A Low-Latency High-Order Arithmetic to Boolean Masking Conversion

Jiangxue Liu[1], Cankun Zhao[1], Shuohang Peng[1], Bohan Yang[1], Hang Zhao[1], Xiangdong Han[1], Min Zhu[2], Shaojun Wei[1] and Leibo Liu[1]

[1] Beijing National Research Center for Information Science and Technology, School of Integrated Circuits, Tsinghua University, Beijing, China.
[2] Wuxi Micro Innovation Integrated Circuit Design Co., Ltd., Wuxi, China.
{liujx21,zck22,psh20}@mails.tsinghua.edu.cn;bohanyang@tsinghua.edu.cn;zhao-h21@mails.tsinghua.edu.cn;hanxd2023@tsinghua.edu.cn;zhumin@mucse.com;wsj@mail.tsinghua.edu.cn;liulb@tsinghua.edu.cn

Abstract. Masking, an effective countermeasure against side-channel attacks, is commonly applied in modern cryptographic implementations. Considering cryptographic algorithms that utilize both Boolean and arithmetic masking, the conversion algorithm between arithmetic masking and Boolean masking is required. Conventional high-order arithmetic masking to Boolean masking conversion algorithms based on Boolean circuits suffer from performance overhead, especially in terms of hardware implementation. In this work, we analyze high latency for the conversion and propose an improved high-order A2B conversion algorithm. For the conversion of 16-bit variables, the hardware latency can be reduced by 47% in the best scenario. For the case study of second-order 32-bit conversion, the implementation results show that the improved scheme reduces the clock cycle latency by 42% in hardware and achieves a 30% speed performance improvement in software. Theoretically, a security proof of arbitrary order is provided for the proposed high-order A2B conversion. Experimental validations are performed to verify the second-order DPA resistance of second-order implementation. The Test Vector Leakage Assessment does not observe side-channel leakage for hardware and software implementations.

Keywords: Masking · Arithmetic masking to Boolean masking conversion algorithms · Probe Isolating Non-Interference · Hardware Private Circuit · Test Vector Leakage Assessment

1 Introduction

Side-channel attacks (SCA) can recover keys using side information leaked from the physical interaction between cryptographic devices and the external environment during the operation of cryptographic algorithms. These side information include execution time [Koc96], consumed power [KJJ99] and electromagnetic radiation [QS01]. Among them, the Differential Power Analysis (DPA) [KJJ99] based on the minimal assumptions about the adversary [MME10] is a powerful attack method, even for devices with a low signal-to-noise ratio.

A mainstream countermeasure against DPA is masking [QS01], which divides sensitive variables and related intermediate states into independent and random shares to cut off the direct dependence between sensitive variables (such as the key-dependent information) and power consumption. According to the mathematical relationship between the original sensitive variable and shares, it can be divided into Boolean masking, arithmetic masking, and multiplicative masking [TDG03]. Boolean masking excels at logical operations like XOR and shifting, while arithmetic and multiplicative masking apply to addition/subtraction and multiplication, respectively. This signifies that an efficient masking scheme necessitates the use of corresponding masking for different types of operations. For algorithms that combine arithmetic operations and Boolean operations, side-channel protection schemes need to use both arithmetic masking, Boolean masking, and more importantly the conversion between the two masking countermeasures.

The first conversion algorithm against side-channel attacks was proposed by Goubin [Gou01], including converting from Boolean masking to arithmetic masking (B2A) and arithmetic masking to Boolean masking (A2B). Goubin's B2A with optimal computational complexity $\mathcal{O}(1)$ is efficient. Note that the complexity mentioned here aligns with [CGMZ22], indicating the number of elementary operations. However, the A2B algorithm with complexity $\mathcal{O}(k)$ for k-bit variables is less efficient, and the conversion in both directions is limited to first-order security. The table-based A2B conversion algorithm proposed in [CT03] can significantly improve the efficiency of A2B at a particular expense of additional memory consumption. However, there is a flaw in their algorithm, which has been continuously revised in [Deb12] and [VDV21].

The high-order A2B algorithm was first introduced by Coron in [CGV14], which based on the secure addition over the Boolean circuit, has complexity $\mathcal{O}(n^2 \cdot k)$ for n shares and k-bit variables. The complexity can be improved to $\mathcal{O}(n^2 \cdot \log k)$ by applying the Kogge-Stone carry look-ahead adder (KSA) instead of the ripple-carry adder (RCA), just like the first-order case in [CGTV15]. Coron et al. generalized table-based A2B conversion to high-order [CGMZ22], but only efficient in specific applications, which was further accelerated by Jan-Pieter D'Anvers [D'A22]. For hardware implementation, Boolean circuit-based A2B conversion is widely selected since it can share the underlying SCA-secure adder module with B2A conversion and is more efficient in general applications [FBR$^+$22].

For secure hardware implementations against side-channel attacks, the masked implementations of nonlinear functions need to insert registers to prevent potential leakage caused by glitches. In conventional hardware circuit design, inserting registers to cut critical paths in combinatorial circuits improves the frequency and overall performance. However, in side-channel secure hardware implementations inserting these registers that are not for performance purposes increases the delay in clock cycles and often degrades the overall performance. Previous studies have focused on the delay issue in secure hardware implementations and proposed solutions to reduce the latency of clock cycles. The low-latency masking was first introduced by Moradi et al., who used asynchronous design methods to decrease the delay of first-order threshold implementation [MS16]. The first high-order general S-box low-latency masking scheme (GLM) was proposed by Gross et al. [GIB18]. The essential idea of the GLM scheme is to skip the shares compression stage after the nonlinear operation, thereby eliminating the need for a register stage to increase the number of shared values. Arribas et al. later proposed a Low-Latency Threshold Implementations (LLTI) technique [AZN22] using a divide-and-conquer strategy for nonlinear functions with an algebraic degree greater than two to reduce area and delay.

In the widely studied low-latency masked implementation of lightweight ciphers, the AND gate depth of low-degree S-boxes in the round function is only one to two. For A2B conversion circuits, the number of nonlinear function layers far exceeds this number and increases with the width of the variables. The additional cost of hardware latency due to multi-layer nonlinear functions was not considered or evaluated at the outset of the A2B conversion algorithm design. The first-order hardware implementation [GPM23] of Coron et al.-A2B [CGV14] with 16-bit shares requires 34 cycles. The hardware latency issue in A2B conversion is notably exacerbated in high-order scenarios. Using the secure threshold implementations of adder in [SMG15] to build a second-order 32-bit A2B conversion according to the general hardware implementation scheme of first-order A2B in [FBR$^+$22], the conversion takes 20 clock cycles for the A2B conversion based on secure KSA or 99 clock cycles for that based on secure RCA. At the same time, we also observed that the A2B module with the multi-register stage would not become the frequency bottleneck of the entire system on the FPGA platform. The synthesis results of hardware acceleration of Kyber and Saber nonlinear operations in [FBR+22] also illustrate this point, and the maximum frequency of the secure adder is higher than other modules. Reducing the number of register stages in the A2B conversion algorithm, thereby reducing the delay in the number of clock cycles, helps to improve the system's overall performance. In this work, to the best of our knowledge, we are the first to address this issue and propose an improved high-order A2B conversion algorithm that avoids performance loss in terms of latency.

Our Contributions. Our contributions are summarized as follows:

1. We analyze the high latency problem of the existing A2B high-order conversion algorithm in hardware implementation and find the cause is that the higher-degree nonlinear function required for the carry calculation introduces multi-level registers in the SCA-secure implementation to prevent glitch propagation.

2. To avoid the performance overhead caused by carry calculation, we propose a high-order A2B conversion algorithm based on Carry-Save Adder (CSA), which implements secure carry-free addition by redundantly expressing intermediate variables. In the best scenario, hardware latency can be reduced by 50%.

3. We provide formal security proofs for the proposed A2B algorithm at arbitrary order. The experimental validation is launched to prove the security of the proposed second-order hardware and software implementations of improved A2B conversion.

4. Through a case study of 32-bit variables A2B conversion, we compare the performance of the CSA-based A2B conversion and the conventional KSA-based scheme in terms of hardware and software implementations. The results show that the second-order secure hardware implementation based on CSA reduces clock cycle latency by 42% while improving the time delay, area, and randomness consumption. The software implementation reduces the number of clock cycles required to execute a conversion by 30%.

Paper Organisation. We briefly introduce the Probe Isolating Non-Interference theorems and the high-order Boolean circuit-based A2B conversion algorithm in Section 2. In Section 3, an improved scheme is introduced to enhance the performance of high-order A2B, especially in terms of latency. In Section 4, the security of the proposed scheme is verified from both theoretical and experimental aspects. In Section 5, we conduct a performance evaluation of the proposed A2B conversion method, comparing it with other state-of-the-art implementations. Additionally, we explore the application of CSA-based A2B conversion to prime modulus and B2A scenarios.

2 Preliminaries

2.1 Notion

For a k-bit variable x, the n-share arithmetic masking is represented as $\boldsymbol{x}^{\boldsymbol{A_{2^k}}} = \left(x_i^{A_{2^k}}\right)_{i=1,\dots,n} \in \mathbb{F}_{2^k}$. The i^{th} share is represented by $x_i^{A_{2^k}}$, and share index $i \in [1, n]$. The sum of n arithmetic shares is expressed as

$$x = \sum_{i=1}^{n} x_i^{A_{2^k}} = x_1^{A_{2^k}} + ... + x_n^{A_{2^k}} \bmod 2^k. \tag{1}$$

Similarly, the set of k-bit Boolean sharing of a secret variable x is $\boldsymbol{x}^{\boldsymbol{B,k}}$. For n-share Boolean masking $\boldsymbol{x}^{\boldsymbol{B,k}} = \left(x_i^{B,k}\right)_{i=1,\dots,n}$, the i^{th} share is represented by $x_i^{B,k}$. The XOR of n shares is denoted as

$$x = \bigoplus_{i=1}^{n} x_i^{B,k} = x_1^{B,k} \oplus ... \oplus x_n^{B,k}. \tag{2}$$

When describing bit-level operations, we denote the i^{th} bit of x as $x[i]$ and the i^{th} bit of $\boldsymbol{x^{B,k}}$ is denoted as $\boldsymbol{x^{B,k}}[i]$.

2.2 Composable Probing Security

The probing model proposed by [ISW03] is the most commonly used attack model in the field of side-channel security. It assumes the adversary can place probes anywhere in the circuit and obtain corresponding intermediate variables. For the t-probing secure circuit C against t-order DPA, the tuple of any t probes in the circuit is independent of the sensitive variables. The probing model is often used to analyze the overall security of the circuit. The probing security cannot guarantee composable security, therefore the combination of multiple probing secure circuits is not necessarily probing secure. With the increase of circuit scale and security order t, the complexity of the overall probing security analysis of the circuit increases sharply. Barthe et al. [BBD+16] subsequently proposed strong security notions such as Non-Interfering (NI) and Strong-Non-Interfering (SNI), which simplified the security analysis of large-scale circuits and increased the reusability of secure gadgets.

