Efficiently Masking Binomial Sampling at Arbitrary Orders for Lattice-Based Crypto

Tobias Schneider[1], Clara Paglialonga[2], Tobias Oder[3], and Tim Güneysu[3,4]

[1]ICTEAM/ELEN/Crypto Group, Université Catholique de Louvain, Belgium
tobias.schneider@uclouvain.be
[2]Technische Universität Darmstadt, Germany
clara.paglialonga@crisp-da.de
[3]Horst Görtz Institute for IT Security, Ruhr-Universität Bochum, Germany
{tobias.oder,tim.gueneysu}@rub.de
[4]DFKI, Germany

Abstract. With the rising popularity of lattice-based cryptography, the Learning with Errors (LWE) problem has emerged as a fundamental core of numerous encryption and key exchange schemes. Many LWE-based schemes have in common that they require sampling from a discrete Gaussian distribution which comes with a number of challenges for the practical instantiation of those schemes. One of these is the inclusion of countermeasures against a physical side-channel adversary. While several works discuss the protection of samplers against timing leaks, only few publications explore resistance against other side-channels, e.g., power. The most recent example of a protected binomial sampler (as used in key encapsulation mechanisms to sufficiently approximate Gaussian distributions) from CHES 2018 is restricted to a first-order adversary and cannot be easily extended to higher protection orders.
In this work, we present the first protected binomial sampler which provides provable security against a side-channel adversary at arbitrary orders. Our construction relies on a new conversion between Boolean and arithmetic (B2A) masking schemes for prime moduli which outperforms previous algorithms significantly for the relevant parameters, and is paired with a new masked bitsliced sampler allowing secure and efficient sampling even at larger protection orders. Since our proposed solution supports arbitrary moduli, it can be utilized in a large variety of lattice-based constructions, like NewHope, LIMA, Saber, Kyber, HILA5, or Ding Key Exchange.

1 Introduction

Ever since the first publication by Kocher [24], protection against side-channel analysis (SCA) has become an essential optimization goal for designers of security-critical applications. It has been shown numerous times that an adversary, which does not only have access to the inputs and outputs of a system but can also measure some physical characteristics (e.g., timing) of the target, is capable of extracting sensitive information from an unprotected implementation with

ease [18]. This affects especially embedded systems, as they (a) allow extensive physical access to the adversary enabling the exploitation of additional side-channels (e.g., power, EM) which requires more sophisticated protection schemes, and (b) often possess only limited resources which makes the integration of dedicated countermeasures a complex task.

One important type of countermeasure in this context is masking [9]. Its core idea is splitting the sensitive information into multiple shares and transforming the target implementation to securely compute on these shares. Based on certain assumptions, it is possible to prove that an adversary needs to combine at least t intermediate values (so called *probes*) of the circuit to extract any sensitive data. In practice, an increase in the number of required probes exponentially increases the attack complexity given a sufficient level of noise in the measurements [32]. The seminal work of Ishai et al. [22] introduces a security model, which identifies the attack situation described above. It introduces the *probing model*, that allows to prove the security of a masked gadget. Still, the imprudent composition of such gadgets can introduce security flaws, due to the extra information that an adversary might be able to extract when using the outputs of a gadget as input to another one. The notion of Strong Non-Interference (SNI), introduced in [5], provides stronger conditions that ensure a secure composition of masked gadgets.

While for common symmetric and asymmetric constructions the application of masking schemes has been extensively examined and achieves a high level of security and efficiency, this is still ongoing research for most post-quantum schemes. First approaches to achieve first-order side-channel security were made [10, 11, 25, 28], but the overhead for higher-order masking of post-quantum cryptography has not been studied, except in [6] for the specific GLP signature scheme [20]. However, GLP has not been submitted to the standardization process for post-quantum cryptography run by the National Institute of Standards and Technology (NIST) [26]. Among those 69 complete and proper submissions to this competition, lattice-based cryptography is clearly the largest group with a total of 29 submissions due to advantages, such as reasonable parameter sizes, simple implementation, as well as advanced constructions like identity-based encryption and homomorphic encryption. The competition explicitly takes into account the ease of integration of physical countermeasures as an evaluation metric [27], which is therefore a primary contribution of this work.

Most of the lattice-based encryption and key exchange schemes rely on the Learning with Errors (LWE) problem or variants like Ring-LWE for which many of them require noise polynomials with small Gaussian distributed coefficients. For small standard deviations, the binomial sampler as presented in [1] and implemented in [2] is the best choice as it has a constant run time by design, is easy to implement and does not require any precomputed tables. However, applying masking schemes to sampling algorithms for protection against side-channel analysis is a non-trivial task. It has been shown in [25] by Oder et al. that this can easily become even the bottleneck of an implementation, causing a performance overhead of more than 400%.

This reduction in performance stems mostly from the mixture of different masking schemes. Oder et al. initially generate uniformly-random bits by running SHAKE for a given seed. Using a pseudo random number generator (PRNG) is common as a first step in such samplers and, as shown in [25], needs to include side-channel countermeasures as well. From this uniform randomness, the approximated Gaussian randomness is derived, e.g., [1] proposes to subtract the Hamming weight of two uniform bit vectors. For both steps, Boolean masking is an obvious choice, since many PRNGs rely on Boolean operations and the computation of the Hamming weight requires bit-wise manipulations of the shares, which is complex for arithmetic masking (assuming not every bit is separately masked). However, the subsequent operations in the lattice scheme favor arithmetic masking. Therefore, it is necessary to apply a conversion at some point of the protected sampler.

Related work. In [6], Barthe et al. propose the first provably-secure sampler for arbitrary orders. However, the targeted GLP signature schemes requires a uniform sampler, which poses different challenges to the implementer than a binomial one. Thus, their results cannot be straight-forwardly transferred to protect a binomial sampler, as required for many NIST submissions.

One of these submissions is NewHope [1] and its potential for physical protection was evaluated in [25], where, as part of their construction, Oder et al. presented a masked binomial sampler. However, it was heavily-optimized for first-order protection and not easily extendable to higher orders. In addition, their results show that masking such a sampler can severely impact the performance of the whole scheme. Therefore, it is an open problem how to efficiently protect binomial samplers against side-channel adversaries.

While the authors of [25] convert Boolean-masked bits to arithmetic shares, they do not formally specify the conversion algorithm. Such Boolean-to-arithmetic (`B2A`) conversions have been extensively researched [7, 8, 12–14, 16, 21, 23, 30, 33] as many symmetric primitives mix Boolean and arithmetic operations. However, they focus solely on power-of-two moduli. In contrast, many lattice schemes employ other moduli (not necessarily power-of-two) to allow optimizations. For these cases, it is necessary to apply specific conversions (in the following denoted as $\texttt{B2A}_q$). The first of such conversion algorithms which work with arbitrary moduli was proposed in [6] based on the cubic conversion of [14] and proven to be secure at arbitrary orders. Since these algorithms are a non-negligible part of masking a binomial sampler, any improvements regarding the conversion will also improve the sampling performance.

Our contribution. Our first contribution is a revised version of the quadratic `B2A` algorithm of [14] which is combined with some of the ideas outlined in [6]. While this is technically straightforward, we still provide the essential proof of security, resulting in $\texttt{A2B}_q$ and $\texttt{B2A}_q$ algorithms with the best asymptotic run time complexity (i.e., $\mathcal{O}(n^2 \log k)$, where k denotes the bit size of the operands and n the number of shares) to date.

As a second contribution, we present a new $\texttt{B2A}_q$ conversion with a run time complexity of $\mathcal{O}(n^2 \cdot k)$. Our new algorithm is proven to be $t - \mathsf{SNI}$ enabling composition with $n = t+1$ shares. While its asymptotic complexity is worse than our quadratic adaption of [6], it still significantly reduces the conversion time for relevant values of k. The new algorithm even outperforms the established standard B2A algorithms for power-of-two moduli for certain parameters (e.g., $n \geq 11$ for $k = 32$). Therefore, it is not only relevant for our considered use case, but might also improve the performance of masked symmetric algorithms.

Thirdly, relying on $\texttt{B2A}_q$ conversions we propose two masked binomial samplers for lattice-based encryption and key exchange schemes. The first sampler is a higher-order extension of the approach from [25], while the second variant uses bitslicing to further increase the throughput. Again, we prove both solutions to be $t - \mathsf{SNI}$ to enable easy composition in larger constructions, e.g., for CCA2-transformations. This results in the first provable SCA-protected binomial samplers at arbitrary orders. Since our proposed solutions support arbitrary moduli, they can be adopted in a large variety of NIST submissions, e.g., NewHope [1], LIMA [31], Saber [15], Kyber [3], HILA5 [29], or Ding Key Exchange [17].

Finally, we present an ARM Cortex-M4F microcontroller implementation of our constructions to evaluate their performance on a popular embedded processor platform. We find that our new $\texttt{B2A}_q$ conversion improves the performance of the samplers over the adaption of [6] up to a factor of 46, while our new bitsliced sampler improves the performance over a generalized version of the approach of [25] up to a factor of 15. The combination of both approaches results in the currently most-efficient masked binomial sampler. At first-order, it even outperforms the implementation from [25], which is highly-optimized for first-order security, while our contribution provides generic protection for arbitrary orders.

Organization of the paper. In Section 2, we start by briefly reviewing the state-of-the-art regarding B2A conversions and masked binomial sampling. In Section 3, we provide the adaption of the quadratic $\texttt{B2A}_q$ from [14] using the prime addition from [6] and its corresponding proof of security. In Section 4, our new $\texttt{B2A}_q$ algorithm is presented and compared to the current state-of-the-art considering both standard and prime moduli. Finally in Section 5, we propose our new masked sampling algorithms and conduct a case study using the parameters of the NIST submission NewHope.

2 Background

In this section, we first introduce the notation used throughout this paper and the considered notions of physical security. Then we briefly introduce the B2A algorithms relevant to our work and review the masked binomial sampler of [25].

2.1 Notation

In the rest of the paper we denote with q a prime number, with B2A the standard conversion from Boolean to arithmetic masking and with $\texttt{B2A}_\texttt{q}$ the same transfor-

mation for prime moduli. We will indicate with k the bit size of the conversion, κ the bit size of the input vectors of the samplers, n the number of shares and t the security order. Moreover, the lower case will be used for Boolean encoding and the upper one for arithmetic encoding. Operations on the whole encoding, i.e., vector of shares, are denoted in bold and performed share-wise, e.g., $\mathbf{z} = \mathbf{x} \oplus \mathbf{y}$. We denote the k bits of a share as $\left(x_i^{(k)} \dots \ x_i^{(1)}\right) = x_i$, i.e., the least-significant bit (LSB) of x_i is written as $x_i^{(1)}$ (resp. $\mathbf{x}^{(1)}$ denotes the LSB of $\mathbf{x}$).

2.2 Notions of Physical Security

Counteracting side-channel analysis via masking informally means to randomize the information leaked during the computation of sensitive variables in order to make the representation of such variables independent from the actual processed data. The security of a masking scheme has been formalized for the first time by Ishai et al. in [22]. Their seminal work introduces the *t-probing model*, where an adversary is allowed to read up to t wires in a circuit (the so called *probes*). Every sensitive variable s is encoded into n shares s_i, such that the sum of them gives the original masked variable s and only the combination of all the shares gives some information about s. In order to keep this independence throughout the whole circuit, every operation is performed on the shares s_i and, especially in the case of non-linear operations, it makes use of additional randomness, which helps to guarantee that every t-tuple of intermediate variables is independent from at least one of the shares of s. Such transformed gates are commonly called gadgets and, in order to provide sound claims of their composability, they must satisfy the conditions of one of the following definitions.

