# X2X: Low-Randomness and High-Throughput A2B and B2A Conversions for $d + 1$ shares in Hardware

Quinten Norga, Jan-Pieter D'Anvers, Suparna Kundu, and Ingrid Verbauwhede

COSIC, KU Leuven, Leuven, Belgium
{firstname}.{lastname}@esat.kuleuven.be

**Abstract.** The conversion between arithmetic and Boolean masking representations (A2B & B2A) is a crucial component for side-channel resistant implementations of lattice-based (post-quantum) cryptography. In this paper, we first propose novel $d$-order algorithms for the secure addition ($\texttt{SecADDChain}_q$) and B2A (`B2X2A`). Our secure adder is well-suited for repeated ('chained') executions, achieved through an improved method for repeated masked modular reduction. The optimized `B2X2A` gadget removes a full secure addition compared to state-of-the-art B2A approaches, by relying on the `X2B` operation. This component directly converts a simultaneously Boolean and arithmetically shared variable to $d + 1$ Boolean shares. This approach reduces the required amount of `SecADD`s to $2d$, of which $2 \cdot \lceil \log_2(d) \rceil$ are max-order.
Secondly, we develop both a first- and high-order masked, unified hardware implementation that can compute both A2B & B2A conversions for power-of-two ($p$) and prime ($q$) moduli. Compared to state-of-the-art (high-throughput) hardware implementations that only support $\texttt{A2B}_p$, we reduce area utilization for a second-order implementation by 45% up to 60% and fresh randomness up to 62%, while supporting all four types of additive mask conversions. Our first-order design only requires 1,133/2,170 [LUT/FF] on Kintex-7 FPGAs.
Our proposed algorithms are proven secure in the $d$-probing model and their implementations are validated via practical lab analysis using the TVLA methodology. We experimentally show that our masked implementation is hardened against first- and second-order univariate and multivariate power-based side-channel attacks using 100 million traces, for each mode of operation.

**Keywords:** PQC · Hardware · Masking · Side-Channel Analysis

## 1 Introduction

The security of currently deployed public key cryptographic algorithms is typically based on the integer factorization or elliptic curve discrete logarithm problem. The threat of large-scale quantum computers is ever-increasing, potentially

leaving current algorithms and their implementations vulnerable to potential quantum attacks [61] in the (near) future. The term 'Post-Quantum Cryptography' (PQC) encompasses all alternative cryptographic algorithms, that can resist these attacks and are soon to replace vulnerable algorithms and their implementations.

The National Institute of Standards and Technology (NIST) has recognized the need for replacing the existing public-key standards. It launched an initial PQC standardization effort in 2016 [52] and is continuing with an additional Digital Signature (DS) competition, launched in 2023 [54]. Noticeably, lattice-based schemes and their promising security and performance features, are popular candidates for both competitions. The final Kyber [50] and Dilithium [49] standards were published in Summer 2024, while seven out of 40 accepted submissions for the PQC DS competition (Round 1) are lattice-based [54]. One of the challenges for the deployment of new post-quantum schemes is protection against (physical) side-channel attacks.

Side-Channel Analysis (SCA) attacks aim at extracting sensitive information from electronic devices performing security-critical applications, by observing the physical characteristics of the calculations. First discovered and published by Kocher [38] in 1996, many types of physical behavior exist and can be abused by adversaries: execution time, instantaneous power consumption [39] or Electromagnetic (EM) radiation [29]. The security and confidentiality of a cryptographic implementation can be completely broken if its physical characteristics correlate to a secret key, typically called (side-channel) *leakage*. Many insecure implementations, including of lattice-based schemes, have been successfully attacked using side channels [22,35,55,56,63]. Therefore, protection against SCA attacks is a critical factor for the security of a physical device and remains an open challenge in academia and industry.

Masking is an algorithmic and well-studied approach for protecting cryptographic hardware or software implementations against (passive) EM or power side-channel attacks. Following the concept of secret sharing by Shamir et al. [60], a sensitive variable $x$ is split into $(d+1)$ uniform random shares $(x^{\{i\}})$ for achieving security order $d$. Each of the shares separately is independent to the secret and only if an adversary combines information of all $d+1$ shares, it can learn something about the original secret $x$. The masking countermeasure [37,51,57,32,33] is popular because it can provide physical security through formal security and adversary models.

Masking the operations of lattice-based crypto schemes requires a mix of both Boolean and arithmetic mask representations. More precisely, polynomial multiplication and addition are preferably performed on arithmetic shares, whereas hashing (Keccak) inherently is a bitwise operation and thus prefers Boolean masking. Hence, there is a need for converting between both sharing types: from arithmetic to Boolean (A2B) and Boolean to arithmetic (B2A). These conversions are costly, even more so at higher protection orders, and are one of the major bottlenecks in masked implementations.

**Related Work.** Masking techniques have been applied to lattice-based cryptography in other work, mostly targeting software implementations. This includes PQC candidates Dilithium [45], Saber [6,20,41,27], Kyber [7,36,27,9] and NTRU [21,40]. A first-order A2B conversion was originally proposed by Goubin in [31], with Coron et al. proposing higher-order conversions [16,17] for power-of-two moduli. They propose to construct the A2B conversion from the Secure Addition (`SecADD`) operation, which can be seen as an arithmetic addition of two Boolean shared variables.

However, most lattice-based schemes (incl. Kyber and Dilithium) operate on polynomials with coefficients modulo a prime integer $q$. A secure addition modulo a prime integer $q$ ($\texttt{SecADD}_q$) can be constructed from a regular `SecADD` and additional explicit modular reduction. This expensive procedure typically involves a combination of additional `SecADD`'s and Secure Multiplexers (`SecMUX`). Techniques for the $\texttt{A2B}_q/\texttt{B2A}_q$ operation have been proposed by Barthe et al. [5] and in [59]. An alternate approach for $\texttt{A2B}_q/\texttt{B2A}_q$ was proposed in [9,45], where first a modulus conversion from a prime integer ($q$) to a power-of-two one ($p$) is performed after which the masked operations are performed.

More recently, table-based approaches have received more attention as they are becoming viable for high-order conversions [62,20,25], yet not as efficient as computational approaches. These techniques are out-of-scope for this work.

**Contribution.** We propose improvements from the algorithmic level down to the circuit level, applicable to arbitrary protection orders.

- State-of-the-art $\texttt{SecADD}_q$ strategies require $3 \times$ `SecADD` or $2 \times$ `SecADD` and a `SecMUX`. We propose a novel gadget, $\texttt{SecADDChain}_q$, which utilizes *interleaved modular reduction* and can be efficiently chained for repeated executions, requiring $2 \times$ `SecADD` at all protection orders (Section 3).
- `B2A` conversions are typically computed by combining an A2B operation with expensive *pre-and post-processing stages*. Our `B2X2A` gadget consists of a simplified post-processing stage (without `SecADD`) and the novel `X2B`, which directly converts a mix of arithmetic and Boolean shares to an equivalent Boolean sharing (Section 3). By operating on a mix of arithmetic and Boolean shares, $d{+}1$ Boolean shares are directly computed instead of through an A2B and a $d$-order `SecADD`. Compared to state-of-the-art techniques, our approach requires 2 or 3 fewer `SecADDs` at second protection order and 2 up to 4 fewer `SecADD` operations at third order.
- Through *careful, manual masking* of all operations, we significantly reduce the masking overhead (area, latency, randomness). Our `SecADD` implementation requires 50% fewer random bits and clock cycles by not relying on universally composable gadgets, but is paired with an increased verification cost (Section 4 & 5). Additionally, we show that *half-cycle datapaths* can reduce the total execution time of highly non-linear, masked operations from 36% to 42% ($d = 1$) and 42% to 47% ($d = 2$).
- The side-channel resistance of our implementation is formally proven and experimentally verified in our Security Evaluations Lab using the Test Vector Leakage Assessment (TVLA) methodology (Section 5).

– Our RTL source code is made publicly available at https://github.com/KULeuven-COSIC/X2X.

Our unified, streaming hardware architecture can be *dynamically* configured to perform any type of mask conversion: $\texttt{A2B}_{2^k}/\texttt{A2B}_q/\texttt{B2A}_{2^k}/\texttt{B2A}_q$. Our work is directly applicable to any lattice-based PQC scheme, we specifically target Kyber parameters in our unified implementation. To the best of our knowledge, our design strategy results in the lowest overhead cost (latency, fresh randomness and area) compared to the current state-of-the-art.

## 2 Background & Preliminaries

### 2.1 Notation

A bit position (index) is indicated by the subscript, with the LSB at bit 0 ($x_0$) and MSB at position $k-1$ ($x_{k-1}$) for $k$-bit data words. For power-of-two moduli ($2^p$), $k = p$; for prime moduli $q$, the word width is $k = \lceil \log_2(q) \rceil$. All operations and units/costs are expressed in terms of $k$-bit data words/shares, unless explicitly specified. Rounding up to the next integer is denoted by $\lceil \cdot \rceil$.

### 2.2 Arithmetic, Boolean and Composite Sharing

At protection order $d$, a secret value $x \in \mathbb{F}_k^n$ is arithmetically masked by converting it into $d+1$ shares $x^{\{0:d\}}$, such that $x = \sum_{i=0}^{d} x^{\{i\}}$ modulo a predefined integer $q$. For Boolean masking, the sharing of a secret value $x$ can be reconstructed as $x = \bigoplus_{i=0}^{d} x^{\{i\}}$. A negation (`SecNOT`) ($\sim$) on Boolean shared data is equivalent to performing binary invert on a single share. Throughout this work, all sharing is considered uniformly random.

We introduce the term *composite* sharing for secret values that consist of a combination of arithmetic and Boolean shares. $x^{\{a,b\}}$ corresponds to a secret value $x$ consisting of $a$ arithmetic shares, each shared as $b$ Boolean shares. Or alternatively: $x = \sum_{i=1}^{a}(\bigoplus_{j=1}^{b} x^{\{i,j\}})$ with a (total) masking order $d = (a*b)-1$. Note that Boolean masking can be seen as a special form of composite masking where all $d+1$ Boolean shares belong to the same arithmetic share ($a = 1, b = d+1$). An arithmetically shared variable consists of $d+1$ arithmetic shares ($a = d+1, b = 1$).

### 2.3 (Extended) *d*-Probing Model

The most prominent and well-studied adversary and security model, the Ishai, Sahai, and Wagner (ISW) $d$-probing model [37], aims at capturing the capabilities of real-world adversaries. In such a context, the adversary can probe and observe up to $d$ wires (intermediate values) of an ideal (glitch-less) circuit performing sensitive operations. In this model, a (masked) circuit is $d^{th}$- order probing secure if and only if the information gained from $d$ (noise-free and instantaneous) probes does not reveal any information of any secret variable.

However, the discrepancy between theoretical and practical security has been shown to be problematic in the case of the original ISW $d$-probing model. This has resulted in the compromised security of theoretically secure designs and implementations [43,48]. An extended (and more robust) security model that captures different physical effects (naturally) present in digital logic circuits (CMOS) and hardware, was proposed by Faust et al. in [26]. It introduces glitch-extended [43,44], transition-extended [15,2] and coupling-extended probes [23], and incorporate such (natural) physical defects as part of the adversarial model.

### 2.4 Masking: a Side-Channel Leakage Countermeasure

By introducing masking countermeasures, an attacker can only obtain information about any sensitive value if they have access to all shares at once, while an incomplete set of shares results in strictly random information.

Following the Domain-Oriented Masking (DOM) scheme [33], $d$-order secure masked circuits are achieved by splitting sensitive variables into $d+1$ (independent) shares. 'Manually' creating a complex $d$-probing secure circuit, consisting of multiple such secure (DOM) gates, requires careful analysis. Several security notions for composability have been proposed: Non-Interference (NI) [3] and Strong Non-Interference (SNI) [4] in the presence of glitches (and transitions) in hardware.

**Definition 1 (t-Non-Interference [4]).** *A gadget with one output sharing and $m_i$ input sharings is t-Non-Interferent (t-NI) if any set of at most $t_1$ probes on its internal wires and $t_2$ probes on wires from its output sharings such that $t_1 + t_2 \leq t$ can be simulated with $t_1 + t_2$ shares of each of its $m_i$ input sharings.*

**Definition 2 (t-Strong-Non-Interference [4]).** *A gadget with one output sharing and $m_i$ input sharings is t-Strong Non-Interferent (t-SNI) if any set of at most $t_1$ probes on its internal wires and $t_2$ probes on wires from its output sharings such that $t_1 + t_2 \leq t$ can be simulated with $t_1$ shares of each of its $m_i$ input sharings.*

A different approach is based on 'trivial composability' and the security notion of Probe-Isolating Non-Interference (PINI) [10]: HPC gadgets [14], which are derived from the DOM scheme. Introduced by Cassiers et al., the proposed gadgets can be instantiated at arbitrary protection orders and trivially combined into a larger circuit. In general, trivial composability and its low verification cost and guaranteed $d$-probing security comes at a high (overhead) cost, due to being overly conservative in applying certain countermeasures.

We target 'optimized composition' in the $d$-probing model in this work using glitch-robust (DOM) gates and avoiding transitional leakage through a fully pipelined design. As a result, the overhead introduced when masking A2B/B2A operations, is significantly reduced compared to strictly using (PINI) HPC gadgets. This manual analysis and algorithmic masking of operations results in a lower overhead, but requires a higher verification cost and can be error-prone in the case of larger and more complex circuits [46].

### 2.5 Masking Lattice-based PQC: ML-KEM

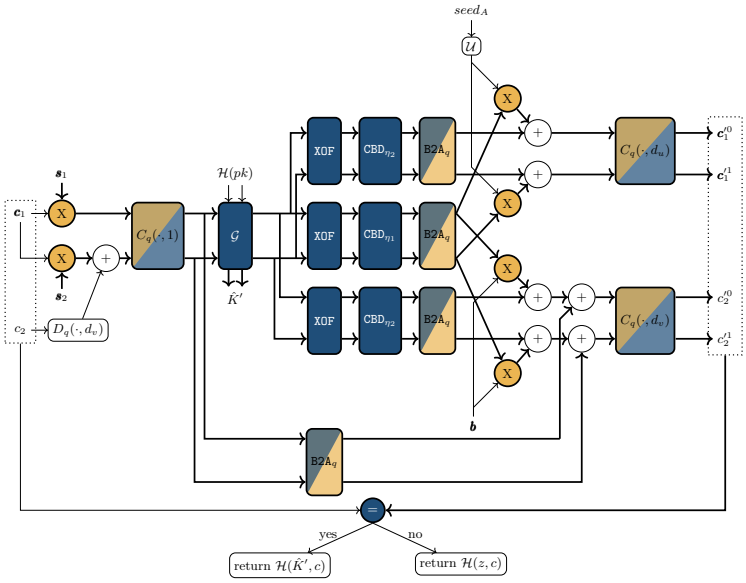



**Fig. 1.** The masked Decapsulation procedure for ML-KEM (FIPS 203). Operations that require Boolean masking are highlighted in blue, operations that prefer arithmetic masking are highlighted in yellow. Mask conversions are required to convert between these representations. [27]

Figure 1 illustrates the impact of different mask domains and the need for switching between both during several sub-operations for Kyber (or ML-KEM). The decryption procedure requires performing Boolean operations, like binomial sampling and hashing. The re-encryption stage requires performing polynomial multiplication, which is an arithmetic operation, after which a masked comparison is performed in the Boolean domain. Note again, these conversions are extremely costly and result in being (one of) the main contributor(s) of run time latency. For the pseudocode of the full algorithms of all (future) PQC standards, we refer to the initial/final NIST FIPS standards [53].