Algorithm 1 $SecAnd_k^n$ (bitwise AND of Boolean maskings) [CS20] [CGLS20a]

Input: n shares Boolean sharing $\boldsymbol{x^{B,k}}, \boldsymbol{y^{B,k}}$.
Output: n shares Boolean sharing $\boldsymbol{z^{B,k}}$ such that $z = x \wedge y$.

```
1: for $i = 1$ to $n$ do
2:     for $j = i + 1$ to $n$ do
3:         $r_{ij} = r_{ji} \xleftarrow{\$} \mathbb{F}_{2^k}$
4: for $i = 1$ to $n$ do
5:     for $j = 1$ to $n, j \neq i$ do
6:         $u_{ij} \leftarrow \bar{x}_i^{B,k} \wedge \mathrm{Reg}[r_{ij}]$
7:         $v_{ij} \leftarrow y_j^{B,k} \oplus r_{ij}$
8: for $i = 1$ to $n$ do
9:     $z_i^{B,k} \leftarrow \mathrm{Reg}[x_i^{B,k} \wedge \mathrm{Reg}[y_i^{B,k}]] \oplus \bigoplus_{j=1,j\neq i}^{n} (\mathrm{Reg}[u_{ij}] \oplus \mathrm{Reg}[x_i^{B,k} \wedge \mathrm{Reg}[v_{ij}]])$
```

For trivial composition theorems, Cassiers and Standaert [CS20] further propose the notion of Probe Isolating Non-Interference (PINI). Consistent with [CS20], the circuit is PINI if an n-share circuit is $(n-1)$-PINI. They also provide the PINI multiplication gadgets PINI1 and PINI2. The $SecAnd_k^n$ gadget computing the bitwise AND of k-bit Boolean masking in Algorithm 1, is adapted from the parallelization of k PINI1 multiplication gadgets. By setting $k = 1$ and ignoring the registers needed in hardware implementation, $SecAnd_1^n$ is the PINI1 gadget. For all share-isolating gadgets are PINI, the trivial implementations of linear functions are directly PINI. Without the need for refresh gadgets, the composition of PINI gadgets maintains the PINI property. Finally, any t-PINI gadget is t-probing secure.

Definition 1. (t-PINI [CS20]) In the circuit G of d-share, P is a set of t_1 probes inside the circuit, and A is a set of t_2 share indexes. G is t-PINI if and only if for all A and P with $t_1 + t_2 \leq t$, there is a share index set B with $|B| \leq t_1$ so that the P probe set and outputs whose share index in A can be simulated by the corresponding input probes whose share indexes in $A \cup B$.

Algorithm 2 $SecA2B_k^n$ based on Carry-Propagation adders [BC22]

```
Input: n shares arithmetic sharing $\boldsymbol{x}^{\boldsymbol{A_{2^k}}}$, such that $x \in \mathbb{F}_{2^k}$.
Output: n shares Boolean sharing $\boldsymbol{z}^{\boldsymbol{B,k}}$, such that $z = x$.
1: if $n = 1$ then
2:     $\boldsymbol{z}^{\boldsymbol{B,k}} \leftarrow \boldsymbol{x}^{\boldsymbol{A_{2^k}}}$
3: else
4:     $\boldsymbol{y}^{\boldsymbol{B,k}} \leftarrow SecA2B_k^{\lfloor n/2 \rfloor}((x_1^{A_{2^k}}, ..., x_{\lfloor n/2 \rfloor}^{A_{2^k}}))$
5:     $\boldsymbol{y'}^{\boldsymbol{B,k}} \leftarrow SecA2B_k^{n-\lfloor n/2 \rfloor}((x_{\lfloor n/2 \rfloor+1}^{A_{2^k}}, ..., x_n^{A_{2^k}}))$
6:     $\boldsymbol{s}^{\boldsymbol{B,k}} \leftarrow (y_1^{B,k}, y_2^{B,k}, ..., y_{\lfloor n/2 \rfloor}^{B,k}, 0, ..., 0)$
7:     $\boldsymbol{s'}^{\boldsymbol{B,k}} \leftarrow (0, ..., 0, y_1'^{B,k}, ..., y_{\lceil n/2 \rceil}'^{B,k})$
8:     $\boldsymbol{z}^{\boldsymbol{B,k}} \leftarrow SecAdd_k^n(\boldsymbol{s}^{\boldsymbol{B,k}}, \boldsymbol{s'}^{\boldsymbol{B,k}})$        ▷ Algorithm 3 or Algorithm 4.
```

When considering hardware-oriented masking implementations, physical defeats are an unavoidable threat, and formal security proof is necessary. Many mainstream masking schemes have shown local or composability flaws for a lack of arbitrary-order proof in the robust probing model [MMSS19].

The compositional strategy in the glitch-robust probing model is formalized and proved in [CGLS20a]. Hardware Private Circuits (HPCs) were proposed for glitch-robust PINI. The HPC1 and HPC2 in [CGLS20a] are arbitrary-order secure multiplication gadgets in hardware. We employ the HPC2 gadget taking two cycles of latency and optimized randomness demand, which is evolved from the PINI1 multiplication by adding registers. Similarly, our $SecAnd_k^n$ parallelizes k HPC2 gadgets by considering the registers in Algorithm 1 and takes two clock cycles.

Subsequently, Cassiers and Standaert [CS21] studied the security under the transition-robust and glitch-robust probing model, considering the possible leakage caused by the transition of registered values and the glitches existing on the combinational circuit. They prove that if the adjacent executions of the HPC2 are parallelized, it is still safe under the model combining glitches and transitions. Therefore, the $SecAnd_k^n$ constitutes the basic building block of our A2B conversion under the glitch+transition-robust probing model.

2.3 High-Order Boolean Circuit-Based A2B Conversions

Assume n arithmetic masks for sensitive variable x in $\mathbb{F}_{2^k}$ are known, such that $\sum_{i=1}^n x_i^{A_{2^k}} = x$. On the premise that the t^{th}-order attack does not reveal x, the goal of the A2B conversion algorithm is to obtain n Boolean masks of x as $\bigoplus_{i=1}^{n} x_i^{B,k} = x$.

The basic idea of high-order Boolean circuit-based A2B conversions [CGV14] is to re-share each arithmetic share $x_i^{A,k}$ into Boolean shares and then use the addition on Boolean shares to obtain the Boolean masking representation of the arithmetic shares sum (that is x). Here we denote the i^{th} Boolean share of the j^{th} arithmetic share by $(x_j^{A_{2^k}})_i^{B,k}$ ($1 \leq i \leq n, 1 \leq j \leq n$), and n shares in all need to be converted. The original A2B scheme directly transforms the addition circuit into a circuit of XOR and AND gates, applying the ISW multiplication technique [ISW03] to ensure the side-channel security under the ISW t-Probing framework. For t^{th}-order security, the ISW framework requires $n > 2t$.

A straightforward way to convert masking is adding n arithmetic shares sequentially

Algorithm 3 $SecRCA_k^n$ [BC22]

Input: n shares Boolean sharing $\boldsymbol{x^{B,k}}, \boldsymbol{y^{B,k}} \in \mathbb{F}_{2^k}$.
Output: n shares Boolean sharing $\boldsymbol{z^{B,k}} \in \mathbb{F}_{2^k}$, such that $z = x + y \bmod 2^k$.

```
1: $\boldsymbol{c^{B,1}} \leftarrow (0, 0, ..., 0)$
2: for $j = 0$ to $k - 2$ do
3:     $\boldsymbol{a^{B,1} \leftarrow x^{B,k}[j] \oplus y^{B,k}[j]}$
4:     $\boldsymbol{z^{B,k}}[j] \leftarrow \boldsymbol{c^{B,1}} \oplus \boldsymbol{a^{B,1}}$
5:     $\boldsymbol{c^{B,1}} \leftarrow \boldsymbol{x^{B,k}}[j] \oplus SecAnd_1^n(\boldsymbol{a^{B,1}}, \boldsymbol{x^{B,k}}[j] \oplus \boldsymbol{c^{B,1}})$
6: $\boldsymbol{z^{B,k}[k-1] \leftarrow x^{B,k}[k-1] \oplus y^{B,k}[k-1] \oplus c^{B,1}}$
```

one by one:

$$\bigoplus_{i=1}^{n} x_i^{B,k} = ((x_1^{A_{2^k}})_1^{B,k} \oplus ... \oplus (x_1^{A_{2^k}})_n^{B,k}) + ... + (x_n^{A_{2^k}})_1^{B,k} \oplus ... \oplus (x_n^{A_{2^k}})_n^{B,k}). \tag{3}$$

In this way, secure addition on n Boolean shares is the underlying element with complexity $\mathcal{O}(n^2 \cdot k)$, and n calls to secure addition result in complexity $\mathcal{O}(n^3 \cdot k)$. In the improved version [CGV14], the algorithm with complexity $\mathcal{O}(n^2 \cdot k)$ divides n arithmetic shares into two parts: the first $\lfloor n/2 \rfloor$ and the last $\lceil n/2 \rceil$. Applying this conversion recursively results in

$$\begin{aligned}
\bigoplus_{i=1}^{n} x_i^{B,k} &= (x_1^{A_{2^k}} + ... + x_{\lfloor n/2 \rfloor}^{A_{2^k}}) + (x_{\lfloor n/2 \rfloor+1}^{A_{2^k}} + ... + x_n^{A_{2^k}}) \\
&= (x_1^{B,k} \oplus ... \oplus x_{\lfloor n/2 \rfloor}^{B,k}) + (y_1^{B,k} \oplus ... \oplus y_{\lceil n/2 \rceil}^{B,k}) \\
&= x_1'^{B,k} \oplus ... \oplus x_n'^{B,k} + y_1'^{B,k} \oplus ... \oplus y_n'^{B,k}.
\end{aligned} \tag{4}$$

Boolean shares $x_i^{B,k}$ $(1 \le i \le \lfloor n/2 \rfloor)$ and $y_i^{B,k}$ $(1 \le i \le \lceil n/2 \rceil)$ are converted from the sum of the first $\lfloor n/2 \rfloor$ arithmetic shares and the sum of the last $\lceil n/2 \rceil$ arithmetic shares respectively. Then n-sharing $\boldsymbol{x'^{B,k}}$ and $\boldsymbol{y'^{B,k}}$ are obtained by refreshing $\boldsymbol{x^{B,k}}$ and $\boldsymbol{y^{B,k}}$. A tree structure can be established by recursive layer by layer as shown in Figure 1, where the number of additive terms is divided by two at each layer. For A2B conversion hardware implementation, n arithmetic input shares require the addition of $\lceil \log_2 n \rceil$ layers.

On the basis of Coron's recursive A2B conversion, Bronchain and Cassiers [BC22] introduced the provable PINI A2B conversion shown in Algorithm 2, which applies PINI multiplication to construct secure adder and removes the refresh gadget based on the PINI combinatorial theorem. The $SecAdd_k^n$ in [BC22] uses a chain of full adders with complexity $\mathcal{O}(k)$ depicted as $SecRCA_k^n$ in Algorithm 3.

