\in E_2}$
$P_{\in E_1} \leftarrow P_{\in E_3} \oplus (\texttt{0b 1100...})$
$P'_{\in E_1} \leftarrow P_{\in E_1}$
$G_{\in E_1} \leftarrow \texttt{ConstAND}(x_{\in E_1}, y_{\in E_2})$
$G_{\in E_2} \leftarrow G_{\in E_1} \oplus (\texttt{0b 0110...})$
for $i = 0$ to $l - 2$ **do**
  $T \leftarrow 0; \quad T \leftarrow G_{\in E_2} \ll (4 \ll i)$
  $T_{\in E_2} \leftarrow T \oplus (\texttt{0b ...0000 1100...})$
  $T_{\in E_1} \leftarrow \texttt{ConstAND}(P_{\in E_1}, T_{\in E_2})$
  $G_{\in E_3} \leftarrow T_{\in E_1} \oplus G_{\in E_2}$
  $G_{\in E_2} \leftarrow G_{\in E_3} \oplus (\texttt{0b 1010...})$
  $T \leftarrow 0; \quad T \leftarrow P_{\in E_1} \ll (4 \ll i)$
  $T_{\in E_1} \leftarrow T \oplus (\texttt{0b ...0000 1010...})$
  $P_{\in E_2} \leftarrow P_{\in E_1} \oplus (\texttt{0b 0110...})$
  $P_{\in E_1} \leftarrow \texttt{ConstAND}(T_{\in E_1}, P_{\in E_2})$
end for
$T \leftarrow 0; \quad T \leftarrow G_{\in E_2} \ll (4 \ll (l-1))$
$T_{\in E_2} \leftarrow T \oplus (\texttt{0b ...0000 1100...})$
$T_{\in E_1} \leftarrow \texttt{ConstAND}(P_{\in E_1}, T_{\in E_2})$
$G_{\in E_3} \leftarrow T_{\in E_1} \oplus G_{\in E_2}$
$c \leftarrow 0; \quad c \leftarrow \texttt{MSCW of } G_{\in E_3}$ {MSCW : Most Significant Codeword}
$c_{\in E_2} \leftarrow c \oplus (\texttt{0b ...1100 1010})$
$T \leftarrow 0; \quad T \leftarrow G_{\in E_3} \ll 4$
$T_{\in E_2} \leftarrow T \oplus (\texttt{0b ...1010 1100})$
$z_{\in E_2} \leftarrow P'_{\in E_1} \oplus T_{\in E_2}$
return $(c_{\in E_2}, z_{\in E_3})$

Algorithm A3: $\texttt{KSAdder}_{\texttt{Won}}$, Modified Won's Kogge–Stone Adder

Require: $x', y' \in \{0,1\}^{k-1} || \{0\}$, Carry $c \in \{0,1\}$, and $l = max(\lceil \log_2(k-1) \rceil, 1)$
Ensure: Carry $c \in \{0,1\}$, $z'(= x' + y' + 2c \mod 2^k) \in \{0,1\}^{k-1} || \{0\}$

$P \leftarrow x' \oplus y'$
$P' \leftarrow P$
$G \leftarrow (x' \wedge y') \oplus c$
for $i = 0$ to $l - 2$ **do**
  $G \leftarrow (G \wedge (G \ll (1 \ll i))) \oplus G$
  $P \leftarrow (P \wedge (P \ll (1 \ll i)))$
end for
$G \leftarrow (G \wedge (G \ll (1 \ll (l-1)))) \oplus G$
$c || z' \leftarrow P' \oplus (2G)$ {Carry bit is MSB of G}
return (c, z')

Algorithm A4: $\texttt{BKSAdder}_{\texttt{Won}}$, Won's KS Adder Applied to Power-balancing Methods