## 3 Optimized Secure Gadgets for Mask Conversions

The following section follows a bottom-up approach. Firstly, we optimize the secure addition ($\texttt{SecADD}_q$) by focusing on the masked modular reduction, requiring strictly $2 \times$ SecADDs in total, even when chained repeatedly. Our strategy can be directly applied for arbitrary moduli $q$ and arbitrary protection orders $d$, including Kyber ($q = 3329$) and Dilithium ($q = 8380417$).

Secondly, we propose a novel B2A gadget: `B2X2A`. We introduce a new primitive `X2B`, which operates on composite shares and eliminates a full secure addition from the post-processing stage, reducing the latency and randomness requirements at all protection orders. Compared to the A2B, which operates on an arithmetically shared variable, the `X2B` gadget can convert a mix of arithmetic and Boolean shares to $d+1$ boolean shares. The composability of all proposed gadgets is proven secure in the $d$-probing model and tested in TVLA setting. Table 1 gives an overview of all the gadgets used throughout this work, including a short description of their functionality and assumed security properties, on which we rely for proving their security in larger compositions.

**Table 1.** Overview of used gadgets in this work, with $t = d + 1$ shares. All gadgets operate on full ($k$-bit) data words, unless explicitly stated.

| Algorithm | Description | Security | Reference |
|---|---|---|---|
| `SecXOR` | Masked logical XOR | $t$-NI | Alg. 6 |
| `SecNOT` | Masked Boolean negation | $t$-NI | Alg. 7 |
| `SecAND` | Masked logical AND | $t$-NI | [33] & Alg. 8 |
| `SecOR` | Masked logical OR | $t$-NI | Alg. 9 |
| `SecREF` | Strong refresh of Boolean masking | $t$-SNI | [4,14] & Alg. 11 |
| `SecEXP` | Doubling/Expanding of Boolean shares | $t$-NI | [17] & Alg. 10 |
| `SecADD` | Arithmetic addition of Boolean shares | $t$-NI | [5] & App. 4.1 |
| `FullXOR` | Refresh and combine Boolean shares | $t$-NI | [5] & Alg. 13 |
| $\texttt{SecAddChain}_q$ | Arithmetic addition mod (prime) $q$ of Boolean shares | $t$-NI | Alg. 2 |
| `A2B` | Arithmetic to Boolean mask conversion | $t$-NI | [16] & Alg. 3 |
| `X2B` | Composite to Boolean mask conversion | $t$-NI | Alg. 4 |
| `B2X2A` | Boolean to arithmetic mask conversion | $t$-NI | Alg. 5 |

### 3.1 Secure Addition (SecADD)

The secure addition is equivalent to performing an arithmetic addition ($s = x+y \mod q$) in the Boolean domain (Equation 1). As we will demonstrate, it serves as the primary building block for higher-order mask conversions.

$$s^{\{0:d\}} = x^{\{0:d\}} + y^{\{0:d\}} \bmod q = \bigoplus_{i=0}^{d} x^{\{i\}} + \bigoplus_{i=0}^{d} y^{\{i\}} \bmod q \tag{1}$$

For power-of-two moduli ($p = 2^k$), the modular reduction is taken care of implicitly during computation. However, prime moduli ($q$) require explicit (masked) modular reduction. We show that our $d-$order $\texttt{SecADDChain}_q$ gadget outperforms state-of-the-art approaches when performing multiple secure additions in succession (which is the case in mask conversions). We first highlight two (costly) approaches from literature, which rely on an additional `SecMUX` or `SecADD` to complete the masked modular reduction.

**SecMUX-based Modular Reduction** Barthe et al. [5] introduced a simple, yet costly method for performing the $\texttt{SecADD}_q$ at arbitrary protection orders. It requires calculating both $s = x + y$ and $s' = x + y - q$ securely, one of which will be in range $[0, q)$. A costly `SecMUX` (Equation 2) securely selects the desired shared data ($s$ or $s'$) that lies in the $[0 : q - 1]$ interval, based on the carry bits $c$ (index $k - 1$).

$$\texttt{SecMUX}(s^{\{0:d\}}, s'^{\{0:d\}}, c_k^{\{0:d\}}) = \texttt{SecXOR}(\texttt{SecAND}(s, c), \texttt{SecAND}(s', \texttt{SecNOT}(c))) \quad (2)$$

**SecADD-based Modular Reduction** Subsequently, Fritzmann et al. [27] introduced a method for performing a first-order $\texttt{SecADD}_q$, which does not involve a `SecMUX` gadget and which we extend to arbitrary protection orders. By pre-processing the input data, which requires access to the initial masking of either input $y$ (or $x$), the `SecMUX` operation can be removed.

In practice, we need one of the inputs to be in range $[-q, 0)$. This can be achieved by subtracting $q$ from one of the inputs before it is shared: $y' = y - q$. The first `SecADD` operates on $y'$ and $x$ (or $y$ and $x'$) and computes $z = x + y'$ (Alg. 1, Line 1). Next, a correction term $c'$ is constructed and is added to this intermediate result $z$, ensuring that the result of the second `SecADD` $s = z + c' = x + y - q(+q)$ lies in $[0, q)$ (Line 4).

**Algorithm 1** $\texttt{SecADD}_q$ (without `SecMUX`) (extended from [27])

**Input Parameter** : $q$ ▷ $q$ is prime
**Input Data** : $x^{\{0:d\}}$ and $y'^{\{0:d\}} = y^{\{0:d\}} + (2^k - q)$ ▷ Initial masking.
**Output Data** : $s^{\{0:d\}}$ such that $s = x + y \bmod q$

1: $z^{\{0:d\}} \leftarrow \texttt{SecADD}(x^{\{0:d\}}, y'^{\{0:d\}})$
2: $c_0^{\{0:d\}} \leftarrow z^{\{0:d\}} \gg (k - 1)$ ▷ Carry bit (share-wise).
3: $c'^{\{0:d\}} \leftarrow c^{\{0:d\}} \cdot q$ ▷ Share-wise.
4: $s^{\{0:d\}} \leftarrow \texttt{SecADD}(z^{\{0:d\}}, c'^{\{0:d\}})$

Note that the modular reduction and the construction of $c'$ now is a linear (e.g. mask-friendly) operation, except for the secure additions themselves and can be generalized for $d + 1$ shares. There is no longer any need to explicitly select the correct result, using a `SecMUX`.

The main issue with this method arises when one of the Boolean masked inputs is not in range $[-q, 0)$. This is the case if the output of a $\texttt{SecADD}_q$ operation, in range $[0, q)$, is directly used as the input for another $\texttt{SecADD}_q$, as is the case during an A2B (and B2A) conversion. To subtract $q$ from a Boolean masked variable, an additional secure addition with $(2^k - q)$ is required, as proposed in [11] (Algorithm 11). As a result, a full $\texttt{SecADD}_q$ now requires three `SecADD`s, which is costly in the context of mask conversions. More details are provided in Table 2.

**Algorithm 2** $\texttt{SecADDChain}_q$ $[t-\text{NI}]$

**Input Parameter** : $q$ ▷ $q$ is prime
**Input Data** : $x^{\{0:d\}}$ and $y'^{\{0:d\}} = y^{\{0:d\}} + (2^k - q)$ ▷ Initial masking.
**Output Data 1** : $s^{\{0:d\}}$ such that $s = x + y \bmod q$
**Output Data 2** : $s'^{\{0:d\}}$ such that $s' = s + (2^k - q)$

1: $z^{\{0:d\}} \leftarrow \texttt{SecADD}(x^{\{0:d\}}, y'^{\{0:d\}})$
2: $cc_0^{\{0:d\}} \leftarrow z^{\{0:d\}} \gg (k-1)$ ▷ Carry bit.
3: $cc_0^{\{0:d\}} \leftarrow \texttt{SecREF}(cc_0^{\{0:d\}})$ ▷ 1-bit
4: **if** final $\texttt{SecADD}_q$ **then**
5: $\quad c^{\{0:d\}} \leftarrow cc^{\{0:d\}} \cdot q$ ▷ Share-wise.
6: $\quad s^{\{0:d\}} \leftarrow \texttt{SecADD}(z^{\{0:d\}}, c^{\{0:d\}})$ ▷ in $[0:q)$
7: **else**
8: $\quad c'^{\{0:d\}} \leftarrow \texttt{SecNOT}(cc^{\{0:d\}}) \cdot (-q)$ ▷ Share-wise.
9: $\quad s'^{\{0:d\}} \leftarrow \texttt{SecADD}(z^{\{0:d\}}, c'^{\{0:d\}})$ ▷ in $[-q:0)$
10: **end if**

**Interleaved Modular Reduction for Chained SecADDs** We propose the $\texttt{SecADDChain}_q$ gadget (Algorithm 2), which requires only two SecADD operations and is specifically useful in the context of chained additions, as is typically the case for A2B and B2A conversions. To achieve efficient chaining of $\texttt{SecADD}_q$ we provide two possible outputs: one is calculated if the $\texttt{SecADD}_q$ is one of many subsequent secure additions that need to be calculated, or the other if it is the final one in the chain.

For the algorithm in the previous section, if the secure addition is the final operation, the goal is to calculate $s = x + y$ which lies in $[0, q)$ with $x$ and $y$ consisting of $d+1$ Boolean shares. As described above, this can be achieved using strictly two SecADDs (Alg. 1 & Alg. 2, Line 6) if one of the inputs is pre-processed: $y' = y - q$.

If another $\texttt{SecADD}_q$ needs to be performed subsequently, the result of the operation needs to be pre-processed (by subtracting $q$) as it is one of the inputs of the next $\texttt{SecADD}_q$. Instead of doing this explicitly, our novel gadget $\texttt{SecADDChain}_q$ allows for this to be computed directly. The result will now be $s' = s + (2^k - q)$ (Alg. 2, Line 9), which lies in $[-q, 0)$, allowing for the output to be used directly as an input for the next $\texttt{SecADD}_q$.

More specifically, first $z = x + y' (= x + y - q)$ is calculated. Using this intermediate result $z$, a different correction term $c'$ is constructed in Line 8: $c' = (\sim z_{k-1}) \cdot (-q)$. This term is eventually added together with the intermediate result, in order to obtain the final result: $s' = z + c'$. Intuitively, if the intermediate result lies in $[-q, 0)$, the unshared correction term should be zero. If positive, $-q$ should be added back to the intermediate result in order to ensure the final result ($s'$) lies in $[-q, 0)$. This is achieved by using the Boolean inverse of the carry bits ($z_{k-1}$) as a share-wise select signal for a multiplexer. If an uneven amount of carry bits are one, the unshared value is negative and an even amount of shares in $c'$ is set to $-q$. As a result, the unshared $c'$ is equal to zero, which is desired.

This extension allows for multiple secure additions to be directly chained in succession, without the need for repeated and explicit pre-processing of one of the inputs and thus strictly requiring two SecADDs. Such a thing is useful for

$\text{A2B}_q$/$\text{B2A}_q$ conversions, as the masked modular reduction is interleaved during successive operations. The only time when access to the initial masking is required is one of the inputs, $y$, of the very first secure addition of which many are performed in succession. The input is corrected with $-q$ before the initial sharing, so that $\bigoplus_{i=0}^{d} y'^{\{i\}} = y - q$. If not possible, a one-time pre-processing using the `SecADD` is required.

***d*-Probing Security:** We show that the $\texttt{SecADDChain}_q$ gadget is correct and prove it to be $t-$NI, considering the leakage effects from Section 2.3 in Appendix A.

### 3.2 B2A

A method for converting $d+1$ Boolean shares to $d+1$ arithmetic shares (mod $2^k$) was introduced in [17] and extended for arbitrary moduli $q$ in [5]. Generally speaking, the B2A conversion is equivalent to an A2B operation with additional (costly) pre- and post-processing stages. We make several modifications to this procedure and propose a more efficient B2A conversion routine in Algorithm 5: `B2X2A`. It relies on the conversion (e.g., addition) of composite sharing to Boolean sharing through the `X2B`, to remove an additional secure addition in the post-processing. Correctness and security proofs are also provided. We conclude by comparing the overhead of published work and our methods.

**X2B (and A2B)** A B2A operations requires adding a $d+1$ Boolean shared variable with a $d+1$ arithmetically shared one (with one zero share). Traditionally, this is solved by first converting the second value to a $d+1$ Boolean shared variable using an A2B. Secondly, both Boolean shared variables can be securely added together (`SecADD`). Instead, we propose the `X2B` primitive, which is a variant of the A2B proposed in [17] but operates on a (specific) *composite* sharing $z^{\{0:d\}}$ (Eq. 3) and obtains an equivalent $d+1$ Boolean sharing. It directly adds arithmetic and Boolean shared variables together using secure additions that operate on different levels of Boolean sharing, exploiting the structure of the B2A operands. As we will demonstrate in the next section, this approach is more efficient for B2A operations compared to the state-of-the-art, where all secure additions and conversions operate on strictly identical share counts.

As described in Equation 3, the input of `X2B` consists of a specific mix of arithmetic and Boolean sharing: all $d+1$ shares of $z^{\{0:d\}}$ are arithmetically shared, but one share $z^{\{0\}}$ consists in turn of a number of Boolean shares ($2d+1$ shares in total):

$$z^{\{0:d\}} = z^{\{0,0:d\}} + z^{\{1:d\}} = \bigoplus_{i=0}^{d} z^{\{0,i\}} + \sum_{j=1}^{d} z^{\{j\}} \tag{3}$$

The full `X2B` algorithm (and a comparison with the `A2B`) are shown in Algorithms 3 & 4. All terms are added using a tree-like structure ([17], Algorithm

4 & [59], Algorithm 3). In each layer, the first two terms are added using a $d$-order SecADD or $\texttt{SecADDChain}_q$, the other terms using minimal share count. The Boolean share count of each arithmetic share $z^{\{j\}}$ $(2 \leq j \leq d)$ is doubled before the arithmetic share count is halved by securely adding them together, using a $t-$NI SecADD. This process is repeated, as in the A2B operation.