Using the more efficient Kogge-Stone carry look-ahead algorithm instead of the ripple-carry adder in the original scheme, the secure adder complexity can be improved to $\mathcal{O}(n^2 \cdot \log k)$, leading to A2B conversion with complexity $\mathcal{O}(n^2 \cdot \log k)$.

3 An Improved High-Order Conversion Based on Carry-Save Adder

In this section, we first analyze and explain that carry propagation is the root cause of high hardware implementation delay. Then we introduce an improved A2B conversion algorithm, which further reduces hardware latency and improves software speed performance by using carry-free addition.

Algorithm 4 $SecKSA_k^n$, built from the *SecAdd* of [BBE+18] with PINI gadgets

```
Input: n shares Boolean sharing $\boldsymbol{x^{B,k}}, \boldsymbol{y^{B,k}} \in \mathbb{F}_{2^k}$.
Output: n shares Boolean sharing $\boldsymbol{z^{B,k}} \in \mathbb{F}_{2^k}$, such that $z = x + y \bmod 2^k$.
1: $\boldsymbol{p^{B,k}} \leftarrow \boldsymbol{x^{B,k}} \oplus \boldsymbol{y^{B,k}}$
2: $\boldsymbol{g^{B,k}} \leftarrow SecAnd_k^n(\boldsymbol{x^{B,k}}, \boldsymbol{y^{B,k}})$
3: for $j = 0$ to $\lceil log_2(k-1) \rceil - 2$ do
4:     $\boldsymbol{g^{B,k}} \leftarrow \boldsymbol{p^{B,k}} \oplus SecAnd_k^n(\boldsymbol{p^{B,k}}, \boldsymbol{g^{B,k}} \ll 2^j)$
5:     $\boldsymbol{p^{B,k}} \leftarrow SecAnd_k^n(\boldsymbol{p^{B,k}}, \boldsymbol{p^{B,k}} \ll 2^j)$
6: $\boldsymbol{g^{B,k}} \leftarrow \boldsymbol{p^{B,k}} \oplus SecAnd_k^n(\boldsymbol{p^{B,k}}, \boldsymbol{g^{B,k}} \ll 2^{\lceil log_2(k-1) \rceil - 1})$
7: $\boldsymbol{z^{B,k}} \leftarrow \boldsymbol{x^{B,k}} \oplus \boldsymbol{y^{B,k}} \oplus (\boldsymbol{g^{B,k}} \ll 1)$
```

3.1 High Latency Caused by Carry Propagation

The underlying module of the high-order A2B conversion algorithm is a side-channel secure adder that uses Boolean masked bit-level operations, which makes the algorithm complexity and circuit performance directly affected by the internal carry propagation in the adder. Coron et al. proposed the first Boolean circuit-based A2B conversion algorithm based on RCA [CGV14] and then described an improved A2B algorithm based on KSA with $\mathcal{O}(\log k)$ operations where k is the addition bit width [CGTV15]. We first recall the secure ripple-carry adder and Kogge-Stone carry look-ahead adder to illustrate the effect of carry propagation on the latency of a secure hardware implementation.

Given two k-bit addends x and y, let $z = x+y \bmod 2^k$. For RCA, the bitwise operation can be expressed recursively by the following equation,

$$z[i] = x[i] \oplus y[i] \oplus c[i]. \tag{5}$$

For $0 \leq i < k$,

$$\begin{cases} c[0] = 0 \\ c[i+1] = x[i] \oplus ((x[i] \oplus y[i]) \wedge (x[i] \oplus c[i])) \end{cases} \tag{6}$$

where $c[i+1]$ denotes the carry resulting from the i^{th} bit addition. The above equation calculates the sum value after $k-1$ iterations. For side-channel protection, the Algorithm 3 reviews $SecRCA_k^n$, a secure RCA implementation of k-bit width and n shares. The $SecAnd_1^n$ gadget is an HPC2 gadget and requires two cycles as mentioned in Subsection 2.2. The RCA is composed of a chain of k secure Full-Adders (FAs) and the carry input of the highest-bit FA is calculated after $(k-1)$-level AND gates. Therefore, the secure hardware implementation of RCA needs at least $2 \cdot (k-1)$ stage registers. However, introducing multilevel registers will cause the latency of the secure adder to increase linearly with the bit width of variables, resulting in a significant delay in the A2B conversion with a loss of performance and increased hardware complexity.

The Kogge-Stone carry look-ahead adder requires only $\mathcal{O}(\log k)$ iterations, as shown in Algorithm 4. In the preprocessing stage, k-bit carry generation g_0 and carry propagation p_0 are generated:

$$g_0 = x \wedge y, p_0 = x \oplus y. \tag{7}$$

During the processing stage, $\lceil \log_2(k-1) \rceil$ stages of calculations as follows are required:

$$g_{i+1} = g_i \oplus ((g_i \ll 2^i) \wedge p_i), p_{i+1} = p_i \wedge (p_i \ll 2^i), \tag{8}$$

for $0 \leq i < \lceil \log_2(k-1) \rceil$. In [BBE+18], Gilles Barthe et al. proposed an arbitrary-order Boolean masked KSA based on SNI multiplication and refresh gadget. On this basis, we adopt the $SecAnd_k^n$ in Algorithm 1 for PINI composability and therefore omit the refresh gadgets. The $SecKSA_k^n$ takes $\lceil \log_2(k-1) \rceil + 1$ stages of $SecAnd_k^n$ and requires $2(\lceil \log_2(k-1) \rceil + 1)$ stages of registers. Therefore, the clock cycle delay of the secure KSA is proportional to $\log k$.

Carry propagation in both of the above adder structures introduces a performance loss. Based on these two carry propagation adders, the structure of the A2B conversion circuit consistent with Coron's scheme described in Subsection 2.3 is shown in Figure 1. It can be seen that when n arithmetic shares are converted for the high-order masking, the $\lceil \log_2 n \rceil$-layer addition serial structure in the A2B conversion algorithm will further amplify the delay problem of secure addition.

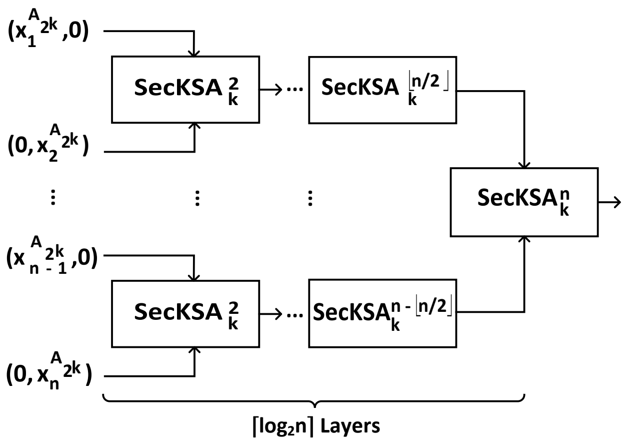


Figure 1: High-order A2B conversion based on Carry-Propagation adder.

3.2 An Improved High-Order A2B Conversion Based on CSA

The improvement from RCA to KSA reduces the complexity of the A2B algorithm from k to $\log k$, and the latency of the hardware implementation is also greatly improved. The main reason for the improved performance is the shortening of the carry chain. Under the glitch-extended model, each level of carry calculation corresponds to a layer of nonlinear functions, resulting in inserting two levels of registers to assure security. We thus consider adopting a redundant number representation with a carry-propagation-free addition to avoid the performance loss from computing carry of intermediate results.

The most commonly used redundancy representations are Carry-Save (C-S) and Redundant Signed Digit (RSD) representations. A number N is expressed as the sum of two numbers N_s and N_c in the C-S representation and is expressed as the difference between two numbers N^+ and N^- in the RSD representation. The C-S representation is more suitable for unsigned Boolean shares in $\mathbb{F}_{2^k}$. Thus, we choose to use the carry-save adder.

The k-bit CSA can be regarded as a 3:2 compressor with three k-bit operands x, y, and c_{in} as inputs and two k-bit operands c and s as outputs. c and s are C-S representations of the sum of three input operands. The CSA consists of k FAs. FA is also the basic unit of RCA. An FA output carry $c[i]$ is the input $c_{in}[i+1]$ of the next FA in RCA, forming a serial structure of k levels. In CSA, k FAs are parallel, and the carriers are preserved,

$$s[i] = x[i] \oplus y[i] \oplus c_{in}[i] \tag{9}$$

$$c[i] = x[i] \oplus ((x[i] \oplus y[i]) \wedge (x[i] \oplus c_{in}[i])). \tag{10}$$

In the modulo 2^k, c shifts left for one bit, and the carry generated by the highest bit is discarded as in Figure 2. According to the above Equation 10, there is only a nonlinear

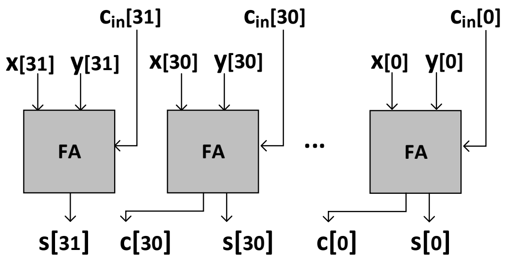


Figure 2: 32-bit Carry-Save adder.

Algorithm 5 $SecCSA_k^n$

```
Input: n shares Boolean sharing $\boldsymbol{x}^{B,k}, \boldsymbol{y}^{B,k}, \boldsymbol{c_{in}}^{B,k} \in \mathbb{F}_{2^k}$.
Output: n shares Boolean sharing $\boldsymbol{s}^{B,k}, \boldsymbol{c}^{B,k}$, such that $s + c = x + y + c_{in} \mod 2^k$.
1: $\boldsymbol{a}^{B,k} \leftarrow \boldsymbol{x}^{B,k} \oplus \boldsymbol{y}^{B,k}$
2: $\boldsymbol{s}^{B,k} \leftarrow \boldsymbol{c_{in}}^{B,k} \oplus \boldsymbol{a}^{B,k}$
3: $\boldsymbol{c}^{B,k} \leftarrow (\boldsymbol{x}^{B,k} \oplus SecAnd_k^n(\boldsymbol{a}^{B,k}, \boldsymbol{x}^{B,k} \oplus \boldsymbol{c_{in}}^{B,k})) \ll 1$
```

function in FA with a degree of only two. For k-bit and n-sharing CSA, the masked implementation is depicted in the Algorithm 5. The $SecCSA_k^n$ consumes two clock cycles for only one level of $SecAnd_k^n$ gadget.

