applied sciences

MDPI

Article

Boolean Masking for Arithmetic Additions at Arbitrary Order in Hardware

Florian Bache [1,*] **and Tim Güneysu** [1,2]

[1] Horst-Görtz Institute for IT-Security, Ruhr-Universität Bochum, 44801 Bochum, Germany; tim.gueneysu@rub.de
[2] German Research Center for Artificial Intelligence, 28359 Bremen, Germany
* Correspondence: florian.bache@rub.de

Abstract: Modular addition is an important component of many cryptographic algorithms such as ARX-ciphers and lattice-based post-quantum secure schemes. In order to protect devices that execute these algorithms against side-channel attacks, countermeasures such as masking must be applied. However, if an implementation needs to be secured against multivariate attacks, univariately secure masking schemes do not suffice. In this work, we focus on hardware architectures for higher-order masked addition circuits. We present and discuss three adder designs that are protected with a provably secure masking scheme. Concretely, we discuss Kogge–Stone, Sklansky and Brent–Kung adders regarding their suitability for high-order masking and their performance in this setting. All architectures are fully pipelined and achieve a throughput of one addition per cycle. In order to achieve multivariate security at arbitrary orders, we use HPC2 Gadgets that satisfy the PINI security notion. Additionally, we apply a first-order secure threshold implementation scheme to the adder variants and compare their performance in the univariate case.

Keywords: side-channel analysis; Boolean masking; hardware; addition; threshold implementation; HPC2

Citation: Bache, F.; Güneysu, T. Boolean Masking for Arithmetic Additions at Arbitrary Order in Hardware. *Appl. Sci.* **2022**, *12*, 2274. https://doi.org/10.3390/app12052274

Academic Editors: Guy Gogniat and Arcangelo Castiglione

Received: 11 January 2022
Accepted: 15 February 2022
Published: 22 February 2022

1. Introduction

Modular addition in the ring $\mathbb{Z}_{2^n}$ is a core part of several cryptographic schemes. For example, in Addition-Rotation-XOR (ARX) ciphers they perform similar function to S-boxes in classical block ciphers, being the only non-linear part of the algorithm. ARX constructions, such as SPECK or SALSA20, combine this arithmetic function with the Boolean exclusive or operation (XOR) and a bit-wise rotation to produce a secure algorithm. When hardware implementations of such algorithms can potentially be targets of side-channel attacks, masking can be used as an effective countermeasure. However, while the separate protection of the arithmetic and Boolean parts of these algorithms is possible using both an arithmetic and a Boolean masking scheme, the conversion between these representations poses significant problems, especially in hardware.

Contribution

In this work we study how addition operations can be protected against side-channel attacks in the hardware context using Boolean masking. We propose three different designs for parallel-prefix addition circuits that can generically be masked at arbitrary order using gadgets that follow the Probe Isolating Non Interference (PINI) security notion. To this end, we study their suitability for masking and compare them regarding their area and randomness requirements as well as their latency. In order to achieve higher-order multivariate security, we employ the HPC2-gadgets that were proposed in [1]. Concretely, the proposed adder structures are:

1. A Kogge–Stone adder with a latency of $\log n$-cycles.

2. A Sklansky adder with the same latency but reduced area and randomness requirements.
3. A Brent–Kung adder which trades off higher latency for an even lower logic area requirement and requires less randomness.

After performing a detailed analysis of their structure, we implement these adder types as 32-bit variants on an FPGA. All implemented variants are fully pipelined and therefore achieve a throughput of one addition per cycle. To our knowledge, the proposed adders are the first arbitrary-order masked hardware designs that are secure against multivariate attacks.

Additionally, we discuss the application of the Threshold Implementation (TI) masking scheme to the Sklansky and Brent–Kung structures, where we use the same sharing that the authors of [2] applied to ripple-carry and Kogge–Stone adders. Only the first order variant is considered, as higher-order TI can not generically provide protection against multivariate attacks [3]. In this case, randomness and area can be saved in comparison to the variants that are secure at arbitrary order. Finally, we provide practical side-channel evaluations of the Sklansky- and Brent–Kung-adders using Test Vector Leakage Assessment (TVLA).

2. Preliminaries

2.1. Notation

The binary operations `and`, `or` and `xor` are denoted by the symbols $\wedge$, $\vee$ and $\oplus$, while the $+$ sign is used for the addition over integers or rings. All logarithms are in base two. The least significant and most significant bits of the n-bit variable a are a_0 and a_{n-1} , respectively. Single subscripts indicate the index of multi-bit variable (e.g., a_i) and consecutive groups of bits between index i and j are indicated as $a_{\{i,j\}}$. In order to simplify equations, indices are treated as zero if they become negative. Superscripts note the respective share of a shared variable.

2.2. Parallel Prefix Adders

In order to compute the sum s of two n-bit inputs over $\mathbb{Z}_n$, an addition circuit needs to compute a sum bit s_i for every pair of input bits (a_i, b_i) using carry bits c_i as:

$$s_i = a_i \oplus b_i \oplus c_i \text{ with}$$

$$c_i = c_{i-1} \wedge (a_{i-1} \oplus b_{i-1}) \vee (a_{i-1} \wedge b_{i-1}) \quad \forall i > 0 \text{ , else } 0$$

A direct realization of these equations leads to a carry-ripple adder with a circuit depth of n due to the dependency of c_i from c_{i-1}. When masking this architecture in hardware where glitches occur, the required registers in each stage lead to a latency of n cycles.

Parallel-prefix adders can reduce this latency by restating the computation of the carry bits using (group-) generate and propagate terms and computing them in parallel. Intuitively, the generate term $p_{\{i,j\}}$ determines if the groups of input bits $a_{\{i,j\}}$ and $b_{\{i,j\}}$ will generate a carry output c_{j+1}, independently of the carry input c_i. The propagate term determines if an input carry c_i will affect the output carry c_{j+1}.