Require: $x'_{\in E_1}, y'_{\in E_2}$, Carry $c_{\in E_2}$, and $l = max(\lceil \log_2(k-1) \rceil, 1)$
Ensure: Carry $c_{\in E_2}$, $z'_{\in E_3} (= E_3(x' + y' + 2c \mod 2^k))$

$P, P', G \leftarrow 0$
$P_{\in E_3} \leftarrow x'_{\in E_1} \oplus y'_{\in E_2}$
$P_{\in E_1} \leftarrow P_{\in E_3} \oplus (\texttt{0b 1100...})$
$P'_{\in E_1} \leftarrow P_{\in E_1}$
$G_{\in E_1} \leftarrow \texttt{ConstAND}(x'_{\in E_1}, y'_{\in E_2})$
$G_{\in E_3} \leftarrow G_{\in E_1} \oplus c_{\in E_2}$
$G_{\in E_2} \leftarrow G_{\in E_1} \oplus (\texttt{0b 1010...})$
for $i = 0$ to $l - 2$ **do**
  $T \leftarrow 0$; $T \leftarrow G_{\in E_2} \ll (4 \ll i)$
  $T_{\in E_2} \leftarrow T \oplus (\texttt{0b ...0000 1100...})$
  $T_{\in E_1} \leftarrow \texttt{ConstAND}(P_{\in E_1}, T_{\in E_2})$
  $G_{\in E_3} \leftarrow T_{\in E_1} \oplus G_{\in E_2}$
  $G_{\in E_2} \leftarrow G_{\in E_3} \oplus (\texttt{0b 1010...})$
  $T \leftarrow 0$; $T \leftarrow P_{\in E_1} \ll (4 \ll i)$
  $T_{\in E_1} \leftarrow T \oplus (\texttt{0b ...0000 1010...})$
  $P_{\in E_2} \leftarrow P_{\in E_1} \oplus (\texttt{0b 0110...})$
  $P_{\in E_1} \leftarrow \texttt{ConstAND}(T_{\in E_1}, P_{\in E_2})$
end for
$T \leftarrow 0$; $T \leftarrow G_{\in E_2} \ll (4 \ll (l-1))$
$T_{\in E_2} \leftarrow T \oplus (\texttt{0b ...0000 1100...})$
$T_{\in E_1} \leftarrow \texttt{ConstAND}(P_{\in E_1}, T_{\in E_2})$
$G_{\in E_3} \leftarrow T_{\in E_1} \oplus G_{\in E_2}$
$c \leftarrow 0$; $c \leftarrow \texttt{MSCW of } G_{\in E_3}$ {MSCW : Most Significant Codeword}
$c_{\in E_2} \leftarrow c \oplus (\texttt{0b ...1100 1010})$
$T \leftarrow 0$; $T \leftarrow G_{\in E_3} \ll 4$
$T_{\in E_2} \leftarrow T \oplus (\texttt{0b ...1010 1100})$
$z'_{\in E_2} \leftarrow P'_{\in E_1} \oplus T_{\in E_2}$
return $(c_{\in E_2}, z'_{\in E_3})$

Algorithm A5: Modified Won's Multi-bit Adder Combining Kogge–Stone Adders

Require: $X(= x_{n-1}||...||x_0)$, $Y(= y_{n-1}||...||y_0)$ $\{x_i, y_i \in \{0,1\}^k\}$
Ensure: $Z(= z_{n-1}||...||z_0)$

$C \leftarrow 0$
for $i = 1$ to n **do**
  $x'_i \leftarrow (x^{i(k-1)+k-1}||...||x^{i(k-1)+1}||0)$
  $y'_i \leftarrow (y^{i(k-1)+k-1}||...||y^{i(k-1)+1}||0)$
end for
$(C, z_0) \leftarrow \texttt{KSAdder}_{\texttt{Coron}}(x_0, y_0)$
for $i = 1$ to n **do**
  $(C, z'_i) \leftarrow \texttt{KSAdder}_{\texttt{Won}}(x'_i, y'_i, C)$
  $z'_i \leftarrow z'_i \gg 1$
end for
$(z_{n-1}||...||z_0) \leftarrow \texttt{Adjustment}(z'_{n-1}||...||z'_0)$ {Bit reordering except 0}
return $(z_{n-1}||...||z_0)$