Algorithm 6 $SecA2B\_CS_k^n$: secure conversion from arithmetic masking to Boolean masking of the C-S representation

```
Input: n shares arithmetic sharing $\boldsymbol{x}^{\boldsymbol{A}_{2^k}}$, such that $x = \sum_{i=1}^{n} x_i^{A_{2^k}} \in \mathbb{F}_{2^k}$ and $n > 2$.
Output: n shares Boolean sharing $\boldsymbol{s}^{B,k}, \boldsymbol{c}^{B,k}$, such that $x = \bigoplus_{i=1}^{n} s_i^{B,k} + \bigoplus_{i=1}^{n} c_i^{B,k}$.
1: if $n = 3$ then
2:     $(\boldsymbol{y_1})^{B,k} \leftarrow (x_1^{A_{2^k}}, 0, 0)$
3:     $(\boldsymbol{y_2})^{B,k} \leftarrow (0, x_2^{A_{2^k}}, 0)$
4:     $(\boldsymbol{y_3})^{B,k} \leftarrow (0, 0, x_3^{A_{2^k}})$
5:     $(\boldsymbol{s}^{B,k}, \boldsymbol{c}^{B,k}) \leftarrow SecCSA_k^3((\boldsymbol{y_1})^{B,k}, (\boldsymbol{y_2})^{B,k}, (\boldsymbol{y_3})^{B,k})$
6: else
7:     $(\boldsymbol{s}^{B,k}, \boldsymbol{c}^{B,k}) \leftarrow SecA2B\_CS_k^{n-1}((x_1^{A_{2^k}}, x_2^{A_{2^k}}, ...x_{n-1}^{A_{2^k}}))$
8:     $(\boldsymbol{y_1})^{B,k} \leftarrow (s_1^{B,k}, .., s_{n-1}^{B,k}, 0)$
9:     $(\boldsymbol{y_2})^{B,k} \leftarrow (c_1^{B,k}, .., c_{n-1}^{B,k}, 0)$
10:    $(\boldsymbol{y_3})^{B,k} \leftarrow (0, ..., 0, x_n^{A_{2^k}})$
11:    $(\boldsymbol{s}^{B,k}, \boldsymbol{c}^{B,k}) \leftarrow SecCSA_k^n((\boldsymbol{y_1})^{B,k}, (\boldsymbol{y_2})^{B,k}, (\boldsymbol{y_3})^{B,k})$
```

By applying secure carry-free adders based on redundant number representation, the improved higher-order A2B is depicted in Algorithm 7 and Figure 5. The input is n arithmetic shares of the k-bit variable $\boldsymbol{x}^{\boldsymbol{A}_{2^k}}$, where $\sum_{i=1}^{n} x_i^{A_{2^k}} = x$. In the new conversion algorithm $SecA2B_k^n$, the $SecA2B\_CS_k^n$ first converts $\boldsymbol{x}^{\boldsymbol{A}_{2^k}}$ into $\boldsymbol{s}^{B,k}$ and $\boldsymbol{c}^{B,k}$, which is the n-share redundant number representation of x. The $SecA2B\_CS_k^n$ shown in Algorithm 6 follows the basic framework of Algorithm 2 [BC22]. Namely, the secure CSA

Algorithm 7 $SecA2B_k^n$ based on Carry-Save adders

Input: n shares arithmetic sharing $\boldsymbol{x^{A_{2^k}}}$, such that $x = \sum_{i=1}^{n} x_i^{A_{2^k}}$ and $x \in \mathbb{F}_{2^k}$.

Output: n shares Boolean sharing $\boldsymbol{z^{B,k}}$, such that $z = x = \bigoplus_{i=1}^{n} z_i^{B,k}$.

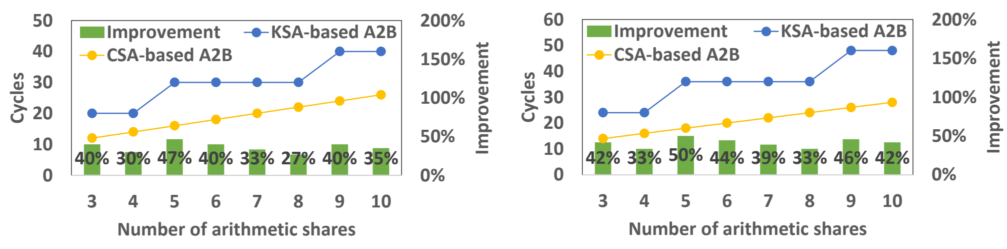
```
1: if $n = 1$ then
2:     $\boldsymbol{z^{B,k}} \leftarrow \boldsymbol{x^{A_{2^k}}}$
3: else
4:     if $n = 2$ then
5:         $\boldsymbol{s^{B,k}} \leftarrow (x_1^{A_{2^k}}, 0)$
6:         $\boldsymbol{c^{B,k}} \leftarrow (0, x_2^{A_{2^k}})$
7:     else
8:         $(\boldsymbol{s^{B,k}}, \boldsymbol{c^{B,k}}) \leftarrow SecA2B\_CS_k^n((x_1^{A_{2^k}}, x_2^{A_{2^k}}, ...x_n^{A_{2^k}}))$
9:     $\boldsymbol{z^{B,k}} \leftarrow SecKSA_k^n(\boldsymbol{s^{B,k}}, \boldsymbol{c^{B,k}})$
```

of three Boolean shares is taken at the first layer. Then, the Boolean shares are expanded layer by layer until it is equal to the input arithmetic share. The last level of CSA output is the C-S representation of x, and both c and s are represented as n Boolean shares. In order to convert to a normal binary Boolean masking, carry-propagation addition only needs to be done once, and the $SecKSA_k^n$ is selected here.

The expand method also follows the [BC22], adding zeros before or after the Boolean shares output by the previous layer to form a gadget embedding structure in Definition 2 and Figure 4. As for the choice of padding zeros in front or behind the Boolean share, follow the principle of mapping arithmetic shares to Boolean shares with the same index. This ensures that both linear operations and PINI gadgets cause variables to propagate only in isolated share index domains.

The advantage of the improved architecture in terms of latency in hardware implementation is evident. The A2B conversion based on a carry-propagation adder needs $\lceil \log_2 n \rceil$-layer additions for n shares. As mentioned above, the $SecRCA_k^n$ takes at least $2(k-1)$ cycles. Shown in Algorithm 2, the PINI $SecA2B_k^n$ based on $SecRCA_k^n$ needs $2(k-1)\lceil \log_2 n \rceil$ cycles. In later improved versions, $SecRCA_k^n$ in $SecA2B_k^n$ is replaced by $SecKSA_k^n$. The $SecKSA_k^n$ takes $2(\lceil \log_2(k-1) \rceil + 1)$ cycles, and on the basis $SecA2B_k^n$ needs $2(\lceil \log_2(k-1) \rceil + 1)\lceil \log_2 n \rceil$ clock cycles. The latency of the masked implementation of the CSA is consistent with an FA and requires two clock cycles since there is only one layer of nonlinear function. For the $SecA2B_k^n$ based on CSA, a total of $n-2$ levels of CSA are required, and the total latency in terms of clock cycles is $2(\lceil \log_2(k-1) \rceil + n - 1)$. Generally, the larger the variable bit width, the better the improvement effect of the CSA-based A2B conversion.

For the common high-order case with $2 < n \leq 10$ and $9 < k \leq 33$, our estimated latency results are shown in Figure 3. It can be seen that the best improvement occurs in five arithmetic shares with $17 < k \leq 33$, and the latency is reduced by 50%. At the same time, the carry calculation not only increases the additional register stage in the hardware implementation but also increases the number of instructions in the software implementation. Therefore, CSA-based A2B conversion will also improve software performance, which we will evaluate later through software implementation.



(a) $9 < k \leq 17$. (b) $17 < k \leq 33$.

Figure 3: Latency comparison of A2B conversion.

4 Security Analysis of the Proposed A2B Conversion

This section first theoretically proves the formal security of the new $SecA2B_k^n$ based on CSA under the transition-robust and glitch-robust probing model. Then, we further analyze the safety of second-order A2B conversion through side-channel evaluation experiments.

4.1 Theoretical Security Proof

We first prove that the algorithm $SecA2B_k^n$ is PINI under the ideal probing model. Then it is explained that the PINI gadget evolves into an HPC gadget by adding registers and $SecA2B_k^n$ is guaranteed to be PINI under the probing model considering glitches and transitions in the hardware.

Lemma 1. *(PINI composability [CS20]). Any composite gadget made of t-PINI composing gadgets is t-PINI.*

Lemma 2. *Without considering physical defeats and registers* Reg[.] *marked in gray in Algorithm 1, the $SecKSA_k^n$ in Algorithm 4 and $SecCSA_k^n$ in Algorithm 5 are t-PINI, for $t = n - 1$.*

Proof. The $SecAnd_k^n$ gadget shown in Algorithm 1, which computes the bitwise AND of k-bit Boolean maskings, is the parallelization of k PINI1 gadgets. The PINI1 gadget is PINI multiplication gadget for $\mathbb{F}_2$ in [CS20], therefore according to the composability rule in Lemma 1 the $SecAnd_k^n$ gadget is $(n-1)$-PINI.

In accordance with [CGLS20b] Proposition 4, the trivial masking implementation of a linear function is shares-isolating and PINI. The two gadgets $SecCSA_k^n$ and $SecKSA_k^n$ are the composition of PINI $SecAnd_k^n$ and linear gadget, therefore they are $(n-1)$-PINI. □

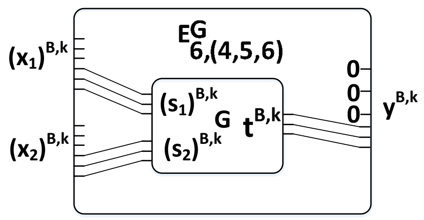


Figure 4: Example of 3-share to 6-share gadget embedding.

To illustrate the security of the algorithm $SecA2B_k^n$, we review here the gadget embedding technique proposed in [BC22] for combining PINI gadgets with various numbers of shares. The gadgets embedding maps the embedded gadget with a lower number to the embedding gadgets that use more shares. Unused input shares of the embedding gadget

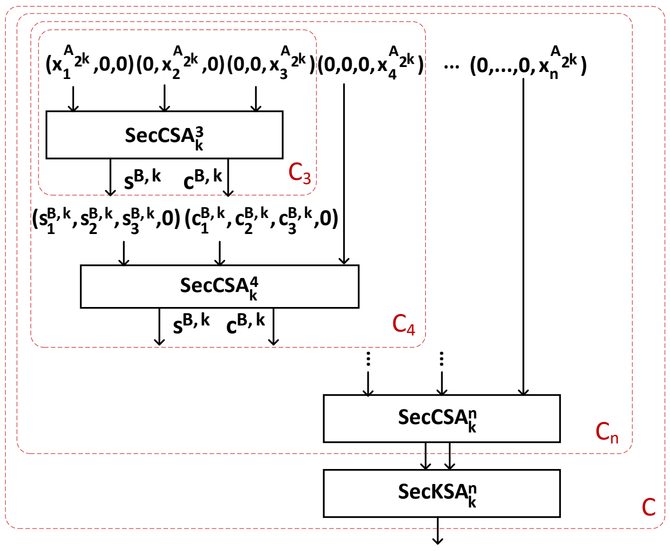


Figure 5: High-order A2B conversion based on Carry-Save adder.

are discarded, while output shares not generated from the embedded gadget are assessed to a value of zero. The example of 3-share to 6-share gadget embedding is depicted in Figure 4. For $E^G_{6,(4,5,6)}$, the embedded gadget G maps 3-share inputs and outputs into the last three shares of the embedding gadget, padding the first three shares with zero. Here we recall the PINI embedding as a lemma.