For single-bit inputs (a_i, b_i), the generate and propagate terms are computed as $g_i = a_i \wedge b_i$ and $p_i = a_i \oplus b_i$, respectively. We call this initial step, which is the same for all parallel-prefix adders, the preprocessing step. Given $i > k \geq j$, the group-generate term is calculated as

$$g_{\{i,j\}} = g_{\{k,j\}} \oplus \left(p_{\{k,j\}} \wedge g_{\{i,k+1\}} \right) \tag{1}$$

and the group-propagate term as

$$p_{\{i,j\}} = p_{\{i,k+1\}} \wedge p_{\{k,j\}} \, . \tag{2}$$

The final sum bits can then be computed as $s_i = g_{\{i-1,0\}} \oplus p_i$, where $g_{\{-1,0\}} = 0$. The costs in XOR and AND operations for implementing each of the functions mentioned above are itemized in Table 1.

Table 1. Number of elementary XOR and AND operations per basic function.

Function	$g(i)$	$p(i)$	$g(\{i,j\})$	$p(\{i,j\})$	$s(i)$
AND	1	0	1	1	0
XOR	0	1	1	0	1

In the following we use $pg_{\{i,j\}}$ (or PG term where the indices are not relevant) as a short form for the tuple $(p_{\{i,j\}}, g_{\{i,j\}})$. We define the function block that computes these terms by combining existing terms as $PG(\{i,j\},\{k,l\}) = (p(\{i,l\}), g(\{i,l\}))$ and call them PG blocks.

The different variants of parallel-prefix adders, such as the ones discussed here, all apply this same principle but differ in the way PG terms of larger groups of bits are constructed from smaller ones.

2.3. Masking Countermeasures

A common and effective countermeasure against side-channel attacks is masking. Here, the relation between the sensitive data and the side-channel is obscured by removing the dependencies between the two to a certain degree. In a d-th order masking scheme, sensitive values are split into multiple (at least $d+1$) shares and a successful attacker is required to combine information from at least $d+1$ of these shares to be able to deduce any information about the original value. As the measurement of a physical side-channel always introduces noise, the complexity of attacks increases exponentially with the masking order under most practical conditions [4]. Masking schemes for hardware implementations should provide security in the presence of glitches.

A multitude of masking schemes, such as the Consolidated Masking Scheme [5] and Domain-Oriented Masking [6], have been proposed in the literature of which we will discuss Threshold Implementation [7] and Hardware Private Circuits [1] as they are applied in this contribution.

Threshold Implementation

A Threshold Implementation (TI) is a Boolean masking scheme, i.e., a sensitive value x is split into shares x^i such that $x = \bigoplus x^i$. The number of input (d_{in}) and output (d_{out}) shares of a function following the TI notion depends on the masking order and the algebraic degree of the function. However, we will mainly focus on the first-order secure variant here, where $d_{in} > d$ and $d_{out} > d$. While higher-order TI instantiations are possible [8], their security is limited to univariate attacks in practice [3]. In order to correctly instantiate a threshold implementation, three properties have to be fulfilled. The *non-completeness* property requires any subset of output shares smaller than $d+1$ to be independent of at least one input share. The *correctness* property guarantees that the output of a shared function can be unmasked yielding the result an unprotected realization of the function would produce, i.e., $\bigoplus f_{shared}(x^1, ..., x^{d_{in}}) = f_{unshared}(x)$. Finally, *uniformity* requires each possible output sharing to be equally likely when the input sharing is drawn from a uniform distribution. Note, that this last property is usually the hardest to fulfill and frequently requires the addition of fresh randomness. As with most masking schemes, linear functions (e.g., XOR) are trivially shared by applying the unshared function to all input shares independently, while non-linear functions (e.g., AND) are more difficult to share correctly. To ensure the resistance of TI against leakage through glitches, registers have to be placed between components when composing larger circuits from multiple functions.

Probe-Isolating Non-Interference and Hardware Private Circuits

Several attack models have been developed in order to abstract the observation of side-channels in a meaningful way and allow the construction of masking schemes that are provably secure in the respective model. The probing model, formalized in [9], generally requires any set of values that can be accessed by up to d probes in a circuit to be indepen-

dent of the sensitive variables. As this property is insufficient to guarantee the security of a composition of gadgets, which are themselves secure in this model [10], the more restrictive notions of Non-Interference (NI) and Strong Non-Interference (SNI) [11] were introduced. In order to satisfy NI, a set of $t \leq d$ probes on a gadget needs to be simulatable using at most t shares of each input. For SNI-secure gadgets, every set of t_{int} internal probes and t_{out} probes on the gadget output, where $t_{in} + t_{out} \leq d$, can be simulated using only t_{int} shares of each input. While the authors of [11] showed that gadgets can be securely composed using NI or SNI gadgets, the area and randomness costs are typically high because refresh gadgets are needed to allow composition. The Probe Isolating Non Interference (PINI) notion proposed in [12] aims to solve this problem by developing a model which allows trivial composition with reasonable cost. For a PINI-secure gadget, the simulatability is constraint by the share index a probe is associated with. The resulting model allows trivial realizations of linear functions as with TI.

Following this notion, the authors of [1] proposed several hardware realizations of PINI-secure gadgets. The HPC2 AND gadget (which we use in this work) provides dth-order security using $d + 1$ input and output shares. It has an asymmetric latency with respect to the two input arguments, where the first argument influences the output after one cycle and the second after two cycles. The required fresh randomness per gadget amounts to $d \cdot (d + 1)/2$ bits. We used the source code for the gadgets from the library provided by the authors [13] in our implementations.

3. Boolean Masking for Addition Circuits

In this section, we discuss the application of three different architectures for parallel addition circuits that can be effectively used for higher-order masked applications. In many cryptographic schemes, e.g., in ARX algorithms, the addition operation is performed in the ring $\mathbb{Z}_n$, where $n = 2^m, m \in \mathbb{Z}$. Therefore, we assume the width of the addition circuits to be 2^m and do not consider input and output carry bits. However, the proposed designs can be adapted to other bit widths and to the handling of carries, if required by an application.

Throughout this section we use diagrams to illustrate the structures of the algorithms under discussion. In these diagrams, the preprocessing blocks are represented by oval shapes while all other blocks are rectangular. A block is shaded gray if both, the generate and propagate terms, are computed by it. If a block only needs to calculate the generate or propagate term, it is shaded green or orange, respectively. Blocks that are only needed if an output carry is required are not filled. All described adder designs require a final stage for the calculation of the sum bits, which is not shown in the diagrams.