Definition 1 ($t - \mathsf{NI}$). *A given gadget $\mathcal{G}$ is t-Non-Interfering ($t - \mathsf{NI}$), if every set of t probes can be simulated by using at most t shares of each input.*

Definition 2 ($t - \mathsf{SNI}$). *A given gadget $\mathcal{G}$ is t-Strong-Non-Interfering ($t - \mathsf{SNI}$), if every set of t_1 probes on the internal values and t_2 probes on the output values, with $t_1 + t_2 \leq t$, can be simulated by using at most t_1 shares of each input.*

Both definitions have been introduced in [4] and are refinements of the original definition of probing security, given in [22]. They require the existence of a simulator, which can simulate the adversary's view, without accessing the sensitive variables, using only part of their shares. In particular, the definition of $t - \mathsf{SNI}$ is important when designing a gadget, which is supposed to be part of a bigger algorithm. In this case indeed, using the output of a gadget as input to another one might add sensitive information to the knowledge of the adversary and $t - \mathsf{NI}$ would not be sufficient to ensure global security. On the other hand, the conditions of $t - \mathsf{SNI}$ are stronger, since they require an independence between the number of probes on the output shares and the number of the shares needed to perform the simulation, therefore they allow to compose gadgets securely. In the rest of the paper, we will prove our algorithms to be $t - \mathsf{SNI}$ making them suited to be composed with other gadgets as part of a larger circuit.

2.3 Conversion from Boolean to Arithmetic Masking

The first sound transformation from Boolean to arithmetic masking (`B2A`) has been proposed by Goubin in [19] and it is based on the fact that the function $\Phi(x, r) : \mathbb{F}_{2^k} \times \mathbb{F}_{2^k} \mapsto \mathbb{F}_{2^k}$ such that

$$\Phi(x, r) = (x \oplus r) - r \mod 2^k \tag{1}$$

is affine in r over $\mathbb{F}_2$. The algorithm is very efficient, since it has a run time complexity of $\mathcal{O}(1)$, i.e., independent of the size of the inputs k. However, the initial algorithm was only proven secure against a first-order adversary ($t = 1$).

The first `B2A` algorithm secure at higher orders was presented in [14]. Instead of relying on the aforementioned affine relationship $\Phi(x, r)$, the main idea is to first initialize the shares $(A_i)_{1 \leq i \leq n-1}$ with random samples in $\mathbb{F}_{2^k}$. Then they are used to generate a random encoding $\mathbf{A'}$ of the form $\sum_{i=1}^{n} A'_i = -\sum_{i=1}^{n-1} A_i \mod 2^k$. This encoding is given to a higher-order secure arithmetic-to-Boolean (`A2B`) conversion algorithm to compute the Boolean shares $\bigoplus_i y_i = \sum_i A'_i \mod 2^k$, which are then added to the input encoding $\mathbf{x}$ via a Boolean-masked addition algorithm. This results in

$$\bigoplus_i z_i = \bigoplus_i x_i + \bigoplus_i y_i = x - \sum_{i=1}^{n-1} A_i \mod 2^k.$$

Using the function $\texttt{FullXOR} : \mathbb{F}_{2^k}^n \mapsto \mathbb{F}_{2^k}$ [14], which securely decodes a given input encoding, the remaining share of $\mathbf{A}$ is set to $A_n = \texttt{FullXOR}(\mathbf{z})$, which completes the transformation given

$$\sum_{i=1}^{n} A_i = \sum_{i=1}^{n-1} A_i + (x - \sum_{i=1}^{n-1} A_i) = x \mod 2^k.$$

In theory, the aforementioned framework can be instantiated with any secure `A2B` and Boolean-masked addition algorithm. In practice, however, there is only one secure higher-order `A2B` algorithm to the best of our knowledge, which was also published in the same work [14]. Its underlying concept is rather simple. Each share of the arithmetic input encoding $\mathbf{A}$ is first transformed to a Boolean encoding with n shares. These n Boolean encodings are then added together using a Boolean-masked addition algorithm resulting in a Boolean encoding $\mathbf{x}$ with $\bigoplus_i x_i = \sum_i A_i \mod 2^k$. With a simple addition algorithm this basic version has a cubic complexity of $\mathcal{O}(n^3 \cdot k)$. In [14], the authors also propose an improved version which relies on recursion and has a quadratic complexity of $\mathcal{O}(n^2 \cdot k)$. Instead of summing the input shares A_i sequentially one by one, the main algorithm splits the n shares into two halves of $\lfloor \frac{n}{2} \rfloor$ and $\lceil \frac{n}{2} \rceil$, and recursively calls itself for the two halves. The resulting encodings are then added together as

$$\begin{aligned} x &= \left(A_1 + \cdots + A_{\lfloor n/2 \rfloor}\right) + \left(A_{\lfloor n/2 \rfloor + 1} + \cdots + A_n\right) \\ &= \left(y_1 \oplus \cdots \oplus y_{\lfloor n/2 \rfloor}\right) + \left(z_1 \oplus \cdots \oplus z_{\lceil n/2 \rceil}\right) \\ x &= x_1 \oplus \cdots \oplus x_n. \end{aligned} \tag{2}$$

The complexity of both these B2A and A2B algorithms can be improved by using a more efficient masked addition algorithm, e.g., using the logarithmic addition from [13] reduces the run time complexity of both directions to $\mathcal{O}(n^2 \cdot \log k)$.

An alternative higher-order B2A conversion algorithm relying on the affine relation given in Equation (1) was recently presented by Bettale et al. in [7] following previous work of [21] and [12]. Since it is not based on the Boolean-masked addition of shares, its run time is independent of the input bit size. This makes it especially efficient for small values of n for which it is possible to achieve a significant speedup over the previous solutions based on [14]. For increasing number of shares, however, the performance of the new algorithm quickly deteriorates given its asymptotic run time complexity of $\mathcal{O}(2^n)$.

Conversion with prime moduli. All the aforementioned approaches assume an arithmetic modulus of the form 2^k, i.e., a power of two, and cannot be directly applied for prime moduli. The foundation for some very basic solutions for $\texttt{A2B}_q$ and $\texttt{B2A}_q$ conversions were given in [25]. However, the algorithms were not strictly formalized, their application is very specific to the presented use case, and their security is limited to a first-order adversary. Therefore, we refrain from giving further details on both algorithms and will only briefly mention their $\texttt{B2A}_q$ approach when discussing the masked binomial sampler in the next subsection.

The first generic solution for arbitrary orders was recently given in [6]. The authors present a new Boolean-masked addition algorithm which works modulo a prime q with $2^k \geq 2q$. Based on this adder, both the A2B and B2A algorithms from [14] can be changed to feature moduli other than a power-of-two. Their $\texttt{B2A}_q$ solution is given in Algorithm 1 which as the approach from [14] relies on an A2B conversion. Due to page limitations, we do not discuss all algorithms necessary for the conversion inside the paper (e.g., FullXOR) and instead refer to their original publications[1]. In their work, the authors only provide the proofs for an adaption of the A2B algorithm from [14] with cubic complexity (cf. Algorithm 2). In Section 3, we complement their work by providing the security proofs for an adaption of the conversion algorithm with quadratic complexity (cf. Algorithm 3) which leads to conversions for both directions with currently the lowest complexity of $\mathcal{O}(n^2 \cdot \log k)$.[2]

2.4 Masked Sampler from [25]

The construction of [25] initially uses a Boolean-masked SHAKE core to generate masked pseudo-random bits with a uniform distribution. As defined in

[1] In the full version of this paper, all algorithms are given as supplementary material.

[2] The authors of [6] also hint that the k-independent algorithm of [7] can be adopted to other moduli. However, we did not find a working solution. Nevertheless, an adapted algorithm would share the exponential complexity of the original making it only viable for small number of shares and, therefore, not a generic solution for masking at any order. In addition, the bit sizes considered in our case study are relatively small which would further decrease the benefit of a prime-adjusted algorithm.

Algorithm 1 `SecBoolArithModp` [6]

Input: $\mathbf{x} = (x_i)_{1 \le i \le n} \in \mathbb{F}_2^k$ such that $\bigoplus_i x_i = x \in \mathbb{F}_2^k$
Output: $\mathbf{A} = (A_i)_{1 \le i \le n} \in \mathbb{F}_q$ such that $\sum_i A_i = x \mod q$

1: $(A_i)_{1 \le i \le n-1} \xleftarrow{\$} \mathbb{F}_q$
2: $(A_i)'_{1 \le i \le n-1} \leftarrow q - (A_i)_{1 \le i \le n-1}$
3: $A'_n \leftarrow 0$
4: $\mathbf{y} \leftarrow \texttt{SecArithBoolModp}(\mathbf{A}')$
5: $\mathbf{z} \leftarrow \texttt{SecAddModp}(\mathbf{x}, \mathbf{y})$
6: $A_n \leftarrow \texttt{FullXOR}(\mathbf{z})$

Algorithm 2 `SecArithBoolModp` (cubic) [6]

Input: $\mathbf{A} = (A_i)_{1 \le i \le n} \in \mathbb{F}_q$ such that $\sum_i A_i = x \mod q \in \mathbb{F}_q$
Output: $\mathbf{x} = (x_i)_{1 \le i \le n} \in \mathbb{F}_2^k$ with $2^k > 2q$ such that $\bigoplus_i x_i = x$

1: $(x_i)_{1 \le i \le n} \leftarrow 0$
2: **for** $j = 1$ to n **do**
3: $\quad (y_i)_{1 \le i \le n} \leftarrow (A_j, 0, \ldots, 0)$
4: $\quad \mathbf{y} \leftarrow \texttt{RefreshXOR}(\mathbf{y}, k)$
5: $\quad \mathbf{x} \leftarrow \texttt{SecAddModp}(\mathbf{x}, \mathbf{y}, k)$
6: **end for**

the specification of NewHope, the sampler takes two 8-bit vectors (x, y) of this pseudo-randomness as input and computes the difference of the Hamming weight between them. Since the subsequent operations of the lattice scheme are performed modulo $q = 12289$, the masked sampler needs to convert from Boolean masking to arithmetic masking modulo the prime q. Again, the $\texttt{B2A}_q$ conversion is not explicitly given and analyzed in [25], but rather directly integrated in the overlying algorithm. In particular, the authors exploit that Boolean-masked bits (x_1, x_2) with $x_1 \oplus x_2 = x$ provide the following arithmetic relation

$$x = x_1 + x_2 - 2 \cdot x_1 \cdot x_2 \tag{3}$$

which is used to transform these Boolean masked bits to an arithmetic encoding modulo q. The resulting masked sampler (cf. Algorithm 4 in [25]) is highly-customized for a first-order adversary and the authors do not provide any discussion on how to extend this approach to higher orders. Our new higher-order $\texttt{B2A}_q$ conversion algorithm, given in Section 4, relies on this arithmetic relation as well, but contains further optimizations to significantly improve the efficiency and security at higher orders.