Definition 2. (Gadget embedding [BC22]) Let G be a n'-share gadget, and let mapping $m \in [1,n]^{n'}$ (with $n' \leq n$) have unique components ($m_i \neq m_j$ for all i,j). The embedding of the n'-share gadget G to n shares with mapping m denotes $E^G_{n,m}$.

Lemma 3. *(PINI embedding [BC22]). If G is $(n'-1)$-PINI gadget, its embedding $E^G_{n,m}$ is $(n-1)$-PINI for any shares n and mapping m.*

Theorem 1. *The $SecA2B\_CS_k^n$ in Algorithm 6 is t-PINI, for $t = n-1$.*

Proof. In the case of $n = 3$, the three arithmetic shares are expanded to $(x_1^{A_{2k}}, 0, 0)$, $(0, x_2^{A_{2k}}, 0)$ and $(0, 0, x_3^{A_{2k}})$ as circuit C_3 in Figure 5. Then the 3-sharing variables are input into $SecCSA_k^3$, which is a 2-PINI gadget in Lemma 2. According to the probe propagation rules of the PINI gadgets as in Definition 1, t_1 internal probes and t_2 output probes with $t_1 + t_2 \leq t$ can be simulated by input shares with up to t different index values. The share index value remains unchanged during the process of mapping the arithmetic shares to the Boolean shares. This share-isolating property similar to linear operation ensures that t_1 internal probes and t_2 output probes can be also simulated by input arithmetic shares with up to t index. Therefore, the $SecA2B\_CS_k^3$ is 2-PINI.

In the case $n > 3$, we prove security by induction on the number of shares n. The C_n circuit in Figure 5 represents $SecA2B\_CS_k^n$. Assuming that C_{n-1} is $(n-2)$-PINI , we prove that the C_n is also $(n-1)$-PINI. The condition is satisfied for the case of $n = 3$, and the C_3 is 2-PINI. The circuit C_n can be divided into three parts: embedded gate $E^{C_{n-1}}_{n,(1,\ldots,n-1)}$, expanded arithmetic share $(0,\ldots,0,x_n^{A_{2k}})$ and $SecCSA_k^n$. Since C_{n-1} is $(n-2)$-PINI, the embedding gadget $E^{C_{n-1}}_{n,(1,\ldots,n-1)}$ is $(n-1)$-PINI according to Lemma 3. The expansion of arithmetic share is share-isolating as described above, so it is PINI. Finally, since $SecCSA_k^n$ is $(n-1)-$PINI, $SecA2B\_CS_k^n$ a combination of PINI gadgets is $(n-1)$-PINI. The above proves that $SecA2B\_CS_k^n$ is PINI for arbitrary order with $n > 2$. □

Theorem 2. *The $SecA2B_k^n$ in Algorithm 7 is t-PINI, for $t = n - 1$.*

Proof. In the case n=1, this is a non-masked implementation and trivial to prove. In the case n=2, the arithmetic shares are expanded to $(x_1^{A_{2^k}}, 0)$ and $(0, x_2^{A_{2^k}})$ and input to the gadget $SecKSA_k^2$. The expansion is share-isolating, and the $SecKSA_k^2$ is 1-PINI as proved in Lemma 2. The $SecA2B_k^2$ is 1-PINI.

In the case $n > 2$, the gadget is composed of the $SecA2B\_CS_k^n$ and $SecKSA_k^n$ two parts. The $SecA2B\_CS_k^n$ is PINI by Theorem 1, and $SecKSA_k^n$ is PINI by Lemma 2. Therefore, Algorithm 7 is a composition of PINI gadgets and is PINI. □

When considered under the probing model with glitches and transitions, the registers Reg[.] marked in gray of Algorithm 1 are used in the hardware implementation. Here we prove the $SecA2B_k^n$ with registers still secure under this condition. Transitions in iterative circuits can weaken the security level [MKSM22], so the hardware implementations of the algorithms are fully pipelined and have no dependencies between successive inputs. For using the concept of glitch+transition-robust PINI in [CS21], we review the relevant theorems in it here.

Lemma 4. *(O-PINI share-isolating gadgets [CS21]). Share-isolating structural gadgets are iterated glitch+transition-robust O-PINI.*

Lemma 5. *(glitch+transition-robust PINI composition [CS21]). Let S be a structural gadget composition of iterated glitch+transition-robust t-O-PINI gadgets and glitch+transition-robust t-PINI gadgets. If every PINI gadget in S has no adjacent executions, then S is glitch+transition-robust t-PINI.*

Theorem 3. *The $SecA2B_k^n$ with registers in Algorithm 7 is glitch+transition-robust t-PINI, for $t = n - 1$.*

Proof. According to [CGLS20b] Section 4.4, the PINI1 multiplication is adjusted to HPC2 by adding the registers Reg[.] of Algorithm 1. The HPC2 multiplication gadget in fully pipelined circuit is glitch+transition-robust $(n-1)$-PINI with n shares [CS21]. According to the Lemma 5, the $SecAnd_k^n$ gadget parallel of HPC2 is glitch+transition-robust t-PINI.

The trivial parallel n-sharing implementation of the linear function is a share-isolating gadget, which is iterated glitch+transition-robust $(n-1)$-O-PINI according to Lemma 4. Thanks to Lemma 5, the $SecCSA_k^n$ and $SecKSA_k^n$ combining the parallel implementation for linear gadget and $SecAnd_k^n$ are glitch+transition-robust t-PINI.

The $SecA2B\_CS_k^n$ in case of $n = 3$ consists of arithmetic share expansion and $SecCSA_k^3$. Expansion is a share-isolating linear operation and is O-PINI. Furthermore, $SecCSA_k^3$ is glitch+transition-robust 2-PINI, so $SecA2B\_CS_k^3$ is glitch+transition-robust 2-PINI. For the other cases, circuit C_n in Figure 5 consists of embedding gates, arithmetic share expansion, and $SecCSA_k^n$. The embedding gadget mapping with share indexes unchanged by wires is share-isolating and O-PINI, so the Lemma 3 still holds under the probing model with glitches and transitions. Expansion is O-PINI, and $SecCSA_k^n$ is glitch+transition-robust t-PINI. Therefore, the recurrence relationship in proof of Theorem 2 still holds that assuming C_{n-1} is glitch+transition-robust $(n-2)$-PINI, the C_n is also glitch+transition-robust $(n-1)$-PINI. Consequently the $SecA2B\_CS_k^n$ is glitch+transition-robust $(n-1)$-PINI. The composition of $SecA2B\_CS_k^n$ and $SecKSA_k^n$, that is $SecA2B_k^n$, is glitch+transition-robust $(n-1)$-PINI. □

4.2 Experimental Leakage Evaluation

A widely used side-channel leakage evaluation method is Test Vector Leakage Assessment (TVLA) [GGJR+11], and we use the non-specific statistical t-test (also called fixed vs. random t-test) to evaluate our A2B implementations. The basic principle is to set two sets of inputs. One set is the shared input of random variables, and the other is the shared input of fixed variables. These two sets of variables are randomly interleaved and input into the design under test, and the corresponding two sets of power traces are recorded. Under the independent leakage assumption, the statistical distributions of these two sets of power traces should not be significantly different. Thus we assumed that the two sets had an indistinguishable statistical distribution and used the Welch t-statistic to calculate the credibility of this assumption:

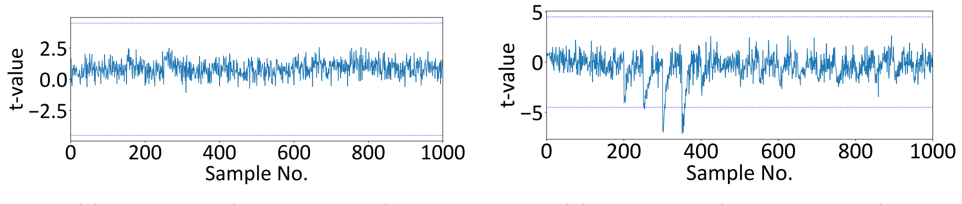
$$t = \frac{\mu_{fixed} - \mu_{random}}{\sqrt{\frac{\sigma^2_{fixed}}{N_{fixed}}} + \sqrt{\frac{\sigma^2_{random}}{N_{random}}}}, \tag{11}$$

where μ_{fixed}, σ_{fixed}, and N_{fixed} (resp. μ_{random}, σ_{random}, and N_{random}) represent the mean, variance, and number of measurements of the power traces of the fixed input group (resp. random input group). A widely adopted empirical value for $|t|$ is 4.5 [DAP+22] [MBFC23], and $|t| > 4.5$ means that the null hypothesis is rejected with 99.999% confidence. In Figure 6, Figure 7, and Figure 8, ± 4.5 is represented by dotted lines.

The Equation 11 is applied directly to each sample point on the power traces for the first-order t-test. For high-order univariate tests, Equation 11 is also used to calculate the t value after power trace preprocessing [SM15]. The mean-free squared traces are calculated for the second-order univariate t-test. To conduct the multivariate second-order t-test, we need to perform individual t-tests for every pair of sample points by multiplying their respective mean-free power values. We conduct TVLA on second-order CSA-based A2B hardware and software implementations.

1) Hardware Implementation: The experimental platform used for side channel evaluation is the SAKURA-G development board, which mainly includes two Xilinx Spartan-6 FPGAs. The design under test is instantiated on the target FPGA (XC6SLX75-2CSG484C), and the control FPGA (XC6SLX9-2CSG225C) is employed for communication with the host and randomness generation. The development board operates at 2MHz and is powered by KEYSIGHT B2961A with a 5V supply voltage. To avoid the noise caused by randomness generation, we pre-generate random numbers and store them in BRAM before testing. The voltage drop across a 1 Ω shunt resistor, which is amplified by the AD8000YRDZ amplifier on the target FPGA, is used for measuring the design's power consumption. The Teledyne LeCroy WaveRunner 8404M digital oscilloscope is used to sample the instantaneous power consumption of the design under test at a sampling frequency of 100MHz.

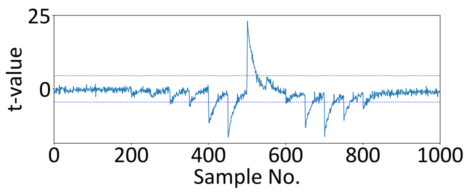
Under the above experimental environment and evaluation method, we conduct a side-channel security evaluation of the CSA-based A2B conversion hardware implementation.



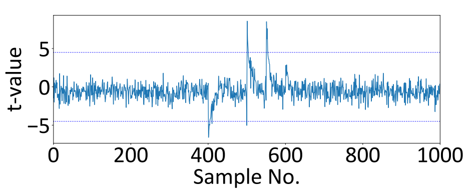
(a) PRNG ON (1st-order t-test). (b) PRNG ON (2nd-order t-test).

