Analysis and Comparison of Table-based Arithmetic to Boolean Masking

Michiel Van Beirendonck, Jan-Pieter D'Anvers and Ingrid Verbauwhede

imec-COSIC KU Leuven
Kasteelpark Arenberg 10 - bus 2452, 3001 Leuven, Belgium
{firstname}.{lastname}@esat.kuleuven.be

Abstract. Masking is a popular technique to protect cryptographic implementations against side-channel attacks and comes in several variants including Boolean and arithmetic masking. Some masked implementations require conversion between these two variants, which is increasingly the case for masking of post-quantum encryption and signature schemes. One way to perform Arithmetic to Boolean (A2B) mask conversion is a table-based approach first introduced by Coron and Tchulkine, and later corrected and adapted by Debraize in CHES 2012. In this work, we show both analytically and experimentally that the table-based A2B conversion algorithm proposed by Debraize does not achieve the claimed resistance against differential power analysis due to a non-uniform masking of an intermediate variable. This non-uniformity is hard to find analytically but leads to clear leakage in experimental validation. To address the non-uniform masking issue, we propose two new A2B conversions: one that maintains efficiency at the cost of additional memory and one that trades efficiency for a reduced memory footprint. We give analytical and experimental evidence for their security, and will make their implementations, which are shown to be free from side-channel leakage in 100.000 power traces collected on the ARM Cortex-M4, available online. We conclude that when designing side-channel protection mechanisms, it is of paramount importance to perform both a theoretical analysis and an experimental validation of the method.

Keywords: Masking · A2B conversion · ARM Cortex-M4 · Post-Quantum Cryptography

1 Introduction

While the underlying algorithms might be black-box secure, cryptographic applications deployed in real-world applications can be subject to side-channel attacks, in which an adversary uses information based on physical artefacts of the computation to break the cryptosystem. These artefacts could include the timing of the execution [Koc96], the power usage [KJJ99], or electromagnetic radiation [QS01] inadvertently sent out due to the electronic calculations.

Attacks based on Differential Power Analysis (DPA) [KJJ99] are particularly powerful, since they require only minimal assumptions about the underlying device and are able to function even with low signal-to-noise ratios. Especially for embedded cryptographic devices deployed in real-world applications, security against side-channel attacks is of vital importance as these devices can typically be directly observed and manipulated by malicious entities.

A popular countermeasure against DPA is masking [CJRR99], in which sensitive values are split into several parts, typically called shares, so that the device running the algorithm never directly computes on sensitive values. Masking methods are designed so that even when an adversary can completely retrieve the information about all but one share, he can not infer any information about the masked secret value.

Two popular masking methods are Boolean and arithmetic masking. In Boolean masking, shares are obtained by xor-ing the sensitive value with uniformly random bitstrings, while arithmetic masking is achieved by adding uniformly random integers. As can be expected, Boolean masking is typically more efficient for Boolean operations while arithmetic masking excels at arithmetic operations. In some designs, it leads therefore to more optimal results when arithmetic and Boolean masking are combined. Such designs require conversion algorithms that allow to convert from Arithmetic to Boolean (A2B) and from Boolean to Arithmetic (B2A) masking.

Previously, use of these conversions was limited to a select number of ciphers and hash functions, typically based on Add-Rotate-XOR operations. Nowadays they are increasingly being used to secure post-quantum public-key algorithms, including the masked implementations of several candidates of the NIST Post-Quantum Standardization process such as NewHope [OSPG18], Dilithium [MGTF19], and Saber [VBDK$^+$20]. In the future, other lattice-based candidates can be expected to also use these conversion primitives to secure their implementations.

First-order secure B2A and A2B conversion algorithms were initially proposed by Goubin [Gou01]. Goubin's B2A method remains popular due to its efficiency, while the time complexity of the A2B method was later improved by Coron et al. [CGTV15]. Higher-order secure conversion algorithms have been described and subsequently improved in [CGV14, Cor17, BCZ18, HT19].

Another method to perform first-order A2B conversions was proposed by Coron and Tchulkine [CT03]. Their innovation was to divide masked variables into smaller chunks and convert each chunk individually using a pre-computed table. Neiße and Pulkus [NP04] modified this method, resulting in a conversion that was secure against DPA at the cost of weakening the security against Simple Power Analysis (SPA) attacks. In 2012, Debraize [Deb12] corrected a bug in the original Coron and Tchulkine [CT03] conversion and introduced a new method, aimed at improving the time complexity of the conversion on 8-bit and 16-bit microprocessors.

In this paper, we show that this last method is flawed. The specific format of the table in [Deb12], which aims at reducing the number of arithmetic operations in the main conversion loop, introduces an intermediate value that is correlated with its unmasked equivalent. We give both a theoretical analysis of this flaw and an experimental validation using the well-known Test Vector Leakage Assessment (TVLA) [GJJR11] approach, showing clear side-channel leakage of the conversion. This flaw undermines the side-channel security of the conversion algorithm and of implementations that make use of it such as [OSPG18]. Subsequently, we propose and experimentally validate two solutions. The first solution keeps the efficient conversion of Debraize, but increases the memory consumption proportionally to the number of converted chunks. The second solution reintroduces an omitted arithmetic operation, resulting in a conversion that is similar to [CT03], but with more efficient pre-computed tables. The flaw in [Deb12] is difficult to recognize from the algorithm, but quickly shows up in leakage tests, highlighting the importance of practical validation in side-channel research.

2 Preliminaries

Let $\oplus$ denote a bitwise exclusive or operation and let $\|$ denote bitstring concatenation. Let $x \leftarrow \mathcal{U}(\{0,1\}^n)$ denote uniformly sampling a n-bit number x. In this paper, we will regularly divide bitstrings into chunks of k bits, where k is determined by the context. For a bitstring y, we will denote with y_i the i^{th} such chunk so that:

$$y = y_{n-1} \| y_{n-2} \| \cdots \| y_0.$$

Masking splits a sensitive variable x in a cryptographic algorithm into multiple shares. A first-order masking scheme typically uses two shares x_1 and x_2, such that $x = x_1 \odot x_2$, and performs all operations on each of the shares separately. Since these shares are randomized at

Table 1: Overview of the different table-based A2B conversions.

	Precomputation	Conversion	Correct	Secure
CT [CT03]	Algorithm 1, 2*	-†	✗	✓
CT (fixed) [Deb12]	Algorithm 1, 2	-†	✓	✓
CT (imp.) [Deb12]	Algorithm 3	Algorithm 4	✓	✓
Debraize [Deb12]	Algorithm 5	Algorithm 6	✓	✗
Debraize (fixed) [§5.1]	Algorithm 7	Algorithm 8	✓	✓
Dual-lookup [§5.2]	Algorithm 9, 10	Algorithm 11	✓	✓

* with k-bit γ
† explained in Subsection 3.1

each execution of the algorithm, the actual leakage from only one share is no longer correlated to the sensitive variable x. More specifically, an adversary that sees the information of all but one share will not be able to obtain any information about the secret.

The random shares (x_1,x_2) are created by uniformly sampling a mask $R \leftarrow \mathcal{U}(\{0,1\}^n)$, with n the bitlength of the sensitive value. Subsequently, $x_2 = R$, and x_1 is computed to satisfy the relationship $x = x_1 \odot x_2$. The operator $\odot$ denotes the type of masking. This paper considers conversions between two types of masking: Boolean masking, i.e. $x = B \oplus R$ where (B,R) is the mask for a sensitive value x, and arithmetic masking with a power-of-two modulus, i.e. $x = A + R \bmod 2^n$ where (A,R) is the mask for the sensitive value x. Modular reduction with a power-of-two modulus consists of a simple bit truncation. All arithmetic operations throughout this paper use such a truncation, and therefore we sometimes omit it for clarity. We use $c_{out}(y+z)$ to denote the carry from a modular addition, i.e. $c_{out}(y+z) = \lfloor (y+z)/2^n \rfloor$, where n is the bitlength of the inputs y and z.

3 Table-based A2B conversion

In this section we first introduce the table-based A2B conversion due to Coron and Tchulkine, which will be referred to as CT. We then discuss a correctness problem of this conversion as noticed by Debraize [Deb12] and will give both a fixed version of the conversion, denoted CT (fixed), and an improved version by Debraize, denoted CT (imp.). Finally, we present the A2B conversion method as introduced by Debraize [Deb12]. An overview of these different conversions can be found in Table 1.

3.1 Coron and Tchulkine

An arithmetic masking $x = A + R$ can be converted to a Boolean masking $x = B \oplus R$ by computing $B = (A+R) \oplus R$. For security reasons, one must be careful not to unmask x in the intermediate computation $(A+R)$. Table-based conversions prevent the intermediate unmasking by using a pre-computed table, $G[A] = (A+r) \oplus r$, which stores the conversion for a fixed mask r. However, as generating and storing a full table for every possible input value A would be inefficient, this table typically only provides conversion for short input chunks, e.g. $G[A_l] = (A_l+r) \oplus r$ for a k-bit input A_l and mask r.

To convert a masked input (A,R), the conversion algorithm iteratively takes the k least significant bits of (A,R), denoted (A_l,R_l) and performs the following steps: first (A_l,R_l) is remasked to (A_l,r), then (A_l,r) is converted into a Boolean masking (B_l,r) using the conversion table, and finally (B_l,r) is remasked to the original masking (B_l,R_l). When proceeding to the next k bits to be converted, care should be taken for the possibility of a carry that needs to be propagated to the next chunk in the arithmetic masking.