**Algorithm 3** A2B [$t$-NI]

**Input Parameter**: modulus $m$ ($q$ or $p$)
**Input Data**: An arithmetic sharing $z^{\{0:d\}}$ of coefficient $z$
**Output Data**: A Boolean sharing $B^{\{0:d\}}$ such that $\bigoplus_{i=0}^{d} B = z \bmod m$

1: **if** d=0 **then**
2: **return** $B^{\{0\}} \leftarrow z^{\{0\}}$
3: **end if**
4: $x^{\{0:\lfloor (d+1)/2 \rfloor - 1\}} \leftarrow \texttt{A2B}(z^{\{0:\lfloor (d+1)/2 \rfloor - 1\}})$
5: $x^{\{0:d\}} \leftarrow \texttt{SecEXP}(x^{\{0:\lfloor (d+1)/2 \rfloor - 1\}})$
6: $y^{\{0:\lceil (d+1)/2 \rceil - 1\}} \leftarrow \texttt{A2B}(z^{\{\lfloor (d+1)/2 \rfloor :d\}})$
7: $y^{\{0:d\}} \leftarrow \texttt{SecEXP}(y^{\{0:\lceil (d+1)/2 \rceil - 1\}})$
8: **return** $B^{\{0:d\}} \leftarrow \texttt{SecADD}_m(x^{\{0:d\}}, y^{\{0:d\}})$

**Algorithm 4** X2B [$t$-NI]

**Input Parameter**: modulus $m$ ($q$ or $p$)
**Input Data**: A *composite* sharing $z^{\{0,d\}}$ as in Eq. 3 of coefficient $z$
**Output Data**: A Boolean sharing $B^{\{0:d\}}$ such that $\bigoplus_{i=0}^{d} B = z \bmod m$

1: **if** d=0 **then**
2: **return** $B^{\{0,0:d\}} \leftarrow z^{\{0,0:d\}}$
3: **end if**
4: $x^{\{0:\lfloor (d+1)/2 \rfloor - 1\}} \leftarrow \texttt{A2B}(z^{\{0:\lfloor (d+1)/2 \rfloor - 1\}})$
5: **if** $d \geq 3$ **then**
6: $x^{\{0:d\}} \leftarrow \texttt{SecEXP}(x^{\{0:\lfloor (d+1)/2 \rfloor - 1\}})$
7: **else**
8: $x^{\{0:d\}} \leftarrow x^{\{0:\lfloor (d+1)/2 \rfloor - 1\}}$
9: **end if**
10: $y^{\{0:\lceil (d+1)/2 \rceil - 1\}} \leftarrow \texttt{A2B}(z^{\{\lfloor (d+1)/2 \rfloor :d\}})$
11: $y^{\{0:d\}} \leftarrow \texttt{SecEXP}(y^{\{0:\lceil (d+1)/2 \rceil - 1\}})$
12: **return** $B^{\{0:d\}} \leftarrow \texttt{SecADD}_m(x^{\{0:d\}}, y^{\{0:d\}})$

In each step, the first element consists of $d+1$ shares, the second is expanded to $d+1$ shares, and the remaining terms double their Boolean share count using the $t-$NI SecEXP (Expand) gadget. Eventually, we obtain an equivalent representation of $z^{\{0:d\}}$ consisting of $d+1$ Boolean shares. Notice that for $d = 1, 2$ the X2B is equivalent to state-of-the-art A2B methods [17,16], and for higher orders is slightly more expensive due to non-minimal share count.

***d*-Probing Security:** We show that the X2B gadget is $t-$NI, considering the leakage effects from Section 2.3 in Appendix B.

**B2X2A** The goal of the B2A operation is to convert $d+1$ Boolean shares $B^{\{0:d\}}$ to $d+1$ arithmetic shares $A^{\{0:d\}}$. The first $d$ output shares are newly sampled, random shares: $A^{\{0:d-1\}} = R_A^{\{0:d-1\}}$. The final output share $A^{\{d\}}$ is computed as $B - R_A$, using the $d$ previously sampled random, arithmetic shares $R_A^{\{0:d-1\}}$. In the following sections, we will denote with superscript-free variables (e.g. $R_A$) the unshared value: $R_A = \sum_{i=0}^{d-1} R_A^{\{i\}} \bmod q$.

In other published work, $R_A^{\{0:d-1\}}$ is first converted to the Boolean domain (using an A2B), resulting in $R_B^{\{0:d\}}$: $\bigoplus_{i=0}^{d} R_B^{\{i\}} = \sum_{i=0}^{d-1} -R_A^{\{i\}} \bmod q$. Next, $B + R_B$ is computed using a secure addition, as both are Boolean shared operands. In total, a full A2B and ($d-$order) SecADD are required.

Our B2X2A gadget combines the A2B conversion and SecADD in a single operation: the X2B input $z^{\{1:d\}}$ is equal to $R_A^{\{0:d-1\}}$ and $z^{\{0\}}$ to $\sim B$. Note that

$B$ consist of $d+1$ Boolean shares which means that $z$ is compositely shared, consisting of $d$ arithmetic shares and one Boolean sharing. The `X2B` is required to convert the compositely shared input, equal to $R_A + (\sim B) = R_A - B - 1$, to a Boolean sharing. As opposed to other work [17,5,27], all inversions are performed as negations in the Boolean domain[1]. The `SecNOT` performed on the `X2B` output, to obtain the desired result $B - R_A$, requires the Boolean inversion of only a single share ($\mathcal{O}(1)$) instead of a share-wise effort ($\mathcal{O}(d)$) in the case of negation on arithmetic shares. In the final step of post-processing, $d+1$ Boolean shares are securely combined using the `FullXOR` gadget [17,9,19], to obtain the final output share: $A^{\{d\}} = \bigoplus_{i=0}^{d} (B - R_A)^{\{0:d\}}$.

**Algorithm 5** `B2X2A` [$t-$NI]

**Input Parameter/Data** : $q$ ▷ $q = 2^n$ $(n = 1..k)$ or prime
**Input Data** : $B^{\{0:d\}}$
**Output Data** : $A^{\{0:d\}}$ such that $\bigoplus_{i=0}^{d} B^{\{i\}} = \sum_{i=0}^{d} A^{\{i\}} \bmod q$

1: $A^{\{0:d-1\}}, R_A^{\{0:d-1\}} \leftarrow$Rand($[0 : q-1]$)
2: **if** $q$ is prime **then** ▷ Modify initial masking for `SecADD`$_q$.
3: **for** $i = 0, 2...d-2$ **do**
4: $z^{\{i+1\}} \leftarrow R_A^{\{i\}}$
5: $z^{\{i+2\}} \leftarrow R_A^{\{i+1\}} - q$ ▷ $-q$ correction.
6: **end for**
7: **else**
8: $z^{\{1:d\}} \leftarrow R_A^{\{0:d-1\}}$
9: **end if**
10: $z^{\{0,0:d\}} \leftarrow$ `SecNOT`$(B^{\{0:d\}})$ ▷ $z = R_A - B - 1$
11: $y'^{\{0:d\}} \leftarrow$ `X2B`$(z^{\{0:d\}})$
12: $y^{\{0:d\}} \leftarrow$ `SecNOT`$(y'^{\{0:d\}})$ ▷ $y = -z - 1 = B - R_A$
13: $A^{\{d\}} \leftarrow$`FullXOR`$(y^{\{0:d\}})$ ▷ $A^{\{d\}} = y$, [17]

**Cost:** This approach is an improvement over the state-of-the-art, as $B - R_A$ is directly computed during the `X2B` as the inputs consist of different amounts of Boolean shares, and thus one does not need to perform the explicit secure addition during post-processing, on two inputs which now consist of $d+1$ Boolean shares. In the original method one needs to compute one `A2B` and one secure addition, while our improved method requires only the `X2B` operation. The `X2B` operation has the same computational cost as `A2B` for first and second security order, and only slightly higher than `A2B` for higher orders. When $d \geq 3$, minimal share count is no longer achieved, as a $d-$order secure adder is required in non-final layers of the tree. The total amount `SecADD`s is reduced in all cases, for high orders an additional max-order secure addition is required. For first-order implementations, only two `SecADD`s are required, for second order one-third of secure additions is removed, etc. For prime moduli, we give a comparison in Table 2. In all cases we obtain a more efficient end result, especially for practical masking orders.

[1] $\sim x = -(x+1)$

**Table 2.** Detailed B2A$_q$ operation cost comparison ($d+1$ shares, $k$-bit words). Max-order `SecADD` operations are explicitly listed, as their cost is relatively high compared to low-order operations.

| | | # `SecADD` | | | # `SecMUX` | |
|---|---|---|---|---|---|---|
| | Order | 1 2 3 | $d$ | Total | 1 2 3 $d$ | Total |
| [5] | 1 | 4 - - | - | 4 | 2 - - - | 2 |
| | 2 | 2 4 - | - | 6 | 1 2 - - | 3 |
| | 3 | 4 - 4 | - | 8 | 2 - 2 - | 4 |
| | $d$ | - - - | 4 | $2(d+1)$ | - - - 2 | $d+1$ |
| [27] | 1 | 2 - - | - | 2 | - - - - | - |
| [11] | 1 | 2 - - | - | 2 | - - - - | - |
| | 2 | 2 5 - | - | 7 | - - - - | - |
| | 3 | 4 0 6 | - | 10 | - - - - | - |
| | $d$ | - - - | 5 or 6[a] | $3d$ or $3d+1$[a] | - - - - | - |
| `B2X2A` (Alg. 5) | 1 | 2 - - | - | 2 | - - - - | - |
| | 2 | 2 2 - | - | **4** | - - - - | - |
| | 3 | 2 0 4 | - | **6** | - - - - | - |
| | $d$ | - - - | $2 \cdot \lceil \log_2(d) \rceil$ | **2d** | - - - - | - |

[a] For *complete* or *incomplete* tree-structure.

**$d$-Probing Security:** We show that the `B2X2A` gadget is correct and prove it to be $t-$NI, considering the leakage effects from Section 2.3 in Appendix C.

## 4 High-Throughput & Low-Randomness Mask Conversions in Hardware

In this section, we first introduce our strategy and novel techniques for implementing the proposed secure gadgets and then demonstrate how a ($d+1$ share) A2B & B2A for prime and power-of-two moduli can be combined in a unified and compact accelerator in hardware: `X2X`. Our implementation follows a streaming approach, in which data flows through the entire pipelined (and unrolled) circuit, ensures all logic is maximally active, high throughput is achieved and transitional leakage in memory elements is avoided. We provide the SystemVerilog source code for all our designs, which we experimentally verify to be first- and high-order secure in the next section.

### 4.1 SecADD$_p$ & SecADD$_q$

A masked ripple-carry adder was proposed by Coron et al. [17], and more hardware-focused parallel prefix-type adders in [1,13]. We propose a Brent-Kung adder architecture [8] because it is more area-efficient than a Kogge-Stone or Schlansky architecture, at the cost of an increased latency yet high throughput. In general, our masking strategy relies on (selectively) combining share-wise gadgets (XOR, NOT...), $t$-NI (DOM-indep AND) and $t$-SNI refresh gadgets to ensure composability. A full description of `SecADD`$_{\texttt{BK}}$ and a security proof are given in Appendix D.

Our fully unrolled and pipelined implementation can compute the secure addition for power-of-two and prime moduli (see `SecADDChain`$_q$), on the same

hardware by setting the appropriate control signals and using dynamic reconfiguration. Two SecADDs are instantiated, which are chained when the modulus is prime, computing either $s$ or $s' = s - q$. Alternatively, for the secure addition modulo a power-of-two integer, we propose using both SecADDs in parallel instead of one being idle in this mode. Throughput is doubled in this mode, as two shared data words ($x_1, y_1$ and $x_2, y_2$) can be accepted each clock cycle.

**Masking Techniques** We now illustrate the difference between masking techniques in hardware on the (Brent-Kung) SecADD operation (Table 3). We take into account two factors: implementation cost (or masking overhead) and verification cost (formal and/or experimental). Firstly, masking a Brent-Kung adder using HPC1 gadgets results in low verification cost. As they are PINI secure, they can be freely composed into a larger circuit. However, due to its (implicit) refreshing stage, both randomness cost and latency are high. Secondly, a $t$-NI SecADD can be constructed using DOM AND gates and explicit $t$-SNI refresh gadgets (see Appendix D). We formally prove its security and observe that the masking overhead is reduced, as expected. Thirdly, we include the overhead for a $t$-probing secure implementation, which we show in Section 5 results in a practically secure circuit and does not require any refresh gadgets, reducing the implementation cost even further.

**Table 3.** Comparison of first-order masking techniques of a Brent-Kung SecADD ($k = 13$).

| Masking Technique | RND [bits] | Latency [cycles] | Verification |
|---|---|---|---|
| HPC1 (PINI) | 228 | 18 | *Low* |
| DOM + SecREF ($t-$NI) | 176 | 11 | *High* |
| DOM ($t$-probing) | 114 | 9 | *High* |

In conclusion, the SecADD operation is a crucial operation in mask conversions, thus optimizing its masking overhead is critical. Our design has minimal implementation cost and is practically secure, yet has a higher verification cost compared to implementations that rely on universally composable gadgets. The security of the larger gadget is formally proven and verified by composing smaller gadgets and verifying their properties.

**Half-Cycle Path** The Domain-Oriented Masking (DOM) scheme and Threshold Implementations (TI) both guarantee glitch-immunity, which means they proveably stay probing secure for every possible occurrence of a glitch. This is achieved by introducing register stages, which essentially result in a 'free' pipelining of the datapath. The SecADD datapath is dominated by chaining non-linear SecAND and SecOR gates, which require at least one register stage to stop the propagation of glitches when crossing domain borders.

We propose interleaving registers clocked at the positive and negative edge in the $\texttt{SecADD}_{\texttt{BK}}$, resulting in half-cycle path implementation[34,24]. This circuit-

level technique can halve the latency of a tightly pipelined secure gadget, (ideally) without significantly impacting the maximal operating frequency of the implementation.From Table 4, for our first-order implementation, using halfcycle datapaths reduces the operating frequency by 21% (176 MHz vs. 139 MHz), while the latency is halved (5 cycles vs 10/11 cycles). As a result, the total execution time is reduced by 36.6%/42.4% compared to the fullcycle implementation, for $\mathrm{A2B}_p/\mathrm{B2A}_p$ respectively. For second order, the maximum operating frequency is naturally lower due to increased circuit complexity. Half-cycle data paths reduces the operating frequency by 9%, and the execution time is reduced by 42% up to 47%. In conclusion, the total execution time is significantly reduced because the latency is reduced while not massively impacting the maximum operating frequency, illustrating why highly non-linear operations are a good target for such circuit-level optimizations.

**Table 4.** Timing performance of half-cycle and full-cycle `X2X` hardware implementation (Kintex-7).

| Masking Order | Design | Max. Freq. [MHz] | Latency[a] [cycles] | Time [ns] |
|---|---|---|---|---|
| $d = 1$ | FULL | **176** | 10/11 | 56.8/62.5 |
| | HALF | 139 | **5/5** | **36.0/36.0** |
| $d = 2$ | FULL | **144** | 20/21 | 139.9/145.8 |
| | HALF | 130 | **10/10** | **76.9/76.9** |

[a] $\mathrm{A2B}_p/\mathrm{B2A}_p$

Similarly, this technique can be directly applied to HPC1 gadgets, performing the refresh and DOM AND gate in a single clock cycle instead of two. Still, since we only include refresh stages when explicitly required (to ensure independent inputs and simulability), our manual masking approach results in a lower total latency.

### 4.2 A2B & B2A

In our `X2X` implementation, the `A2B` and `X2B` are computed using (mostly) the same physical instances in hardware. A tree-structure of `SecEXP` and `SecADD` components is instantiated, maximizing the parallelism available in hardware by operating on all shares simultaneously. As seen in Figure 2, this combination of operations first doubles the level of Boolean sharing, after which the amount of arithmetic shares is halved. This process is repeated on all shares in parallel, $L = \lceil \log(d+1) \rceil$ times to obtain $d+1$ Boolean shares.

Technically, only the `SecADD` operations involving the actual, secret input data and final XOR (Line 13) need to be computed at run-time in order to obtain the final share $A^{\{d\}}$.