Figure 6: t-test results of 1st-order KSA hardware implementation.

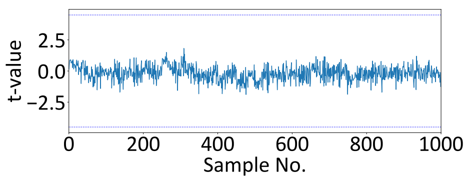
First, to illustrate that the platform can detect second-order leaks, we collect ten million power traces for first-order A2B conversion and conduct first-order and second-order t-test. The experimental results shown in Figure 6 are consistent with the theory, with no first-order leakage in Figure 6(a) but second-order leakage in Figure 6(b).



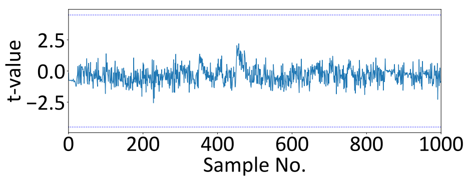
(a) PRNG OFF (1st-order t-test).



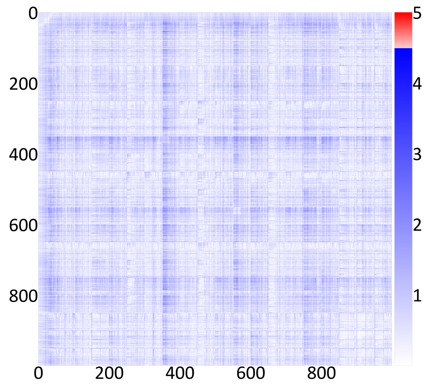
(b) PRNG OFF (2nd-order t-test).



(c) PRNG ON (1st-order t-test).



(d) PRNG ON (2nd-order t-test).



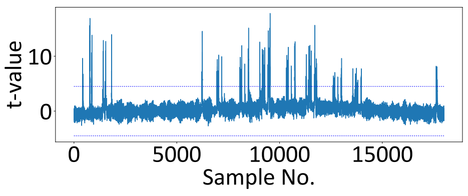
(e) PRNG ON (2nd-order bivariate t-test).

Figure 7: t-test results of 2nd-order CSA-based A2B conversion hardware implementation.

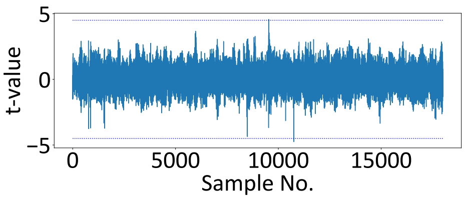
For the second-order CSA-based A2B, 100,000 power traces were collected with the PRNG turned off for comparison. Figure 7(a) and Figure 7(b) illustrate that this implementation has first-order and second-order leakage without PRNG masking. The PRNG is then opened, and 30 million power traces are collected. The results are shown in Figure 7(c) and Figure 7(d). The t-values of the first-order t-test and the second-order t-test are within ± 4.5, indicating no second-order univariate leakage. For the bivariate t-test, we calculated one million energy traces. Points in Figure 7(e) are red if their absolute t-value is greater than 4.5, and blue otherwise. It can be seen that there is no multivariate leakage.

2) Software Implementation: The software evaluation platform is CW308 UFO board with STM32F415 target board. The power consumption of the design under test is obtained by measuring the voltage drop on the 12 Ω shunt resistor on the board. The operating frequency of the target board is 8MHz, and the oscilloscope used to collect power traces is a Teledyne LeCroy WaveRunner 8404M with a sampling rate set to 25MHz.

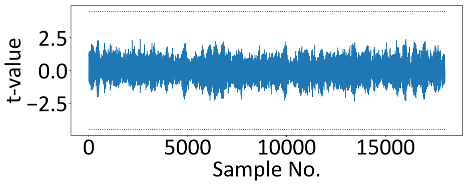
We performed a fixed vs. random TVLA for software implementation of second-order CSA-based A2B conversion. We first turned off the PRNG and collected 200,000 energy traces. The first-order and second-order t-test results are shown in Figure 8(a) and Figure 8(b), and there is noticeable energy leakage. The PRNG is then turned on, and the t-test results for ten million power traces are shown in Figure 8(c) and Figure 8(d), with no second-order univariate leakage. Each power trace contains 18,000 sample points, and the total number of bivariate t-tests between all points amounts to 324 million for one traces. The computations required for such tests are time-consuming even with 32 cores. The maximum distance of multivariate analysis is set at 3,000 points since we continuously introduce fresh randomness through PINI multiplications [ZSS+21]. The results of the one million bivariate t-test analysis are shown in Figure 8(e), and no leakage was observed.



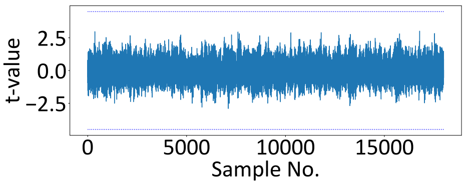
(a) PRNG OFF (1st-order t-test).



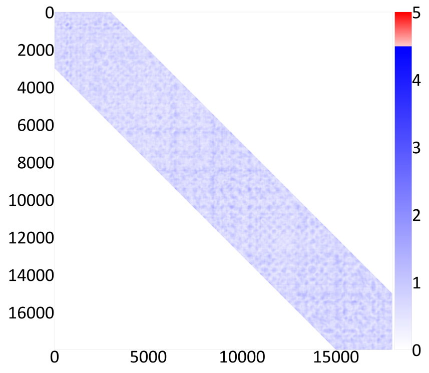
(b) PRNG OFF (2nd-order t-test).



(c) PRNG ON (1st-order t-test).



(d) PRNG ON (2nd-order t-test).



(e) PRNG ON (2nd-order bivariate t-test).

Figure 8: t-test results of 2nd-order CSA-based A2B conversion software implementation.

5 Implementation Results and Extensions

In this section, we will analyze the performance parameters of CSA-based A2B designs and compare them with other state-of-the-art solutions. Subsequently, we will explore A2B conversion based on CSA to expand the applications of prime modulus and B2A.

5.1 Performance Evaluation

In this paper, we have evaluated and compared the hardware and software performance of our new A2B implementation based on CSA with bit width $k = 32$, which is chosen for supporting A2B conversion for both Saber and Kyber in [FBR+22] and is the common width of ARX cipher implementations like SHA-256 [Lan19] or ChaCha [B+08].

1) Hardware Implementation: We performed performance evaluations under ASIC standard cell technology and FPGA for the hardware implementation. On the one hand, our design was evaluated at the TSMC 28nm (ssg0p81v125c) process node with ss corner and 0.9v power supply by the Design Compiler, providing the performance under the worst timing. We use the *set_dont_touch* during synthesis to prevent optimizations that might compromise the security requirements. Table 1 compares the performance of CSA-based A2B conversion with other state-of-the-art designs in terms of clock cycle latency, maximum frequency, time delay, area, and randomness consumption. To the best of our knowledge, no literature gives detailed hardware performance data for individual A2B conversion. For a fair comparison, we implement the RCA-based A2B conversion in [BC22] using our glitch+transition-robust PINI $SecAnd_k^n$ gadget as shown in Algorithm 2 and Algorithm 3. Also referring to [BBE+18], we adopt $SecKSA_k^n$ in Algorithm 4, and use the high-order implementation of A2B based on KSA in [CGTV15] as our comparison. All of the above hardware implementations are available at `https://github.com/ybhphoenix/A_Low_latency_A2B`.

The results show that under the ASIC application, our A2B conversion scheme improves latency, area, and time delay. Compared with the second-order RCA-based A2B [BC22] and KSA-based A2B [CGTV15], the clock cycle latency required for one conversion is reduced by 89% and 42% respectively. The reduction in the maximum clock frequency is within 3%, which results in a corresponding reduction in the time delay required for one conversion by 88% and 41%. Our approach costs more randomness than the RCA-based approach, primarily because of the last level of $SecKSA_k^n$ in Figure 5, but it uses 17% less randomness than the KSA-base approach. The area of full pipelined second-order A2B conversion is reduced by 20% and 25% under the TSMC 28nm process node.

Table 1: Resource utilization and performance for 32-bit conversions at TSMC 28nm.

Security order	Design	Latency (CLK)	Max. Freq. (GHz)	Delay (ns)	Area (kGE)	Randomness (bits)
2	[BC22]	124	5.02	24.70	197.20	124
	[CGTV15]	24	4.90	4.90	208.57	1,280
	This work	14	4.87	2.87	157.42	1,056
3	[BC22]	124	4.44	27.93	294.85	248
	[CGTV15]	24	4.31	5.57	371.34	2,560
	This work	16	4.31	3.25	287.34	2,208
4	[BC22]	186	3.95	47.09	496.76	465
	[CGTV15]	36	3.81	9.49	641.59	4,800
	This work	18	3.84	4.69	456.32	3,808

For the maximum frequency, the CSA-based A2B conversion is slightly dropped within 3%. But it should be emphasized that these designs are inserted with multi-level registers due to security requirements, making the frequency close to 5 GHz which far exceeds the frequency of masked implementations for most cryptographic algorithms [FBR+22][LMRBG23]. Therefore, the improved A2B as a masking conversion module does not limit the maximum frequency of the cryptographic algorithm, making latency in terms of clock cycles a much-needed aspect of improvement.

On the other hand, the resource utilization and performance of our design are evaluated on Artix-7 FPGA (XC7A200TFFG1156-3). Table 2 compares the designs in terms of latency, maximum frequency, time delay, resource utilization, and randomness. Among these, the clock cycle latency and randomness consumption determined during design align with ASIC performance. The evaluation results show that the improved CSA-based second-order A2B conversion also greatly improves delay on the FPGA as well. The second-order conversion using CSA has 84% and 45% smaller delay than RCA-based and KSA-based implementations, respectively. Our second-order CSA-based solution also results in the smallest number of Slices used among all three solutions. The maximum

Table 2: Resource utilization and performance for 32-bit conversions on Artix-7.

Security order	Design	Latency (CLK)	Max. Freq. (MHz)	Delay (ns)	LUTs	FFs	Slices	Randomness (bits)
2	[BC22]	124	512.82	241.80	2,234	20,423	4,352	124
	[CGTV15]	24	350.87	68.40	13,064	17,952	4,450	1,280
	This work	14	370.37	37.80	11,196	14,550	3,715	1,056
3	[BC22]	124	444.44	279.00	3,818	30,555	6,753	248
	[CGTV15]	24	298.50	80.40	25,838	32,099	8,225	2,560
	This work	16	300.30	53.28	22,189	26,093	7,275	2,208
4	[BC22]	186	370.37	502.20	6,642	51,567	10,852	465
	[CGTV15]	36	232.55	154.81	47,000	56,308	14,895	4,800
	This work	18	219.78	81.90	36,885	41,331	10,879	3,808

frequency on the FPGA of the improved A2B conversion is lower than that of RCA-based conversion. But similar to the ASIC implementation, the maximum frequency of the A2B conversion far exceeds that of many first-order masked cipher algorithm implementations [AMD+21][KNAH22].