In order to do so, the method of Coron and Tchulkine employs two tables, G and C. Table G is the aforementioned table that converts an arithmetic mask A_l to a Boolean mask $B_l = (A_l+r) \oplus r$, whereas table C tracks the carry $c_{out}(A_l+r)$ that must be added to the

Algorithm 1: Pre-computation of G [CT03]

```
1 $r \leftarrow \mathcal{U}(\{0,1\}^k)$
2 for $A=0$ to $2^k-1$ do
3     $G[A]=(A+r)\oplus r$
4 end
5 return $G,r$
```

Algorithm 2: Pre-computation of C [CT03, Deb12]

```
1 $\gamma \leftarrow \mathcal{U}(\{0,1\}^{(n-1)\cdot k})$
2 for $A=0$ to $2^k-1$ do
3     $C[A]=c_{out}(A+r)+\gamma \bmod 2^{(n-1)\cdot k}$
4 end
5 return $C$, $\gamma$
```

Algorithm 3: Table T generation [Deb12]

```
1 $r \leftarrow \mathcal{U}(\{0,1\}^k)$
2 $\gamma \leftarrow \mathcal{U}(\{0,1\}^{(n-1)\cdot k})$
3 for $A=0$ to $2^k-1$ do
4     $T[A]=(A+(\gamma \| r) \bmod 2^{n\cdot k})\oplus r$
5 end
6 return $T,r,\gamma$
```

remaining unconverted chunks A_h. This carry is masked with an arithmetic mask γ, as its value is correlated to the unmasked x, i.e. $C[A_l]=c_{out}(A_l+r)+\gamma$. It is then possible to securely add the carry to A_h in two subsequent steps, $A_h \leftarrow A_h+C[A_l]$ and $A_h \leftarrow A_h-\gamma$. The pre-computation of G and C is illustrated in Algorithms 1 and 2, respectively. Note that table C includes a correction from [Deb12], which is explained later in this section.

The original A2B conversion of Coron and Tchulkine was improved and corrected by Debraize [Deb12], who noted that both the information provided by tables G and C in the Coron-Tchulkine method can be summarized in one unique table T, which is pre-computed as illustrated in Algorithm 3. In this table, the carry $c_{out}(A+r)$ is automatically propagated to the $(k+1)^{th}$ least-significant bit, where it is masked by the addition of γ. Furthermore, Debraize proposed to move the masking with γ and r outside of the main algorithm loop, by pre-masking with γ and $(r \| \ldots \| r)$. Using these two approaches, the improved Coron-Tchulkine conversion proceeds as given in Algorithm 4.

At the same time, it was Debraize who noted that a correct conversion requires that γ in table C is of size $(n-1)\cdot k$ bits, rather than k bits as in its initial specification. Previously, the computation was incorrect when $\gamma = 2^k-1$ and $c_{out}(A_l+r)=1$, such that $C[A_l] = \gamma + 1 \bmod 2^k = 0$. Then, when the first chunk gets converted, $A_h \leftarrow A_h + C[A_l] - \gamma \bmod 2^{(n-1)\cdot k}$ is not correctly equal to A_h+1. The main problem are the differing moduli, originally 2^k in the table C and $2^{(n-1)\cdot k}$ in the addition. For a correct calculation, γ must be at least of size $(n-1)\cdot k$ bits. We will distinguish between the original Coron-Tchulkine method from [CT03], the fixed Coron-Tchulkine method with table C computed as in Algorithm 2, and the improved Coron-Tchulkine method as in Algorithm 4.

3.2 Debraize

Apart from correcting a bug and improving the previous method, Debraize introduced a new method that further improves the performance of the conversion on 8-bit or 16-bit microprocessors. This new method groups two conversion tables G into a table T, where one half of T converts the input when no carry is present, and the other half does the conversion when a carry from a previous conversion is present. On top of returning the masked value $B_l=(A_l+r)\oplus r$, this table also returns a one-bit carry that is Boolean masked, $\beta=c_{out}(A_l+r)\oplus\rho$. The table T is generated following Algorithm 5 and makes the table

Algorithm 4: A2B conversion of a $(n{\cdot}k)$-bit variable [CT03, Deb12]

```
   input  : $(A,R)$ such that $x = A + R \bmod 2^{n\cdot k}$,
            $T$, $r$, $\gamma$
   output : $B$ such that $x = B \oplus R$
   /* Let $A = (A_h \| A_l)$, $R = (R_h \| R_l)$ with $A_l$, $R_l$ the $k$ least significant
      bits. $A_h$, $A_l$, $R_h$, $R_l$ are updated at the same time as $A$, $R$.        */
 1 $\Gamma \leftarrow \sum_{i=1}^{n-1} 2^{i\cdot k} \cdot \gamma \bmod 2^{n\cdot k}$
 2 $A \leftarrow A - (r \| ... \| r) \bmod 2^{n\cdot k}$
 3 $A \leftarrow A - \Gamma \bmod 2^{n\cdot k}$
 4 for $i = 0$ to $n-1$ do
 5     $A \leftarrow A + R_l \bmod 2^{(n-i)\cdot k}$
 6     $A \leftarrow A_h \| 0 + T[A_l] \bmod 2^{(n-i)\cdot k}$
 7     $B_i \leftarrow A_l \oplus R_l$
 8     $A \leftarrow A_h$
 9     $R \leftarrow R_h$
10 end
11 return $(B_{n-1} \| ... \| B_0) \oplus (r \| ... \| r)$
```

Algorithm 5: Table T generation [Deb12]

```
1 $r \leftarrow \mathcal{U}(\{0,1\}^k)$
2 $\rho \leftarrow \mathcal{U}(\{0,1\})$
3 for $A = 0$ to $2^k - 1$ do
4     $T[\rho][A] = (A + r) \oplus (\rho \| r)$
5     $T[(\rho \oplus 1)][A] = (A + r + 1) \oplus (\rho \| r)$
6 end
7 return $T, r, \rho$
```

C obsolete in the A2B conversion. The main benefit of Debraize's new method is that the arithmetic operation $A \leftarrow A_h \| 0 + T[A_l] \bmod 2^{(n-i)\cdot k}$ is no longer necessary. In this operation, the intermediate data is of the same size as the total conversion, which is typically expensive when 32-bit conversions are executed on 8-bit or 16-bit embedded processors.

A2B conversion using Debraize's method then proceeds similarly to the improved method of Coron and Tchulkine. Given a masked input (A,R), the algorithm iteratively takes the k least significant bits (A_l, R_l), removes the mask R_l from A_l such that it is only masked with r, and uses the table to transform A_l into the Boolean masked chunk B_l and a masked carry β. Depending on the previous carry, the appropriate half of T is used for the A2B transformation lookup. The chunk B_l is then remasked with the original mask R_l before continuing on to the next chunk. Algorithm 6 gives an overview of this procedure. Note that in the new method only a single table lookup is necessary per iteration, similarly to the improved CT method.

4 Security weakness in Debraize's A2B conversion

In this section, we show that Debraize's A2B conversion is still vulnerable to differential power attacks due to a bias in the masking of one of the intermediate variables. We do so first experimentally, showing side-channel leakage in actual measurements, and subsequently we give the theoretical analysis of the vulnerability.

Algorithm 6: Conversion of a $n{\cdot}k$-bit variable [Deb12]

```
   input  : $(A,R)$ such that $x = A + R \bmod 2^{n\cdot k}$,
            $T$, $r$, $\rho$
   output : $B$ such that $x = B \oplus R$
   /* Let $A = (A_h \| A_l)$, $R = (R_h \| R_l)$ with $A_l$, $R_l$ the $k$ least significant
      bits. $A_h$, $A_l$, $R_h$, $R_l$ are updated at the same time as $A$, $R$.  */
1  $A \leftarrow A - (r\|...\|r) \bmod 2^{n\cdot k}$
2  $\beta \leftarrow \rho$
3  for $i = 0$ to $n-1$ do
4      $A \leftarrow A + R_l \bmod 2^{(n-i)\cdot k}$
5      $\beta \| B_i \leftarrow T[\beta][A_l]$
6      $B_i \leftarrow B_i \oplus R_l$
7      $A \leftarrow A_h$
8      $R \leftarrow R_h$
9  end
10 return $B = (B_{n-1}\|...\|B_0) \oplus (r\|...\|r)$
```

4.1 Experimental validation

Setup We implemented Debraize's A2B conversion on an STM32F303 microcontroller manufactured by ST Microelectronics. The microcontroller features the popular 32-bit ARM-Cortex M4 RISC core operating at a frequency of up to 72 MHz, as well as 256 kB of flash memory and 40 kB of SRAM.

To facilitate side-channel measurements, the STM32F303 is mounted on a custom PCB target board that guarantees stable behaviour of the chip. This PCB is stripped of all of the unnecessary components that would otherwise introduce additional noise into the measurements. It contains a dedicated shunt resistor to monitor side-channel information through the chip's instantaneous power consumption, and, to ensure maximal stability, is driven by an external power supply at 3.2 V.