3 Improved Higher-Order $\texttt{B2A}_q$ from [6]

In this section, we discuss the adaption of the $\texttt{B2A}_q$ conversion algorithm with quadratic complexity from [14] to the setting of prime moduli. The basic idea was already proposed in [6] without any concrete instantiation or proof. Nevertheless,

Algorithm 3 `SecArithBoolModp` (quadratic)

Input: $\mathbf{A} = (A_i)_{1 \leq i \leq n} \in \mathbb{F}_q$ such that $\sum_i A_i = x \mod q \in \mathbb{F}_q$
Output: $\mathbf{x} = (x_i)_{1 \leq i \leq n} \in \mathbb{F}_2^k$ with $2^k > 2q$ such that $\bigoplus_i x_i = x$

1: **if** n=1 **then**
2: $\quad x_1 \leftarrow A_1$
3: **end if**
4: $(y_i)_{1 \leq i \leq \lfloor n/2 \rfloor} \leftarrow$ `SecArithBoolModp` $\left((A_i)_{1 \leq i \leq \lfloor n/2 \rfloor}\right)$
5: $(y_i)_{1 \leq i \leq n} \leftarrow$ `RefreshXOR` $\left((y_1, \ldots, y_{\lfloor n/2 \rfloor}, 0, \ldots, 0), k\right)$
6: $(z_i)_{1 \leq i \leq \lceil n/2 \rceil} \leftarrow$ `SecArithBoolModp` $\left((A_i)_{\lfloor n/2 \rfloor + 1 \leq i \leq n}\right)$
7: $(z_i)_{1 \leq i \leq n} \leftarrow$ `RefreshXOR` $\left((z_1, \ldots, z_{\lceil n/2 \rceil}, 0, \ldots, 0), k\right)$
8: $\mathbf{x} \leftarrow$ `SecAddModp`$(\mathbf{y}, \mathbf{z})$

we provide the algorithmic description of the adjusted conversion and prove its $t - \mathsf{SNI}$ property to enable comparison with our new approach of Section 4.

3.1 Construction

In the original algorithm from [6], `SecBoolArithModp` calls the simple version of `SecArithBoolModp`, i.e., the shares are added sequentially as depicted in Algorithm 2, which leads to a run time complexity of $\mathcal{O}(n^3 \cdot \log k)$. To improve this, we adapt the recursive structure previously discussed in Equation (2). In particular, the input encoding given to `SecArithBoolModp` is split into two sets of $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$ elements. These are then separately given to a new call of `SecArithBoolModp` and the resulting Boolean encodings are summed to derive the correct result. This comes with the advantage that each sub routine processes a smaller number of shares which reduces the complexity of the refresh and addition, decreasing the overall complexity to $\mathcal{O}(n^2 \cdot \log k)$.

The complete $\texttt{A2B}_q$ algorithm with quadratic complexity is given in Algorithm 3 which would simply replace the call to `SecArithBoolModp` in Algorithm 1 to derive a corresponding quadratic $\texttt{B2A}_q$ conversion. To map the $\lfloor n/2 \rfloor$ (resp. $\lceil n/2 \rceil$) Boolean shares to the n shares required for the secure masked addition, we choose to rely on the $t - \mathsf{SNI}$-refresh `RefreshXOR` from [6] instead of the `Expand` function from [14]. To this end, the Boolean encodings are first padded with zeros and the resulting n shares are refreshed. An exemplary structure for the case $n = 4$ is depicted in Figure 2.

3.2 Security

The $t - \mathsf{SNI}$ security of `SecBoolArithModp` in Algorithm 1, when using the quadratic version of `SecArithBoolModp`, relies on the fact that `SecArithBoolModp` itself is $t - \mathsf{SNI}$. Before proceeding with proving that Algorithm 3 is $t - \mathsf{SNI}$, we give the following Lemma.

Lemma 1. *Given a circuit $\mathcal{C}$ as in Figure 1, where $\mathtt{f}, \mathtt{g}$ are $t - \mathsf{SNI}$ gadgets and $\mathtt{h}$ is $t - \mathsf{NI}$, the circuit $\mathcal{C}$ is $t - \mathsf{SNI}$.*

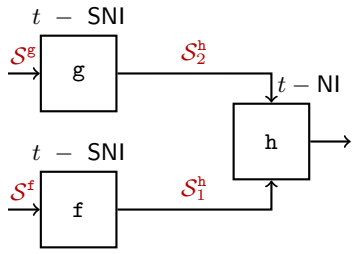

Fig. 1: Recurrent scheme in Algorithm 3

Proof. Let $\Omega = (\mathcal{I}, \mathcal{O})$ be the set of adversarial observations on $\mathcal{C}$, where $\mathcal{I}$ are the probes on the internal values and $\mathcal{O}$ the ones on the output values, with $|\mathcal{I}| + |\mathcal{O}| \leq t$. In particular, let $\mathcal{I}_{\mathtt{f}}$ be the set of probes on $\mathtt{f}$, $\mathcal{I}_{\mathtt{g}}$ the set of probes on $\mathtt{g}$ and $\mathcal{I}_{\mathtt{h}}$ be the set of probes on $\mathtt{h}$, with $|\mathcal{I}_{\mathtt{f}} \cup \mathcal{I}_{\mathtt{g}} \cup \mathcal{I}_{\mathtt{h}}| \leq |\mathcal{I}|$.

We prove the existence of a simulator which can simulate the adversary's view by using at most $|\mathcal{I}|$ input shares, analyzing the circuit from right to left.

Gadget $\mathtt{h}$. Since $\mathtt{h}$ is $t-\mathsf{NI}$ and $|I_{\mathtt{h}} \cup \mathcal{O}| \leq t$, then there exist two observation sets $\mathcal{S}_1^{\mathtt{h}}, \mathcal{S}_2^{\mathtt{h}}$ such that $|\mathcal{S}_1^{\mathtt{h}}| \leq |I_{\mathtt{h}} \cup \mathcal{O}|$, $|\mathcal{S}_2^{\mathtt{h}}| \leq |I_{\mathtt{h}} \cup \mathcal{O}|$ and the gadget can be simulated using at most $|\mathcal{S}_1^{\mathtt{h}}| + |\mathcal{S}_2^{\mathtt{h}}|$ shares of the inputs.

Gadget $\mathtt{f}$. Since $\mathtt{f}$ is t − SNI and $|I_{\mathtt{f}} \cup \mathcal{S}_1^{\mathtt{h}}| \leq t$, then there exists an observation set $\mathcal{S}^{\mathtt{f}}$ such that $|\mathcal{S}^{\mathtt{f}}| \leq |I_{\mathtt{f}}|$ and the gadget can be simulated using at most $|\mathcal{S}^{\mathtt{f}}|$ shares of the inputs.

Gadget $\mathtt{g}$. Since $\mathtt{g}$ is t − SNI and $|I_{\mathtt{g}} \cup \mathcal{S}_2^{\mathtt{h}}| \leq t$, then there exists an observation set $\mathcal{S}^{\mathtt{g}}$ such that $|\mathcal{S}^{\mathtt{g}}| \leq |I_{\mathtt{g}}|$ and the gadget can be simulated using at most $|\mathcal{S}^{\mathtt{g}}|$ shares of the inputs.

Combining the previous steps of the simulation, we can claim that $\mathcal{C}$ can be simulated by using at most $|\mathcal{S}^{\mathtt{f}} \cup \mathcal{S}^{\mathtt{g}}| \leq |I_{\mathtt{f}}| + |I_{\mathtt{g}}| \leq |I|$ shares of the inputs, completing the proof. □

Proposition 1. `SecArithBoolModp` *in Algorithm 3 is* t − SNI, *for* $t = n - 1$.

Proof. In order to prove `SecArithBoolModp` to be t − SNI we iteratively split the circuit in sub-circuits $\mathcal{C}_i$, for $i = 2, \ldots, n$ as in Figure 2, where in particular $\mathcal{C}_n :=$ `SecArithBoolModp` and we prove the thesis by induction on $i \in \mathbb{N}$.

We remark that $\mathcal{C}_2$ is of the form of the circuit in Figure 1. Indeed `RefreshXOR` is t − SNI and `SecAddModp` is $t-\mathsf{NI}$, as proven in [6]. Therefore thanks to Lemma 1, the circuit $\mathcal{C}_2$ is t − SNI.

Let us suppose now that $\mathcal{C}_{n-2}$ is t − SNI and we prove the thesis for $\mathcal{C}_n$.

Since the composition of t − SNI gadgets is still t − SNI, as pointed out in [4], we know that both $\mathcal{C}_{n-2}((A_i)_{1 \leq i \leq \lfloor n/2 \rfloor})$ and $\mathcal{C}_{n-2}((A_i)_{\lfloor n/2 \rfloor + 1 \leq i \leq n})$ composed with the `RefreshXOR` gadget are t − SNI. Therefore, the circuit $\mathcal{C}_n$ can be represented as the circuit in Figure 1, with $\mathtt{f} =$ `RefreshXOR`$(\mathcal{C}_{n-2}((A_i)_{1 \leq i \leq \lfloor n/2 \rfloor}))$, $\mathtt{g} =$ `RefreshXOR`$(\mathcal{C}_{n-2}((A_i)_{\lfloor n/2 \rfloor + 1 \leq i \leq n}))$ and $\mathtt{h} =$ `SecAddModp`. From Lemma 1 we conclude therefore that $\mathcal{C}_n$ is t − SNI, completing the proof. □

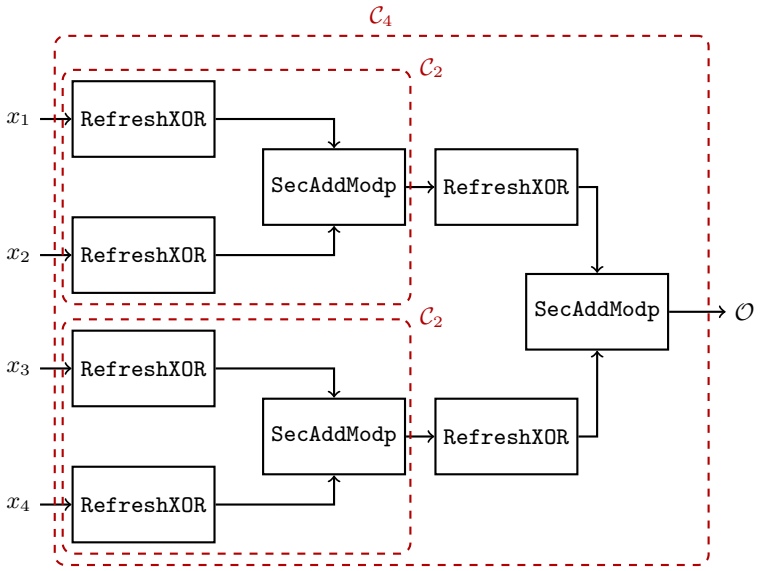

Fig. 2: Structure of `SecArithBoolModp` in Algorithm 3 for $n = 4$

4 A New $\texttt{B2A}_q$ Conversion Algorithm

In this section, we present our new $\texttt{B2A}_q$ conversion algorithm. Initially, we describe how the aforementioned arithmetic relationship of Boolean-masked bits can be used to construct a higher-order-secure conversion algorithm. We use this bit-wise transformation to derive a generic method to convert Boolean encodings to arithmetic encodings for arbitrary bit size k, number of shares n, and modulus q. The security of our solution is extensively proven and the $t - \mathsf{SNI}$ property is shown. In the last subsection, we compare the performance of our proposal not only against the new `SecArithBoolModp` (quadratic) algorithm for a prime modulus, but also against standard `B2A` conversions assuming a modulus of 2^k.