3.1. Kogge–Stone Adder (KSA)

KSAs [14] are parallel-prefix adders that are similar to greedy algorithms in the sense that the maximal number of PG terms are combined in each stage. An n-bit KSA requires $\log n$ stages to compute all $g(\{i, 0\})$ terms and one additional stage to compute the final sum bits. In order to better illustrate the architecture, an 8-bit KSA is depicted in Figure 1. When excluding the preprocessing step and the final sum calculation, it requires 3 stages and 14 PG blocks.

In the first stage after the preprocessing step, every PG term is combined with its neighbor, i.e., $PG(\{i, i\}\ ,\ \{i-1, i-1\})\ \forall 0 < i < n-1$ is computed. Therefore, $n-1$ PG function blocks need to be instantiated in that stage. In the second stage, $PG(\{i, i-1\}\ ,\ \{i-2, i-4\})\ \forall 1 < i < n-1$ is computed, requiring $n-2$ PG function blocks. Note that $p(\{1, 0\})$ is not needed in this computation and the hardware for its generation can therefore be omitted in the first stage. In general, stage k computes $n - 2^k$ PG terms as (when numbering the stages, the stage number k of the first stage after preprocessing is zero):

$$PG\left(\left\{i, i-2^k+1\right\}\ ,\ \left\{i-2^k, i-2^{k+1}+1\right\}\right)\ \forall k < i < n-1.$$

As we do not consider the output carry generation, the calculation of $pg(\{n-1, i\})$ $\forall i \neq n-1$ can be skipped, saving $\log n$ PG blocks. This reasoning does not only apply to Kogge–Stone adders but also to the other structures discussed below. Over all stages excluding the preprocessing, $(n-1) \cdot \log n - n + 1$ PG blocks are required. Of these blocks, $n-2$ blocks do not need to compute the propagate term. In the preprocessing step, which is the same for all parallel-prefix adders, n generate functions and $n-1$ propagate functions are needed. A summary of the required numbers of all the basic blocks for all adder types is provided in Table 2. By combining Tables 1 and 2, we can summarize that an n-bit Kogge–Stone adder requires $(n-1) \cdot \log n - 3 \cdot n + 1$ XOR gates and $2 \cdot (n-1) \cdot \log n - 2 \cdot n + 3$ AND gates. The number of elementary operations per adder is depicted in Table 3.

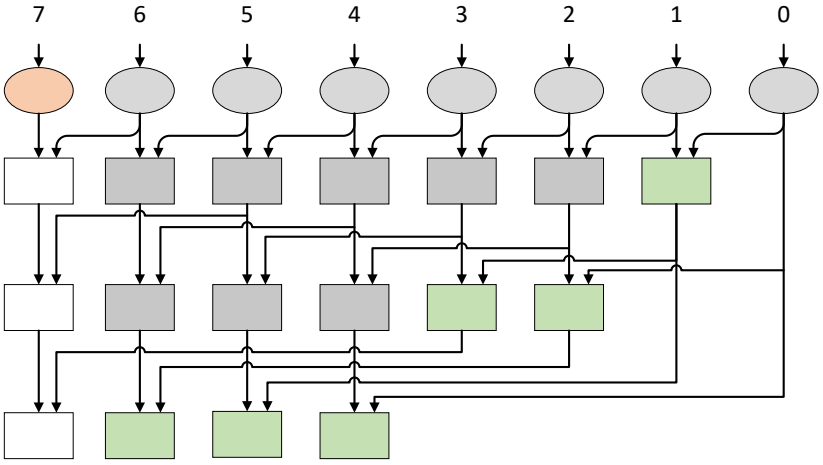


Figure 1. 8-bit Kogge–Stone Adder.

Table 2. Number of required basic functions for different parallel-prefix adders.

	$g(i)$	$p(i)$	$g(\{i,j\})$	$p(\{i,j\})$	$s(i)$
Kogge–Stone	$n-1$	n	$(n-1) \cdot \log n - n + 1$	$(n-1) \cdot \log n - 2 \cdot n + 3$	n
Sklansky	$n-1$	n	$(n/2-1) \cdot \log n$	$(n/2-1) \cdot \log n - n + 2$	n
Brent–Kung	$n-1$	n	$2 \cdot n - 2 \cdot \log n - 2$	$n - 2 \cdot \log n$	n

Table 3. Number of elementary XOR and AND operations for different parallel-prefix adders including the preprocessing stage and final computation of the sum bits.

	Kogge–Stone	**Sklansky**	**Brent–Kung**
AND	$2 \cdot (n-1) \cdot \log n - 2 \cdot n + 3$	$(n-2) \cdot \log n - n + 1$	$3 \cdot n - 4 \cdot \log n - 1$
XOR	$(n-1) \cdot \log n - 3 \cdot n$	$(n/2-1) \cdot \log n + 2 \cdot n$	$4 \cdot n - 2 \cdot \log n - 2$

Masking KSAs

When compared to the other designs discussed in this work, an implementation of a KSA requires the highest number of PG blocks, resulting in the largest area requirement when implemented using masking. As dth-order secure HPC2 AND-Gadgets require $d \cdot (d+1)/2$ bits of fresh randomness, the total randomness requirement of a HPC2-masked n-bit KSA is the highest of the discussed variants at $d \cdot (d+1) \cdot ((n-1) \cdot (\log n - 1) + 1)$ bits. However, the required randomness can be drastically reduced if only univariate security is considered, as shown in [2]. Here, a first-order secure TI of a KSA using three shares that only needs n bits of fresh randomness is proposed. As noted by the authors, their second-order variant does not provide protection against multivariate attacks.

3.2. Sklansky Adder (SA)

The Sklansky Adder was introduced in 1960 [15] as an efficient parallel adder with low area requirements. It has the same latency as a KSA at $\log n + 1$ cycles but requires a lower

total number of PG blocks. In contrast to the previously described circuit, the number of PG blocks per stage is constant at $n/2$. Figure 2 depicts an 8-bit Sklansky adder, realized with 12 PG blocks in 3 stages, excluding preprocessing.