When computing the `X2B` for orders $d \geq 3$, minimal share count is no longer achieved, as a $d+1$ share secure adder is instantiated in all but the final layer. For prime moduli $q$, some pre-processing is required before the initial masking

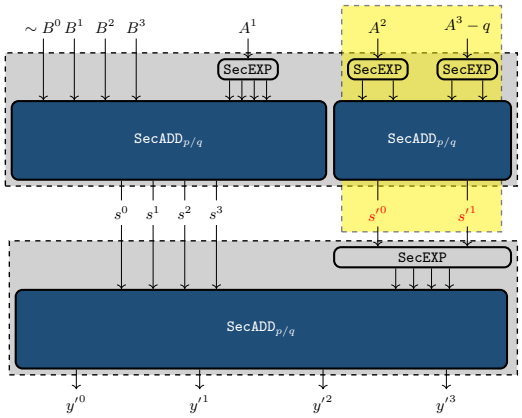



**Fig. 2.** 4-share `X2B` (& `A2B`) in hardware: `SecEXP` and `SecADD`. The right side (yellow) can be pre-computed. $y = B - A$ is directly computed during `X2B`, removing the need for a secure addition in post-processing.

and the `B2X2A` requires additional post-processing, compared to the `A2B`. Interestingly, a portion of the `X2B` computation can be a target for pre-computation as it only involves random data (indicated in yellow). These operations can be computed when the random shares are generated and the result temporarily stored in memory. This optimization is left as future work.

## 5 Performance & Security Evaluation

### 5.1 Measurement Setup

In this section, we describe the practical evaluation of our masked designs on the Xilinx Kintex7 FPGA[2]. We utilize the *keep_hierarchy* pragma to prevent the compiler from optimizing masking countermeasures away. This may result in a less-than-optimal overhead but ensures the desired security.

For the security evaluation, we collect power traces from the measurement point on the SASIMI evaluation board [28] containing an Kintex7 `XCKU040` FPGA. The traces are captured by a Tektronix DPO7254 oscilloscope at a sample rate of 1GS/s while the FPGA is externally clocked at 6MHz. We synchronized the oscilloscope and the external clock for all our measurements. Also, the mask conversion accelerator instance is duplicated several times on FPGA for lab evaluations, to guarantee satisfactory SNR for statistical analysis, illustrated in the mean measurement traces. All instances operate on a single, identical input data and fresh randomness in parallel. The randomness required by our design is supplied by a PRNG that runs on the crypto FPGA. The PRNG consists of an AES-CTR and Trivium cipher implementation, which is re-seeded with fresh

[2] `XC7K160T` & `XCKU040`, Xilinx Vivado v2021.1

**Table 5.** Mask conversion hardware implementation: performance comparison.

| Design | Mask. | Tech. | Device | k | d | Util. [LUT/FF] | Freq. [MHz] | OP | mod | Rand.[a] [bits] | Lat. [cycles] | TP [coeff/cycle] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| [58] | TI | | Spartan-6 | 32 | 1 | 937/1,330 | 62 | SecADD | $2^k$ | 32 | 6 | 0.167 |
| | | | | | 2 | 4,223/5,509 | 63 | | | 128 | 12 | 0.083 |
| [27] | TI | | Artix-7 | 32 | 1 | 2,464/1,323 | 454 | SecADD | $2^k$ | - | 6 | - |
| [1] | TI | | Spartan-6 | 32 | 1 | 487/2352 | 280 | SecADD | $2^k$ | 31 | 9 | 1 |
| [1] | PINI (HPC) | | Spartan-6 | 32 | 1 | 1588/4317 | 173 | SecADD | $2^k$ | 74 | 18 | 1 |
| | | | | | 2 | 1666/7122 | 158 | | | 222 | 18 | 1 |
| [13] | PINI (HPC) | | -[c] | 32 | 1 | - | - | SecADD | $2^k$ | 122 | 10 | 1 |
| | | | | | 2 | - | - | | | 366 | 10 | 1 |
| **This Work (Full-cycle)** | DOM | | Kintex-7[d] | 13 | 1 | 400/989 | - | SecADD | $2^k$ | 114 | 9 | 1 |
| | | | | | 2 | 761/2,028 | - | | | 342 | 9 | 1 |
| **This Work (Half-cycle)** | DOM | | Kintex-7[d] | 13 | 1 | 405/715 | - | SecADD | $2^k$ | 114 | 5 | 1 |
| | | | | | 2 | 762/1515 | - | | | 342 | 5 | 1 |
| [16][b] | PINI (HPC) | | Artix-7 | 32 | 2 | 13,064/17,952 | 351 | A2B | $2^k$ | 1,280 | 24 | 1 |
| [9][b] | PINI (HPC) | | Artix-7 | 32 | 2 | 2,234/20,423 | 512 | A2B | $2^k$ | 124 | 124 | 0.008 |
| [42] | PINI (HPC) | | Artix-7 | 32 | 2 | 11,196/14,550 | 370 | A2B | $2^k$ | 1,056 | 14 | 1 |
| **This Work (Full-cycle)** | **DOM** | | Kintex-7[d] | 13 | 1 | 1,150/3,335 | 176 | A2B | $2^k$ | **140** | 10 | **2** |
| | | | | | | | | | 3329 | **255** | 20 | **1** |
| | | | | | | | | B2A | $2^k$ | **140** | 11 | **2** |
| | | | | | | | | | 3329 | **255** | 21 | **1** |
| | | | | | 2 | 3,128/16,774 | 144 | A2B | $2^k$ | 534 | 20 | **2** |
| | | | | | | | | | 3329 | **993** | 40 | **1** |
| | | | | | | | | B2A | $2^k$ | **534** | 21 | **2** |
| | | | | | | | | | 3329 | **993** | 41 | **1** |
| **This Work (Half-cycle)** | **DOM** | | Kintex-7[d] | **13** | 1 | **1,133/2,170** | 139 | A2B | $2^k$ | **140** | **5** | **2** |
| | | | | | | | | | 3329 | **255** | **10** | **1** |
| | | | | | | | | B2A | $2^k$ | **140** | **5** | **2** |
| | | | | | | | | | 3329 | **255** | **10** | **1** |
| | | | | | 2 | **3,105/9,376** | 130 | A2B | $2^k$ | 534 | **10** | **2** |
| | | | | | | | | | 3329 | **993** | **20** | **1** |
| | | | | | | | | B2A | $2^k$ | **534** | **10** | **2** |
| | | | | | | | | | 3329 | **993** | **20** | **1** |

[a] Total random bits (full operation, per coefficient).
[b] Numbers taken from [42].
[c] No numbers for FPGA given, only ASIC.
[d] XC7K160T

randomness for each mask conversion. We interleave the execution of the PRNG with the execution of the full mask conversion to decrease the impact of noice induced by the PRNG.

### 5.2 Performance Comparison

We now give an overview of existing implementations and our optimized design for different mask conversions in hardware (Table 5). It is important to note that the compared hardware implementations target different operations, platforms, data word sizes, masking strategies, and more. As a result, a direct and/or fair comparison is not always possible. We compare our work with other (SecADD-based) A2B/B2A strategies, which are favored for hardware implementations as both operations rely on similar arithmetic and benefit physical from instance reuse. Through algorithmic, gadget- and circuit-level optimizations, we reduce latency, maximize throughput and minimize area cost.

Firstly, we observe that our design is the only one to directly support the computation all of the types of mask conversions required in the Kyber decapsulation (or any lattice-based PQC scheme). Our streaming hardware design

can be dynamically reconfigured, maximally reusing physical instances between operations. The `A2B` and `B2A` operations require identical amounts of fresh randomness, thanks to the `B2X2A` gadget. For power-of-two modes, the throughput can be doubled by utilizing both instantiated `SecADD` gadgets.

As expected, the half-cycle design outperforms the full-cycle design (without circuit-level modifications): flipflop utilization is reduced due to the shorter pipeline, which also reduces latency at the cost of slightly decreased operating frequency.

We are aware of one other work which directly implements the (second order) $\texttt{A2B}_p$ operation [42], with $k = 32$ and using the PINI security notion (HPC gadgets). By relying on universal composability, this work significantly reduces the verification and design cost. However, the randomness cost and latency are 1.98× and 1.4× higher, respectively, compared to our design. For reference, this work also implements the designs proposed in [16] and [9] using HPC gadgets. Both are less efficient (throughput, latency and/or randomness) compared to the design proposed in [42]. We rely on manual masking and DOM gadgets, which allow to reduce the implementation overhead. We repeat that our design is optimized for Kyber parameters ($k = 13$) but supports four different modes of operation. While we note that it is difficult to compare implementations on different platforms, our design has a significantly lower area utilization and also maximum operating frequency. We would expect the operating frequency of this design to reduce if other types of mask conversions are supported, as the critical path is situated in the B2A and mod $q$ circuit.

Next, we remark that our design achieves the highest throughput for power-of-two moduli by using both instantiated secure adders for two different inputs at once. As our design is fully pipelined, as is the case with HPC gadgets, one coefficient is processed per cycle. Oppositely, [9] achieves low area utilization by reusing physical instances over time (ripple-carry), resulting in a low throughput, high latency and lower randomness cost. More specifically, compared to our design, [9] requires 4.3× less random bits but a 12.4× lower throughput.

Several other works have presented performance results for secure adder implementations, the most crucial component for mask conversions. Where possible, we compare with the most optimized and/or Brent-Kung adder architecture with this work. In [58] a 32-bit secure adder is presented, which has a low randomness cost but has lower throughput and operating frequency compared to our implementation. Compared to the adders presented in [1], our full-cycle design has half the latency but requires around 50% more fresh randomness at first- and second-order. The authors also propose a first-order secure TI secure adder, which outperforms our design at first-order. The main disadvantage is that this design cannot be scaled to higher protection orders. In [13] techniques were presented to reduce latency of large chains of PINI gadgets, at the cost of increased fresh randomness requirements.

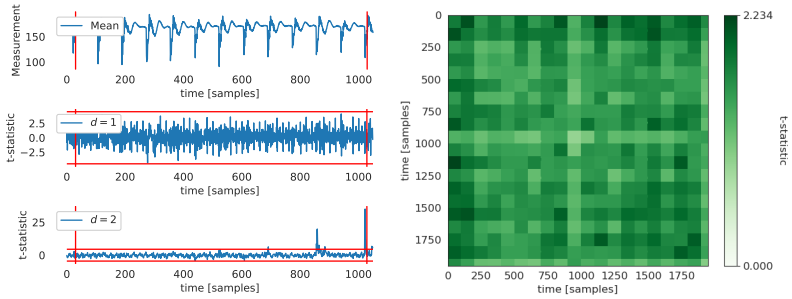
We conclude that using HPC gadgets and the PINI security notion results in a low verification cost, through universal composability. However the implementation overhead, especially randomness and latency, are higher compared to

manually (algorithmically masked) designs for highly non-linear operations (`A2B` & `B2A`). In this case, switching away from HPC gadgets results in 62% improvement in randomness cost, 29% up to 92% lower latency and 45% up to 60%, at the cost of increased design and verification complexity, which is error-prone.

### 5.3 Test Vector Leakage Assessment Results

We verify that our implementations do not show first-order (or second-order) univariate and bivariate leakage. The *non-specific, fixed vs. random* t-test statistic [30] is calculated for the implementation of all different mask conversion operations. The TVLA results for the $\text{A2B}_p$ mode are shown below (Figure 3), with all (extended) figures shown in Appendix F. The threshold value of the t-test commonly used by the side-channel research community is 4.5 which provides a confidence of roughly 0.99999. If the t-test value of the measured power trace grows over 4.5, the implementation under test is considered as insecure. The regions of interest are indicated on all figures between vertical red lines, which indicate the start and end of mask conversion.

Figure 9 illustrates the TVLA results of the first-order masked $\text{A2B}_p$, $\text{A2B}_q$, $\text{B2A}_p$ and $\text{B2A}_q$ operations (i.e., Fig. 9a - 9d), which rely on the novel `X2B` and $\texttt{SecADDChain}_q$ gadgets. The mean trace, first- and second-order statistical moments with the PRNG activated are displayed. Each of the subplots confirm our theoretical expectation, as no significant evidence of first-order leakage was detected for 100 million measurements. The second-order leakages show as anticipated. In contrast, we also include t-test results for the implementation with the randomness turned off (set to zero), guaranteeing that our test set-up is sound and can detect leakage (Fig. 10a) with only 500K traces.




(a) Univariate (b) Bivariate

**Fig. 3.** Univariate and bivariate TVLA analysis (fixed-vs.-random) of $1^{\text{st}}$- and $2^{\text{nd}}$-order $\texttt{A2B}_{2^{13}}$, 100M traces, PRNG ON. The $\pm$ 4.5 threshold is marked by red lines and max-pooling guarantees that t-test peaks are visible.

Fig. 11 illustrates the TVLA result of the second-order masked A2B and B2A operations (mod $2^{13}$ and 3329). The mean trace and first, second (and third)

order statistical moments with the RNG activated are displayed (Fig. 11a - 11d). First- and second-order (univariate) leakages are not present. Again, we verified our measurement setup by turning off the randomness source (Fig. 10b), with all present leakages not appearing when the randomness is turned on again.

We also performed second-order bivariate leakage detection tests [12], illustrated in Fig. 13. To alleviate the computational complexity of this analysis, we set the point of interests at every 10 sample points. First, we verified that our second-order implementation shows leakages with the PRNG turned off, with only 500K traces (Fig. 12). We confirm the measurement setup is sound and can detect bivariate leakages (t-values exceeding 4.5). With the PRNG switched on, no excursions of the t-values beyond $\pm$ 4.5 occur and thus the test is passed with 100M traces. In our figures, we use max-pooling sub-sampling, displaying the largest (absolute) t-value for every $10\times10$ square in the bivariate plot.
**Conclusion.** From these first-and high-order univ- and bi-variate tests using TVLA methodology, we can conclude our proposed techniques and their first and second-order implementations are secure.

## 6 Conclusion

In this work, a first- and high-order hardware implementation of the mask conversion operation, secure against differential power analysis attacks were described. These leverage novel $d-$order secure gadgets and circuit-level optimizations to improve performance at all protection orders. Including a novel $\texttt{SecADDChain}_q$ gadget, which relies on repeated, implicit modular reduction for improved chaining and the `B2X2A`, which relies on the novel `X2B`. The univariate and multivariate security is formally proven and experimentally validated in various modes.

This work leverages careful, manual masking to achieve first- and high-order protection, which is demonstrated to lead to reasonable overheads. An interesting direction for future work is to investigate both the reuse of random masks and physical instances and machine-assisted verification of our implementation.

In summary, the presented techniques result in hardware implementations with the lowest area utilization, fresh randomness cost and latency published for Kyber parameters. The amount of clock cycles required for a mask conversion is reduced by 29% up to 92%, the required amount of fresh randomness by up to 62%. The presented second-order implementation requires 3,105/9,376 [LUT/FF] on FPGA, which is a reduction of 45% to 60% compared to the state-of-the-art and up to 62% fewer random bits.

**Acknowledgements** We thank Lennert Wouters, Zhenda Zhang, John Gaspoz and Siemen Dhooghe for the interesting discussions. This work was partially supported by Horizon 2020 ERC Advanced Grant (101020005 Belfort), Horizon Europe (101070008 ORSHIN), CyberSecurity Research Flanders with reference number VOEWICS02, BE QCI: Belgian-QCI (3E230370) (see beqci.eu), and Intel Corporation.