Algorithm A6: Multi-bit Adder Combining Won's Kogge–Stone Adders Applied to Power-balancing Methods

Require: $X_{\in E_1}(= x_{n-1}||...||x_0)$, $Y_{\in E_2}(= y_{n-1}||...||y_0)$ $\{x_i, y_i \in \{0,1\}^k\}$
Ensure: $Z_{\in E_3}(= z_{n-1}||...||z_0)$
1: $C \leftarrow 0$
2: **for** $i = 1$ to n **do**
3: $x'_i \leftarrow (x^{4i(k-1)+4(k-1)}||...||x^{4i(k-1)+4}||\texttt{0b 1010})$
4: $y'_i \leftarrow (y^{4i(k-1)+4(k-1)}||...||y^{4i(k-1)+4}||\texttt{0b 1100})$
5: **end for**
6: $(C, z_0) \leftarrow \texttt{BKSAdder}_{\texttt{Coron}}(x_0, y_0)$
7: **for** $i = 1$ to n **do**
8: $(C, z'_i) \leftarrow \texttt{BKSAdder}_{\texttt{Won}}(x'_i, y'_i, C)$
9: $z'_i \leftarrow z'_i \gg 4$
10: **end for**
11: $(z_{n-1}||...||z_0) \leftarrow \texttt{Adjustment}(z'_{n-1}||...||z'_0)$ {Bit reordering except $0_{\in E_3}$}
12: **return** $(z_{n-1}||...||z_0)$

References

1. Ors, S.B.; Gurkaynak, F.; Oswald, E.; Preneel, B. Power-Analysis Attack on an ASIC AES implementation. In Proceedings of the IEEE International Conference on Information Technology: Coding and Computing, ITCC 2004, Las Vegas, NV, USA, 5–7 April 2004; Volume 2, pp. 546–552.
2. Kocher, P.C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Annual International Cryptology Conference*; Springer: Berlin, Germany, 1996; pp. 104–113.
3. Gandolfi, K.; Mourtel, C.; Olivier, F. Electromagnetic analysis: Concrete results. In *International Workshop on Cryptographic Hardware and Embedded Systems*; Springer: Berlin, Germany, 2001; pp. 251–261.
4. Messerges, T.S. *Power Analysis Attacks and Countermeasures for Cryptographic Algorithms*; University of Illinois at Chicago: Chicago, IL, USA, 2000.
5. Mangard, S.; Oswald, E.; Popp, T. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*; Springer Science & Business Media: Berlin, Germany, 2008; Volume 31.
6. Rivain, M.; Prouff, E. Provably secure higher-order masking of AES. In *International Workshop on Cryptographic Hardware and Embedded Systems*; Springer: Berlin, Germany, 2010; pp. 413–427.
7. Hoogvorst, P.; Duc, G.; Danger, J.L. Software implementation of dualrail representation. In *COSADE*; Springer: Darmstadt, Germany, 2011; pp. 24–25.
8. Tiri, K.; Akmal, M.; Verbauwhede, I. A dynamic and differential CMOS logic with signal independent power consumption to withstand differential power analysis on smart cards. In Proceedings of the 28th European Solid-state Circuits Conference, Seoul, Korea, 28–29 November 2002; pp. 403–406.
9. Rivain, M.; Prouff, E.; Doget, J. Higher-order masking and shuffling for software implementations of block ciphers. In *International Workshop on Cryptographic Hardware and Embedded Systems*; Springer: Berlin, Germany, 2009; pp. 171–188.
10. Han, Y.; Zhou, Y.; Liu, J. Securing lightweight block cipher against power analysis attacks. In *Future Wireless Networks and Information Systems*; Springer: Berlin, Germany, 2012; pp. 379–390.
11. Chen, C.; Eisenbarth, T.; Shahverdi, A.; Ye, X. Balanced encoding to mitigate power analysis: a case study. In *International Conference on Smart Card Research and Advanced Applications*; Springer: Berlin, Germany, 2014; pp. 49–63.
12. Won, Y.S.; Hodgers, P.; O'Neill, M.; Han, D.G. On the Security of Balanced Encoding Countermeasures. In *International Conference on Smart Card Research and Advanced Applications*; Springer: Berlin, Germany, 2015; pp. 242–256.
13. Won, Y.S.; Choi, S.W.; Park, D.W.; Han, D.G. Security of Constant Weight Countermeasures. *ETRI J.* **2017**, *39*, 417–427. [CrossRef]