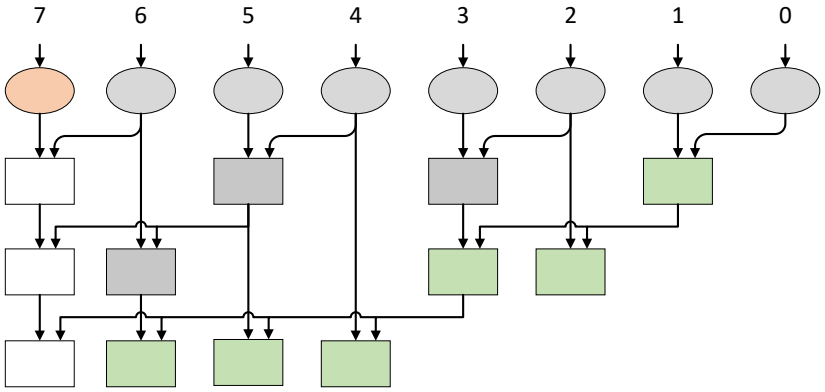


Figure 2. 8-bit Sklansky Adder.

In the first stage, every other PG term is combined with its neighbor, i.e.,

$$PG(\{2 \cdot i + 1, 2 \cdot i + 1\}\ ,\ \{2 \cdot i, 2 \cdot i\})\ \forall 0 \leq i < n/2$$

is computed. The second stage combines these 2-bit PG terms as

$$PG\left(\left\{4 \cdot \left\lfloor \frac{i}{2} \right\rfloor + 2 + (i \bmod 2), 4 \cdot \left\lfloor \frac{i}{2} \right\rfloor + 2\right\}\ ,\ \left\{4 \cdot \left\lfloor \frac{i}{2} \right\rfloor + 1, 4 \cdot \left\lfloor \frac{i}{2} \right\rfloor\right\}\right)\ \forall 0 \leq i < n/2.$$

In general, the k-th stage for $k > 0$ combines the previous PG terms using $n/2$ PG blocks in the following way:

$$PG\left(\left\{\left\lfloor \frac{i}{2^k} \right\rfloor \cdot 2^{k+1} + 2^k + (i \bmod 2^k), \left\lfloor \frac{i}{2^k} \right\rfloor \cdot 2^{k+1} + 2^k\right\}\ ,\right.$$
$$\left.\left\{\left\lfloor \frac{i}{2^k} \right\rfloor \cdot 2^{k+1} + 2^k - 1, \left\lfloor \frac{i}{2^k} \right\rfloor \cdot 2^{k+1} + 2^{k-1} - 1\right\}\right)\ \forall 0 \leq i < n/2$$

Following the reasoning of Section 3.1, $(n/2 - 1) \cdot \log n$ PG blocks are needed to build a SA when excluding the preprocessing stage. As in in the Kogge–Stone case, $n - 2$ of these do not need to compute the propagate term. The resulting number of required the basic blocks is provided in Table 2.

Masking SAs

When compared to a KSA, the main advantage of an SA in the context of masked implementations lies in the reduced number of required PG blocks, which directly results in a lower area requirement. If HPC2 gadgets are used this also leads to reduces randomness use of $d \cdot (d + 1) \cdot ((n - 2) \cdot \log n - n + 1)$ bit.

If only first-order security is desired, a TI similar to [2] can be used to reduce the randomness requirements in comparison to HPC2 gadgets. In this publication, the authors re-use shares of the generate terms to reduce the required fresh randomness in the computation of the propagate term. Specifically, when computing $p_{\{i,j\}}$ as a masked version of Equation (2), the first share of $g_{\{k,j\}}$ is used to achieve the uniformity of the tuple $(p_{\{i,j\}}, g_{\{i,j\}})$. In the KSA-case, each p-term is not used more than once per stage as the rightmost term in Equation (2). Therefore, each g-term is not used more than once to replace a bit of fresh randomness, preventing potential violations of the joint uniformity of the PG tuples in later stages. Unfortunately, this randomness reduction approach can not directly be applied to SAs, due to the higher fan-out of the PG blocks of up to $n/2 - 1$ in this case. Therefore, SAs require additional randomness when masked with TI. However, the number of required fresh random bits can be reduced by following the the construction of the second-order secure KSA presented in [2]. Here, the authors take four shares from $g_{\{k,j\}}$ to replace fresh mask bits. Following this approach, instead of taking the same first share as mask replacement, we can use up to three different shares. In stages with a fan-out higher than three, additional fresh random bits need to be inserted. This results in 10 additional random bits required by a 32-bit first-order secure SA when masked with TI.

3.3. Brent–Kung Adder (BKA)

A BKA [16] allows a further reduction in the number of PG blocks in comparison to an SA, albeit with the cost of increased latency. The general structure of BKA is composed of two trees, where the first tree computes the group generate and propagate terms of increasingly larger groups of bits until the carry bit for the most significant output can be determined. The resulting structure can be viewed as a binary tree where the output carry for the most significant bit represents the root and the PG terms generated by the preprocessing represent the leafs. An 8-bit version of this tree is shown in Figure 3.

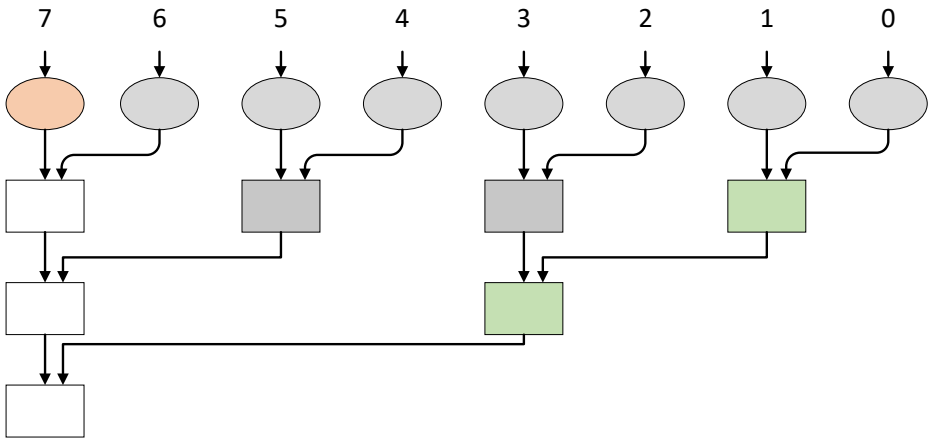


Figure 3. 8-bit Brent–Kung Adder, generation of the MSB carry bit.