2) Software Implementation: In order to compare under the same platform and compilation options, we also implemented Algorithm 2 based on $SecRCA_k^n$ [BC22] and KSA-based A2B conversion [CGTV15] using our PINI $SecAnd_k^n$ gadget. All code implementations used for evaluation are available at https://github.com/ybhphoenix/A_Low_latency_A2B. At the same time, we also added table-based A2B conversion commonly used in software implementation for comparison, and [D'A22] has benchmarked the algorithms on an Intel(R) Core(TM) i5-6500 @ 3.20GHz CPU. For comparison, we conducted performance evaluations on the Intel(R) Core(TM) i5-1135G7 @ 2.40GHz CPU. To ensure security the compilation option -O0 is consistent with the TVLA experiment. For 32-bit variables conversion, the results are shown in Table 3. The number of clock cycles of A2B conversion based on CSA is about 30% less than that based on KSA. For the RCA-based A2B conversion in [BC22], we consider individual non-bitsliced A2B implementations, which can bitsliced acceleration in the masked implementation of Kyber to obtain 18.8 times faster than KSA-based conversion. Compared with table-based A2B, our CSA-based A2B is faster than [CGMZ22] at the same compilation optimization level. The third-order case of the number of clock cycles for our A2B is similar to [D'A22], and the second-order case is faster than [D'A22].

Table 3: Software performance of different 32-bit A2B conversions in cycles.

Design	Compilation option	Security order 2	3	4
Bool. circ. [BC22]*	-O0	13,124	23,049	42,212
Bool. circ. [CGTV15]*	-O0	3,914	6,944	12,788
Bool. circ. **(This work)***	-O0	2,725	4,950	8,593
Bool. circ. **(This work)***	-O2	986	1,895	2,947
Table-based [CGMZ22]†	-O2	52,779	105,613	-
Table-based [D'A22]†	-O2	1,337	1,814	-

*Evaluated on the Intel(R) Core(TM) i5-1135G7 @ 2.40GHz CPU.
†Data from [D'A22] evaluated on Intel(R) Core(TM) i5-6500 @ 3.20GHz CPU.

Analyzing the 32-bit conversions for second-order to fourth-order security, the A2B conversion based on CSA improves performance in many aspects in both hardware and software implementation. Since CSA is a 3:2 compressed adder applicable to the case of $n > 2$, the first-order in Algorithm 7 is consistent with that based on KSA. For other

Algorithm 8 $SecAddModp_k^n$

```
Input: $n$ shares Boolean sharing $\boldsymbol{x}^{B,k}, \boldsymbol{y}^{B,k}$, integer $p < 2^k$ and $x, y \in [0, p)$.
Output: $n$ shares Boolean sharing $\boldsymbol{z}^{B,k}$, such that $z = x + y \bmod p$.
1: $\boldsymbol{p}^{B,k+1} \leftarrow (2^k - p, 0, ..., 0)$
2: $(\boldsymbol{s}^{B,k+1}, \boldsymbol{c}^{B,k+1}) \leftarrow SecCSA_{k+1}^n(\boldsymbol{p}^{B,k+1}, \boldsymbol{x}^{B,k}, \boldsymbol{y}^{B,k})$
3: $\boldsymbol{u}^{B,k+1} \leftarrow SecKSA_{k+1}^n(\boldsymbol{s}^{B,k+1}, \boldsymbol{c}^{B,k+1})$
4: $\boldsymbol{b}^{B,1} \leftarrow \neg\boldsymbol{u}^{B,k+1}[k]$
5: $\boldsymbol{a}^{B,k} \leftarrow BitCopyMask_k^n(\boldsymbol{b}^{B,1}, p)$        ▷ Use algorithm 1 of [BC22].
6: $\boldsymbol{z}^{B,k} \leftarrow SecKSA_k^n(\boldsymbol{a}^{B,k}, \boldsymbol{u}^{B,k+1})$
```

cases referring to the theoretical estimation in Subsection 3.2, CSA-based high-order A2B conversion can exhibit better performance for larger bit width, and the improvement in latency is at least 27% under the parameters of $2 < n \le 10$ and $9 < k \le 33$.

5.2 Expansion Applications

The above A2B conversion based on the 2^k modulus can be extended to the application of the prime modulus and B2A conversion.

Prime modulus conversion. A simple A2B conversion based on prime modulus can replace the adder of $SecA2B_k^n$ in Algorithm 2 with a prime modulus adder. In order to calculate $z = x + y \mod p$, the prime modulus adder in [BC22] calls secure adder three times sequentially to compute $s' = x + y + (2^k - p) \mod 2^{k+1}$ and $z = s' + p \cdot \neg MSB(s') \mod 2^k$. If $SecRCA_k^n$ is used, it takes $6 \cdot k - 2$ clock cycles. If $SecKSA_k^n$ is used, it takes $4\lceil log_2 k\rceil + 2\lceil log_2(k-1)\rceil + 6$ clock cycles. In Algorithm 8, We implement the $SecAddModp_k^n$ based on $SecCSA_k^n$, which only requires $2\lceil log_2 k\rceil + 2\lceil log_2(k-1)\rceil + 6$. Furthermore, we adopt the technique in [BC22] to optimize the conversion of $n = 2$ and $n = 3$, reducing p in advance before expanding the Boolean shares. Finally, we propose a CSA-based $SecA2BModp_k^n$ in Algorithm 9. The $BitCopyMask_k^n$ in our $SecAddModp_k^n$ and $SecA2BModp_k^n$ uses algorithm 1 of [BC22], which is a share-isolating gadget.

Theorem 4. *The $SecA2BModp_k^n$ in Algorithm 9 is t-PINI, for $t = n - 1$.*

Proof. The $SecKSA_k^n$ and $SecCSA_k^n$ are PINI as proved in Lemma 2, and the extension and $BitCopyMask_k^n$ are share isolating. In the case of $n = 2$, $SecA2BModp_k^2$ is 1-PINI by Lemma 1. For $n = 3$, steps 9 to 13 are 1-PINI, and as Lemma 3 steps 9 to 16 are 2-PINI. Therefore, the $SecA2BModp_k^3$ is 2-PINI.

For other cases, the divide-and-conquer structure is the same as that of origin A2B shown in Figure 1. The gadget can be split into three parts: $E_{n,(1,\dots,\lfloor n/2\rfloor)}^{SecA2BModp_k^{\lfloor n/2\rfloor}}$, $E_{n,(\lfloor n/2\rfloor+1,\dots,n)}^{SecA2BModp_k^{n-\lfloor n/2\rfloor}}$ and $SecAddModp_k^n$. By recursing n, $SecA2BModp_k^{\lfloor n/2\rfloor}$ and $SecA2BModp_k^{n-\lfloor n/2\rfloor}$ is PINI, so the corresponding embedding gadgets is PINI according to Lemma 3. Moreover, $SecAddMop_k^n$ consisting of $SecCSA_k^n$, $SecKSA_k^n$, and share isolating operations is PINI. Finally, $SecA2BModp_k^n$, a composition of PINI gadgets, is PINI. □

Like $SecA2B_k^n$, $SecA2BModp_k^n$ can also use HPC multiplication gadgets by adding registers and guaranteed full pipeline in hardware implementation to achieve glitch+transition-robust PINI.

Using Kyber's prime modulus $p = 3329$, $k = \lceil log_2(p)\rceil = 12$ in A2B conversion. We compare the latency between our $SecA2BModp_k^n$ based CSA and that of [BC22] (algorithm 10, $SecA2BModp_k^n$) in hardware implementation. For a fairer comparison, $SecAdd_k^n$ of [BC22] use $SecKSA_k^n$ with lower latency as a secure adder instead of the original RCA.

Algorithm 9 $SecA2BModp_k^n$

Input: n shares arithmetic sharing $\boldsymbol{x^{A_p}}$,such that $x \in [0, p)$.
Output: n shares Boolean sharing $\boldsymbol{z^{B,k}}$, such that $z = x$ and $p < 2^k$.

```
1: if $n = 2$ then
2:     $\boldsymbol{s^{B,k+1}} \leftarrow (x_1^{A_p} + 2^k - p, 0)$
3:     $\boldsymbol{s'^{B,k}} \leftarrow (0, x_2^{A_p})$
4:     $\boldsymbol{u^{B,k+1}} \leftarrow SecKSA_{k+1}^2(\boldsymbol{s^{B,k+1}}, \boldsymbol{s'^{B,k}})$
5:     $\boldsymbol{b^{B,1}} \leftarrow \neg \boldsymbol{u^{B,k+1}}[k]$
6:     $\boldsymbol{a^{B,k}} \leftarrow BitCopyMask_k^2(\boldsymbol{b^{B,1}}, p)$        ▷ Use algorithm 1 of [BC22].
7:     $\boldsymbol{z^{B,k}} \leftarrow SecKSA_k^2(\boldsymbol{a^{B,k}}, \boldsymbol{u^{B,k+1}})$
8: else if $n = 3$ then
9:     $\boldsymbol{s^{B,k+1}} \leftarrow (x_1^{A_p} + 2^k - p, 0)$
10:    $\boldsymbol{s'^{B,k}} \leftarrow (0, x_2^{A_p})$
11:    $\boldsymbol{u^{B,k+1}} \leftarrow SecKSA_{k+1}^2(\boldsymbol{s^{B,k+1}}, \boldsymbol{s'^{B,k}})$
12:    $\boldsymbol{b^{B,1}} \leftarrow \neg \boldsymbol{u^{B,k+1}}[k]$
13:    $\boldsymbol{a^{B,k}} \leftarrow BitCopyMask_k^2(\boldsymbol{b^{B,1}}, p)$
14:    $\boldsymbol{(y_1)^{B,k}} \leftarrow (a_1^{B,k}, a_2^{B,k}, 0)$
15:    $\boldsymbol{(y_2)^{B,k+1}} \leftarrow (u_1^{B,k+1}, u_2^{B,k+1}, 0)$
16:    $\boldsymbol{(y_3)^{B,k+1}} \leftarrow (0, 0, x_3^{A_p} - p)$
17:    $(\boldsymbol{s^{B,k+1}}, \boldsymbol{c^{B,k+1}}) \leftarrow SecCSA_{k+1}^3(\boldsymbol{(y_1)^{B,k}}, \boldsymbol{(y_2)^{B,k+1}}, \boldsymbol{(y_3)^{B,k+1}})$
18:    $\boldsymbol{u^{B,k+1}} \leftarrow SecKSA_{k+1}^3(\boldsymbol{s^{B,k+1}}, \boldsymbol{c^{B,k+1}})$
19:    $\boldsymbol{b^{B,1}} \leftarrow \neg \boldsymbol{u^{B,k+1}}[k]$
20:    $\boldsymbol{a^{B,k}} \leftarrow BitCopyMask_k^3(\boldsymbol{b^{B,1}}, p)$
21:    $\boldsymbol{z^{B,k}} \leftarrow SecKSA_k^3(\boldsymbol{a^{B,k}}, \boldsymbol{u^{B,k+1}})$
22: else
23:    $\boldsymbol{y^{B,k}} \leftarrow SecA2BModp_k^{\lfloor n/2 \rfloor}((x_1^{A_{2^k}}, ..., x_{\lfloor n/2 \rfloor}^{A_{2^k}}))$
24:    $\boldsymbol{y'^{B,k}} \leftarrow SecA2BModp_k^{n-\lfloor n/2 \rfloor}((x_{\lfloor n/2 \rfloor+1}^{A_{2^k}}, ..., x_n^{A_{2^k}}))$
25:    $\boldsymbol{s^{B,k}} \leftarrow (y_1^{B,k}, y_2^{B,k}, ..., y_{\lfloor n/2 \rfloor}^{B,k}, 0, ..., 0)$
26:    $\boldsymbol{s'^{B,k}} \leftarrow (0, ..., 0, y_1'^{B,k}, ..., y_{\lceil n/2 \rceil}'^{B,k})$
27:    $\boldsymbol{z^{B,k}} \leftarrow SecAddModp_k^n(\boldsymbol{s^{B,k}}, \boldsymbol{s'^{B,k}})$
```

As shown in Table 4, the latency is reduced by 16% to 22% under 2nd-order to 10th-order security.