## References


1. Bache, F., Güneysu, T.: Boolean masking for arithmetic additions at arbitrary order in hardware. Applied Sciences **12**(5) (2022). https://doi.org/10.3390/app12052274, https://www.mdpi.com/2076-3417/12/5/2274
2. Balasch, J., Gierlichs, B., Grosso, V., Reparaz, O., Standaert, F.: On the cost of lazy engineering for masked software implementations. In: Joye, M., Moradi, A. (eds.) Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8968, pp. 64–81. Springer (2014). https://doi.org/10.1007/978-3-319-16763-3_5, https://doi.org/10.1007/978-3-319-16763-3_5
3. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.A., Grégoire, B., Strub, P.Y.: Verified proofs of higher-order masking. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015. pp. 457–485. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
4. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.A., Grégoire, B., Strub, P.Y., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 116–129. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978427
5. Barthe, G., Belaïd, S., Espitau, T., Fouque, P.A., Grégoire, B., Rossi, M., Tibouchi, M.: Masking the GLP lattice-based signature scheme at any order. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2018. pp. 354–384. Springer International Publishing, Cham (2018)
6. Beirendonck, M.V., D'anvers, J.P., Karmakar, A., Balasch, J., Verbauwhede, I.: A side-channel-resistant implementation of saber. J. Emerg. Technol. Comput. Syst. **17**(2) (apr 2021). https://doi.org/10.1145/3429983, https://doi.org/10.1145/3429983
7. Bos, J.W., Gourjon, M., Renes, J., Schneider, T., van Vredendaal, C.: Masking kyber: First- and higher-order implementations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**(4), 173–214 (Aug 2021). https://doi.org/10.46586/tches.v2021.i4.173-214, https://tches.iacr.org/index.php/TCHES/article/view/9064
8. Brent, Kung: A regular layout for parallel adders. IEEE Transactions on Computers **C-31**(3), 260–264 (1982). https://doi.org/10.1109/TC.1982.1675982
9. Bronchain, O., Cassiers, G.: Bitslicing arithmetic/boolean masking conversions for fun and profit with application to lattice-based kems. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022**(4), 553–588 (2022). https://doi.org/10.46586/tches.v2022.i4.553-588, https://doi.org/10.46586/tches.v2022.i4.553-588
10. Cassiers, G., Standaert, F.X.: Trivially and efficiently composing masked gadgets with probe isolating non-interference. IEEE Transactions on Information Forensics and Security **PP**, 1–1 (02 2020). https://doi.org/10.1109/TIFS.2020.2971153
11. Cassiers, G.: Composable and efficient masking schemes for side-channel secure implementations. Ph.D. thesis, École polytechnique de Louvain and Université catholique de Louvain (2022)
12. Cassiers, G., Bronchain, O.: Scalib: A side-channel analysis library. Journal of Open Source Software **8**(86), 5196 (2023). https://doi.org/10.21105/joss.05196, https://doi.org/10.21105/joss.05196
13. Cassiers, G., Gigerl, B., Mangard, S., Momin, C., Nagpal, R.: Compress: Generate small and fast masked pipelined circuits. Cryptology ePrint Archive, Pa-

per 2023/1600 (2023), https://eprint.iacr.org/2023/1600, https://eprint.iacr.org/2023/1600
14. Cassiers, G., Grégoire, B., Levi, I., Standaert, F.X.: Hardware private circuits: From trivial composition to full verification. IEEE Transactions on Computers **70**(10), 1677–1690 (2021). https://doi.org/10.1109/TC.2020.3022979
15. Coron, J.S., Giraud, C., Prouff, E., Renner, S., Rivain, M., Vadnala, P.K.: Conversion of security proofs from one leakage model to another: A new issue. In: Schindler, W., Huss, S.A. (eds.) COSADE 2012. LNCS, vol. 7275, pp. 69–81. Springer, Heidelberg (May 2012). https://doi.org/10.1007/978-3-642-29912-4_6
16. Coron, J.S., Großschädl, J., Tibouchi, M., Vadnala, P.K.: Conversion from arithmetic to boolean masking with logarithmic complexity. In: Leander, G. (ed.) Fast Software Encryption. pp. 130–149. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
17. Coron, J.S., Großschädl, J., Vadnala, P.K.: Secure conversion between Boolean and arithmetic masking of any order. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 188–205. Springer, Heidelberg (Sep 2014). https://doi.org/10.1007/978-3-662-44709-3_11
18. Coron, J.S., Spignoli, L.: Secure wire shuffling in the probing model. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part III. LNCS, vol. 12827, pp. 215–244. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84252-9_8
19. Coron, J.S., Gérard, F., Montoya, S., Zeitoun, R.: High-order polynomial comparison and masking lattice-based encryption. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 153–192 (11 2022). https://doi.org/10.46586/tches.v2023.i1.153-192
20. Coron, J.S., Gérard, F., Montoya, S., Zeitoun, R.: High-order table-based conversion algorithms and masking lattice-based encryption. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(2), 1–40 (Feb 2022). https://doi.org/10.46586/tches.v2022.i2.1-40, https://tches.iacr.org/index.php/TCHES/article/view/9479
21. Coron, J.S., Gérard, F., Trannoy, M., Zeitoun, R.: High-order masking of NTRU. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(2), 180–211 (Mar 2023). https://doi.org/10.46586/tches.v2023.i2.180-211, https://tches.iacr.org/index.php/TCHES/article/view/10281
22. D'Anvers, J.P., Tiepelt, M., Vercauteren, F., Verbauwhede, I.: Timing attacks on error correcting codes in post-quantum schemes. In: Proceedings of ACM Workshop on Theory of Implementation Security Workshop. p. 2–9. TIS'19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3338467.3358948, https://doi.org/10.1145/3338467.3358948
23. De Cnudde, T., Bilgin, B., Gierlichs, B., Nikov, V., Nikova, S., Rijmen, V.: Does coupling affect the security of masked implementations? In: Guilley, S. (ed.) COSADE 2017. LNCS, vol. 10348, pp. 1–18. Springer, Heidelberg (Apr 2017). https://doi.org/10.1007/978-3-319-64647-3_1
24. De Meyer, L., Reparaz, O., Bilgin, B.: Multiplicative masking for aes in hardware. IACR Transactions on Cryptographic Hardware and Embedded Systems **2018**(3), 431–468 (Aug 2018). https://doi.org/10.13154/tches.v2018.i3.431-468, https://tches.iacr.org/index.php/TCHES/article/view/7282
25. D'Anvers, J.P.: One-hot conversion: Towards faster table-based a2b conversion. In: Advances in Cryptology – EUROCRYPT 2023: 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon,

France, April 23-27, 2023, Proceedings, Part IV. p. 628–657. Springer-Verlag, Berlin, Heidelberg (2023). https://doi.org/10.1007/978-3-031-30634-1_21, https://doi.org/10.1007/978-3-031-30634-1_21
26. Faust, S., Grosso, V., Merino Del Pozo, S., Paglialonga, C., Standaert, F.X.: Composable masking schemes in the presence of physical defaults & the robust probing model. IACR Transactions on Cryptographic Hardware and Embedded Systems **2018**(3), 89–120 (Aug 2018). https://doi.org/10.13154/tches.v2018.i3.89-120, https://tches.iacr.org/index.php/TCHES/article/view/7270
27. Fritzmann, T., Van Beirendonck, M., Basu Roy, D., Karl, P., Schamberger, T., Verbauwhede, I., Sigl, G.: Masked accelerators and instruction set extensions for post-quantum cryptography. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(1), 414–460 (Nov 2021). https://doi.org/10.46586/tches.v2022.i1.414-460, https://tches.iacr.org/index.php/TCHES/article/view/9303
28. Fujimoto, D., Kim, Y., Hayashi, Y., Homma, N., Hashimoto, M., Sato, T., Danger, J.L.: Sasimi: Evaluation board for em information leakage from large scale cryptographic circuits. In: 2022 IEEE International Symposium on Electromagnetic Compatibility & Signal/Power Integrity (EMCSI). pp. 299–302 (2022). https://doi.org/10.1109/EMCSI39492.2022.9889445
29. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Koç, Çetin Kaya., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (May 2001). https://doi.org/10.1007/3-540-44709-1_21
30. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side channel resistance. https://csrc.nist.gov/csrc/media/events/non-invasive-attack-testing-workshop/documents/08_goodwill.pdf (2011), [Online; accessed 6-November-2023]
31. Goubin, L.: A sound method for switching between Boolean and arithmetic masking. In: Koç, Çetin Kaya., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 3–15. Springer, Heidelberg (May 2001). https://doi.org/10.1007/3-540-44709-1_2
32. Groß, H., Mangard, S.: Reconciling d+1 masking in hardware and software. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 115–136. Springer, Heidelberg (Sep 2017). https://doi.org/10.1007/978-3-319-66787-4_6
33. Gross, H., Mangard, S., Korak, T.: Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In: Proceedings of the 2016 ACM Workshop on Theory of Implementation Security. p. 3. TIS '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2996366.2996426, https://doi.org/10.1145/2996366.2996426
34. Gross, H., Schaffenrath, D., Mangard, S.: Higher-order side-channel protected implementations of KECCAK. In: 2017 Euromicro Conference on Digital System Design (DSD). pp. 205–212 (2017). https://doi.org/10.1109/DSD.2017.21
35. Guo, Q., Johansson, T., Nilsson, A.: A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 359–386. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_13
36. Heinz, D., Kannwischer, M.J., Land, G., Pöppelmann, T., Schwabe, P., Sprenkels, D.: First-order masked kyber on ARM cortex-m4. Cryptology ePrint Archive, Paper 2022/058 (2022), https://eprint.iacr.org/2022/058, https://eprint.iacr.org/2022/058

37. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_27
38. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (Aug 1996). https://doi.org/10.1007/3-540-68697-5_9
39. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_25
40. Krausz, M., Land, G., Stolz, F., Naujoks, D., Richter-Brockmann, J., Güneysu, T., Kogelheide, L.: Generic accelerators for costly-to-mask pqc components. Cryptology ePrint Archive, Paper 2023/1287 (2023), https://eprint.iacr.org/2023/1287, https://eprint.iacr.org/2023/1287
41. Kundu, S., D'Anvers, J.P., Van Beirendonck, M., Karmakar, A., Verbauwhede, I.: Higher-order masked saber. In: Galdi, C., Jarecki, S. (eds.) Security and Cryptography for Networks. pp. 93–116. Springer International Publishing, Cham (2022)
42. Liu, J., Zhao, C., Peng, S., Yang, B., Zhao, H., Han, X., Zhu, M., Wei, S., Liu, L.: A low-latency high-order arithmetic to boolean masking conversion. Cryptology ePrint Archive, Paper 2024/045 (2024), https://eprint.iacr.org/2024/045, https://eprint.iacr.org/2024/045
43. Mangard, S., Popp, T., Gammel, B.M.: Side-channel leakage of masked cmos gates. In: Menezes, A. (ed.) Topics in Cryptology – CT-RSA 2005. pp. 351–365. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
44. Mangard, S., Schramm, K.: Pinpointing the side-channel leakage of masked AES hardware implementations. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 76–90. Springer, Heidelberg (Oct 2006). https://doi.org/10.1007/11894063_7
45. Migliore, V., Gérard, B., Tibouchi, M., Fouque, P.A.: Masking Dilithium - efficient implementation and side-channel evaluation. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) ACNS 19. LNCS, vol. 11464, pp. 344–362. Springer, Heidelberg (Jun 2019). https://doi.org/10.1007/978-3-030-21568-2_17
46. Moos, T., Moradi, A., Schneider, T., Standaert, F.X.: Glitch-resistant masking revisited: or why proofs in the robust probing model are needed. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(2), 256–292 (Feb 2019). https://doi.org/10.13154/tches.v2019.i2.256-292, https://tches.iacr.org/index.php/TCHES/article/view/7392
47. Moradi, A., Wild, A.: Assessment of hiding the higher-order leakages in hardware - what are the achievements versus overheads? In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 453–474. Springer, Heidelberg (Sep 2015). https://doi.org/10.1007/978-3-662-48324-4_23
48. Müller, N., Knichel, D., Sasdrich, P., Moradi, A.: Transitional leakage in theory and practice: Unveiling security flaws in masked circuits. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(2), 266–288 (Feb 2022). https://doi.org/10.46586/tches.v2022.i2.266-288, https://tches.iacr.org/index.php/TCHES/article/view/9488
49. National Institute of Standards and Technology: Module-lattice-based digital signature standard. Tech. rep., U.S. Department of Commerce, Washington, D.C. (2024). https://doi.org/10.6028/NIST.FIPS.204
50. National Institute of Standards and Technology: Module-lattice-based key-encapsulation mechanism standard. Tech. rep., U.S. Department of Commerce, Washington, D.C. (2024). https://doi.org/10.6028/NIST.FIPS.203

51. Nikova, S., Rechberger, C., Rijmen, V.: Threshold implementations against side-channel attacks and glitches. In: Ning, P., Qing, S., Li, N. (eds.) Information and Communications Security. pp. 529–545. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
52. NIST Computer Security Division: Post-quantum cryptography standardization. https://csrc.nist.gov/projects/post-quantum-cryptography (2016), [Online; accessed 17-August-2023]
53. NIST Computer Security Division: Comments requested on three draft fips for post-quantum cryptography. https://csrc.nist.gov/news/2023/three-draft-fips-for-post-quantum-cryptography (2023), [Online; accessed 30-October-2023]
54. NIST Computer Security Division: Post-quantum cryptography: Digital signature schemes. https://csrc.nist.gov/projects/pqc-dig-sig/round-1-additional-signatures (2023), [Online; accessed 7-September-2023]
55. Primas, R., Pessl, P., Mangard, S.: Single-trace side-channel attacks on masked lattice-based encryption. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 513–533. Springer, Heidelberg (Sep 2017). https://doi.org/10.1007/978-3-319-66787-4_25
56. Ravi, P., Sinha Roy, S., Chattopadhyay, A., Bhasin, S.: Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(3), 307–335 (Jun 2020). https://doi.org/10.13154/tches.v2020.i3.307-335, https://tches.iacr.org/index.php/TCHES/article/view/8592
57. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 764–783. Springer, Heidelberg (Aug 2015). https://doi.org/10.1007/978-3-662-47989-6_37
58. Schneider, T., Moradi, A., Güneysu, T.: Arithmetic addition over boolean masking. In: Malkin, T., Kolesnikov, V., Lewko, A.B., Polychronakis, M. (eds.) Applied Cryptography and Network Security. pp. 559–578. Springer International Publishing, Cham (2015)
59. Schneider, T., Paglialonga, C., Oder, T., Güneysu, T.: Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto. In: Lin, D., Sako, K. (eds.) Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11443, pp. 534–564. Springer (2019). https://doi.org/10.1007/978-3-030-17259-6_18, https://doi.org/10.1007/978-3-030-17259-6_18
60. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (nov 1979). https://doi.org/10.1145/359168.359176, https://doi.org/10.1145/359168.359176
61. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing **26**(5), 1484–1509 (1997). https://doi.org/10.1137/S0097539795293172, https://doi.org/10.1137/S0097539795293172
62. Van Beirendonck, M., D'Anvers, J.P., Verbauwhede, I.: Analysis and comparison of table-based arithmetic to boolean masking. IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**(3), 275–297 (Jul 2021). https://doi.org/10.46586/tches.v2021.i3.275-297, https://tches.iacr.org/index.php/TCHES/article/view/8975

63. Xu, Z., Pemberton, O., Roy, S.S., Oswald, D.F., Yao, W., Zheng, Z.: Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of kyber. IEEE Transactions on Computers **71**, 2163–2176 (2022), https://api.semanticscholar.org/CorpusID:220794801

# Appendices

## Appendix A Correctness & Security Proof $\texttt{SecADDChain}_q$ (Algorithm 2)

Note that Algorithm 2 is independent of the specific masked algorithms used for `SecADD`, `SecNOT` or `SecREF`. `SecNOT` refers to the sharewise $t-$NI computation of the Boolean negation, where only the first share of the Boolean-masked input is negated (Alg. 7). `SecADD` denotes the (robust) $t-$NI arithmetic addition of Boolean shares, as from [16,5] or Appendix D. `SecREF` describes a $t-$SNI algorithm to refresh Boolean shares, as proposed in [5,14] and Alg. 11. We note that explicit mask refresh operations can be avoided by switching to a more strict security notion, e.g., PINI [10]. We place the refresh in this position, in order to only have to refresh 1-bit shares (MSB), instead of $k-$bit shares. Furthermore, we use $\cdot$ to indicate the linear $t-$NI computation integer multiplication, on each of the shares separately. We use $\gg$ to denote the linear $t-$NI bitwise shift of Boolean shares, achieved by shifting each input share separately.