As PG terms, which are not required to compute the most significant carry bit, are not considered in this process, a second reversed tree is needed in order to generate them. A full 8-bit BKA that can compute all bits of the final sum is depicted in Figure 4. Note that in this 8-bit case, the carry for the most significant bit is already available after $\log 8 = 3$ stages. However, an additional stage is required for the reverse tree, which computes the remaining carry bits. As a result, the 8-bit BKA requires a total of 4 stages and 11 PG blocks, excluding the preprocessing stage.

The first stage in a BKA is the same as for the SA, combining every other PG term with its neighbor and therefore generating $\frac{n}{2}$ 2-bit PG terms. Following the tree structure of BKAs, the second stage combines adjacent 2-bit PG terms to 4-bit terms as:

$$PG(\{4 \cdot i + 3, 4 \cdot i + 2\}\ ,\ \{4 \cdot i + 1, 4 \cdot i\})\ \forall 0 \leq i < n/4.$$

This pattern is repeated until the final carry can be computed, requiring $\log n$ stages. The k-th stage therefore computes PG terms for increasingly larger groups of bits using $\frac{n}{2^{k+1}}$ PG blocks according to:

$$PG\Big(\Big\{i \cdot 2^{k+1} - 1, (2 \cdot i - 1) \cdot 2^k\Big\}\ ,\ \Big\{(2 \cdot i - 1) \cdot 2^k - 1, (i - 1) \cdot 2^{k+1}\Big\}\Big)\ \forall 1 \leq i < \frac{n}{2^{k+1}}.$$

The resulting structure consists of $n - 1$ PG blocks, of which $\log n$ are only needed for the output carry generation. As explained above, this binary tree does only generate the group PG terms that are necessary to calculate carry for the MSB. The reversed tree is therefore inserted to the circuit below the initial binary tree, generating the remaining terms. In order to improve the readability of the equations describing the PG term generation, we count the stages of this subtree beginning with the output stage of the adder, i.e., the stage number l of the inverse tree is related to the stage of the complete adder k through $l = 2 \cdot \log n - 2 - k$. On the lowest level ($l = 0$), the inverse tree generates all even PG terms, i.e., $pg(\{2 \cdot i, 0\})\ \forall 1 \leq i < n/2$ as:

$$PG(\{2 \cdot i, 2 \cdot i\}\ ,\ \{2 \cdot i - 1, 0\})\ \forall 1 \leq i < n/2.$$

In general, the inverse tree in a BKA computes $\frac{n}{2^{k+1}} - 1$ PG terms in $\log n - 1$ stages as:

$$PG\Big(\Big\{i \cdot 2^{l+1} + 2^l - 1, i \cdot 2^{l+1}\Big\}\ ,\ \Big\{i \cdot 2^{l+1} - 1, 0\Big\}\Big)\ \forall 1 \leq i < \frac{n}{2^{l+1}} - 1.$$

The total number of PG terms in this lower tree is equal to $n - \log n - 1$.

When joining both tree structures, the last stage of the upper tree and the first stage of the lower tree can be combined in one stage, as there is no direct dependency between them. If no carry output is required, this step is trivial as the last stage of the upper tree can be omitted. As a result, a BKA can be realized in with a total of $2 \cdot (n - \log n - 1)$ PG blocks in $2 \cdot \log n - 2$ stages, when no carry output is computed.

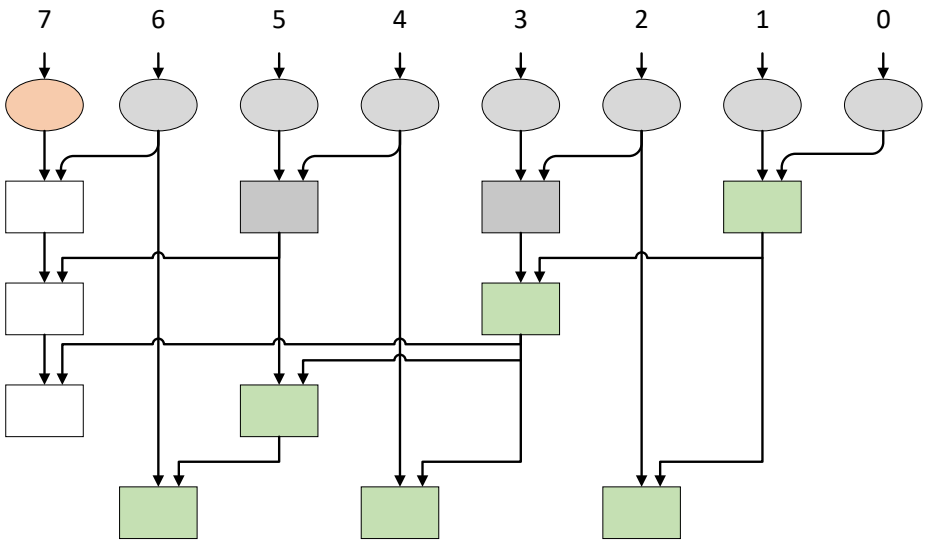


Figure 4. Complete 8-bit Brent–Kung Adder.

Masking BKAs

The asymptotic complexity of the number of PG blocks in a BKA is $\mathcal{O}(n)$ in comparison to $\mathcal{O}(n \cdot \log n)$ in the KSA and SA cases. This leads directly to smaller implementations if masking countermeasures are employed. Additionally, as the number of AND-gates has the same linear complexity, the required number of fresh random bits is reduced further to $d \cdot (d+1) \cdot (3 \cdot n - 4 \cdot \log n - 1)$. These improvements are bought with a higher latency due to BKAs requiring $2 \cdot \log n - 2$ stages, while KSAs and SAs can be realized in only $\log n$ stages.