We use a Tektronix DPO 70404C digital oscilloscope to collect instantaneous power measurements with a sample rate of 625 MS/s during A2B conversion. In between the oscilloscope and the PCB we add a PA 303 SMA pre-amplifier to perform analog pre-processing of the collected traces. A central PC is used to communicate data to the board through a serial USART connection, as well as to collect and analyze power measurements. We use the on-chip clock at 48 MHz, and disable the data cache and interrupts to enforce constant-time execution.

Method To show whether or not the implementation exhibits side-channel leakage, we perform the well-known Test Vector Leakage Assessment (TVLA) [GJJR11]. This method works by creating two sets of measurements, which are partitioned according to the sensitive unmasked data that is processed. We use a *fixed vs. random* partitioning of the data. The fixed class corresponds to conversions where $A+R=x_{fix}$ and x_{fix} is set to 0, whereas the random class corresponds to measurements where $A+R=x_{rand}$. Under the independent leakage assumption that the instantaneous power consumption depends only on A and R, but not on the sensitive unmasked $x=A+R$, these two classes should have no observable differences in their mean power consumption. Differences are detected by computing *Welch's t-statistic* for every sample in the measurements as:

$$t = \frac{\overline{X}_{fix} - \overline{X}_{rand}}{\sqrt{\frac{\sigma^2_{fix}}{N_{fix}} + \frac{\sigma^2_{rand}}{N_{rand}}}},$$

where $\overline{X}$ denotes the means of a set, σ^2 the variance, and N the number of samples. The null-hypothesis of equal means is rejected with a confidence greater than 99.999% when the

t-statistic exceeds the value ± 4.5 for a large number of measurements. In other words, t values outside this range indicate that $\overline{X}_{fix}$ and $\overline{X}_{rand}$ are distinguishable, and thus that $A+R$ is leaked in the first statistical moment of the power traces.

Independent leakage in practice It is well known that the independent leakage assumption is easily violated by microarchitectural effects on an embedded microcontroller. Transitions of the memory bus, overwrites in the register file or in hidden registers in the ALU are all examples that can cause leakage that depends on the combined value of both shares. For example, consider the load operation on line 5 of Algorithm 6, for which the compiler might place the result in a dedicated register. In the first loop iteration, this register will be assigned

$$\beta \| B_0 = (c_{out}(x_0 - r) \oplus \rho) \| (x_0 \oplus r).$$

In the next loop iteration, this register will be overwritten by

$$\beta \| B_1 = (c_{out}((x_1 \| x_0) - (r \| r)) \oplus \rho) \| (x_1 \oplus r).$$

Any power consumption that depends on the transition of this register value, i.e. the Hamming distance, will cancel out the masks ρ and r, such that the power consumption depends on

$$(c_{out}((x_1 \| x_0) - (r \| r)) \oplus c_{out}(x_0 - r)) \| (x_1 \oplus x_0),$$

which is directly correlated to the unmasked chunks $(x_1 \| x_0)$ of x. Implementations can be carefully constructed to avoid this issue, e.g. by clearing registers before sensitive transitions or assigning different registers to different shares, the above microarchitectural leakage can be prevented. We take this same approach in our implementation of Debraize's A2B conversion, where we directly integrate these countermeasures into a hand-crafted assembly routine.

Measurements We first verify our setup by collecting 10.000 measurements with the PRNG disabled. In this case, the sets correspond to $A = x_{fix} = 0$ and $A = x_{rand}$ for fixed and random, respectively. Furthermore, all randomness used in generating the table, i.e. r and ρ, is likewise forced to 0. In all our experiments, we use 32-bit A2B conversion with $n = 8$ chunks and $k = 4$ bits. Figure 1 shows the mean power consumption of the two sets of measurements collected through this experiment. The power traces show a pattern that repeats 8 times, corresponding to the n iterations of the inner loop. As these are measurements with the PRNG OFF, differences in the mean power consumption of the fixed and random sets are in some places visible with the naked eye. Figure 2a collects the t-statistic as a function of time, which indeed shows values that cross ± 4.5 by a very large margin.

In a second experiment, we collect 100.000 measurements with the PRNG enabled. Results of this second experiment are collected in Figure 2b. Even though the PRNG is enabled, Figure 2b still shows t-values that cross ± 4.5. We additionally add a yellow trigger signal to mark computations on A_l, which we identify as the leaking variable in all but the first loop iteration. Finally, Figure 2c shows the evolution of the maximum t-statistic value over the 100.000 measurements. This value is monotonically increasing, giving another strong indication of leakage.

4.2 Theoretical analysis

In each iteration of the loop of the A2B conversion (Algorithm 6) a table lookup is performed based on A_l, which represents k masked bits of x. We will show that the masking applied on a part of x to obtain A_l is not uniformly distributed and thus leaks information about x.

To this end, we calculate the value of all intermediate variables used in the algorithm. First, note that the values of R and x can be written in terms of their bitchunks as $R = \sum_{j=0}^{n-1} 2^{jk} R_j$ and $x = \sum_{j=0}^{n-1} 2^{jk} x_j$, and therefore $A = \sum_{j=0}^{n-1} 2^{jk}(x_j - R_j)$. In our analysis, we write intermediate values with tilde $\tilde{\cdot}$, and variables without are fixed input values. Writing out the intermediate values on each line of Algorithm 6 in the first iteration, we get:

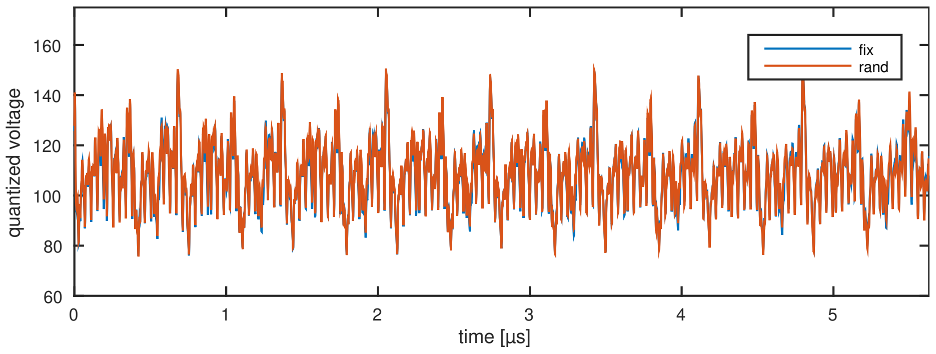
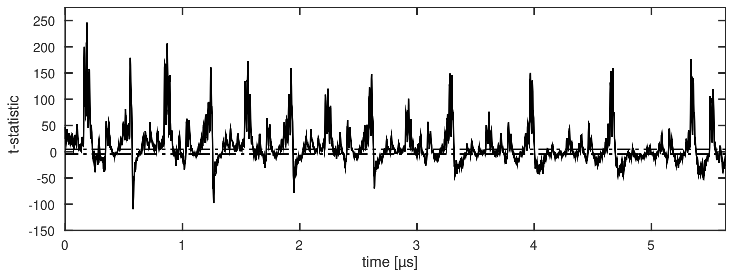
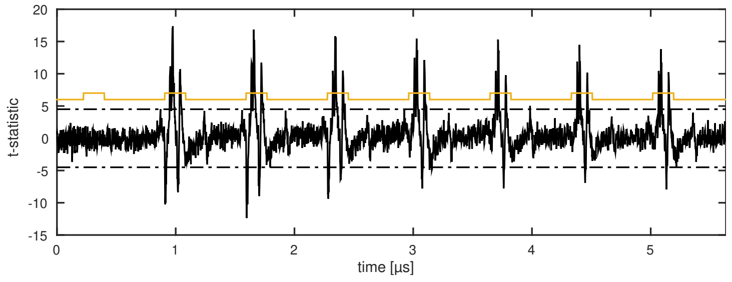


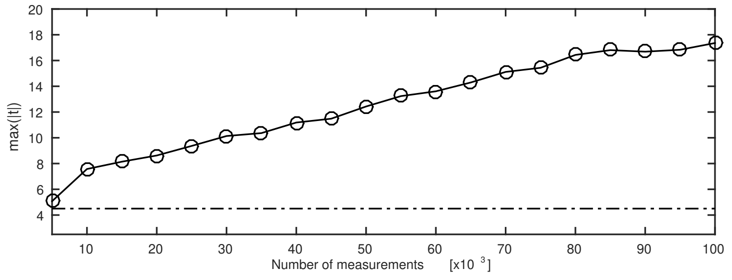
Figure 1: Mean power consumption of Debraize's A2B conversion with a pool of 10.000 measurements and PRNG OFF



(a) t-statistic as
a function of time for a pool of 10.000 measurements and PRNG OFF



(b) t-statistic as
a function of time for a pool of 100.000 measurements and PRNG ON



(c) Absolute
maximum t-statistic of (b) as a function of number of measurements.