**Correctness.** For prime $q$, explicit modular reduction is performed on $z = x + y' = x + y - q \in [-q : q - 2]$, because $y'$ lies in $[-q : -1]$.

- $z \in [-q : -1]$: $c = q$, so $s = z + q$ lies in $[0 : q - 1]$. $c' = 0$, because an uneven amount of carry bits $cc$ will be '1' as both $x$ and $y$ are mod $q$. This ensures $s' = z + 0$ lies in $[-q : -1]$.
- $z \in [0 : q - 2]$: $c = 0$, so $s = z + q$ lies in $[0 : q - 2]$. $c' = -q$, because an even amount of carry bits $cc$ will be '1' as both $x$ and $y$ are mod $q$. This ensures $s' = z - q$ lies in $[-q : -2]$.

The algorithm returns either a value modulo $q$, or (mod $q$) - $q$.

**Security.** To argue about the higher-order security of Algorithm 2, we prove it to be (word-level) $t-$NI with $t + 1$ shares. As a result, all gadgets are modeled to operate on word-level shares. This provides resistance against a probing adversary with $t$ word-level probes and allows the use of the gadget in larger compositions. We show how probes on intermediate values in the algorithm can be perfectly simulated with only a limited number of input shares, by iterating over all possible intermediate variables. We provide formal arguments on how they can be simulated relying on the $t-$(S)NI properties of the sub-operations. We show that all word-level probes can be simulated with no more number of input shares.

**Theorem 1.** *The gadget $\texttt{SecADDChain}_q$ (Algorithm 2) is word-level t-NI secure.*

*Proof.* We model Algorithm 2 as a sequence of $t$-(S)NI gadgets, as shown in Figure 4. For simplicity, we model (and combine) the linear operations in Lines 2 and 5/8 as $t$-NI gadgets ($G_2$ and $G_4$), which can be trivially shown as the operations are linear and process the inputs share-wise. We map all three $t-$NI SecADD gadgets as follows: $G_1$ (Line 1) and $G_5$ (Line 6 or 9). Due to the *if..else* statement, either Lines 5-6 or 8-9 are executed and can both be represented by

gadgets $G_4$ and $G_5$. The $t$-SNI refresh gadget on Line 3 is mapped to gadget $G_3$. An adversary can probe the intermediate values and outputs of all gadgets $G_i$ (except the output shares of the complete algorithm), $t_{G_i}$ refers to the number of internal probes and $o_{G_i}$ the number of output probes.

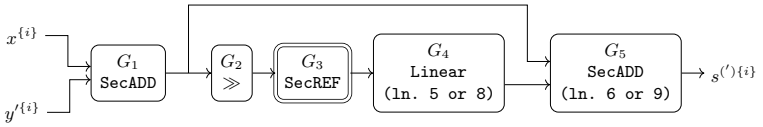



**Fig. 4.** An abstract diagram of $\texttt{SecADDChain}_q$ (Algorithm 2). The $t$-NI gadgets are depicted with a single border, the $t$-SNI gadgets with a double border.

To prove Theorem 1, we show that the internal ($t_{G_i}$) and output ($o_{G_i}$) probes of each gadget in Algorithm 2, except the output shares of the full composition, can be perfectly simulated with no more number of input shares $x^{\{i\}}$ and $y'^{\{i\}}$ ($\leq t_{A_2}$) with:

$$t_{A_2} = \sum_{i=1}^{5} t_{G_i} + \sum_{i=1}^{4} o_{G_i}$$

As defined earlier, for the $t$-NI notion, if any probes are placed on the final output of Algorithm 2 (entire composition), the simulator will get access to a same amount of input shares, for free. Or, as stated above, the simulation must succeed for any set of $t$ intermediate probes. In contrast, for $t$-SNI security, an probing adversary can place $t$ probes on intermediate values and at the final output and simulation must succeed without access to additional input shares. We rely on the $t-$(S)NI properties of each gadget to argue about their internal and output probes. For the simulation of a larger composition, the required shares are added up. To ensure the simulation is sound, we will show that it is necessary to insert a $t-$SNI `SecREF` gadget for the MSB of the result of the first `SecADD` and to ensure the inputs of the second `SecADD` are independent. This is because a $t-$SNI gadget stops the propagation of probes from the output shares to the input shares, allowing its simulation to be performed independent of the number of probed output shares.

To simulate the $t_{G_5}$ intermediate probes of the $t-$NI gadget $G_5$, $t_{G_5}$ shares of both inputs $G_4$ and $G_1$ are required. Given the share-wise operation of $G_4$, simulating $t_{G_4} + o_{G_4}$ intermediate and output probes requires $t_{G_4} + o_{G_4}$ shares of input $G_3$. Without a $t-$SNI refresh $G_3$, the simulation of gadgets $G_1$-$G_5$ would require $t_{G_5}$ shares of both $t_{G_1}$ and $t_{G_4}$. As a result, the required set of input shares ($I$) would include duplicate entries $2 \cdot t_{G_5}$, which cannot be simulated for $t_{G_5} = t$. From this it is clear that one of the inputs to $G_5$ needs to be refreshed. By further following the flow from gadgets $G_3$ through $G_1$ (the input), we conclude that the simulation of Algorithm 2 requires $|I| = t_{G_1} + o_{G_1} + t_{G_2} + o_{G_2} + t_{G_3} + t_{G_5} \leq t_{A_2}$ of the input shares, and thus is $t-$NI. □

## Appendix B Security Proof X2B (Algorithm 4)

Note that the X2B gadget description is independent of the specific masked algorithm used for SecADD/SecADD$_q$. SecADD denotes the $t-$NI arithmetic addition of Boolean shares, as from [5] or Algorithm 2 & Appendix D. Furthermore, SecEXP refers to the $t-$NI expansion of Boolean shares, as proposed in [17] and formalised in Appendix E (Algorithm 10). As opposed to a $t-$SNI SecREF, it requires less fresh randomness (strictly $d+1$ shares) and doubles the Boolean share count.

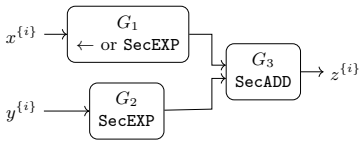



**Fig. 5.** An abstract diagram of X2B & A2B (Algorithm 4 & 3): the $t$-NI gadgets are depicted with a single border. This structure is recursively applied and combined for higher masking orders.

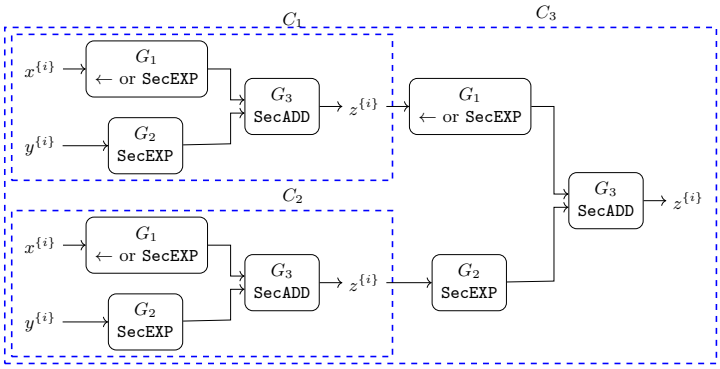



**Fig. 6.** Recursive structure of X2B (Algorithm 4) and A2B (Alg 3).

To argue about the higher-order security of Algorithm 3/4, we prove it to be word-level $t$-NI secure with $t+1$ shares. This provides resistance against a probing adversary (A2B) or allows the use of the gadget in larger compositions (X2B). We show how probes on intermediate word-level values in the algorithm can be perfectly simulated with only a limited number of input shares, by iterating over all possible intermediate variables. We show that all word-level probes can be simulated with no more number of input shares. To this end, we model all gadgets to operate on word-level data shares.

**Theorem 2.** *The gadgets* `A2B` *and* `X2B` *(Alg. 3 & 4) are word-level t-NI secure.*

*Proof.* We model Algorithms 3 & 4 as a sequence of $t$-NI gadgets, as shown in Figure 5. For the `A2B` gadget, the $t$-NI `SecEXP` gadgets (Line 5 and 7) are modeled as gadgets $G_1$ and $G_2$. For the `X2B` gadget, the signal assignment (Line 8) and/or $t$-NI `SecEXP` gadgets (Line 6 and 11) are modeled as gadgets $G_1$ and $G_2$. The $t$-NI gadget `SecADD` (Line 8 and 12, respectively) is modeled as gadget $G_3$. An adversary can probe the intermediate values and outputs of all gadgets $G_i$, except the output shares of the complete algorithm. The proof of Theorem 2 is a direct result from the direct chaining of $t$-NI gadgets $G_1$-$G_3$ resulting in a $t$-NI circuit (Figure 5) and the recursive application also results in such a structure (Figure 6). Each of the sub-circuits ($C_1$-$C_3$) is of the form shown in Figure 5, as well as the larger circuit (`A2B` & `X2B`). All sub-circuits are $t$-NI and directly chained, from which it follows that the entire structure is $t$-NI. □

## Appendix C Correctness & Security Proof `B2X2A` (Algorithm 5)

Note that Algorithm 5 is independent of the specific masked algorithms used for `SecNOT`, `X2B` or `FullXOR`. `SecNOT` refers to the $t-$NI computation of the Boolean negation, where only the first share of the Boolean-masked input is negated (Alg. 7). `FullXOR` is a $t$-NI secure unmasking of a Boolean sharing, consisting of a strong (free-$t$-SNI) mask refreshing and XOR'ing of all shares ([9,19] or Appendix E). `X2B` refers to the $t-$NI secure conversion of composite shares to Boolean shares encoding the same value (Section 3.2), a variant of the `A2B` operation (ALgorithm 4).

**Correctness.** Algorithm 5 securely converts Boolean shares $B$ to arithmetic shares $A$ encoding the same value. $y'$ is equivalent to $R_A + (\sim B) = R_A - B - 1$, with $R_A$ randomly sampled data. As a result $y$ is equal to $(-R_A + B + 1) - 1$ or $B - R_A$ (`SecNOT`). All resulting shares are XOR'd into a single share in Line 13, ensuring the full output $A$ is equal to $R_A + B - R_A = B$, which is the same data as input but shared differently.

**Security.** To argue about the higher-order security of Algorithm 5, we prove it to be word-level $t$-NI with $t + 1$ shares. This provides resistance against a probing adversary with $t$ word-level probes and allows the use of the gadget in larger compositions. Again, we show how probes on intermediate word values in the algorithm can be perfectly simulated with only a limited number of input shares, by iterating over all possible intermediate variables. We provide formal arguments on how they can be simulated relying on the $t-$NI properties of the sub-operations. We show that all probes can be simulated with no more number of input shares.

**Theorem 3.** *The gadget* `B2X2A` *(Algorithm 5) is word-level t-NI secure.*

*Proof.* We model Algorithm 5 as a sequence of $t$-NI gadgets, as shown in Figure 7. All gadgets are assumed to operate on full data words. The operations in

Lines 1-9 are not considered, as they operate on fresh random shares (non-input data). We model the linear operations in Line 10 and 12 as $t-$NI gadgets $G_1$ and $G_3$, respectively, which is trivially shown as they process their inputs in a share-wise manner. Line 11 (`X2B`) is mapped to $t-$NI gadget $G_2$. The final operation on Line 13 is represented by $t-$NI gadget $G_4$. An adversary can probe the intermediate values and outputs of all gadgets $G_i$, $t_{G_i}$ refers to the number of internal probes and $o_{G_i}$ the number of output probes.

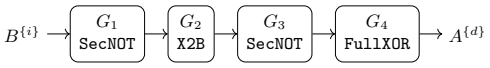



**Fig. 7.** An abstract diagram of `B2X2A` (Algorithm 5), $t$-NI gadgets are depicted with a single border.

To prove Theorem 3, we show that any set of probes on intermediate values $t_{G_i}$ and output shares $o_{G_i}$ of complete Algorithm 5 (Gadget 1-4), except for the output shares of the full composition, can be perfectly simulated with no more number of input shares $B^{\{i\}}$ ($\leq t_{A_4}$), with:

$$t_{A_4} = \sum_{i=1}^{4} t_{G_i} + \sum_{i=1}^{3} o_{G_i}$$

To simulate the $t_{G_4}$ intermediate probes of $t-$NI gadget $G_4$, $t_{G_4}$ shares of its input is required. To simulate the $t_{G_3}$ intermediate and $o_{G_3}$ output probes of gadget $G_3$, $t_{G_3} + o_{G_3}$ shares of the output of $G_2$ are required. This reasoning can be directly extended for all remaining gadgets in Algorithm 5. By following the flow from the output, through all gadgets, to the input, we conclude that the simulation of Algorithm 5 requires $|I| = t_{G_1} + o_{G_1} + t_{G_2} + o_{G_2} + t_{G_3} + o_{G_3} + t_{G_4} \leq t_{A_4}$ of the input shares and thus is $t$-NI. □

## Appendix D Our Masked Brent-Kung `SecADD` ($\texttt{SecADD}_{\texttt{BK}}$) Design & Security Proof (Section 4.1)

We construct a secure Brent-Kung adder from several masked components: `SecAND`, `SecOR`, `SecXOR` and `SecREF`. `SecAND` refers to the $t$-NI masked computation of the logical AND (Alg. 8). `SecXOR` refers to the sharewise $t$-NI computation of the logical XOR of Boolean shares (Alg. 6). `SecREF` describes a $t$-SNI algorithm to refresh Boolean shares, as proposed in [5,14] and included in Appendix E. `SecOR` denotes the $t$-NI masked computation of the logical OR, as described in Appendix E.

**Correctness.** We refer to [8] for the correctness proof. Our Brent-Kung adder architecture is optimized for the parameters of CRYSTALS-Kyber ($q = 3329$). Increasing or decreasing the amount of carry-generation and carry-propagation stages, allows to increase or decrease the operand bit width.