However, if a first-order TI is used as the masking scheme, the randomness requirements are higher than in the KSA case. As BKAs have a maximal fan-out of $\log n - 1$, additional randomness is needed, similar to the SA case. Due to the tree-like structure of the BKA the randomness overhead is lower than for SA adders. This results in no additional randomness requirements in a 32-bit adder, as a sufficient number of shares can be re-used as described in Section 3.2.

4. Implementation Results and Discussion

This section provides implementation details for the proposed addition structures. We implemented 32-bit versions of the three algorithms as this size is commonly needed in cryptographic algorithms such as SALSA20. Note that adaptions to other widths are trivial. All designs were specified in VHDL and VERILOG and were synthesized for a Xilinx Spartan 6 XC6SLX75 Field-Programmable Gate Array (FPGA) with speed grade -3 using Xilinx ISE 14.7. In order to assure the correct realization of masked gadgets the relevant signals were exclude from optimizations and the design hierarchy was preserved. The implementation results are presented in Table 4.

Table 4. Implementation results for different 32-bit adder designs.

Design	LUTs	Flip-Flops	Freq. (MHz)	Latency	Rand. (Bit)
TI KSA [2]	937	1330	62	6	32
TI KSA	873	1416	228	6	31
1st-order HPC2-KSA	2936	3981	176	12	249
2nd-order HPC2-KSA	3915	8001	148	12	747
TI SA	579	1416	174	6	41
1st-order HPC2-SA	1801	3166	153	12	119
2nd-order HPC2-SA	1994	5979	128	12	357
TI BKA	487	2352	280	9	31
1st-order HPC2-BKA	1588	4317	173	18	74
2nd-order HPC2-BKA	1666	7122	158	18	222

4.1. TI Implementations

If first-order security is sufficient in an application, TI is a valid choice for a masking scheme. Regarding area, our results for the KSA adder are similar to the figures from [2]. The clock frequency that was estimated by the synthesis tool differs significantly from our results, although the authors performed their benchmark on an FPGA from the same family as we did. Different speed grades of the devices and different constraining of the synthesis might explain this discrepancy. Note that our implementation only uses 31 bit of randomness as the carry output is not computed. The SA design utilizes significantly less logic than the KSA, which can be attributed to the lower number of PG blocks. The number of flip-flops is the same as it is dominated by the number of stages in a pipelined architecture. When masking using TI, the BKA utilizes even less logic than the SA design but the higher number of stages lead to a higher flip-flop requirement. In a scenario where logic is expensive and high throughput is demanded but memory and latency are less important, the BKA is a suitable design.

4.2. HPC2 Implementations

If resistance against multivariate attacks is demanded, the KSA shows the worst performance in most categories. It should therefore only be considered in this scenario if the combination of latency and throughput are the primary optimization goals. The SA can effectively be utilized if the latency of the adder and the area requirements are the most important factors. The logic and memory requirements are significantly lower and the randomness requirement is less than half when compared to the KSA, although the throughput is slightly lower. If latency and memory are less important and the randomness consumption is critical, the BKA should be preferred. The logic area is lower and the randomness requirement can be further reduced by almost 40 % in comparison to the SA. Due to the differing asymptotic complexity this difference increases for larger widths.

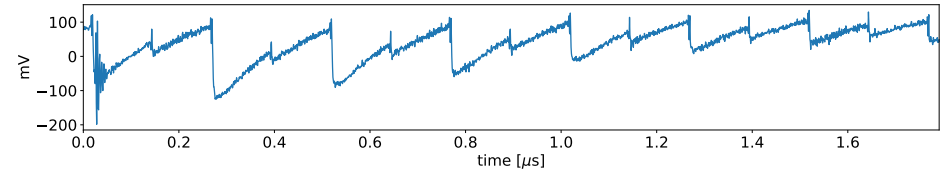
5. Side-Channel Evaluation

In this section, we provide a side-channel evaluation for the TI Sklansky- and Brent–Kung-Adders using the well-established TVLA [17,18]. As the authors of [2] already performed a side-channel evaluation of the TI-KSA we studied, we did not repeat this evaluation. Our HPC2-based adder designs rely on AND- and XOR-Gadgets, that are secure in the PINI model. Following this model, the gadgets can be composed arbitrarily in the presence of glitches. Note that the authors of [19] showed, that this composability can break if transitional leakage occurs. However, as our implementations are fully pipelined and successive inputs are assumed to be shared independently, this vulnerability does not apply in this case. The authors of [20,21] previously demonstrated the side-channel resistance of the HPC2 gadgets using TVLA. Therefore, we did not perform side-channel measurements for the HPC2 adder designs.

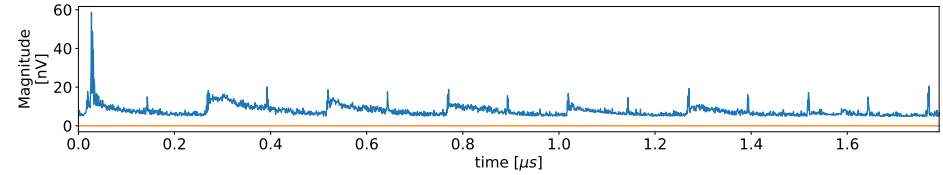
The power side-channel traces were obtained from a Sakura G [22] board specifically designed for side-channel evaluation. The clock rate was set to 4 MHz in order to collect clean traces. The internal amplifier of the board was combined with an external ZFL1000LN+ low-noise amplifier with a gain of approximately 21 dB. The samples were collected with an Spectrum M4i22 PCIe digitizer at a sample rate of $1.25\,\mathrm{GS\,s^{-1}}$. The internal 400 MHz low-pass filter was enabled for the purpose of preventing aliasing of the signal.