Figure 2: TVLA of Debraize's A2B conversion

1. $\widetilde{A} = \sum_{j=0}^{n-1} 2^{jk}(x_j - R_j - r)$
2. $\widetilde{\beta} = \rho$
4. $\widetilde{A} = \left(\sum_{j=1}^{n-1} 2^{jk}(x_j - R_j - r)\right) + (x_0 - r)$
5. $\widetilde{\beta} = \rho \oplus (-\lfloor (x_0 - r)/2^k \rfloor)$
   $\widetilde{B_0} = x_0 \oplus r$
6. $\widetilde{B_0} = x_0 \oplus r \oplus R_0$
7. $\widetilde{A} = \left(\sum_{j=1}^{n-1} 2^{(j-1)k}(x_j - R_j - r)\right) + \lfloor (x_0 - r)/2^k \rfloor$
8. $\widetilde{R} = \sum_{j=1}^{n-1} 2^{(j-i-1)k} R_j$

Note that $\tilde{A}$ on line 7 contains a carry $c = -\lfloor (x_0 - r)/2^k \rfloor$ that is dependent on the lower bits of x. The information to correctly add this carry is present in β, which is $\rho \oplus c$. However, the carry c is not added to $\tilde{A}$ which will cause leakage in the next iteration.

Generalizing this calculation over all iterations, the description for all intermediate values of Algorithm 6 can then be found in Figure 3.

```
1.  $\widetilde{A} = \sum_{j=0}^{n-1} 2^{jk}(x_j - R_j - r)$                                                        mod $2^{n \cdot k}$
2.  $\widetilde{\beta} = \rho$                                                                                     mod 2
3.  for $i = 0$ to $n-1$ do:
4.      $\widetilde{A} = \left(\sum_{j=i+1}^{n-1} 2^{(j-i)k}(x_j - R_j - r)\right) + (x_i - r) + \lfloor \sum_{j=0}^{i-1} (x_j - r)/2^{(i-j)k} \rfloor$      mod $2^{(n-i) \cdot k}$
5.      $\widetilde{\beta} = \rho \oplus (-\lfloor \sum_{j=0}^{i} (x_j - r)/2^{(i-j)k} \rfloor)$                          mod 2
        $\widetilde{B_i} = x_i \oplus r$                                                                           mod $2^k$
6.      $\widetilde{B_i} = x_i \oplus r \oplus R_i$                                                                mod $2^k$
7.      $\widetilde{A} = \left(\sum_{j=i+1}^{n-1} 2^{(j-i-1)k}(x_j - R_j - r)\right) + \lfloor \sum_{j=0}^{i} (x_j - r)/2^{(i+1-j)k} \rfloor$        mod $2^{(n-i-1) \cdot k}$
8.      $\widetilde{R} = \sum_{j=i+1}^{n-1} 2^{(j-i-1)k} R_j$                                                      mod $2^{(n-i-1) \cdot k}$
```

Figure 3: Intermediate values of all variables used in Algorithm 6.

In particular, we will look at the value of $A_l = A \bmod 2^k$ at line 4 of Figure 3. In the first loop iteration $i = 0$ we get:

$$A_l = x_0 - r \bmod 2^k,$$

which is a correct masking of x_0 with uniformly random variable r and thus does not lead to leakage as can also be seen from the experimental validation in Figure 2b.

In the second loop iteration $i = 1$, the value of A_l is determined as follows:

$$\begin{aligned} A_l &= (x_1 - r) + \lfloor (x_0 - r)/2^k \rfloor \bmod 2^k \\ &= x_1 - (r + c) \bmod 2^k \end{aligned}$$

where $c = (-\lfloor (x_0 - r)/2^k \rfloor)$, which is the carry that is saved in $\beta \oplus \rho$. The carry c added to the mask r is equal to 1 if $x_0 < r$ and otherwise equal to 0. A higher value of r increases the probability of the carry being 1, which implies that the distribution of the actual mask $(r + c)$ will not be uniform. Consequently, the mask $(r + c) = 0$ will be more probable than other mask values as on one hand $r = 0$ implies that no carry occurs $(c = 0)$ and thus $(r + c) = 0$, but $r = 2^k - 1$ implies that a carry occurs with high probability which also results in $(r + c) = 0$. The full distribution of the mask values can be found in Figure 4. These values were calculated by determining the mask value of A_l on line 5 of Algorithm 6 for all possible combinations of x, r, R.

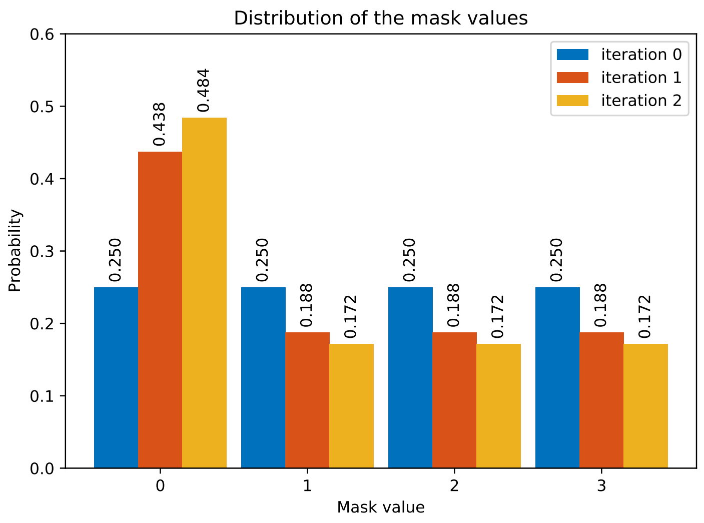


Figure 4: Distribution of mask value $(r+c)$ per iteration for $k=2$.

In the general case, A_l in loop i can be written as:

$$\begin{aligned} A_l &= (x_i - r) + \lfloor \sum_{j=0}^{i-1} (x_j - r)/2^{(i-j)k} \rfloor \\ &= x_i - (r+c) \bmod 2^k \end{aligned}$$

where $c = (-\lfloor \sum_{j=0}^{i-1} (x_j - r)/2^{(i-j)k} \rfloor)$.

With higher i, the distribution will become more skewed as a higher r value results in a higher probability of carries, which in turn also increase the chance of higher-up carries. As can be seen in Figure 4, the probability of $(r+c)=0$ will roughly double, while all other probabilities decrease uniformly. The effect of this non-uniform masking can be seen in Figure 2b, where clear leakage is present starting from the second iteration.

The security reasoning of Algorithm 6 as given in [Deb12] assumes that all intermediate values are masked with a uniformly random mask, without explicitly checking this for all intermediate variables. In this section we explicitly showed why this assumption is not valid in this case, which explains the leakage found in Figure 2b.

4.3 Attack possibilities

In this section we will give a short reasoning of how this leakage might be exploited in an attack scenario. Our goal is not to present a full attack, but rather to give an intuitive idea of why the leakage does open a path to a practical attacks. From Figure 4 we can see that for the later iterations, the mask has a higher probability of being 0, than being other values. This means that the k least significant bits of the variable calculated in line 4 of the algorithm is the sensitive value itself with higher than average probability.

In particular, for lattice-based encryption schemes, the decryption step can usually be seen as an operation $\texttt{Decode}(v - s \cdot u)$, where (u,v) is the ciphertext and s is the secret key. In a masked setting, `Decode` often involves A2B conversion of the coefficients of $v - s \cdot u$, as is the case in [OSPG18] and [VBDK+20]. Then, similarly to attacks described in [DTVV19, RRCB20, GJN20], an attacker can easily craft input ciphertexts such that the A2B routine directly computes on the secret key s. From our analysis we know that the mask applied to the sensitive variable is 0 with a higher probability. Therefore, the power consumption of this routine will be correlated with s, and s can be coefficient-wise extracted using DPA or CPA-style attacks.

The masked lattice based encryption scheme of [OSPG18] uses Debraize's insecure A2B method as a subroutine in their decoding step and is therefore vulnerable to such an attack. However, their choice for $k=8$ makes the probability of the mask 0 tend to 0.78% for later iterations, compared to the probability of 0.39% for the other mask values. The attack is more difficult in this setting, and might require a large amount of traces before the correlation with s is visible in the power consumption. This could explain why the leakage did not show up in TVLA experiments, where only a limited amount of power traces were collected. However, given enough traces one can expect leakage of the sensitive value which would make it possible to mount an attack similar to [DTVV19, RRCB20, GJN20]. A simple fix to resolve this insecurity in the implementation of [OSPG18] is to replace the Debraize A2B conversion for an alternative conversion algorithm as discussed in Section 6.

5 Two secure A2B conversions

Debraize's A2B conversion is not secure against DPA because the mask $(r+c)$ is not uniformly distributed as a result of carry propagation. In this section, we propose two solutions and evaluate them experimentally. The first solution is a direct but somewhat expensive fix of Debraize's method. The idea is to avoid the dependence between the consecutive mask values that creates the non-uniform distribution, by using a different and independent mask r_i in each iteration. The second solution can be understood as a hybrid between Debraize's and Coron-Tchulkine's methods, and offers an interesting time/memory trade-off. The main idea here is to explicitly compensate for the carry by calculating it, and adding it to the masked value before starting the next iteration.

5.1 Debraize (fixed)

A straightforward way to remove the non-uniformity from $(r+c)$, is to use a different mask r_i in every iteration of the loop. In this case, the mask r_i and the carry c, which equals $-\lfloor\sum_{j=0}^{i-1}(x_j - r_j)/2^{(i-j)k}\rfloor$, are independent and thus the actual mask r_i+c is uniformly distributed.