4.1 Conversion for $x \in \mathbb{F}_2$

Firstly, we discuss how to securely transform the Boolean shares $(x_1, x_2) \in \mathbb{F}_2$ with $x_1 \oplus x_2 = x$ into arithmetic shares (A_1, A_2) with $A_1 + A_2 = x \mod q$ for some arbitrary modulus q. For conciseness, we will not explicitly indicate every time that the arithmetic operations are done modulo q. To perform the conversion, the Boolean shares are transformed to the arithmetic encodings $B_1 + B_2 = x_1$ and $C_1 + C_2 = x_2$. This results in the following equations

$$\begin{aligned} x = x_1 \oplus x_2 &= (B_1 + B_2) \oplus (C_1 + C_2) \\ &= (B_1 + B_2) + (C_1 + C_2) - 2 \cdot (B_1 + B_2) \cdot (C_1 + C_2). \end{aligned}$$

Algorithm 4 $\texttt{SecB2A}_{q-Bit}$ (simple)

Input: $\mathbf{x} = (x_i)_{1 \leq i \leq n} \in \mathbb{F}_2$ such that $\bigoplus_i x_i = x$
Output: $\mathbf{A} = (A_i)_{1 \leq i \leq n} \in \mathbb{F}_q$ such that $\sum_i A_i = x \mod q$
1: $(A_i)_{1 \leq i \leq n} \leftarrow 0$
2: **for** $j = 1$ to n **do**
3: $\quad (B_i)_{1 \leq i \leq n} \leftarrow (x_j, 0, \ldots, 0)$
4: $\quad \mathbf{B} \leftarrow \texttt{RefreshADD}(\mathbf{B}, q)$
5: $\quad \mathbf{C} \leftarrow \texttt{SecMul}(\mathbf{A}, \mathbf{B})$
6: $\quad \mathbf{A} \leftarrow \mathbf{A} + \mathbf{B} - 2 \cdot \mathbf{C}$
7: **end for**

Before deriving the final arithmetic encoding $\mathbf{A}$, we have to sample a fresh random element $R \in \mathbb{F}_q$ to secure the shared multiplication $(B_1 + B_2) \cdot (C_1 + C_2)$. Now, it is easily possible to compute the shares as

$$A_1 = B_1 + C_1 + R - 2 \cdot B_1 \cdot C_1 - 2 \cdot B_1 \cdot C_2$$
$$A_2 = B_2 + C_2 - R - 2 \cdot B_2 \cdot C_1 - 2 \cdot B_2 \cdot C_2.$$

Extending this simple first-order approach to arbitrary n simply requires a proper refresh and multiplication algorithm, as described in Algorithm 4.

However, similarly as before, the algorithm has a run time complexity of $\mathcal{O}(n^3)$, i.e., cubic in the number of shares, and, therefore, quickly becomes inefficient for increasing security orders. Therefore, we further optimize the simple version of the conversion to increase the performance significantly.

1. Instead of using the t − SNI refresh `RefreshADD` for every iteration round, we found that it is sufficient to use a $t-$NI refresh for every round and only perform a final t − SNI refresh at the end. This reduces the complexity of the refresh for every iteration from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$.
2. Instead of multiplying two complete encodings as $\texttt{SecMul}(\mathbf{A}, \mathbf{B})$[3], we do not refresh x_j and instead compute the component-wise multiplication $\mathbf{A} \cdot x_j$. Obviously, this requires that the encoding $\mathbf{A}$ is independent of x_j, which we achieve with the aforementioned $t-$NI refresh. In this way, we save another operation with $\mathcal{O}(n^2)$ and reduce it to $\mathcal{O}(n)$. Furthermore, we can save $n-1$ operations of the addition $\mathbf{A} + x_j$.
3. Similar to the previous conversions, we vary the number of considered shares in each iteration, e.g., for $j = 2$ the operations are done on two shares. Note that we cannot use the same recursive approach as the previous examples, because it would not allow us to employ the second optimization.

The optimized conversion is given in Algorithm 5, which now has a run time complexity of $\mathcal{O}(n^2)$. Its structure is depicted in Figure 3.

Correctness. We prove the correctness of $\texttt{B2A}_{q-Bit}$, since the refreshing afterwards does not change the decoded output. The proof is based on the following

[3] `SecMul` is implemented similar to `SecAnd` of [14].

Algorithm 5 $\texttt{SecB2A}_{q-Bit}$ (optimized)

Input: $\mathbf{x} = (x_i)_{1 \le i \le n} \in \mathbb{F}_2$ such that $\bigoplus_i x_i = x \in \mathbb{F}_2$
Output: $\mathbf{A} = (A_i)_{1 \le i \le n} \in \mathbb{F}_q$ such that $\sum_i A_i = x \mod q$
$\mathbf{A} \leftarrow \texttt{B2A}_{q-Bit}(\mathbf{x})$
$\mathbf{A} \leftarrow \texttt{RefreshADD}(\mathbf{A}, q)$

Algorithm 6 $\texttt{B2A}_{q-Bit}$

Input: $\mathbf{x} = (x_i)_{1 \le i \le n} \in \mathbb{F}_2$ such that $\bigoplus_i x_i = x \in \mathbb{F}_2$
Output: $\mathbf{A} = (A_i)_{1 \le i \le n} \in \mathbb{F}_q$ such that $\sum_i A_i = x \mod q$
1: $\mathbf{A} \leftarrow x_1$
2: **for** $j = 2$ to n **do**
3: $\quad \mathbf{A} \leftarrow \texttt{B2A}^{(j)}_{q-Bit}(\mathbf{A}, x_j)$
4: **end for**

property, already mentioned in Equation (3). Given $x_1, x_2 \in \mathbb{F}_2$, the XOR between the two bits can be written as $x_1 \oplus x_2 = x_1 + x_2 - 2 \cdot x_1 \cdot x_2$. Generalizing to the case of n values, it is easy to see that

$$\bigoplus_{i=1}^{n} x_i = ((((x_1 \oplus x_2) \oplus x_3) \dots) \oplus x_n) = \bigoplus_{i=1}^{n-1} x_i + x_n - 2 \cdot \bigoplus_{i=1}^{n-1} x_i \cdot x_n \quad (4)$$

Now, adding the output shares of Algorithm 6 we get

$$\begin{aligned} \sum_{i=1}^{n} A_i &= \sum_{i=1}^{n} B_i - 2 \cdot \sum_{i=1}^{n} B_i \cdot x_n + x_n = \sum_{i=1}^{n-1} A_i - 2 \cdot \sum_{i=1}^{n-1} A_i \cdot x_n + x_n \\ &= \bigoplus_{i=1}^{n-1} x_i - 2 \cdot \bigoplus_{i=1}^{n-1} x_i \cdot x_n + x_n \end{aligned}$$

which, for Equation 4, is exactly $\bigoplus_{i=1}^{n} x_i$.

Security. In the following propositions we show that our conversion scheme $\texttt{SecB2A}_{q-Bit}$ in Algorithm 5 satisfies the $t - \mathsf{SNI}$ property, when $t = n-1$. We underline that $t - \mathsf{SNI}$ ensures that our conversion algorithm can be securely composed in larger circuits.

Proposition 2. $\texttt{B2A}^{(2)}_{q-Bit}$ *in Algorithm 7 is* $1 - \mathsf{NI}$.

Proof. Let us suppose that the adversary has exactly 1 probe w in Algorithm 7. This belongs to the following possible groups:

(1) x_1, x_2
(2) B_2

Algorithm 7 $\texttt{B2A}^{(n)}_{q-Bit}$

Input: $\mathbf{A} = (A_i)_{1 \le i \le n-1} \in \mathbb{F}_q$ such that $\sum_i A_i = x$; $x_n \in \mathbb{F}_2$
Output: $\mathbf{C} = (C_i)_{1 \le i \le n} \in \mathbb{F}_q$ such that $\sum_i C_i = (x \oplus x_n) \mod q$

1: $B_n \xleftarrow{\$} \mathbb{F}_q$
2: $B_1 \leftarrow A_1 - B_n \mod q$
3: **for** $j = 2$ to $n-1$ **do**
4: $\quad R \xleftarrow{\$} \mathbb{F}_q$
5: $\quad B_j \leftarrow A_j - R \mod q$
6: $\quad B_n \leftarrow B_n + R \mod q$
7: **end for**
8: **for** $j = 1$ to n **do**
9: $\quad C_j \leftarrow B_j - 2 \cdot (B_j \cdot x_n) \mod q$
10: **end for**
11: $C_1 \leftarrow C_1 + x_n \mod q$

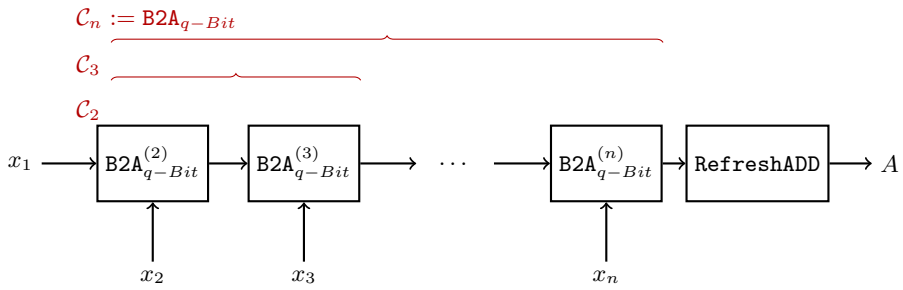

Fig. 3: Structure of $\texttt{SecB2A}_{q-Bit}$ in Algorithm 5.

(3) $B_1 := A_1 - B_2 = x_1 - B_2$
(4) $B_1 \cdot x_2 = (x_1 - B_2) \cdot x_2$, $b := B_1 - 2(B_1 \cdot x_2)$
(5) $B_2 \cdot x_2$, $B_2 - 2(B_2 \cdot x_2)$
(6) $b + x_2 = (B_1 - 2(B_1 \cdot x_2)) + x_2$

We construct a set of indexes I of cardinality 1, by adding the index 1 (resp. 2) if $w = x_1$ (resp. $w = x_2$). The simulation of the probe w, by using at most 1 share of the inputs, is straightforward.

- If the probe w is in group (1), by construction $1 \in I$ or respectively $2 \in I$. Thus the values can be simulated as x_1 (resp. x_2) as in the real algorithm.
- If the probe w is in one of the groups from (2) to (6), thanks to the presence in the computation of w of the random values B_1 or B_2, it is simulated by assigning it to a random and independent value in $\mathbb{F}_q$. □

We remark that the algorithm is not $t-\mathsf{SNI}$. Indeed, if an adversary probes the output share $(B_1 - 2(B_1 \cdot x_2)) + x_2 = ((x_1 - B_2) - 2((x_1 - B_2) \cdot x_2)) + x_2$ and the

internal value B_2, then the adversary gets the knowledge of two inputs shares, contradicting the definition of t − SNI.