B2A conversion. The B2A algorithm based on Boolean circuits was first proposed in [CGV14] and was extended to the case of prime modulus in [BBE+18]. The basic idea is to first generate $n - 1$ random arithmetic shares under the 2^k modulus (resp. prime modulus), and then obtain the n-share Boolean masking of the negation of their sum through $SecA2B_k^n$ (resp. $SecA2BModp_k^n$), add it to the input Boolean masking by $SecKSA_k^n$ (resp. $SecAddModp_k^n$). Lastly, the algorithm unmasks the Boolean masking of the result as the last arithmetic share.

In B2A conversion, the $SecA2B_k^n$ and $SecA2BModp_k^n$, as well as $SecAddModp_k^n$, can leverage the low-latency algorithm introduced in this work. This can also reduce the clock cycle delay required for a B2A conversion.

Table 4: Number of cycles to perform an A2B conversion in hardware with p=3329.

Design	Security order								
	2	3	4	5	6	7	8	9	10
[BC22]	40	50	80	80	80	80	110	110	110
This work	32	42	64	64	64	64	86	86	86

6 Conclusion

In this paper, we analyze the high-latency problem of the existing high-order A2B conversion scheme. The root cause is believed to lie in the high degree nonlinear functions to compute the carry in the secure adder. Based on this analysis, we propose a CSA-based high-order A2B conversion algorithm, which expresses intermediate variables through redundant numbers to avoid carry calculations in the process of arithmetic shares summation.

As a case study, we implemented an A2B conversion of 32-bit variables side-channel protection for both hardware and software platforms. Compared with existing KSA-based A2B conversion, our design reduces latency by 27% to 50% for parameters $9 < k \leq 33$ and $2 < n \leq 10$ under both ASIC standard cell technology and FPGA. Meanwhile, the proposed A2B conversion has a speedup in software performance.

The security assessment is completed using side-channel evaluation experiments and theoretical analysis. On the ChipWhisperer development board and SAKURA-G platform, the software and hardware implementation carried out TVLA experiments, respectively, demonstrating second-order side-channel security.

Acknowledgments

This work is supported in part by the National Key R&D Program of China (Grant No. 2023YFB4403500), and in part by the National Natural Science Foundation of China (Grant No. 62274102, No. 62104129), and in part by the National Key R&D Program of China (Grant No.2021YFB2701201).

References

[AMD$^+$21] Abubakr Abdulgadir, Kamyar Mohajerani, Viet Ba Dang, Jens-Peter Kaps, and Kris Gaj. A lightweight implementation of saber resistant against side-channel attacks. In *Progress in Cryptology–INDOCRYPT 2021: 22nd International Conference on Cryptology in India, Jaipur, India, December 12–15, 2021, Proceedings 22*, pages 224–245. Springer, 2021.

[AZN22] Victor Arribas, Zhenda Zhang, and Svetla Nikova. LLTI: low-latency threshold implementations. *IACR Cryptol. ePrint Arch.*, page 418, 2022.

[B$^+$08] Daniel J Bernstein et al. Chacha, a variant of salsa20. In *Workshop record of SASC*, pages 3–5, 2008.

[BBD$^+$16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 116–129. ACM Press, October 2016.

[BBE$^+$18] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. Masking the glp lattice-based signature scheme at any order. In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37*, pages 354–384. Springer, 2018.

[BC22] Olivier Bronchain and Gaëtan Cassiers. Bitslicing arithmetic/boolean masking conversions for fun and profit: with application to lattice-based kems. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 553–588, 2022.

[CGLS20a] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Transactions on Computers*, 70(10):1677–1690, 2020.

[CGLS20b] Gaëtan Cassiers, Benjamin Grégoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. Cryptology ePrint Archive, Report 2020/185, 2020. https://eprint.iacr.org/2020/185.

[CGMZ22] Jean-Sébastien Coron, François Gérard, Simon Montoya, and Rina Zeitoun. High-order table-based conversion algorithms and masking lattice-based encryption. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):1–40, 2022.

[CGTV15] Jean-Sébastien Coron, Johann Großschädl, Mehdi Tibouchi, and Praveen Kumar Vadnala. Conversion from arithmetic to Boolean masking with logarithmic complexity. In Gregor Leander, editor, *FSE 2015*, volume 9054 of *LNCS*, pages 130–149. Springer, Heidelberg, March 2015.

[CGV14] Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vadnala. Secure conversion between Boolean and arithmetic masking of any order. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 188–205. Springer, Heidelberg, September 2014.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Transactions on Information Forensics and Security*, 15:2542–2555, 2020.

[CS21] Gaëtan Cassiers and François-Xavier Standaert. Provably secure hardware masking in the transition-and glitch-robust probing model: Better safe than sorry. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 136–158, 2021.

[CT03] Jean-Sébastien Coron and Alexei Tchulkine. A new algorithm for switching from arithmetic to Boolean masking. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *CHES 2003*, volume 2779 of *LNCS*, pages 89–97. Springer, Heidelberg, September 2003.

[D'A22] Jan-Pieter D'Anvers. One-hot conversion: Towards faster table-based A2B conversion. *IACR Cryptol. ePrint Arch.*, page 1099, 2022.

[DAP$^+$22] Anuj Dubey, Afzal Ahmad, Muhammad Adeel Pasha, Rosario Cammarota, and Aydin Aysu. Modulonet: neural networks meet modular arithmetic for efficient hardware masking. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 506–556, 2022.

[Deb12] Blandine Debraize. Efficient and provably secure methods for switching from arithmetic to Boolean masking. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 107–121. Springer, Heidelberg, September 2012.

[FBR$^+$22] Tim Fritzmann, Michiel Van Beirendonck, Debapriya Basu Roy, Patrick Karl, Thomas Schamberger, Ingrid Verbauwhede, and Georg Sigl. Masked accelerators and instruction set extensions for post-quantum cryptography. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(1):414–460, 2022.

[GGJR$^+$11] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011.

[GIB18] Hannes Gross, Rinat Iusupov, and Roderick Bloem. Generic low-latency masking in hardware. *IACR TCHES*, 2018(2):1–21, 2018. https://tches.iacr.org/index.php/TCHES/article/view/871.

[Gou01] Louis Goubin. A sound method for switching between Boolean and arithmetic masking. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 3–15. Springer, Heidelberg, May 2001.

[GPM23] Barbara Gigerl, Robert Primas, and Stefan Mangard. Formal verification of arithmetic masking in hardware and software. In *International Conference on Applied Cryptography and Network Security*, pages 3–32. Springer, 2023.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, August 2003.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Heidelberg, August 1999.

[KNAH22] Tendayi Kamucheka, Alexander Nelson, David Andrews, and Miaoqing Huang. A masked pure-hardware implementation of kyber cryptographic algorithm. In *International Conference on Field-Programmable Technology, (IC)FPT 2022, Hong Kong, December 5-9, 2022*, page 1. IEEE, 2022.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Heidelberg, August 1996.

[Lan19] Eric Lankford. Nist cryptographic algorithm and module validation programs: Validating new encryption schemes. *ISSA Journal*, 17(5), 2019.

[LMRBG23] Georg Land, Adrian Marotzke, Jan Richter-Brockmann, and Tim Güneysu. Gate-level masking of streamlined ntru prime decapsulation in hardware. *Cryptology ePrint Archive*, 2023.

[MBFC23] Saurav Maji, Utsav Banerjee, Samuel H. Fuller, and Anantha P. Chandrakasan. A threshold implementation-based neural network accelerator with power and electromagnetic side-channel countermeasures. *IEEE J. Solid State Circuits*, 58(1):141–154, 2023.

[MKSM22] Nicolai Müller, David Knichel, Pascal Sasdrich, and Amir Moradi. Transitional leakage in theory and practice: Unveiling security flaws in masked circuits. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 266–288, 2022.

[MME10] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. Correlation-enhanced power analysis collision attack. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 125–139. Springer, Heidelberg, August 2010.

[MMSS19] Thorben Moos, Amir Moradi, Tobias Schneider, and François-Xavier Standaert. Glitch-resistant masking revisited: Or why proofs in the robust probing model are needed. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 256–292, 2019.

[MS16] Amir Moradi and Tobias Schneider. Side-channel analysis protection and low-latency in action – case study of PRINCE and Midori –. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 517–547. Springer, Heidelberg, December 2016.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In Isabelle Attali and Thomas P. Jensen, editors, *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001.

[SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In Tim Güneysu and Helena Handschuh, editors, *CHES 2015*, volume 9293 of *LNCS*, pages 495–513. Springer, Heidelberg, September 2015.

[SMG15] Tobias Schneider, Amir Moradi, and Tim Güneysu. Arithmetic addition over Boolean masking - towards first- and second-order resistance in hardware. In Tal Malkin, Vladimir Kolesnikov, Allison Bishop Lewko, and Michalis Polychronakis, editors, *ACNS 15*, volume 9092 of *LNCS*, pages 559–578. Springer, Heidelberg, June 2015.

[TDG03] Elena Trichina, Domenico De Seta, and Lucia Germani. Simplified adaptive multiplicative masking for AES. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 187–197. Springer, Heidelberg, August 2003.

[VDV21] Michiel Van Beirendonck, Jan-Pieter D'Anvers, and Ingrid Verbauwhede. Analysis and comparison of table-based arithmetic to boolean masking. *IACR TCHES*, 2021(3):275–297, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8975.

[ZSS$^+$21] Sara Zarei, Aein Rezaei Shahmirzadi, Hadi Soleimany, Raziyeh Salarifard, and Amir Moradi. Low-latency keccak at any arbitrary order. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 388–411, 2021.