This change requires that instead of creating a single table T, n tables $T_{i,0\leq i<n}$ are created for each of the n masks $r_{i,0\leq i<n}$, as illustrated in Algorithm 7. The conversion itself remains very similar to Algorithm 6, with only two necessary changes. The table lookup $T[\beta][A_l]$ is replaced by $T_i[\beta][A_l]$, and the initial pre-masking of A must use $(r_{n-1}\|...\|r_0)$ instead of $(r\|...\|r)$. We illustrate Debraize (fixed) in Algorithm 8, with the required changes as compared to Algorithm 6 highlighted in blue.

Algorithm 7: Table $T_{i,0\leq i<n}$ generation

```
1 $\rho \leftarrow \mathcal{U}(\{0,1\})$
2 for $i=0$ to $n-1$ do
3     $r_i \leftarrow \mathcal{U}(\{0,1\}^k)$
4     for $A=0$ to $2^k-1$ do
5         $T_i[\rho][A]=(A+r_i)\oplus(\rho\|r_i)$
6         $T_i[(\rho\oplus 1)][A]=(A+r_i+1)\oplus(\rho\|r_i)$
7     end
8 end
9 return $T_{i,0\leq i<n}, r_{i,0\leq i<n}, \rho$
```

This solution keeps the advantages of Debraize's conversion over that of Coron-Tchulkine.

Algorithm 8: Conversion of a $n\cdot k$-bit variable (**Debraize (fixed)**)

```
input  : $(A,R)$ such that $x = A + R \bmod 2^{n\cdot k}$,
         $T$, $r$, $\rho$
output : $B$ such that $x = B \oplus R$
/* Let $A = (A_h \| A_l)$, $R = (R_h \| R_l)$ with $A_l$, $R_l$ the $k$ least significant
   bits.  $A_h$, $A_l$, $R_h$, $R_l$ are updated at the same time as $A$, $R$.        */
1  $A \leftarrow A - (r_{n-1} \| \ldots \| r_0) \bmod 2^{n\cdot k}$
2  $\beta \leftarrow \rho$
3  for $i = 0$ to $n-1$ do
4      $A \leftarrow A + R_l \bmod 2^{(n-i)\cdot k}$
5      $\beta \| B_i \leftarrow T_i[\beta][A_l]$
6      $B_i \leftarrow B_i \oplus R_l$
7      $A \leftarrow A_h$
8      $R \leftarrow R_h$
9  end
10 return $B = (B_{n-1} \| \ldots \| B_0) \oplus (r_i \| \ldots \| r_0)$
```

The carry is kept in Boolean masked form, and the algorithm avoids the full-width arithmetic operation $A \leftarrow A_h \| 0 + T[A_l] \bmod 2^{(n-i)\cdot k}$. While the new method requires n different tables to be present for each conversion, each one is accessed in only a single loop iteration, rather than n times as in the original method. Therefore, in a setting where a table is refreshed after a fixed amount of accesses, the pre-computation cost can be completely amortized over multiple A2B conversions[1], and the performance remains equal to Debraize's original conversion. In the other scenario where the table(s) are reused indefinitely, the one-time difference in cost between a single or n table generations would in many cases be eclipsed by the cost of the actual conversions, and thus the efficiency of both methods would not differ significantly. The main disadvantage of this solution is the increased storage cost due to the n tables T_i, which is increased to $n \cdot 2^{k+1} \cdot (k+1)$ bits.

5.1.1 Security proof

Theorem 1. *Debraize (fixed) is secure under first order DPA attacks.*

Proof. To prove security under first order DPA attacks, we explicitly show that all intermediate values are independent of the secret x. First note that the table T is generated without knowledge of x and is thus independent of the secret. Looking at the conversion itself, Figure 5 gives all intermediate values during the execution of the conversion algorithm. From this we can see that: $\widetilde{R}$ does not depend on x and is therefore independent; $\widetilde{\beta}$ is always masked with uniform binary mask ρ and is thus independent of x; and $\widetilde{B_i}$ is masked with the uniformly random mask r_i and later with R_i making it independent of the secret.

There remains to show that $\widetilde{A}$ is always independent of x. In line 1 $\widetilde{A}$ is still masked with the original mask R, and as only an independent random value is subtracted, $\widetilde{A}$ is independent of x. We can rewrite line 4 as:

$$\widetilde{A} = x_i + \sum_{j=i+1}^{n-1} 2^{(j-i)k}(x_j - R_j) + \lfloor \sum_{j=0}^{i-1} (x_j - r_j)/2^{(i-j)k} \rfloor + \sum_{j=i}^{n-1} 2^{(j-i)k} r_j \pmod{2^{(n-i)\cdot k}}$$

where $\sum_{j=i}^{n-1} 2^{(j-i)k} r_j$ is a uniformly random variable modulo $2^{(n-i)\cdot k}$. Moreover, this variable is independent of the rest of the expression as the r_j are drawn independently. Therefore, $\sum_{j=i}^{n-1} 2^{(j-i)k} r_j$ acts as a uniformly random mask making $\widetilde{A}$ independent of x.

[1] A recent masked implementation of the lattice-based Saber algorithm requires 1280 A2B conversions [VBDK+20].

```
1. $\widetilde{A} = \sum_{j=0}^{n-1} 2^{jk}(x_j - R_j - r_j)$    mod $2^{n\cdot k}$
2. $\widetilde{\beta} = \rho$    mod 2
3. for $i = 0$ to $n-1$ do:
4.     $\widetilde{A} = \left(\sum_{j=i+1}^{n-1} 2^{(j-i)k}(x_j - R_j - r_j)\right) + (x_i - r_j) + \lfloor \sum_{j=0}^{i-1}(x_j - r_j)/2^{(i-j)k} \rfloor$    mod $2^{(n-i)\cdot k}$
5.     $\widetilde{\beta} = \rho \oplus (-\lfloor \sum_{j=0}^{i}(x_j - r_j)/2^{(i-j)k} \rfloor)$    mod 2
       $\widetilde{B}_i = x_i \oplus r_i$    mod $2^k$
6.     $\widetilde{B}_i = x_i \oplus r_i \oplus R_i$    mod $2^k$
7.     $\widetilde{A} = \left(\sum_{j=i+1}^{n-1} 2^{(j-i-1)k}(x_j - R_j - r_j)\right) + \lfloor \sum_{j=0}^{i}(x_j - r_j)/2^{(i+1-j)k} \rfloor$    mod $2^{(n-i-1)\cdot k}$
8.     $\widetilde{R} = \sum_{j=i+1}^{n-1} 2^{(j-i-1)k} R_j$    mod $2^{(n-i-1)\cdot k}$
```

Figure 5: Intermediate values of all variables used in Algorithm 8. Variables with tilde $\widetilde{\cdot}$ are intermediate values, variables without are fixed input values.

Similarly, we can rewrite line 7 as:

$$\widetilde{A} = \sum_{j=i+1}^{n-1} 2^{(j-i-1)k}(x_j - R_j) + \lfloor \sum_{j=0}^{i}(x_j - r_j)/2^{(i+1-j)k} \rfloor - \sum_{j=i+1}^{n-1} 2^{(j-i-1)k} r_j \pmod{2^{(n-i-1)\cdot k}}$$

where $\sum_{j=i+1}^{n-1} 2^{(j-i-1)k} r_j$ acts as a uniformly random mask modulo $2^{(n-i-1)\cdot k}$ which is independent of the rest of the expression. As such $\widetilde{A}$ is also in this expression independent of x. $\square$

The security claim of Debraize [Deb12] was similarly based on the independence between the intermediate values and x. However, they did not explicitly write out all intermediate values and as such missed the bias in the mask on line 4 of Figure 3. By explicitly giving all intermediate values we want to show the validity of our security claim. Our security claim will be practically validated using side-channel measurements in the next section.

5.2 Secure dual-lookup A2B

The second solution builds on the fact that the Coron-Tchulkine conversion does not feature the vulnerability present in the Debraize conversion. Indeed, Algorithm 4 avoids this issue, since the incoming carry c is explicitly added to A_l before A_l is unmasked with R_l. This way, the uniform mask r appears instead of the biased mask $(r + c)$.

In adapting the method of Coron-Tchulkine, Debraize briefly introduced another table C' for illustrative purposes. This idea can be extended to construct a full A2B conversion method that features a different trade-off.

In Algorithm 3, table T combines tables G and C from Algorithms 1 and 2. Table T is exactly the same size as G and C combined, such that it is simply more efficient to treat this as a single table. However, it is also possible to decompose table T into table C' and a new table G', such that the combined size of these tables is smaller than T. In order to do so, we pre-compute $G'[A_l]$, such that it includes the Boolean masked carry $\beta = c_{out}(A_l + r) \oplus \rho$ as one of its explicit outputs. Using β as an auxiliary value, it is possible to greatly reduce the size of table C' compared to C. This is due to the fact that table C' only needs to store two elements for the two possible values of β, rather than 2^k elements as in C. The pre-computation of G' and C' is illustrated in Algorithms 9 and 10, respectively. We show this new dual-lookup conversion, using tables G' and C', in Algorithm 11, with the changes as compared to Algorithm 4 highlighted in blue.