In the following proposition we prove that the algorithm $\texttt{B2A}_{q-Bit}$ is $t-\mathsf{NI}$.

Proposition 3. $\texttt{B2A}_{q-Bit}$ *in Algorithm 6 is* $t-\mathsf{NI}$.

Proof. In the following, we denote with $\mathcal{C}_i$ the execution of Algorithm 6 until the i^{th} iteration of the **for**, for $i = 2, \ldots, n$, as indicated in Figure 3. In particular $\mathcal{C}_n := \texttt{B2A}_{q-Bit}$.

We prove the thesis by induction on the value i. From Proposition 2 the condition is satisfied for the case of $i = 2$.

Let now assume that $\mathcal{C}_i$ is $(i-1)-\mathsf{NI}$ for all $i \leq n-1$ and we show that under this condition $\mathcal{C}_n$ is $t-\mathsf{NI}$ as well. First of all let us denote with $A_j^{(i)}$ the output shares of $\mathcal{C}_i$ for all $i \leq n-1$ and with $j = 1, \ldots, n$. We can classify the internal and output values of $\mathcal{C}_n$ in the following groups:

(1) B_j for $j = 2, \ldots, n$, r
(2) A_1^{n-1},
(3) $b_1 := A_1^{n-1} - B_2 - \cdots - B_j$, with $j = 2, \ldots, n$
(4) $A_j^{(n-1)} - r_j =: b_j$, with $j = 2, \ldots, n$
(5) $A_1^{(n)} = (A_1^{n-1} - B_2 - \cdots - B_n) + x_n = b_1 + x_n$
(6) $A_j^{(n)} = (A_j^{(n-1)} - r_j) - 2((A_j^{(n-1)} - r_j) \cdot x_n) = b_j - 2(b_j \cdot x_n)$
(7) x_n

Let us suppose w.l.o.g. that an adversary has Ω probes on $\mathcal{C}_n$ with $|\Omega| = t = t_1 + \cdots + t_n$, where $t_1 + \cdots + t_i$ are the probes on $\mathcal{C}_i$. We show that the adversary's observation on $\mathcal{C}_n$ can be simulated by using at most t shares of the input.

We first construct a set of indexes I accordingly to the following instructions: for each probe in group (5), add n to I, and for each probe in group (6), add n to I. The simulation follows the steps below.

- **Step 1.** The probes in group (1) are simulated by assigning them to a random and independent value in $\mathbb{F}_q$.
- **Step 2.** Since by hypothesis $\texttt{B2A}_{q-Bit}^{(n-1)}$ is $t-\mathsf{NI}$, the probes in group (2) can be simulated by using at most $t_1 + \cdots + t_{n-1}$ shares.
- **Step 3.** If a probe is in group (3) and at least one of the $B_2, \ldots, B_j$ is not in Ω, then the values can be assigned to a random and independent value. Otherwise, if $B_2, \ldots, B_j \in \Omega$, then since $\texttt{B2A}_{q-Bit}^{(n-1)}$ is $t-\mathsf{NI}$ the probes can be simulated by using at most $t_1 + \cdots + t_{n-1}$ shares of the input and the assigned values of $B_2, \ldots, B_j$ in Step 1. Otherwise, if a sum $A_1^{n-1} - B_2 - \cdots - B_k \in \Omega$, with $k < j$ and $B_{k+1}, \ldots, B_j \in \Omega$, then the values can be computed as in the real execution of the algorithm, by using the values $B_{k+1}, \ldots, B_j$ assigned in Step 1 and the simulated sum $A_1^{n-1} - B_2 - \cdots - B_k$, in one of the phases of this Step.
- **Step 4.** If a probe is in group (4) and $r_j \notin \Omega$, then the values can be assigned to a random and independent value. Otherwise, if $r_j \in \Omega$, then since $\texttt{B2A}_{q-Bit}^{(n-1)}$ is $t-\mathsf{NI}$ the probes can be simulated by using at most $t_1 + \cdots + t_{n-1}$ shares and the value r_j assigned in Step 1.

– **Step 5.** If a probe is in group (5) and $b_1 \in \Omega$, then by construction $n \in I$ and we can compute the value as in the algorithm, by using the b_1 simulated at Step 3 and x_n. Otherwise, if $b_1 \notin \Omega$, we can simulate b_1 as in Step 3, by using at most $t_1 + \cdots + t_{n+1}$ input shares and x_n, since $n \in I$.
– **Step 6.** If a probe is in group (6) and $b_j \in \Omega$, then by construction $n \in I$ and we can compute the value as in the algorithm, by using the b_j simulated at Step 4 and x_n. Otherwise, if $b_j \notin \Omega$, we can simulate b_j as in Step 4, by using at most $t_1 + \cdots + t_{n+1}$ input shares and x_n, since $n \in I$.
– **Step 7.** If a probe is in group (7), by construction $n \in I$ and we can trivially simulate x_n.

In all the steps listed above, we showed that the simulation uses at most t input shares, as required from Definition 1, completing the proof. □

Before proceeding with the next proposition, we remind that the refreshing scheme added at the end of $\texttt{SecB2A}_{q-Bit}$ is an algorithm presented in [6] and proven to be $t-$ SNI. The $t-$ SNI security of $\texttt{SecB2A}_{q-Bit}$ relies exactly on the introduction of this gadget after the computation of $\texttt{SecB2A}_{q-Bit}$. We see below a more detailed security proof.

Proposition 4. $\texttt{SecB2A}_{q-Bit}$ *in Algorithm 5 is* $t-$ SNI.

Proof. The $t-$ SNI security of $\texttt{SecB2A}_{\texttt{q-Bit}}$ easily follows from the fact that $\texttt{B2A}_{q-Bit}$ is $t-$ NI and RefreshADD is $t-$ SNI.

Let $\Omega = (I, \mathcal{O})$ be the set of probes on $\texttt{SecB2A}_{q-Bit}$, where I_1 are the probes on the internal wires of $\texttt{B2A}_{q-Bit}$ and I_2 are the probes on the internal wires of RefreshADD, with $|I| = |I_1| + |I_2| \leq t_1$ and $|I| + |\mathcal{O}| \leq t$.

Since RefreshADD is $t-$ SNI and $|I_2 \cup \mathcal{O}| \leq t$, then there exists an observation set $\mathcal{S}^2$ such that $|\mathcal{S}^2| \leq |I_2|$ and the gadget can be simulated from its input shares corresponding to the indexes in $\mathcal{S}^2$.

Since $\texttt{SecB2A}_{q-Bit}$ is $t-$ NI and $|I_1 \cup \mathcal{S}^2| \leq |I_1 \cup I_2| \leq t$, then it exists an observation set $\mathcal{S}^1$ such that $|\mathcal{S}^1| \leq |I_1 \cup \mathcal{S}^2|$ and the gadget can be simulated from its input shares corresponding to the indexes in $\mathcal{S}^1$.

Now, composing the simulators that we have for the two gadgets RefreshADD and $\texttt{SecB2A}_{q-Bit}$, all the probes of the circuit can be simulated from $|\mathcal{S}^1| \leq |I_1| + |I_2| \leq t_1$ shares of the input x and therefore, according to Definition 2, the circuit in Figure 3 is $t-$ SNI.

□

4.2 Conversion for $x \in \mathbb{F}_{2^k}$

While the current solution is very efficient and simple, it only computes the correct results for Boolean encodings of bit values. Otherwise, the arithmetic property does not hold anymore. In order to extend our approach to include arbitrary bit sizes, we apply the previous conversion to each input bit separately and combine with component-wise addition. This trivially results in a complexity of $\mathcal{O}(n^2 \cdot k)$, as we have k calls to $\texttt{SecB2A}_{q-Bit}$. The complete conversion is given in Algorithm 9 and its basic structure for $k = 3$ is depicted in Figure 4.

Algorithm 8 `RefreshADD` (based on `RefreshXOR` [6])

Input: $\mathbf{A} = (A_i)_{1 \le i \le n} \in \mathbb{F}_q$ such that $\sum_i A_i = x \mod q$, modulus q
Output: $\mathbf{B} = (B_i)_{1 \le i \le n} \in \mathbb{F}_q$ such that $\sum_i B_i = x \mod q$

1: $\mathbf{B} \leftarrow \mathbf{A}$
2: **for** $i = 1$ to $n - 1$ **do**
3: **for** $j = 1 + i$ to n **do**
4: $R \xleftarrow{\$} \mathbb{F}_q$
5: $B_i \leftarrow B_i + r$
6: $B_j \leftarrow B_j - r$
7: **end for**
8: **end for**

Algorithm 9 $\mathtt{SecB2A}_q$

Input: $\mathbf{x} = (x_i)_{1 \le i \le n} \in \mathbb{F}_{2^k}$ such that $\bigoplus_i x_i = x \in \mathbb{F}_{2^k}$
Output: $\mathbf{A} = (A_i)_{1 \le i \le n} \in \mathbb{F}_q$ such that $\sum_i A_i = x \mod q$

1: $\mathbf{A} \leftarrow \mathtt{SecB2A}_{\mathtt{q-Bit}}\left((\mathbf{x} >> (k-1)) \wedge 1\right)$
2: **for** $j = 2$ to k **do**
3: $\mathbf{B} \leftarrow \mathtt{SecB2A}_{\mathtt{q-Bit}}\left((\mathbf{x} >> (k-j)) \wedge 1\right)$
4: $\mathbf{A} \leftarrow 2 \cdot \mathbf{A} + \mathbf{B} \mod q$
5: **end for**

Correctness. Let us assume that $2^k \le q$. It is easy to see that for each $i = 1, \ldots, n$ the output shares are of the following form

$$A_i = 2^0 \cdot \mathtt{B2A}_{\mathtt{q-Bit}}(x_i^{(1)}) + 2^1 \cdot \mathtt{B2A}_{\mathtt{q-Bit}}(x_i^{(2)}) + \cdots + 2^{k-1} \cdot \mathtt{B2A}_{\mathtt{q-Bit}}(x_i^{(k)})$$

and therefore $\sum_{i=1}^{n} A_i = x \mod q$.

Security. As done for the previous algorithms, we give in the following the proof of security according to the t − SNI property, for $t \le n - 1$.

Proposition 5. $\mathtt{SecB2A}_{\mathtt{q}}$ *in Algorithm 9 is* t − SNI, *with* $t \le n - 1$.

Proof. Proving that Algorithm 9 is t − SNI follows from the fact that $\mathtt{SecB2A}_{q-Bit}$ is t − SNI and from the structure of the algorithm itself, depicted in Figure 4.

First, we define $\mathcal{C}_i$, for $i = 1, \ldots, k$ as in Figure 4, where $\mathcal{C}_1 := \mathtt{SecB2A}_{\mathtt{q-Bit}}$ and $\mathcal{C}_n := \mathtt{SecB2A}_{\mathtt{q}}$. We prove the thesis by induction on $i \in \mathbb{N}$.