We applied the non-specific fixed-vs-random t-test described in [18] and the confidence-interval-based method proposed in [23] as the evaluation metrics in order to discern if the side-channel is data-dependent. For detailed explanations of these methods the reader is referred to the original publications. The analysis was performed for the first and second orders using 300 million traces at a significance level of $\alpha = 0.01$ adjusted for family-wise error rate. The evaluation results are depicted in Figures 5 and 6.

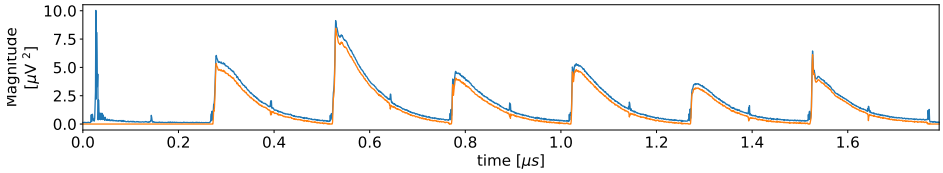
As shown in Figure 5d, no first-order leakage can be detected in the SA implementation with 300 million measurements. Following the plot in Figure 5b if any (undetected) first-order leakage is present, it is below 60 nV after amplification. Second order leakage is clearly detectable, which is expected as the TI masking scheme with three shares only provides first-order security. Similar to the SA case, the BKA does not exhibit detectable first order leakage, but is potentially vulnerable against second-order attacks.



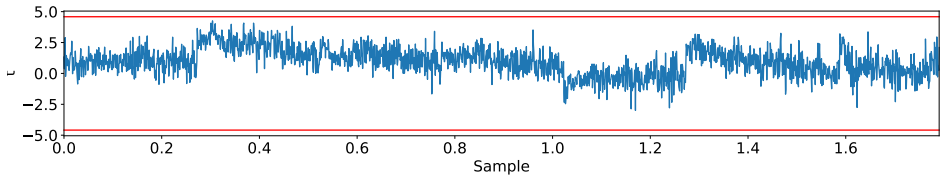
(**a**) sample trace



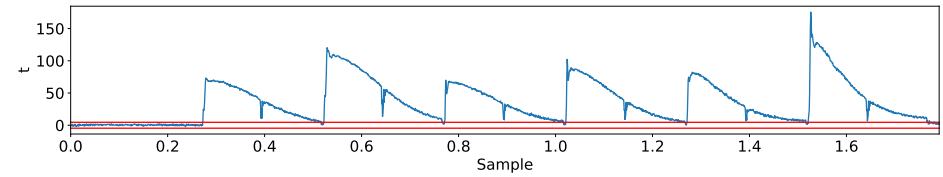
(**b**) 1st-order confidence intervals



(**c**) 2nd-order confidence intervals

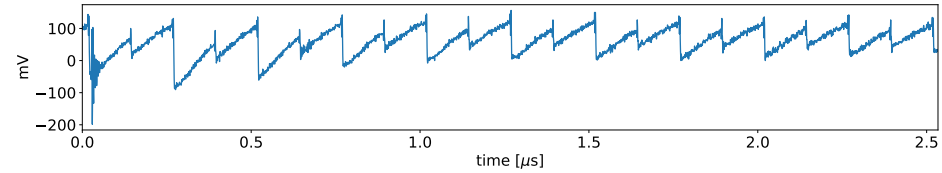


(**d**) 1st-order *t*-test

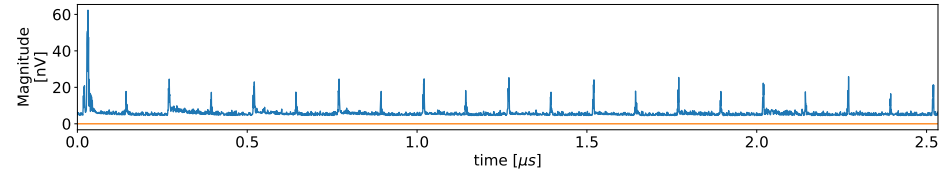


(**e**) 2nd-order *t*-test

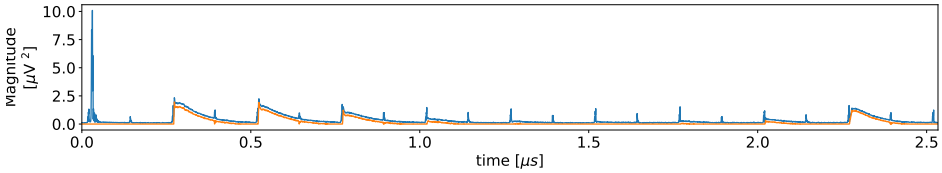
Figure 5. A sample power trace, confidence intervals, and *t*-test results for the TI Sklansky Adder using 300 million traces. No first-order leakage is detectable.



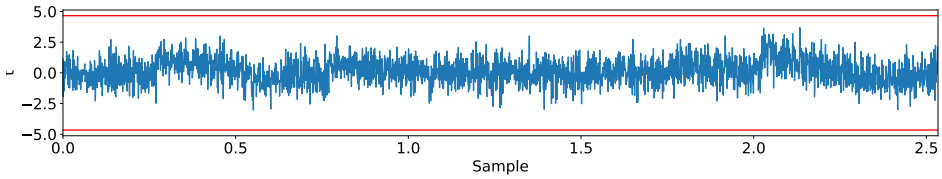
(**a**) sample trace



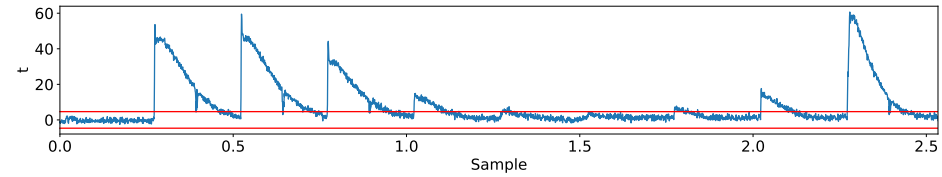
(**b**) First-order confidence intervals



(**c**) Second-order confidence intervals



(**d**) First-order *t*-test



(**e**) Second-order *t*-test

Figure 6. A sample power trace, confidence intervals, and *t*-test results for the TI Brent–Kung Adder using 300 million traces. No first-order leakage is detectable.