Algorithm 9:
Pre-computation of G'

```
1 $r \leftarrow \mathcal{U}(\{0,1\}^k)$
2 $\rho \leftarrow \mathcal{U}(\{0,1\})$
3 for $A = 0$ to $2^k - 1$ do
4     $G'[A] = (A + r) \oplus (\rho \| r)$
5 end
6 return $G', r, \rho$
```

Algorithm 10:
Pre-computation of C' [Deb12]

```
1 $\gamma \leftarrow \mathcal{U}(\{0,1\}^{(n-1)\cdot k})$

2 $C'[\rho] = \gamma$
3 $C'[\rho \oplus 1] = \gamma + 1 \bmod 2^{(n-1)\cdot k}$

4 return $C', \gamma$
```

Algorithm 11: A2B conversion of a $(n \cdot k)$-bit variable (**our dual-lookup A2B**)

```
   input  : $(A, R)$ such that $x = A + R \bmod 2^{n\cdot k}$,
            $G'$, $C'$, $r$, $\rho$, $\gamma$
   output : $B$ such that $x = B \oplus R$
   /* Let $A = (A_h \| A_l)$, $R = (R_h \| R_l)$ with $A_l$, $R_l$ the $k$ least significant
      bits. $A_h$, $A_l$, $R_h$, $R_l$ are updated at the same time as $A$, $R$.   */
1  $\Gamma \leftarrow \sum_{i=1}^{n-1} 2^{i\cdot k} \cdot \gamma \bmod 2^{n\cdot k}$
2  $A \leftarrow A - (r \| \ldots \| r) \bmod 2^{n\cdot k}$
3  $A \leftarrow A - \Gamma \bmod 2^{n\cdot k}$
4  $\beta \leftarrow \rho$
5  for $i = 0$ to $n-1$ do
6      $A \leftarrow A + R_l \bmod 2^{(n-i)\cdot k}$
7      $\beta \| B_i \leftarrow G'[A_l]$
8      $B_i \leftarrow B_i \oplus R_l$
9      $A \leftarrow A_h + C'[\beta] \bmod 2^{(n-i-1)\cdot k}$
10     $R \leftarrow R_h$
11 end
12 return $(B_{n-1} \| \ldots \| B_0) \oplus (r \| \ldots \| r)$
```

This second solution has similar computational efficiency to the fixed Coron-Tchulkine method, with two table lookups and similar computational cost, but it has the advantage of a reduced memory footprint. Compared to the improved Coron-Tchulkine method in Algorithm 4 with a single lookup, there is now a clear trade-off: our method requires more lookups and computations but reduces the required amount of memory. Finally, note that the new method requires the single extra random bit ρ in the pre-computation of G'. A detailed comparison of these conversions will be given in Section 6.

5.2.1 Security proof

Theorem 2. *Our dual-lookup A2B is secure under first order DPA attacks.*

Proof. To prove security under first order DPA attacks, we explicitly show that all intermediate values are independent of the secret x. First note that the precomputed tables G' and C' are generated without knowledge of x and are thus independent of it. To show security of Algorithm 11, Figure 6 gives a list of all intermediate values. Going over all variables we see that: Γ and $\widetilde{R}$ are generated without x and are thus independent of x; $\widetilde{\beta}$ is always masked with a uniformly random value ρ and is therefore independent of x; and $\widetilde{B_i}$ is always masked with the uniformly random variable r, and later also masked with the uniformly random variable R_i

Now we only have to show the independence between $\widetilde{A}$ and x. In line 2 and 3, $\widetilde{A}$ is still masked with the original mask R and as only independent variables are subtracted, $\widetilde{A}$

```
1.  $\Gamma = \sum_{i=1}^{n-1} 2^{ik}\gamma$                                          $\bmod 2^{n\cdot k}$
2.  $\widetilde{A} = \sum_{j=0}^{n-1} 2^{jk}(x_j - R_j - r)$                          $\bmod 2^{n\cdot k}$
3.  $\widetilde{A} = \left(\sum_{j=0}^{n-1} 2^{jk}(x_j - R_j - r - \gamma)\right) + \gamma$      $\bmod 2^{n\cdot k}$
4.  $\widetilde{\beta} = \rho$                                                         $\bmod 2$
5.  for $i=0$ to $n-1$ do:
6.      $\widetilde{A} = \left(\sum_{j=i+1}^{n-1} 2^{(j-i)k}(x_j - R_j - r - \gamma)\right) + (x_i - r)$      $\bmod 2^{(n-i)\cdot k}$
7.      $\widetilde{\beta} = \rho \oplus (-\lfloor \sum_{j=0}^{i}(x_j - r)/2^{(i-j)k} \rfloor)$      $\bmod 2$
        $\widetilde{B}_i = x_i \oplus r$                                                $\bmod 2^k$
8.      $\widetilde{B}_i = x_i \oplus r \oplus R_i$                                     $\bmod 2^k$
9.      $\widetilde{A} = \left(\sum_{j=i+1}^{n-1} 2^{(j-i-1)k}(x_j - R_j - r - \gamma)\right) + \gamma$      $\bmod 2^{(n-i-1)\cdot k}$
10.     $\widetilde{R} = \sum_{j=i+1}^{n-1} 2^{(j-i-1)k} R_j$                           $\bmod 2^{(n-i-1)\cdot k}$
```

Figure 6: Intermediate values of all variables used in Algorithm 11. Variables with tilde $\widetilde{\cdot}$ are intermediate values, variables without are fixed input values.

remains independent of x. Similarly, in line 9, $\widetilde{A}$, which is $2^{(n-i-1)\cdot k}$ bits long, is masked with the $2^{(n-i-1)\cdot k}$ most significant bits of R and thus $\widetilde{A}$ is also here independent of x.

Line 6 can be rewritten as:

$$\begin{aligned}\widetilde{A} &= x_i + \sum_{j=i+1}^{n-1} 2^{(j-i)k}(x_j - \gamma) + r + \sum_{j=i+1}^{n-1} 2^{(j-i)k}(R_j - r) \pmod{2^{(n-i)\cdot k}}\\ &= x_i + \sum_{j=i+1}^{n-1} 2^{(j-i)k}(x_j - \gamma) + r + \sum_{j=1}^{n-i-1} 2^{jk}(R_{j+i} - r) \pmod{2^{(n-i)\cdot k}}\end{aligned}$$

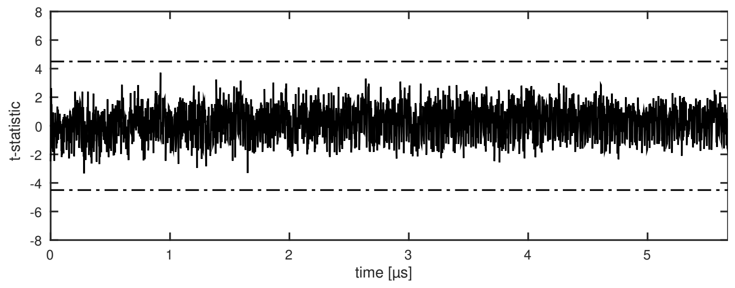
where $r + \sum_{j=1}^{n-i-1} 2^{jk}(R_{j+i} - r) \bmod 2^{(n-i)\cdot k}$ is a uniformly random mask that is independent of the rest of the expression, which makes $\widetilde{A}$ independent of x.

□

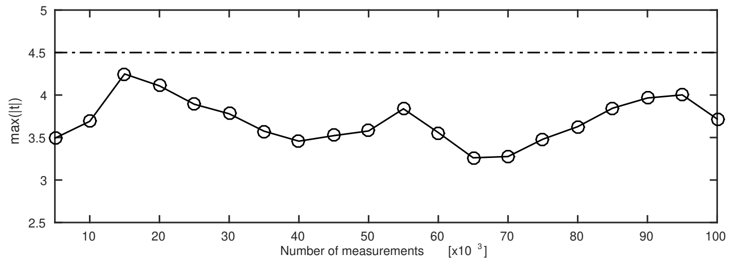
5.3 Experimental validation

We experimentally validate our two solutions, using the same setup and methods as described in Subsection 4.1. Notably, we again create carefully crafted assembly code, including countermeasures that prevent microarchitectural leakage on the Cortex-M4. The results of collecting 100.000 measurements with the PRNG enabled are shown in Figures 7 and 8 for Debraize (fixed) and our dual-lookup A2B, respectively. For both solutions, the t-statistic remains within the ±4.5 boundary after 100.000 measurements, and there is no obvious increase of the t-values over the measurements. Thus, these measures confirm our security claim and proof, showing that the implementation indeed provides first-order side-channel security.

We make our source code publicly available at `https://github.com/KULeuven-COSIC/A2B-table` to allow easily reproducing our results. Furthermore, since we show side-channel security on the Cortex-M4, future implementations that require secure A2B conversions on this platform can readily use the code of one of our methods. In particular, this helps the effort towards masked post-quantum implementations, since NIST has put increasing



(a) t-statistic as a function of time



(b) Absolute maximum t-statistic of (a) as a function of number of measurements

Figure 7: TVLA of Debraize (fixed) for a pool of 100.000 measurements

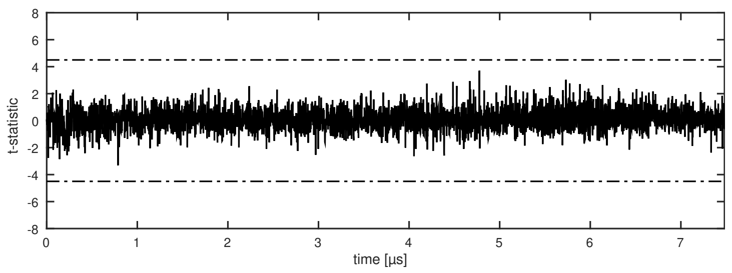
emphasis on side-channel resistance for the finalists of the PQC competition [AASA$^+$20], where the Cortex-M4 is the prime embedded benchmarking platform [KRSS].