Thanks to Proposition 4, $\mathcal{C}_1$ is t − SNI .
We suppose now that $\mathcal{C}_i$ is t − SNI for all $i = 1, \ldots, n-1$ and we prove that $\mathcal{C}_n$ is t − SNI. Let $\Omega = (\mathcal{I}, \mathcal{O})$ be the set of adversarial observations on $\mathcal{C}_n$, where $\mathcal{I}$ are the ones on the internal values and $\mathcal{O}$ the ones on the output values, with $|\mathcal{I}| + |\mathcal{O}| \le t$. In particular, let $\mathcal{I}_1$ be the set of probes on $+$, $\mathcal{I}_2$ the set of probes on $2\cdot$, $\mathcal{I}_3$ be the set of probes on $\mathcal{C}_{n-1}$ and $\mathcal{I}_4$ be the set of probes on $\mathtt{SecB2A}_{\mathtt{q-Bit}}$, with $\sum_j |\mathcal{I}_j| \le |\mathcal{I}|$.

We prove the existence of a simulator which can simulate the adversary's view by using at most $|\mathcal{I}|$ input shares. We proceed with the analysis of the circuit from right to left.

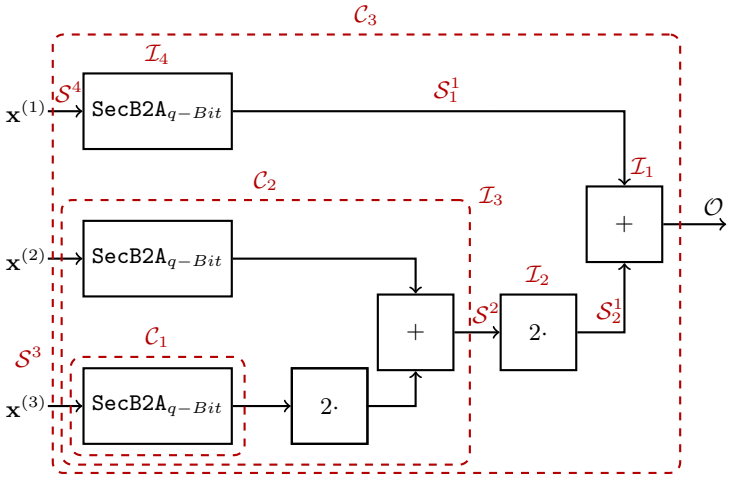

Fig. 4: Structure of $\texttt{SecB2A}_q$ in Algorithm 9 for $k = 3$.

Since $+$ is a linear operation and $|I_1 \cup \mathcal{O}| \leq t$, then there exist two observation sets $\mathcal{S}_1^1, \mathcal{S}_2^1$ such that $|\mathcal{S}_1^1| \leq |I_1 \cup \mathcal{O}|$, $|\mathcal{S}_2^1| \leq |I_1 \cup \mathcal{O}|$ and the gadget can be simulated using at most $|\mathcal{S}_1^1| + |\mathcal{S}_2^1|$ shares of the inputs.

Since $2\cdot$ is a linear operation and $|I_2 \cup \mathcal{S}_2^1| \leq t$, then there exists an observation set $\mathcal{S}^2$ such that $|\mathcal{S}^2| \leq |I_2 \cup \mathcal{S}_2^1|$ and the gadget can be simulated using at most $|\mathcal{S}^2| + |\mathcal{S}_2^1|$ shares of the inputs.

Since, for the assumption step, $\mathcal{C}_{n-1}$ is t − SNI and moreover $|\mathcal{I}_3 \cup \mathcal{S}^2| \leq t$, there exists an observation set $\mathcal{S}^3$ such that $|\mathcal{S}^3| \leq |I_3|$ and the gadget can be simulated using at most $|\mathcal{S}^3|$ shares of the inputs.

Since $\texttt{SecB2A}_{\texttt{q-Bit}}$ is t − SNI and moreover $|\mathcal{I}_4 \cup \mathcal{S}_1^1| \leq t$, there exists an observation set $\mathcal{S}^4$ such that $|\mathcal{S}^4| \leq |I_4|$ and the gadget can be simulated using at most $|\mathcal{S}^4|$ shares of the inputs.

By combining the steps above, we see that $\mathcal{C}_n$ can be simulated by using in total $|\mathcal{S}^3 \cup \mathcal{S}^4| \leq |\mathcal{I}_3| + |\mathcal{I}_4| \leq |\mathcal{I}|$ input shares, completing the proof. □

4.3 Performance Analysis

We analyze both of our new conversion algorithms regarding the number of required operations and randoms. They are compared to the cubic conversion from [6] for a prime modulus $q = 12289$ as used in NewHope. Furthermore, we compare $\texttt{SecB2A}_q$ to the conversions from [14] and [7], since they are currently the fastest algorithms for power-of-two moduli secure at arbitrary orders[4].

[4] Note that there are order-optimized algorithms which can provide an even better performance for specific values of t (i.e., Goubin [19] for $t = 1$, and Hutter and Tunstall [21] for $t = 2$). However, for power-of-two moduli our $\texttt{SecB2A}_q$ is only

Table 1: Operation count for Boolean-to-arithmetic conversions with prime modulo $q = 12289$ for (**A**) $\texttt{SecB2A}_q$, (**B**) `SecBoolArithModp` (cubic) [6], and (**C**) `SecArithBoolModp` (quadratic).

	n	2	3	4	5	6	7	8	9	10	11	16
$k = 4$	(**A**)	76	174	308	478	684	926	1,204	1,518	1,868	2,254	4,724
$k = 8$	(**A**)	156	354	624	966	1,380	1,866	2,424	3,054	3,756	4,530	9,480
$k = 15$	(**A**)	296	669	1,177	1,820	2,598	3,511	4,559	5,742	7,060	8,513	17,803
	(**B**)	765	2,451	5,617	10,722	18,225	28,585	42,261	59,712	81,397	107,775	326,125
	(**C**)	695	1,948	3,513	5,842	8,483	11,592	15,013	19,354	24,007	29,128	60,973

Number of Operations. For comparison, we estimate the number of operations of the different Boolean-to-arithmetic conversions. Similar to [12], we assume that randomness generation takes unit time, and we do not consider the modulo reductions. For [14] we use the estimation given in [12][5], and for [7] the authors provide a closed equation to compute the number of operations. Our newly-proposed $\texttt{SecB2A}_q$ has a run time complexity[6] of

$$T_{\texttt{SecB2Aq}}(n, k) = \frac{9kn^2}{2} + \frac{5kn}{2} - 2n - 3k.$$

We compare the $\texttt{B2A}_q$ conversions considering the prime modulus from the NIST submission NewHope $q = 12289$. It should be noted that the modular addition `SecAddModp` requires $2^k > 2q$. Therefore, we always call the function `SecBoolArithModp` with $k' = \lceil \log_2 2q \rceil = 15$ in this evaluation. The results are presented in Table 1. It is noticeable that $\texttt{SecB2A}_q$ outperforms both versions of `SecBoolArithModp` significantly. This is mostly due to the small values of k. It is expected that for larger k the quadratic variant of `SecBoolArithModp` is the most efficient $\texttt{B2A}_q$ conversion, as it scales with $\mathcal{O}(\log k)$ instead of $\mathcal{O}(k)$. Nevertheless, for our case study we obtain a large performance improvement by using $\texttt{SecB2A}_q$, since we need to transform either 1- or 5-bit variables.

Additionally, we compare our new algorithms with the state-of-the-art `B2A` conversions assuming a modulus of 2^k for different values of k. The resulting performances are given in Table 2. As expected, the algorithm of [7] outperforms all other solutions for small n. Surprisingly, however, our new conversion $\texttt{SecB2A}_q$ outperforms the approach of [14] for the considered values of k. We expect that, for larger bit sizes and by incorporating a logarithmic adder, [14] will eventually outperform our approach.[7] Nevertheless, for $k = 32$, which is used in many sym-

competitive for larger values of t, and we, thus, exclude these specific examples from the comparison.

[5] Note that there is typo in the final equation: $T'_n = 2n + T_n + B_n + 3n^2 + n$.

[6] In the full version of the paper we derive the complexity of the remaining algorithms.

[7] It was shown in [13] that the logarithmic adder offers a significant improvement over the linear approach for $k > 32$ at first-order.

Table 2: Operation count for Boolean-to-arithmetic conversions mod 2^k for (**A**) $\texttt{SecB2A}_q$, (**D**) Bettale et al. [7], and (**E**) Coron et al. [14]. The bold operation counts indicate that our new algorithm provides the best performance of the considered algorithms for a given value of k.

	n	2	3	4	5	6	7	8	9	10	11	16
$\forall k$	(**D**)	15	49	123	277	591	1,225	2,499	5,053	10,167	20,401	655,251
$k=1$	(**A**)	**12**	**33**	**63**	**102**	**150**	**207**	**273**	**348**	**432**	**525**	**1,125**
	(**E**)	56	135	234	374	534	721	928	1,183	1,458	1,760	3,640
$k=4$	(**A**)	76	174	308	478	684	**926**	**1,204**	**1,518**	**1,868**	**2,254**	**4,724**
	(**E**)	134	354	636	1,052	1,530	2,098	2,728	3,520	4,374	5,318	11,248
$k=8$	(**A**)	156	354	624	966	1,380	1,866	**2,424**	**3,054**	**3,756**	**4,530**	**9,480**
	(**E**)	238	646	1,172	1,956	2,858	3,934	5,128	6,636	8,262	10,062	21,392
$k=32$	(**A**)	636	1,434	2,520	3,894	5,556	7,506	9,744	12,270	15,084	**18,186**	**38,016**
	(**E**)	862	2,398	4,388	7,380	10,826	14,950	19,528	25,332	31,590	38,526	82,256

metric algorithms, our $\texttt{SecB2A}_{q-Bit}$ does provide a performance improvement even for standard moduli assuming $n \geq 11$. We further want to note that our algorithm comes with a proof of $t-\mathsf{SNI}$, which allows composability with other modules under certain assumptions [5].

Randomness Complexity We estimate the number of required random bits for the different Boolean-to-arithmetic conversions. We denote the bit size of the input encoding as k_1 and for samples in $\mathbb{F}_q$, where q is not a power-of-two, we assume $k_2 = \lceil \log_2 q \rceil$ bits. A more detailed discussion of the sampling process of such values is provided in the case study. Again, for [7] the authors provide a closed equation to compute the number of random elements. Our newly-proposed $\texttt{SecB2A}_q$ has a randomness complexity of $R_{\texttt{SecB2Aq}}(n, k_1) = k_2 k_1 (n^2 - n)$.

As before, we initially compare the $\texttt{B2A}_q$ conversions considering the prime modulus from the NIST submission NewHope $q = 12289$. The results are presented in Table 3 and again $\texttt{SecB2A}_q$ outperforms `SecBoolArithModp`. However, it should be noted that all random samples for $\texttt{SecB2A}_q$ are from $\mathbb{F}_q$, while `SecBoolArithModp` only requires $(n-1)$ random values from $\mathbb{F}_q$ and the remaining are sampled from $\mathbb{F}_{2^{k_1}}$, which can be more efficient depending on the RNG.

In the typical use cases of a power-of-two modulus (like in symmetric crypto), there is no difference between k_1 and k_2 and thus we evaluate the number of RNG calls instead of random bits for this case. The resulting performances are given in Table 4. Again, the algorithm of [7] outperforms all other solutions for small n, while $\texttt{SecB2A}_q$ provides the best performance for certain values of k and n.