6. Conclusions

In this work we studied three adder designs regarding their suitability for Boolean masking. The algorithms where masked with the TI scheme for first-order security and with HPC2 gadgets that provide resistance against multivariate attacks. After a detailed complexity analysis and practical realization on an FPGA we found different scopes of application for the algorithms. The KSA can effectively be used to achieve univariate security. If randomness requirements and throughput are less important than area, an SA can be considered. In the multivariate case the SA provides the lowest latency, while the BKA can reduce logic area and randomness requirements and improve the throughput at the cost of latency and memory. In future work a study of non-pipelined adder designs for low throughput requirements would be interesting.

Author Contributions: Methodology, F.B. and T.G.; investigation, F.B.; resources, T.G.; writing—original draft preparation, F.B.; writing—review and editing, F.B.; supervision, T.G. All authors have read and agreed to the published version of the manuscript.

Funding: We acknowledge support by the DFG Open Access Publication Funds of the Ruhr-Universität Bochum.

Institutional Review Board Statement: Not applicable.

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

References

1. Cassiers, G.; Grégoire, B.; Levi, I.; Standaert, F. Hardware Private Circuits: From Trivial Composition to Full Verification. *IEEE Trans. Comput.* **2020**, *2020*, 185. [CrossRef]
2. Schneider, T.; Moradi, A.; Güneysu, T. Arithmetic Addition over Boolean Masking-Towards First- and Second-Order Resistance in Hardware. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9092, pp. 559–578.
3. Reparaz, O. A note on the security of Higher-Order Threshold Implementations. *IACR Cryptology ePrint Archive*. 2015. Available online: https://eprint.iacr.org/2015/001 (accessed on 12 February 2022).
4. Duc, A.; Faust, S.; Standaert, F. Making Masking Security Proofs Concrete-Or How to Evaluate the Security of Any Leaking Device. EUROCRYPT (1). In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9056, pp. 401–429.
5. Reparaz, O.; Bilgin, B.; Nikova, S.; Gierlichs, B.; Verbauwhede, I. Consolidating Masking Schemes. CRYPTO (1). In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2015;Volume 9215, pp. 764–783.
6. Groß, H.; Mangard, S.; Korak, T. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016, Vienna, Austria, 24 October 2016; Bilgin, B., Nikova, S., Rijmen, V., Eds.; ACM: New York, NY, USA, 2016; p. 3. [CrossRef]
7. Nikova, S.; Rechberger, C.; Rijmen, V. Threshold Implementations Against Side-Channel Attacks and Glitches. ICICS. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4307, pp. 529–545.
8. Bilgin, B.; Gierlichs, B.; Nikova, S.; Nikov, V.; Rijmen, V. Higher-Order Threshold Implementations. In *Advances in Cryptology–ASIACRYPT 2014*; Sarkar, P., Iwata, T., Eds.; Number 8874 in Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2014; pp. 326–343. [CrossRef]
9. Ishai, Y.; Sahai, A.; Wagner, D. Private Circuits: Securing Hardware against Probing Attacks. In Proceedings of the 23rd Annual International Cryptology Conference, Santa Barbara, CA, USA, 17–21 August 2003; Advances in Cryptology-CRYPTO 2003; Springer: Berlin, Heidelberg, Germany, 2003; pp. 463–481.
10. Coron, J.; Prouff, E.; Rivain, M.; Roche, T. Higher-Order Side Channel Security and Mask Refreshing. FSE. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2013; Volume 8424, pp. 410–424.
11. Barthe, G.; Belaïd, S.; Dupressoir, F.; Fouque, P.; Grégoire, B.; Strub, P.; Zucchini, R. *Strong Non-Interference and Type-Directed Higher-Order Masking*; CCS; ACM: New York, NY, USA, 2016; pp. 116–129.
12. Cassiers, G.; Standaert, F. Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 2542–2555. [CrossRef]
13. Cassiers, G. "fullVerif". Available online: https://github.com/cassiersg/fullverif (accessed on 30 September 2021).
14. Kogge, P.M.; Stone, H.S. A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations. *IEEE Trans. Comput.* **1973**, *22*, 786–793. [CrossRef]
15. Sklansky, J. Conditional-Sum Addition Logic. *IRE Trans. Electron. Comput.* **1960**, *9*, 226–231. [CrossRef]
16. Brent, R.P.; Kung, H.T. A Regular Layout for Parallel Adders. *IEEE Trans. Comput.* **1982**, *31*, 260–264. [CrossRef]
17. Goodwill, G.; Jun, B.; Jaffe, J.; Rohatgi, P. A testing methodology for side channel resistance validation. In Proceedings of the NIST Non-Invasive Attack Testing Workshop, Nara, Japan, 26–27 September 2011; NIST CSRS: Gaithersburg, MD, USA, 2011.
18. Schneider, T.; Moradi, A. Leakage Assessment Methodology-A Clear Roadmap for Side-Channel Evaluations. CHES. In *Lecture Notes in Computer Science*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9293, pp. 495–513.
19. Cassiers, G.; Standaert, F. Provably Secure Hardware Masking in the Transition- and Glitch-Robust Probing Model: Better Safe than Sorry. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**, *2021*, 136–158. [CrossRef]
20. Knichel, D.; Moradi, A.; Müller, N.; Sasdrich, P. Automated Generation of Masked Hardware. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2022**, *2022*, 589–629. [CrossRef]
21. Müller, N.; Knichel, D.; Sasdrich, P.; Moradi, A. Transitional Leakage in Theory and Practice-Unveiling Security Flaws in Masked Circuits. *IACR Cryptology ePrint Archive*. 2022; p. 23. Available online: https://eprint.iacr.org/2022/023 (accessed on 12 February 2022).

22. Side-Channel AttacK User Reference Architecture. Available online: http://satoh.cs.uec.ac.jp/SAKURA/index.html (accessed on 12 February 2022).
23. Bache, F.; Plump, C.; Güneysu, T. Confident leakage assessment-A side-channel evaluation framework based on confidence intervals. In Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 19–23 March 2018; pp. 1117–1122.