6 Discussion

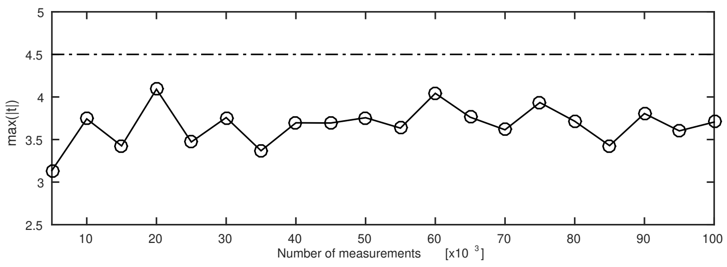
In Sections 5.1.1 and 5.2.1, we showed that our two new methods are probing-secure. Another popular security notion for masked software implementations is that of Strong Non-Interference (SNI) [BBD$^+$16], required for composability of masked routines. Table-based conversions precompute a randomized table and reuse this table for multiple online conversion steps. Because of this implicit randomness reuse, the conversion itself does not strictly adhere to the properties of SNI. Rather, SNI must be examined and proven in higher-level applications, such as the parallel A2B conversion of several polynomial coefficients with a single precomputed table. We note that SNI proofs can be adapted to such routines that share common randomness [CGPZ16]. However, the result is very application-specific, and we therefore leave it as future work.

In Table 2, we compare the different table-based A2B methods in terms of memory, randomness cost and the number of table lookups in the conversion. We do include the original method of Coron-Tchulkine and the method of Debraize for reference purposes, but stress that they should not be used due to correctness and security problems respectively. The table size is given in bits, without taking restrictions of byte-addressable memory into account. For the Debraize (fixed) conversion, the initial randomness cost is $n{\cdot}k{+}1$ bits. However, this procedure produces n tables and when the tables are used for a fixed number of lookups, the tables last n times longer than in other methods. In this case, the equivalent randomness cost is $k + 1/n$. We refrain from giving a theoretical number of arithmetic instructions since this maps poorly to actual instructions, e.g. the Cortex-M4 instruction set allows ALU operations on shifted operands in a single instruction.

The table size can be further reduced using an ‘LSB trick’ from [NP04]. In this work,



(a) t-statistic as a function of time



(b) Absolute
maximum t-statistic of (a) as a function of number of measurements

Figure 8: TVLA of our dual-lookup A2B for a pool of 100.000 measurements

Table 2: Characteristics of the different table-based A2B conversions.

	Mem. ops	Rand. [bits]	Table size [bits]	Secure	Correct
CT [CT03]	$2n$	$2{\cdot}k$	$2^k{\cdot}k{+}2^k{\cdot}k$	✓	✗
CT (fixed) [Deb12]	$2n$	$n{\cdot}k$	$2^k{\cdot}k{+}2^k{\cdot}(n{-}1){\cdot}k$	✓	✓
CT (imp.) [Deb12]	n	$n{\cdot}k$	$2^k{\cdot}n{\cdot}k$	✓	✓
Debraize [Deb12]	n	$k{+}1$	$2^{k+1}{\cdot}(k{+}1)$	✗	✓
Debraize (fixed)	n	$n{\cdot}k{+}1 \mid k{+}1/n$ [§]	$n{\cdot}2^{k+1}{\cdot}(k{+}1)$	✓	✓
Dual-lookup	$2n$	$n{\cdot}k{+}1$	$2^k{\cdot}(k{+}1){+}2{\cdot}(n{-}1){\cdot}k$	✓	✓

[§] average equivalent cost over n conversions

the fact that a Boolean masking and arithmetic masking share the same least-significant bit is exploited, and this bit is removed from all relevant tables. The resulting conversion is then slower, due to the added operations of doing the bit manipulation.

In lattice-based cryptographic implementations the table size can also often be reduced. There, a typical application of these conversion schemes is to determine the most significant bit of an arithmetically masked number. In this case, the lower significant bits do not have to be calculated, and the tables can be reduced in size by only storing the information needed to propagate the carry as explained by Van Beirendonck et al. [VBDK$^+$20]. This reduces the table size of each table T_i in Debraize (fixed) from $2^{k+1} \cdot (k+1)$ to 2^{k+1} and in the dual-lookup for Table G from $2^k \cdot (k+1)$ to 2^k.

The table-based methods described in this section specifically work with power-of-two moduli. Some lattice-based schemes work with prime moduli and similarly need to determine the most significant bit of arithmetically masked numbers. Oder et al. [OSPG18] provide an algorithm to securely compute this most significant bit, by using the power-of-two table-based A2B conversion of Debraize [Deb12] as a subroutine. As we showed, this choice would introduce a security vulnerability. However, it is trivial to replace the Debraize A2B conversion by any of our secure table-based conversion algorithms, by replacing the A2B used in Algorithm 1 of [OSPG18].

Table 3: Performance numbers for 16-bit A2B conversion.

		Precomputation [cycles]		Conversion [cycles]		Table size [bytes]	
	w	$k=4$	$k=8$	$k=4$	$k=8$	$k=4$	$k=8$
Debraize [Deb12]	8	1242	6207	1095	149	32	1024
CT (imp.) [Deb12]	8	**988**	**3643**	**940**	**162**	32	512
Debraize (fixed)	8	3296 | **824**[§]	✗	1178	✗	128	2048[†]
Dual-lookup	8	1062	3658	1282	246	**20**	**516**[‡]
Debraize	32	377	3986	121	88	32	1024
CT (imp.)	32	**197**	**2138**	134	96	32	512
Debraize (fixed)	32	1213 | 303[§]	8432 | 4216[§]	**124**	**89**	128	2048[†]
Dual-lookup	32	331	2266	148	111	**20**	**516**[‡]

[§] average equivalent cost over n conversions
[†] 1024 bytes with the LSB trick
[‡] 260 bytes with the LSB trick

6.1 Performance evaluation

Setup For our performance evaluation, we benchmarked the different A2B conversions on two platforms with different processor word-widths $w=8$ and $w=32$. For $w=8$, we target an 8-bit Atmel ATmega328P on an Arduino Uno, and compile with -Os. For $w=32$, we again target an ARM Cortex-M4, this time on the STM32F407-DISCOVERY development board, and we compile with -O3. The STMF407 is highly similar to the STM303 on our measurement PCB, with the added benefit of an on-chip TRNG from which we sample all masking randomness. In both cases, we include the cost of randomness sampling into the pre-computation cost. For the STM407, the TRNG supplies 32-bit random numbers every 40 cycles of the TRNG clock, which is set to 48 MHz, twice the clock frequency of the 24 MHz system clock. For the ATmega328P, a PRNG is supplied by the random() function that is part of avr-libc. In a small experiment, we find that sampling a 32-bit random number takes 812 CPU clock cycles. Together with all randomness sampling, the computation of Γ and $(r\|...\|r)$ is moved to the pre-computation, such that they only need to be computed once.

Comparing 16-bit conversions In lattice-based cryptography, moduli are often slightly smaller than 16-bit. Therefore, in Table 3, we benchmark 16-bit conversion, using both $(n=4,\ k=4)$ and $(n=2,\ k=8)$. Note that we include the insecure Debraize method for reference purposes only, and it should not be used in secure designs. Again, for the Debraize (fixed) conversion, we report both the initial pre-computation cost and the amortized pre-computation cost resulting from an n times longer lifetime of the tables compared to other methods. This time, we report practical table size for byte-addressable memory. For example, when $k=8$, the word-size of the Debraize (fixed) table is $k+1=9$ bits, for which we use two bytes of memory to prevent packing and unpacking operations. Here it could be beneficial to implement the LSB trick, such that the word-size is reduced to a single byte.

From Table 3, we see that Debraize (fixed) provides the fastest conversion times on the Cortex-M4, slightly outperforming CT (imp.). However, the table size is much larger for Debraize (fixed), due to the added cost of generating and storing n tables T_i. The added memory cost of Debraize (fixed) is further highlighted by the red check mark for $w=8$ and $k=8$, where the 2 KB combined size of the tables T_i completely saturates the 2 KB Atmega328P SRAM.

Debraize's method was designed to appeal to lower word-width processors, by avoiding the full 32-bit arithmetic addition in the main loop in favour of bit-wise operations. This property is preserved in Debraize (fixed). Nevertheless, in our experiments we find that both Debraize and Debraize (fixed) are outperformed by CT (imp.) when $w=8$ and $k=4$. When looking at memory requirements, the dual-lookup method has the smallest memory cost, especially if it would be implemented with the LSB trick.

In Table 4 we repeat the benchmarks for our new methods, this time with microarchitec-

Table 4: Performance numbers for 16-bit A2B conversion with countermeasures for microarchitectural leakage.