Table 3: Required random bits for Boolean-to-arithmetic conversions with $q = 12289$ for (**A**) $\texttt{SecB2A}_q$, (**B**) `SecBoolArithModp` (cubic) [6], and (**C**) `SecArithBoolModp` (quadratic). Note that sampling from $\mathbb{F}_q$ is estimated with $\lceil \log_2 q \rceil$ bits.

	n	2	3	4	5	6	7	8	9	10	11	16
$k=4$	(**A**)	112	336	672	1,120	1,680	2,352	3,136	4,032	5,040	6,160	13,440
$k=8$	(**A**)	224	672	1,344	2,240	3,360	4,704	6,272	8,064	10,080	12,320	26,880
$k=15$	(**A**)	420	1,260	2,520	4,200	6,300	8,820	11,760	15,120	18,900	23,100	50,400
	(**B**)	1,244	4,933	12,282	24,506	42,820	68,439	102,578	146,452	201,276	268,265	828,210
	(**C**)	854	2,968	5,922	10,556	16,030	22,764	30,338	40,012	50,526	62,300	137,970

Table 4: Number of RNG calls for Boolean-to-arithmetic conversions mod 2^k for (**A**) $\texttt{SecB2A}_q$, (**D**) Bettale et al. [7], and (**E**) Coron et al. [14]. The bold call counts indicate that our new algorithm provides the best performance of the considered algorithms for a given value of k.

	n	2	3	4	5	6	7	8	9	10	11	16
$\forall k$	(**D**)	2	7	18	41	88	183	374	757	1,524	3,059	98,286
$k=1$	(**A**)	2	**6**	**12**	**20**	**30**	**42**	**56**	**72**	**90**	**110**	**240**
	(**E**)	9	23	41	66	95	129	167	213	263	318	663
$k=4$	(**A**)	8	24	48	80	120	**168**	**224**	**288**	**360**	**440**	**960**
	(**E**)	15	44	83	141	209	291	383	498	623	762	1,647
$k=8$	(**A**)	16	48	96	160	240	336	448	**576**	**720**	**880**	**1,920**
	(**E**)	23	72	139	241	361	507	671	878	1,103	1,354	2,959
$k=32$	(**A**)	64	192	384	640	960	1,344	1,792	2,304	2,880	3,520	**7,680**
	(**E**)	71	240	475	841	1,273	1,803	2,399	3,158	3,983	4,906	10,831

5 Higher-Order Masked Binomial Sampling

Our sampling algorithms assume that they are given two variables (x, y) that have a length of κ bits and are Boolean-encoded as $(\mathbf{x}, \mathbf{y})$. This is in accordance with many of the aforementioned schemes which rely on a PRNG (Boolean-masked) to produce uniform pseudo-randomness. The sampler needs to compute $\texttt{HW}(x) - \texttt{HW}(y)$ in a secure fashion on these encodings and produce arithmetic shares $\mathbf{A}$ with $\sum_i A_i = \texttt{HW}(x) - \texttt{HW}(y) \mod q$ for a given modulus q to fit the subsequent lattice operations. Since this conversion can be done with any of the aforementioned $\texttt{B2A}_q$ schemes, the algorithms contain a generic function call. Initially, we present a generalization of [25], before proposing a more efficient sampling algorithm based on bitslicing. Both algorithms are proven to be $t - \mathsf{SNI}$ and their performances are compared using NewHope as a case study.

Algorithm 10 $\texttt{SecSampler}_1$

Input: $\mathbf{x} = (x_i)_{1 \le i \le n} \in \mathbb{F}_{2^\kappa}$, $\mathbf{y} = (y_i)_{1 \le i \le n} \in \mathbb{F}_{2^\kappa}$ such that $\bigoplus_i x_i = x$, $\bigoplus_i y_i = y$
Output: $\mathbf{A} = (A_i)_{1 \le i \le n} \in \mathbb{F}_q$ such that $\sum_i A_i = \texttt{HW}(x) - \texttt{HW}(y) \mod q$

1: $(A_i)_{1 \le i \le n} \leftarrow 0$
2: **for** $j = 0$ to $\kappa - 1$ **do**
3: $\quad \mathbf{B} \leftarrow \texttt{B2A}_q\left((\mathbf{x} >> j) \wedge 1\right)$
4: $\quad \mathbf{C} \leftarrow \texttt{B2A}_q\left((\mathbf{y} >> j) \wedge 1\right)$
5: $\quad \mathbf{A} \leftarrow \mathbf{A} + \mathbf{B} \mod q$
6: $\quad \mathbf{A} \leftarrow \mathbf{A} - \mathbf{C} \mod q$
7: **end for**

5.1 Generalization of [25]

As briefly discussed in Section 2.4, the bits of $(\mathbf{x}, \mathbf{y})$ are transformed separately to arithmetic shares. We extend this approach in Algorithm 10 to be generic for any number of shares n, modulus q, and length of the bit-vectors κ. In particular, we first transform each of the 2κ bits separately to an arithmetic encoding modulo q. These are then summed component-wise to compute the Hamming weight of each variable and the results are subtracted again component-wise.

Correctness. The correctness of $\texttt{SecSampler}_1$ follows directly by the construction of the algorithm. Indeed, since at every iteration of the loop

$$\begin{aligned}\mathbf{A} =& \texttt{B2A}_q\left((\mathbf{x} >> 0) \wedge 1\right) - \texttt{B2A}_q\left((\mathbf{y} >> 0) \wedge 1\right) \\ &+ \cdots + \texttt{B2A}_q\left((\mathbf{x} >> \kappa - 1) \wedge 1\right) - \texttt{B2A}_q\left((\mathbf{y} >> \kappa - 1) \wedge 1\right)\end{aligned}$$

we have

$$\begin{aligned}\sum_i A_i =& \sum_i \big(\texttt{B2A}_q\left((x_i >> 0) \wedge 1\right) - \texttt{B2A}_q\left((y_i >> 0) \wedge 1\right) + \dots \\ &+ \texttt{B2A}_q\left((x_i >> \kappa - 1) \wedge 1\right) - \texttt{B2A}_q\left((y_i >> \kappa - 1) \wedge 1\right)\big) \\ =& \texttt{HW}(x) - \texttt{HW}(y) \mod q\end{aligned}$$

Security. The security of the sampler described in Algorithm 10 can be easily derived from its basic structure and utilization of t − SNI gadgets.

Proposition 6. $\texttt{Sampler}_1$ *in Algorithm 10 is* t − SNI, *with* $t \le n - 1$

Proof. We point out that the t − SNI security of both considered $\texttt{B2A}_q$ algorithms, i.e., $\texttt{SecB2A}_{\texttt{q-Bit}}$ proven in Proposition 4 and $\texttt{SecArithBoolModp}$ in Proposition 1, receiving independent inputs, guarantees that every loop of Algorithm 10 represents a t − SNI gadget and therefore the output $\mathbf{A}$ can be securely injected in the sum of the following loop. □

Algorithm 11 `SecBitAdd`

Input: $\mathbf{x} = (x_i)_{1 \le i \le n} \in \mathbb{F}_{2^\kappa}$ such that $\bigoplus_i x_i = x$, $\lambda = \lceil \log_2(\kappa + 1) \rceil + 1$
Output: $\mathbf{z} = (z_i)_{1 \le i \le n} \in \mathbb{F}_{2^\lambda}$ such that $\bigoplus_i z_i = \mathtt{HW}(x)$

1: $(t_i)_{1 \le i \le n} \leftarrow 0$
2: $(z_i)_{1 \le i \le n} \leftarrow 0$
3: **for** $j = 1$ to κ **do**
4: $\quad \mathbf{t}^{(1)} \leftarrow \mathbf{z}^{(1)} \oplus \mathbf{x}^{(j)}$
5: $\quad \mathbf{w} \leftarrow \mathbf{x}^{(j)}$
6: $\quad$ **for** $l = 2$ to λ **do**
7: $\quad\quad \mathbf{w} \leftarrow \mathtt{SecAnd}(\mathbf{w}, \mathbf{z}^{(l-1)})$
8: $\quad\quad \mathbf{t}^{(l)} \leftarrow \mathbf{z}^{(l)} \oplus \mathbf{w}$
9: $\quad$ **end for**
10: $\quad \mathbf{z} \leftarrow \mathbf{t}$
11: **end for**

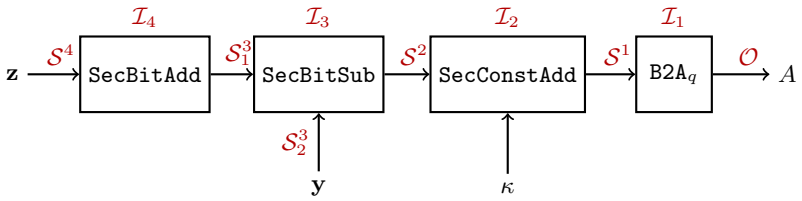

Fig. 5: Structure of $\mathtt{SecSampler}_2$ in Algorithm 15 (lines 1-4).

5.2 New Bitsliced Masked Binomial Sampler

In our improved sampler $\mathtt{SecSampler}_2$, we first compute the Hamming weight of x on the Boolean encodings using bitslicing to significantly increase the throughput. We further improve the performance by directly subtracting the Hamming weight of y from the result, again using bitslicing. In this way, the sampler only requires a single conversion. However, to correctly map the sign of the difference (i.e., negative values would be transformed incorrectly) it is necessary to add κ before converting. A generic algorithm to add such a constant to a Boolean encoding is provided in Algorithm 13. However, for specific values of κ this can be significantly optimized, e.g., for $\kappa = 8$ as in NewHope the addition can be done with only component-wise XOR, as shown in Algorithm 14. Finally, after the $\mathtt{B2A}_q$ conversion, the additional κ needs to be subtracted to recover the correct result, and this can be done component-wise on the arithmetic shares. The complete procedure is given in Algorithm 15. Since the input variables are in a bitsliced format, we directly denote the j-th bit of the l-th share of $\mathbf{x}$ as $x_l^{(j)}$.