14. Servant, V.; Debande, N.; Maghrebi, H.; Bringer, J. Study of a novel software constant weight implementation. In *International Conference on Smart Card Research and Advanced Applications*; Springer: Berlin, Germany, 2014; pp. 35–48.
15. Rauzy, P.; Guilley, S.; Najm, Z. Formally proved security of assembly code against power analysis. *J. Cryptogr. Eng.* **2016**, *6*, 201–216. [CrossRef]
16. Maghrebi, H.; Servant, V.; Bringer, J. There is wisdom in harnessing the strengths of your enemy: Customized encoding to thwart side-channel attacks. In *International Conference on Fast Software Encryption*; Springer: Berlin, Germany, 2016; pp. 223–243.
17. Bhasin, S.; Jap, D.; Peyrin, T. Practical Evaluation of FSE 2016 Customized Encoding Countermeasure. *IACR Trans. Symmetric Cryptol.* **2017**, 108–129. [CrossRef]
18. Petrvalsky, M.; Drutarovsky, M. Constant-weight coding based software implementation of DPA countermeasure in embedded microcontroller. *Microprocess. Microsyst.* **2016**, *47*, 82–89. [CrossRef]
19. Safaei Pour, M.; Salmasizadeh, M. A new CPA resistant software implementation for symmetric ciphers with smoothed power consumption: SIMON case study. *ISC Int. J. Inf. Secur.* **2017**, *9*, 21–32.
20. Coron, J.S.; Großschädl, J.; Tibouchi, M.; Vadnala, P.K. Conversion from arithmetic to boolean masking with logarithmic complexity. In *International Workshop on Fast Software Encryption*; Springer: Berlin, Germany, 2015; pp. 130–149.
21. Won, Y.S.; Han, D.G. Efficient conversion method from arithmetic to Boolean masking in constrained devices. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*; Springer: Berlin, Germany, 2017; pp. 120–137.
22. Kogge, P.M.; Stone, H.S. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. Comput.* **1973**, *100*, 786–793. [CrossRef]
23. Choi, Y.; Choi, D.; Ryou, J. Implementing Side Channel Analysis Evaluation Boards of KLA-SCARF system. *J. Korea Inst. Inf. Secur. Cryptol.* **2014**, *24*, 229–240.
24. Bruneau, N.; Guilley, S.; Najm, Z.; Teglia, Y. Multivariate high-order attacks of shuffled tables recomputation. *J. Cryptol.* **2018**, *31*, 351–393. [CrossRef]
25. Brier, E.; Clavier, C.; Olivier, F. Correlation power analysis with a leakage model. In *International Workshop on Cryptographic Hardware and Embedded Systems*; Springer: Berlin, Germany, 2004; pp. 16–29.
26. Beaulieu, R.; Treatman-Clark, S.; Shors, D.; Weeks, B.; Smith, J.; Wingers, L. The SIMON and SPECK lightweight block ciphers. In Proceedings of the 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 8–12 June 2015; pp. 1–6.