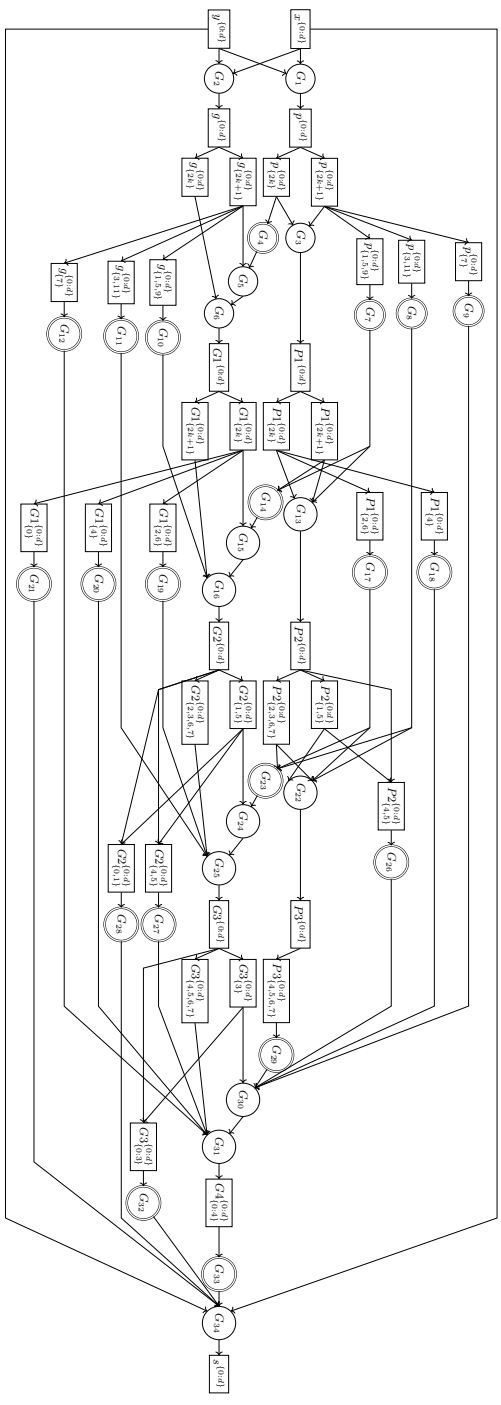


**Fig. 8.** The composition of $\texttt{SecADD}_{\mathsf{BK}}$ (Theorem 4) from $t-$NI gadgets (single circle) and $t-$SNI gadgets (double circle). The even bits of variables are indicated in subscript by $2k$, uneven bits by $2k+1$.

**Security.** To argue about the higher-order security of $\texttt{SecADD}_{\texttt{BK}}$ (Figure 8), we prove it to be bit-level $t$-NI with $t+1$ shares. This provides resistance against a probing adversary with $t$ bit-level probes and allows the use of the gadget in larger compositions. We show how probes on intermediate values in the algorithm can be perfectly simulated with only a limited number of input shares, by iterating over all possible intermediate variables. We provide formal arguments on how they can be simulated relying on the $t$-(S)NI properties of the sub-operations. We show that all probes can be simulated with no more number of input shares.

**Theorem 4.** *The gadget $\texttt{SecADD}_{\texttt{BK}}$ (Figure 8) is bit-level t-NI secure.*

*Proof.* As depicted in Figure 8, our $\texttt{SecADD}_{\texttt{BK}}$ design is modeled as a sequence of $t$-(S)NI gadgets and mapped as follows:

- SecREF [$t$-SNI]: $G_4, G_7 - G_{12}, G_{14}, G_{17} - G_{21}, G_{23}, G_{26} - G_{28}, G_{29}, G_{32} - G_{33}$,
- SecAND [$t$-NI]: $G_2, G_3, G_5, G_{13}, G_{15}, G_{22}, G_{24}, G_{30}$,
- SecOR [$t$-NI]: $G_1, G_6, G_{16}, G_{25}, G_{31}$,
- SecXOR [$t$-NI]: $G_{34}$.

The modelling of share-wise (linear) operations to $t$-NI gadgets can be trivially shown, as inputs are processed in a sharewise manner. An adversary can probe the intermediate values and outputs of all gadgets $G_i$ (except the output shares of the complete algorithm), $t_{G_i}$ refers to the number of internal probes and $o_{G_i}$ the number of output probes.

To prove Theorem 4, we show that the internal (bit-level) probes of complete gadget $\texttt{SecADD}_{\texttt{BK}}$ ($t_{A_{BK}}$) can be perfectly simulated with no more number of input shares $x^{\{i\}}$ and $y^{\{i\}}$ ($\leq t_{A_{BK}}$) with

$$t_{A_{BK}} = \sum_{i=1}^{34} t_{G_i} + \sum_{i=1}^{33} o_{G_i}$$

For $t$-NI security, simulation must succeed for an attacker which can probe any intermediate bit-value of all gadgets $t_{G_i}$ and the output shares of all gadgets $o_{G_i}$, except the output of the full composition. We rely on the $t$-(S)NI properties of each gadget to argue about their internal and output probes. For the simulation of a larger composition, the required shares of the inputs are added up. To ensure the simulation is sound, we will show that it is necessary to insert $t$-SNI SecREF gadgets. This is because a $t$-SNI gadget stops the propagation of probes from the output shares to the input shares, allowing its simulation to be performed independent of the number of probed output shares.

Now, we go over the simulation of the entire design in detail. Starting with the output and gadget $G_{34}$, its $t_{G_{34}}$ internal probes can be simulated with $t_{G_{34}}$ shares of inputs $x^{\{i\}}$ and $y^{\{i\}}$, and $t_{G_{34}}$ of the output shares of $G_{33}$, $G_{32}$, $G_{28}$ and $G_{21}$.

In order for the simulation of $G_3 - G_{33}$ to succeed, we model these gadgets to operate on the bit-level rather than on the complete variables. This means

that we build multi-bit SecOR and SecAND gates, which operate on independent bits in parallel (1-bit gadgets), which are each $t$-NI. The simulation succeeds because the bits are independent. We now give an example of this construction, the simulation of $G_{33}$, which appears several times throughout the composition (four stages of the adder). To simulate $t_{G_{33}}$ of its internal probes and $o_{G_{33}}$ of its output shares, $t_{G_{33}} + t_{G_{31}} + o_{G_{31}} + t_{G_{30}} + o_{G_{30}}$ output shares of $G_{25}$ are required. Because we model these gadgets to operate on individual bits (and these can be probed), there is no problem in simulation: $t_{G_{33}} + t_{G_{31}} + o_{G_{31}} + t_{G_{30}} + o_{G_{30}}$ shares of bit $k = 3$ of the output of $G_{25}$ ($G3$) are required, and $t_{G_{33}} + t_{G_{31}} + o_{G_{31}}$ shares of bits $k = 4, 5, 6, 7$ of $G3$ are required. If we the gadgets are defined on the word-level, $2 \cdot (t_{G_{33}} + t_{G_{31}} + o_{G_{31}})$ shares would be required which results in unsound simulation. However, as there are no duplicate entries (on the bit-level), the simulation succeeds.

Without the insertion of $t$-SNI refresh gadgets, the simulation of the entire design would not be sound. We now provide details for the simulation at particular points in the design, throughout the larger composition the shares required for simulation are added up. For $G_{29}$ to $G_{34}$:

$$t_{G3^{\{i\}}} = t_{G_{30}} + o_{G_{30}} + t_{G_{31}} + o_{G_{31}} + t_{G_{32}} + t_{G_{33}}$$

$$t_{P3^{\{i\}}} = t_{G_{29}}$$

For $G_{22}$ to $G_{34}$:

$$\begin{aligned} t_{G2^{\{i\}}} = t_{G_{24}} + o_{G_{24}} + t_{G_{25}} + o_{G_{25}} + t_{G_{27}} + t_{G_{28}} + t_{G_{30}} + o_{G_{30}} \\ + t_{G_{31}} + o_{G_{31}} + t_{G_{32}} + t_{G_{33}} \end{aligned}$$

$$t_{P2^{\{i\}}} = t_{G_{22}} + o_{G_{22}} + t_{G_{23}} + t_{G_{26}} + t_{G_{29}}$$

For $G_{13}$ to $G_{34}$:

$$\begin{aligned} t_{G1^{\{i\}}} = t_{G_{15}} + o_{G_{15}} + t_{G_{16}} + o_{G_{16}} + t_{G_{19}} + t_{G_{20}} + t_{G_{21}} + t_{G_{24}} + o_{G_{24}} + t_{G_{25}} + o_{G_{25}} \\ + t_{G_{27}} + t_{G_{28}} + t_{G_{30}} + o_{G_{30}} + t_{G_{31}} + o_{G_{31}} + t_{G_{32}} + t_{G_{33}} \end{aligned}$$

$$t_{P1^{\{i\}}} = t_{G_{13}} + o_{G_{13}} + t_{G_{14}} + t_{G_{17}} + t_{G_{18}} + t_{G_{22}} + o_{G_{22}} + t_{G_{23}} + t_{G_{26}} + t_{G_{29}}$$

For $G_3$ to $G_{34}$:

$$\begin{aligned} t_{g^{\{i\}}} = t_{G_5} + o_{G_5} + t_{G_6} + o_{G_6} + t_{G_{10}} + t_{G_{11}} + t_{G_{12}} + t_{G_{15}} + o_{G_{15}} + t_{G_{16}} + o_{G_{16}} \\ + t_{G_{19}} + t_{G_{20}} + t_{G_{21}} + t_{G_{24}} + o_{G_{24}} + t_{G_{25}} + o_{G_{25}} \\ + t_{G_{27}} + t_{G_{28}} + t_{G_{30}} + o_{G_{30}} + t_{G_{31}} + o_{G_{31}} + t_{G_{32}} + t_{G_{33}} \end{aligned}$$

$$\begin{aligned} t_{p^{\{i\}}} = t_{G_3} + o_{G_3} + t_{G_4} + t_{G_7} + t_{G_8} + t_{G_9} + t_{G_{13}} + o_{G_{13}} + t_{G_{14}} + t_{G_{17}} + t_{G_{18}} \\ + t_{G_{22}} + o_{G_{22}} + t_{G_{23}} + t_{G_{26}} + t_{G_{29}} \end{aligned}$$

By following the flow from gadgets $G_{34}$ through $G_1$, we conclude that the simulation of $\mathtt{SecADD}_{\mathtt{BK}}$ requires $|I| = |I_x| + |I_y| = t_{G_1} + o_{G_1} + t_{G_2} + o_{G_2} + t_{g^{\{i\}}} + t_{p^{\{i\}}} + t_{G_{34}} \leq t_{A_{BK}}$ of the input shares $x^{\{i\}}$ and $y^{\{i\}}$, and thus is $t$-NI. □

# Appendix E Remaining Gadgets and their Security Proofs

## E.1 SecXOR, SecNOT and SecAND

We include the `SecXOR`, `SecNOT` and `SecAND` gadgets for completeness. The `SecXOR` is a share-wise XOR of two Boolean shared (bit- or word-level) inputs, the `SecNOT` computes the logical inverse (e.g., Boolean negation) of two (bit- or word-level) Boolean shared inputs. Both are computed using strictly share-wise operations, and thus are $t$-NI.

**Algorithm 6** `SecXOR` [$t$-NI]

**Input Data** : $x^{\{0:d\}}$ and $y^{\{0:d\}}$
**Output Data** : $z^{\{0:d\}}$ such that $\bigoplus_{i=0}^{d} z^{\{i\}} = \bigoplus_{i=0}^{d} x^{\{i\}} \oplus \bigoplus_{i=0}^{d} y^{\{i\}}$

1: **for** $i = 0$ upto $d$ **do**
2: $\quad z^{\{i\}} \leftarrow x^{\{i\}} \oplus y^{\{i\}}$
3: **end for**

**Algorithm 7** `SecNOT` [$t$-NI]

**Input Data** : $x^{\{0:d\}}$
**Output Data** : $y^{\{0:d\}}$ such that $y = \sim x$

1: $y^{\{0\}} \leftarrow\sim x^{\{0\}}$
2: **for** $i = 1$ upto $d$ **do**
3: $\quad y^{\{i\}} \leftarrow x^{\{i\}}$
4: **end for**

The `SecAND` gadget, as proposed in [33], computes the bitwise logical AND of two Boolean shared values and is $t$-NI secure. We refer to [14] for its $t$-NI glitch-robust security proof. In Algorithm 8 we use $\otimes$ to denote a (field) multiplication.

## E.2 SecOR

The `SecOR` gadget computes the OR of two Boolean shared inputs, following De Morgan's law.

**Robust Probing Security:** We now prove that the `SecOR` gadget is word-level $t$-NI with $t + 1$ shares, considering the leakage effects from Section 2.3.

We note that Algorithm 9 is independent of the specific masked algorithms used for `SecNOT` and `SecAND`. `SecNOT` refers to the $t$-NI computation of the Boolean negation, where the first share of the Boolean-masked input is negated. `SecAND` refers to the $t$-NI masked computation of the bitwise AND, e.g., DOM-indep AND [33].

**Algorithm 8** `SecAND` [$t$-NI]

**Input Data** : $x^{\{0:d\}}$ and $y^{\{0:d\}}$

**Output Data** : $z^{\{0:d\}}$ such that $\bigoplus_{i=0}^{d} z^{\{i\}} = \bigoplus_{i=0}^{d} x^{\{i\}} \wedge \bigoplus_{i=0}^{d} y^{\{i\}}$

1: **for** $i = 0$ upto $d$ **do**
2: **for** $j = i + 1$ upto $d$ **do**
3: $r^{\{ij\}} \leftarrow \texttt{Rand}(k)$
4: $u^{\{ij\}} \leftarrow x^{\{i\}} \otimes y^{\{j\}} \oplus r^{\{ij\}}$
5: $u^{\{ji\}} \leftarrow x^{\{j\}} \otimes y^{\{i\}} \oplus r^{\{ij\}}$
6: **end for**
7: **end for**
8: **for** $i = 0$ upto $d$ **do**
9: $z^{\{i\}} \leftarrow \texttt{Reg}[x^{\{i\}} \otimes y^{\{i\}}] \oplus \bigoplus_{j=0, j\neq i}^{d} u^{\{ij\}}$
10: **end for**

**Algorithm 9** `SecOR` [$t$-NI]

**Input Data** : $x^{\{0:d\}}$ and $y^{\{0:d\}}$

**Output Data** : $z^{\{0:d\}}$ such that $\bigoplus_{i=0}^{d} z^{\{i\}} = \bigoplus_{i=0}^{d} x^{\{i\}} \vee \bigoplus_{i=0}^{d} y^{\{i\}}$

1: $a^{\{0:d\}} \leftarrow \texttt{SecNOT}(x^{\{0:d\}})$
2: $b^{\{0:d\}} \leftarrow \texttt{SecNOT}(y^{\{0:d\}})$
3: $z^{\{0:d\}} \leftarrow \texttt{SecAND}(a^{\{0:d\}}, b^{\{0:d\}})$
4: $z^{\{0:d\}} \leftarrow \texttt{SecNOT}(z^{\{0:d\}})$

**Theorem 5.** *The gadget `SecOR` (Algorithm 9) is word-level t-NI secure.*

*Proof.* This is a direct result from the linear `SecNOT` on both independently shared inputs and the `SecAND` gadget is $t$-NI secure. □

### E.3 SecEXP, modified from [17]

The `SecEXP` gadget expands (and doubles) a Boolean sharing, using $d+1$ random shares $r^{\{i\}}$. The first half of the output $y^{\{0:2d+1\}}$ consists of the XOR-ing of $r$ and input $x$, while the second half consists of $r$, resulting in an equivalent Boolean sharing. As such, the unshared output is equal to the unshared input.