		Conversion [cycles]	
	w	$k=4$	$k=8$
Debraize (fixed)	32	233	165
Dual-lookup	32	261	184

tural countermeasures such as clearing registers or flushing the ALU, which were described in Section 4.1. We stress that these are the performance numbers corresponding to the secure implementations that showed no t-test leakage in Subsection 5.3. However, we report them separately, since they are platform-specific numbers and are difficult to compare to other methods which do not have countermeasures against microarchitectural attacks.

Results from Table 3 can also be compared to the prime-modulus algorithm of [BBE+18, `SecArithBoolModp`]. This algorithm was further optimized in [SPOG19], and requires 436[2] 'basic operations' for conversion using the 14-bit prime modulus $q = 12289$. Even though it is difficult to map 436 operations to exact cycle counts since this is usually an understatement, the general trend that power-of-two conversion is much cheaper is immediately clear. `SecArithBoolModp` will likely use more than 436 CPU cycles, whereas our methods can use as few as 89 cycles for the online conversion. We note however that `SecArithBoolModp` is not table-based, and as a result does not require a pre-computation cost.

As mentioned before, our efficient 16-bit A2B conversion methods can also be used as a subroutine to implement prime modulus conversion using the approach of [OSPG18]. We leave an extensive treatment and benchmarking of this combined algorithm as future work.

Comparing 64-bit conversions Table 5 repeats the same experiment, this time for 64-bit conversion using ($n = 16$, $k = 4$) and ($n = 8$, $k = 8$). We benchmark 64-bit conversion only on the Cortex-M4 and omit the insecure Debraize method. For 64-bit conversion, the dual-lookup method is appealing in memory-constrained scenarios. The tables in this method scale mostly with k and only very limitedly with n. As such the memory-cost is comparable to the 16-bit conversion, which stands in contrast to CT (imp.) and Debraize (fixed), where this scales linearly. For 64-bit conversion, the dual-lookup method reduces the memory cost by a factor 4× to 16×, while incurring a conversion cycle overhead of less than ~35%.

We also compare to the non-table based A2B conversion of [WH17], who exclusively reported CPU cycles for 64-bit conversion in [WH17, Table 4]. This work presents two algorithms based on the Kogge-Stone adder, which were improved from [CGTV15]. The first, [WH17, Algorithm 3], is the general algorithm, whereas the second, [WH17, Algorithm 4], is a speciality algorithm only applicable when the conversion bit-width (64) is twice the processor word-width (32). From Table 5, it is noticeable that [WH17] is not table-based, and as such has no pre-computation cost or tables that need to be stored in memory. We find that both of their algorithms are outperformed by table-based methods in terms of online conversion cost when $k=8$. For Debraize (fixed), this remains true when $k=4$. We note that care should be taken when directly comparing these numbers, since they compare actual CPU cycles on the STM407 for our benchmarks with simulated ARM cycles for the benchmarks of [WH17].

Compared to methods based on the Kogge-Stone adder, table-based algorithms have two further benefits. Firstly, Kogge-Stone based algorithms have logarithmic complexity in the conversion width, whereas table-based algorithms have linear complexity. Therefore, when scaling down the conversion width from 64-bit to widths more applicable to PQC, e.g. 16-bit, table-based algorithms will scale better. Secondly, table-based algorithms only sample randomness in the offline precomputation phase. For platforms without a

[2]The reported operation count (695) for `SecArithBoolModp` in [SPOG19] is a typo, and in reality constitutes the operation count of `SecBoolArithModp`. We obtained the correct number 436 for `SecArithBoolModp` after communication with the authors.

Table 5: Performance numbers for 64-bit A2B conversion.

		Precomputation [cycles]		Conversion [cycles]		Table size [bytes]	
	w	$k=4$	$k=8$	$k=4$	$k=8$	$k=4$	$k=8$
CT (imp.) [Deb12]	32	338	2210	407	244	128	2048
Debraize (fixed)	32	4139 | 259[§]	33093 | 4137[§]	**333**	**211**	512	8192
Dual-lookup	32	480	2120	446	270	**32**	**528**
[WH17, Algorithm 3]	32	-	-	526[†]		-	-
[WH17, Algorithm 4]	32	-	-	384[†]		-	-

[§] average equivalent cost over n conversions
[†] 2048 bytes with the LSB trick
[‡] 272 bytes with the LSB trick
[†] Simulated results using ARM Developer Suite v1.2, taken from [WH17, Table 4]. Randomness sampling is not included in the simulated cycles.

TRNG, randomness sampling can quickly dominate the actual conversion cost. For this reason, [WH17], explicitly mentioned that they did not include this cost in their performance numbers. As a downside, table-based algorithms do not extend to higher-order security.

7 Conclusion

Masking cryptographic implementations that convert from arithmetic to Boolean masking is a complex process that makes it prone to errors. It is therefore of paramount importance to provide complete security proofs and experimentally validate new algorithms proposed in this setting, by implementing them and testing for side-channel leakage. In this paper, we showed a security vulnerability in the state-of-the-art table-based A2B conversion introduced by Debraize in CHES 2012, which has been used as a subroutine in practical applications. After analyzing the problem, we proposed two alternative A2B methods that prevent it: a single-table method that maintains efficiency at the cost of additional memory, and a dual-table method that reduces this memory at the cost of computational efficiency. Both methods were verified for correctness and security both theoretically and using experimental side-channel validation. The source code of these conversions will be made available online so that they can be used to support masked implementation efforts, with a particular focus on the field of post-quantum cryptography.

Acknowledgements

We want to thank Siemen Dhooghe for his valuable insights in provable secure masking. This work was supported in part by CyberSecurity Research Flanders with reference number VR20192203, the Research Council KU Leuven (C16/15/058), the Horizon 2020 ERC Advanced Grant (695305 Cathedral), and SRC grant 2909.001. Michiel Van Beirendonck is funded by an FWO PhD fellowship strategic basic research.

References

[AASA+20] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. Status report on the second round of the nist post-quantum cryptography standardization process. *NIST, Tech. Rep., July*, 2020.

[BBD$^+$16] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 116–129, New York, NY, USA, 2016. Association for Computing Machinery.

[BBE$^+$18] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. Masking the GLP lattice-based signature scheme at any order. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 354–384. Springer, Heidelberg, April / May 2018.

[BCZ18] Luk Bettale, Jean-Sébastien Coron, and Rina Zeitoun. Improved high-order conversion from Boolean to arithmetic masking. *IACR TCHES*, 2018(2):22–45, 2018. https://tches.iacr.org/index.php/TCHES/article/view/873.

[CGPZ16] Jean-Sébastien Coron, Aurélien Greuet, Emmanuel Prouff, and Rina Zeitoun. Faster evaluation of SBoxes via common shares. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 498–514. Springer, Heidelberg, August 2016.

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

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

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 398–412. Springer, Heidelberg, August 1999.

[Cor17] Jean-Sébastien Coron. High-order conversion from Boolean to arithmetic masking. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 93–114. Springer, Heidelberg, September 2017.

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

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

[DTVV19] Jan-Pieter D'Anvers, Marcel Tiepelt, Frederik Vercauteren, and Ingrid Verbauwhede. Timing attacks on error correcting codes in post-quantum schemes. Cryptology ePrint Archive, Report 2019/292, 2019. https://eprint.iacr.org/2019/292.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. NIST Non-Invasive Attack Testing Workshop - NIAT, 2011.

[GJN20] Qian Guo, Thomas Johansson, and Alexander Nilsson. A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 359–386. Springer, Heidelberg, August 2020.

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

[HT19] Michael Hutter and Michael Tunstall. Constant-time higher-order Boolean-to-arithmetic masking. *Journal of Cryptographic Engineering*, 9(2):173–184, June 2019.

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

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

[KRSS] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[MGTF19] Vincent Migliore, Benoît Gérard, Mehdi Tibouchi, and Pierre-Alain Fouque. Masking Dilithium - efficient implementation and side-channel evaluation. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 344–362. Springer, Heidelberg, June 2019.

[NP04] Olaf Neiße and Jürgen Pulkus. Switching blindings with a view towards IDEA. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 230–239. Springer, Heidelberg, August 2004.

[OSPG18] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical CCA2-secure masked Ring-LWE implementations. *IACR TCHES*, 2018(1):142–174, 2018. https://tches.iacr.org/index.php/TCHES/article/view/836.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In Isabelle Attali and Thomas Jensen, editors, *Smart Card Programming and Security*, pages 200–210, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[RRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs. *IACR TCHES*, 2020(3):307–335, 2020. https://tches.iacr.org/index.php/TCHES/article/view/8592.

[SPOG19] Tobias Schneider, Clara Paglialonga, Tobias Oder, and Tim Güneysu. Efficiently masking binomial sampling at arbitrary orders for lattice-based crypto. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 534–564. Springer, Heidelberg, April 2019.

[VBDK$^+$20] Michiel Van Beirendonck, Jan-Pieter D'Anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. A side-channel resistant implementation of saber. Cryptology ePrint Archive, Report 2020/733, 2020. https://eprint.iacr.org/2020/733.

[WH17] Yoo-Seung Won and Dong-Guk Han. Efficient conversion method from arithmetic to Boolean masking in constrained devices. In Sylvain Guilley, editor, *COSADE 2017*, volume 10348 of *LNCS*, pages 120–137. Springer, Heidelberg, April 2017.