**Algorithm 10** `SecEXP` [$t$-NI]

**Input Data** : $x^{\{0:d\}}$

**Output Data** : $y^{\{0:2d+1\}}$ such that $\bigoplus_{i=0}^{d} x^{\{i\}} = \bigoplus_{i=0}^{2d+1} y^{\{i\}}$

1: $r^{\{0:d\}} \leftarrow \texttt{Rand}(k)$
2: $y^{\{0:d\}} \leftarrow \texttt{SecXOR}(x^{\{0:d\}}, r^{\{0:d\}})$
3: $y^{\{d+1:2d+1\}} \leftarrow r^{\{0:d\}}$
4: $y^{\{0:2d+1\}} \leftarrow \texttt{Reg}[y^{\{0:2d+1\}}]$

**Robust Probing Security:** We now show that the `SecEXP` gadget is correct and prove it to be word-level $t$-SNI, considering the leakage effects from Section 2.3.

In Algorithm 10, `SecXOR` refers to the sharewise XOR operation, which is $t$-NI as it is linear. `Reg[]` corresponds to adding a register between the computation of $y^{\{0:2d+1\}}$ and the output of the gadget.

**Correctness.** From the description in Algorithm 10, it is trivial that $\bigoplus_{i=0}^{d} x^{\{i\}} = \bigoplus_{i=0}^{2d+1} y^{\{i\}}$ and hence it is correct.

**Security.** To argue about the higher-order security of Algorithm 10, we prove it to be word-level $t$-NI with $t+1$ shares. This provides resistance against a probing adversary with $t$ word-level probes and allows the use of the gadget in larger compositions, such as the `X2B`.

**Theorem 6.** *The gadget `SecEXP` (Algorithm 10) is word-level t-NI secure.*

*Proof.* This is a direct result from the share-by-share XOR with random shares, as all intermediate probes can be simulated with as many input shares. □

### E.4 SecREF, from [14]

Below is the glitch-robust $t$-SNI refresh gadget, as proposed in [14], and which is used throughout this work. The main idea is to add a sharing of zero to the input $x$ and register it. We refer to the original work for the correctness and security proofs.

**Algorithm 11** `SecREF` [$t$-SNI]

**Input Data** : $x^{\{0:d\}}$
**Input Data** : $r^{\{0:d\}}$ such that $\bigoplus_{i=0}^{d} r^{\{i\}} = 0$ ▷ random 0-sharing
**Output Data** : $y^{\{0:d\}}$ such that $\bigoplus_{i=0}^{d} x^{\{i\}} = \bigoplus_{i=0}^{d} y^{\{i\}}$

1: $z^{\{0:d\}} \leftarrow$ `SecXOR`$(x^{\{0:d\}}, r^{\{0:d\}})$
2: $y^{\{0:d\}} \leftarrow$ `Reg`$[z^{\{0:d\}}]$

### E.5 RefreshMasks, from [18]

The `RefreshMasks` gadget consists of $d+1$ mask refresh operations `SecREF`. As shown in [19], the gadget is (free-)$t$-SNI, which means all output variables except one can always be perfectly simulated. The gadget described in [18], which is recalled below, only requires $(d+1)d/2$ random values.

**Algorithm 12** `RefreshMasks` [$t$-SNI]

**Input Data** : $x^{\{0:d\}}$

**Output Data** : $y^{\{0:d\}}$ such that $\bigoplus_{i=0}^{d} x^{\{i\}} = \bigoplus_{i=0}^{d} y^{\{i\}}$

1: $y^{\{0:d\}} \leftarrow x^{\{0:d\}}$
2: **for** $j = 0$ to $d$ **do**
3: $\quad r^{\{j:d\}} \leftarrow \text{Rand}(k)$ $\quad \triangleright \bigoplus_{i=j}^{d} r^{\{i\}} = 0$
4: $\quad y^{\{j:d\}} \leftarrow \texttt{SecREF}(x^{\{j:d\}}, r^{\{j:d\}})$
5: **end for**

### E.6 FullXOR, from [5]

The `FullXOR` gadget first refreshes the input shares, before unmasking the shared value. As shown in [5,19], the gadget is $t$-NI.

**Algorithm 13** `FullXOR` [$t$-NI]

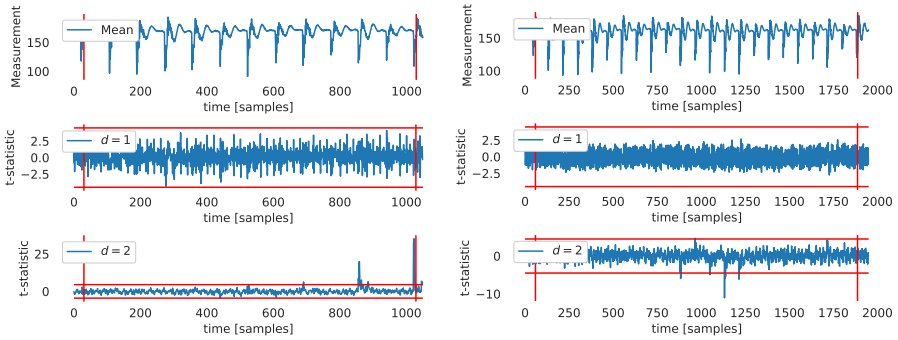
**Input Data** : $x^{\{0:d\}}$

**Output Data** : $y^{\{0\}}$ such that $\bigoplus_{i=0}^{d} x^{\{i\}} = y^{\{0\}}$

1: $z^{\{0:d\}} \leftarrow \texttt{RefreshMasks}(x^{\{0:d\}})$
2: $y^{\{0\}} \leftarrow z^{\{0\}} \oplus \cdots \oplus z^{\{d\}}$

## Appendix F TVLA results

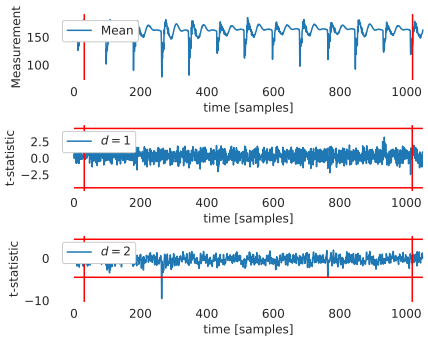
We want to bring the reader's attention to the complexities of observing higher-order leakages. For our second-order implementation, third-order leakages show for certain modes, as anticipated, and not for others. We can attribute this phenomenon to effects described in [47]. More specifically, to observe higher-order leakage one needs to collect much more traces. One could expect that if we continued acquiring traces up to 500M or even 1 billion traces, our second-order implementation would exhibit third-order leakages more clearly in other modes of operation too. We do not include such figures due to the practical and computational infeasibility.



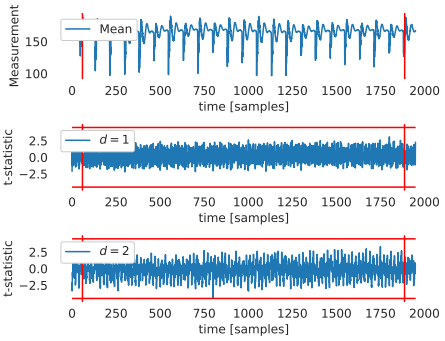

(a) A2B mod $2^{13}$.




(b) A2B mod 3329.

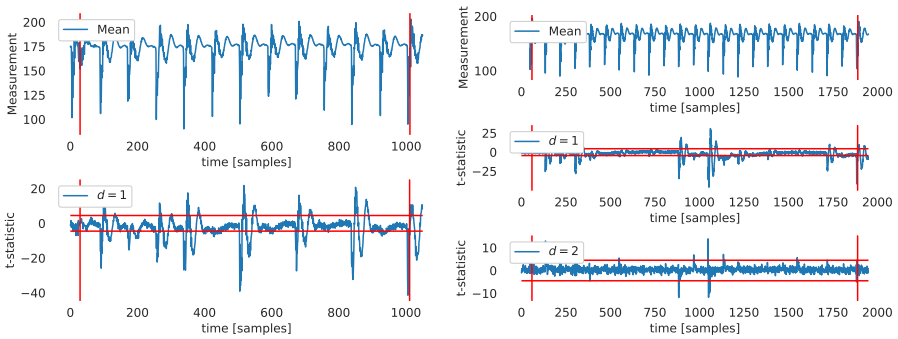



(c) B2A mod $2^{13}$.



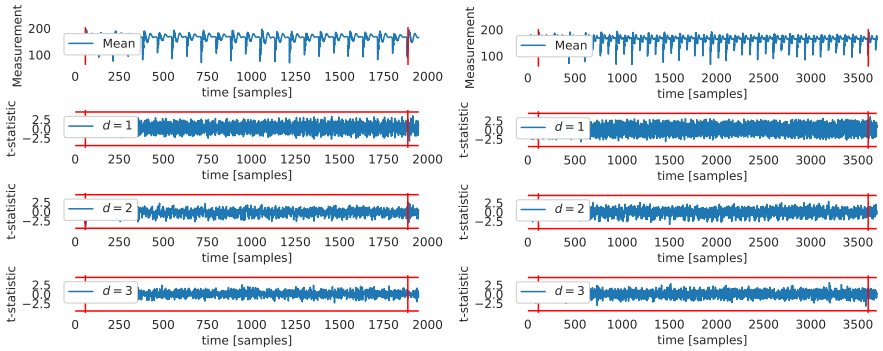

(d) B2A mod 3329.

**Fig. 9.** $1^{st}$ & $2^{nd}$-order univariate fixed-vs.-random TVLA results for first-order mask conversions (2 shares) using 100M traces with PRNG ON. For each subfigure, the upper plot shows the mean trace. The $\pm$ 4.5 threshold is marked by red lines.



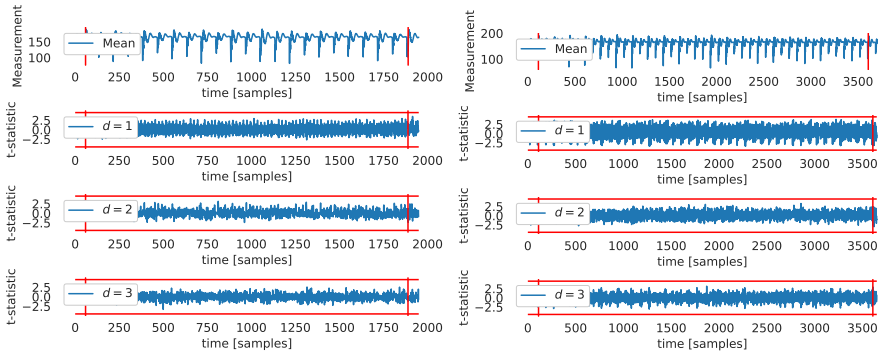

(a) $1^{st}$-order (2 share) implementation (b) $2^{nd}$-order (3 share) implementation

**Fig. 10.** $1^{st}$ (& $2^{nd}$)-order univariate fixed-vs.-random TVLA results for A2B mod $2^{13}$ (2 and 3 shares) using 500K traces with PRNG OFF. For each subfigure, the upper plot shows the mean trace.




(a) A2B mod $2^{13}$.

(b) A2B mod 3329.




(c) B2A mod $2^{13}$.

(d) B2A mod 3329.

**Fig. 11.** $1^{st}$, $2^{nd}$ (&$3^{rd}$)-order univariate fixed-vs.-random TVLA results for second-order mask conversions (3 shares) using 100M traces with PRNG ON. For each subfigure, the upper plot shows the mean trace. The ± 4.5 threshold is marked by red lines.

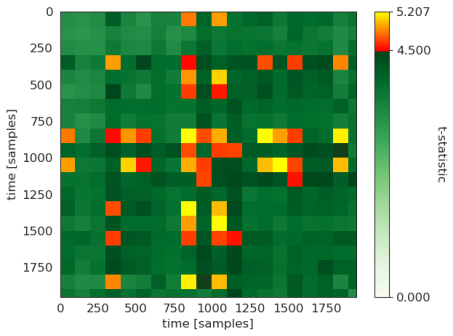
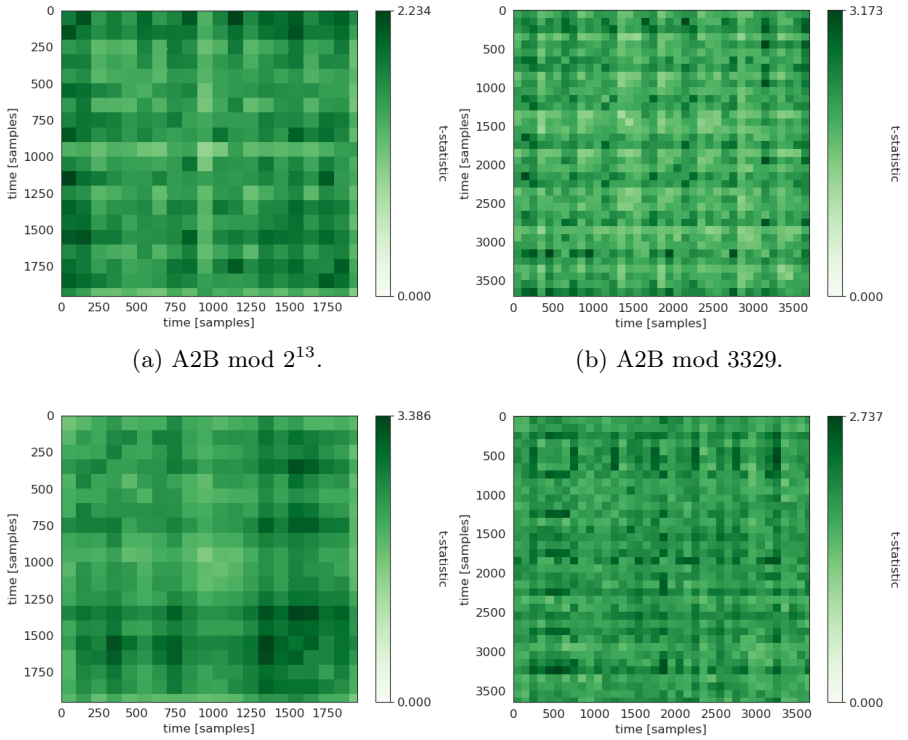



**Fig. 12.** Bivariate analysis of second-order mask conversion implementation (3 shares), 500K traces, PRNG OFF. Max-pooling guarantees that t-test peaks are visible.




(a) A2B mod $2^{13}$.

(b) A2B mod 3329.

(c) B2A mod $2^{13}$.

(d) B2A mod 3329.

**Fig. 13.** (Best viewed on-screen.) Bivariate analysis of second-order mask conversion implementation (3 shares), 100M traces, PRNG ON. Max-pooling guarantees that t-test peaks are visible.