Algorithm 12 `SecBitSub`

Input: $\mathbf{z} = (z_i)_{1 \le i \le n} \in \mathbb{F}_{2^\lambda}, \mathbf{y} = (y_i)_{1 \le i \le n} \in \mathbb{F}_{2^\kappa}$ such that $\bigoplus_i z_i = z$ and $\bigoplus_i y_i = y$, $\lambda = \lceil \log_2(\kappa + 1) \rceil + 1$
Output: $\mathbf{z} = (z_i)_{1 \le i \le n} \in \mathbb{F}_{2^\lambda}$ such that $\bigoplus_i z_i = z - \mathtt{HW}(y)$

1: $(t_i)_{1 \le i \le n} \leftarrow 0$
2: **for** $j = 1$ to κ **do**
3: $\quad \mathbf{t}^{(1)} \leftarrow \mathbf{z}^{(1)} \oplus \mathbf{y}^{(j)}$
4: $\quad \mathbf{w} \leftarrow \mathbf{y}^{(j)}$
5: $\quad$ **for** $l = 2$ to λ **do**
6: $\quad\quad \mathbf{u} \leftarrow \mathbf{z}^{(l-1)}$
7: $\quad\quad u_1 \leftarrow \neg u_1$
8: $\quad\quad \mathbf{w} \leftarrow \mathtt{SecAnd}(\mathbf{w}, \mathbf{u})$
9: $\quad\quad \mathbf{t}^{(l)} \leftarrow \mathbf{z}^{(l)} \oplus \mathbf{w}$
10: $\quad$ **end for**
11: $\quad \mathbf{z} \leftarrow \mathbf{t}$
12: **end for**

Algorithm 13 `SecConstAdd`

Input: $\mathbf{x} = (x_i)_{1 \le i \le n} \in \mathbb{F}_{2^\lambda}$, $\lambda = \lceil \log_2(\kappa + 1) \rceil + 1$
Output: $\mathbf{y} = (y_i)_{1 \le i \le n} \in \mathbb{F}_{2^\lambda}$ such that $\bigoplus_i y_i = x + \kappa$

1: $(t_i)_{1 \le i \le n} \leftarrow (\kappa, 0, \ldots, 0)$
2: $\mathbf{t} \leftarrow \mathtt{RefreshXOR}(\mathbf{t}, \lambda)$
3: $\mathbf{y} \leftarrow \mathtt{SecAdd}(\mathbf{x}, \mathbf{t})$

Correctness. The correctness of $\mathtt{SecSampler}_2$ is easy to show.

$$\begin{aligned}
\sum_i A_i &= \mathtt{B2A}_q(\mathtt{SecConstAdd}(\mathtt{SecBitSub}(\mathtt{SecBitAdd}(\mathbf{x}), \mathbf{y}), \kappa)) - \kappa \\
&= \mathtt{B2A}_q(\mathtt{SecConstAdd}(\mathtt{SecBitSub}(\mathtt{HW}(x), \mathbf{y}), \kappa)) - \kappa \\
&= \mathtt{B2A}_q(\mathtt{SecConstAdd}(\mathtt{HW}(x) - \mathtt{HW}(y), \kappa)) - \kappa \\
&= \mathtt{B2A}_q(\mathtt{HW}(x) - \mathtt{HW}(y) + \kappa) - \kappa = \mathtt{HW}(x) - \mathtt{HW}(y) \mod q.
\end{aligned}$$

Security. Before proving the security of $\mathtt{SecSampler}_2$ in Algorithm 15, we briefly summarize the security properties which its subroutines satisfy.

First we show that `SecBitAdd` in Algorithm 11 is $t - \mathsf{NI}$ and we start the analysis by focusing on its structure. We recall that `SecAnd` [14] is $t - \mathsf{SNI}$ and it receives at every iteration independent inputs (line 7). The output of each `SecAnd` is added with a `XOR` to a value independent from its inputs (line 8), therefore the entire inner loop (lines 6-9) represents a $t - \mathsf{SNI}$ gadget. This is recursively composed in the outer loop (lines 3-11) providing the outputs $(\mathbf{t}^{(2)}, \ldots, \mathbf{t}^{(\lambda)})$ and preserving the $t - \mathsf{SNI}$ property. Additionally, at every iteration of the outer loop, $\mathbf{x}^{(j)}$ is added with a `XOR` to $\mathbf{z}^{(1)}$ (line 4), resulting in the output $\mathbf{t}^{(1)} = \mathbf{x}^{(1)} + \ldots + \mathbf{x}^{(\kappa)}$. Let us suppose an attacker probes a set of t_1 values $\mathcal{P}_1$ on the shares $(\mathbf{t}^{(2)}, \ldots, \mathbf{t}^{(\lambda)})$ or on the internal values produced during the

Algorithm 14 `SecConstAdd` (optimized for $\kappa = 8$)

Input: $\mathbf{x} = (x_i)_{1 \le i \le n} \in \mathbb{F}_{2^\lambda}$
Output: $\mathbf{y} = (y_i)_{1 \le i \le n} \in \mathbb{F}_{2^\lambda}$ such that $\bigoplus_i y_i = x + 8$
1: $\mathbf{y} \leftarrow \mathbf{x}$
2: $\mathbf{y}^{(5)} \leftarrow \mathbf{y}^{(5)} \oplus \mathbf{y}^{(4)}$
3: $y_1^{(4)} \leftarrow y_1^{(4)} \oplus 1$

Algorithm 15 $\texttt{SecSampler}_2$

Input: $\mathbf{x} = (x_i)_{1 \le i \le n} \in \mathbb{F}_{2^\kappa}$, $\mathbf{y} = (y_i)_{1 \le i \le n} \in \mathbb{F}_{2^\kappa}$, κ, such that $\bigoplus_i x_i = x$, $\bigoplus_i y_i = y$
Output: $\mathbf{A} = (A_i)_{1 \le i \le n} \in \mathbb{F}_q$ such that $\sum_i A_i = \texttt{HW}(x) - \texttt{HW}(y) \mod q$
1: $\mathbf{z} \leftarrow \texttt{SecBitAdd}(\mathbf{x})$
2: $\mathbf{z} \leftarrow \texttt{SecBitSub}(\mathbf{z}, \mathbf{y})$
3: $\mathbf{z} \leftarrow \texttt{SecConstAdd}(\mathbf{z}, \kappa)$
4: $\mathbf{A} \leftarrow \texttt{B2A}_q(\mathbf{z})$
5: $A_1 \leftarrow A_1 - \kappa \mod q$

concatenation of the inner loop, and a set of t_2 values $\mathcal{P}_2$ on the computation of $\mathbf{t}^{(1)}$, with $t_1 + t_2 \le t$. In particular let t_1^O, t_2^O be the probes on the output values and t_1^I, t_2^I the ones on the internals, with $t_1^O + t_1^I = t_1$ and $t_2^I + t_2^O = t_2$. The t − SNI of the inner loop guarantees that every value in $\mathcal{P}_1$ can be simulated by using at most t_1^I shares of the inputs. On the other hand, because of the linearity of the computation of $\mathbf{t}^{(1)}$, the probes in $\mathcal{P}_2$ can be simulated using at most $t_2^I + t_2^O$ shares of the input. Therefore, by Definition 1, `SecBitAdd` is $t-$NI.

Now, since `SecBitSub` in Algorithm 12 follows the same procedure as Algorithm 11, with the exception of Lines 6 and 7, which simply add a negation to the interested value, then it is $t-$NI as well.

As for `SecConstAdd` in Algorithm 13, from [6] we know that `RefreshXOR` is t − SNI and `SecAdd` is $t-$NI. Therefore it is easy to see that the composition of them, as it appears in Algorithm 13, is $t-$NI. Regarding the optimized version of `SecConstAdd` in Algorithm 14, here the security comes directly from the fact that the algorithm is linear.

Proposition 7. $\texttt{SecSampler}_2$ *in Algorithm 15 is* t − SNI, *with* $t \le n - 1$

Proof. Before proceeding with the proof, we point out that $\texttt{SecSampler}_2$ is given by the circuit in Figure 5 with the addition of a share-wise sum between the output share A_1 and the public value $-\kappa$ (line 5 of Algorithm 15). Since the simulation of such value depends only on the simulation of A_1, the security level of $\texttt{SecSampler}_2$ is not influenced by this additional operation and it corresponds to the one of the algorithm in Figure 5.

Let $\Omega = (\mathcal{I}, \mathcal{O})$ be the set of adversarial observations on the circuit in Figure 5, where $\mathcal{I}$ are the ones on the internal values and $\mathcal{O}$ on the output shares, with $|\mathcal{I}| + |\mathcal{O}| \le t$. In particular, let $\mathcal{I}_1$ be the set of probes on $\texttt{B2A}_q$, $\mathcal{I}_2$ on `SecConstAdd`, $\mathcal{I}_3$ on `SecBitSub` and $\mathcal{I}_4$ on `SecBitAdd`, with $\sum_j |\mathcal{I}_j| \le |\mathcal{I}|$.

We prove the existence of a simulator which simulates the adversary's view by using at most $|\mathcal{I}|$ input shares, analyzing of the circuit from right to left.

Since $\texttt{B2A}_q$ is t − SNI, there exists an observation set $\mathcal{S}^1$ such that $|\mathcal{S}^1| \leq |\mathcal{I}_1|$ and the gadget can be simulated using at most $|\mathcal{S}^1|$ shares of its input.

Since `SecConstAdd` is $t-\mathsf{NI}$ and $|\mathcal{I}_2 \cup \mathcal{S}^1| \leq t$, then there exist an observation set $\mathcal{S}^2$ such that $|\mathcal{S}^2| \leq |I_2 \cup \mathcal{S}^1|$ and the gadget can be simulated using at most $|\mathcal{S}^2|$ shares of the inputs.

Since `SecBitSub` is $t-\mathsf{NI}$ and $|\mathcal{I}_3 \cup \mathcal{S}^2| \leq t$, then there exist two observation sets $\mathcal{S}^3_1, \mathcal{S}^3_2$ such that $|\mathcal{S}^3_1| \leq |\mathcal{I}_3 \cup \mathcal{S}^2|$, $|\mathcal{S}^3_2| \leq |\mathcal{I}_3 \cup \mathcal{S}^2|$ and the gadget can be simulated using at most $|\mathcal{S}^3_1| + |\mathcal{S}^3_2|$ shares of the inputs.

Since `SecBitAdd` is $t-\mathsf{NI}$ and $|\mathcal{I}_4 \cup \mathcal{S}^3_1| \leq t$, then there exist an observation set $\mathcal{S}^4$ such that $|\mathcal{S}^4| \leq |\mathcal{I}_4 \cup \mathcal{S}^3_1|$ and the gadget can be simulated using at most $|\mathcal{S}^4|$ shares of the inputs.

By combining the steps above, we see that $\texttt{SecSampler}_2$ can be simulated by using in total $|\mathcal{S}^4| \leq |\mathcal{I}_4| + |\mathcal{S}^3_1| \leq |\mathcal{I}_4| + |\mathcal{I}_3| + |\mathcal{S}^2| \leq |\mathcal{I}_4| + |\mathcal{I}_3| + |\mathcal{I}_2| + |\mathcal{S}^1| \leq |\mathcal{I}_4| + |\mathcal{I}_3| + |\mathcal{I}_2| + |\mathcal{I}_1| \leq |\mathcal{I}|$ input shares, proving that it is t − SNI. □

5.3 Performance Analysis

To better compare the two sampling approaches, we derive the run time complexity for both. The calls to $\texttt{B2A}_q$ and `SecConstAdd` are not substituted, since their performance strongly depends on the used parameters, which may allow further optimizations, e.g., Algorithm 14. With $\lambda = \lceil \log_2(\kappa+1) \rceil + 1$, we derive

$$
\begin{aligned}
T_{\texttt{SecSampler}_1}(n,\kappa) &= 2\kappa T_{\texttt{B2A}_q}(n,1) + 6n\kappa, \\
T_{\texttt{SecSampler}_2}(n,\kappa) &= T_{\texttt{SecBitAdd}}(n,\kappa) + T_{\texttt{SecBitSub}}(n,\kappa) + T_{\texttt{SecConstAdd}}(n,\kappa) \\
&\quad + T_{\texttt{B2A}_q}(n,\lambda) + n \\
&= T_{\texttt{B2A}_q}(n,\lambda) + T_{\texttt{SecConstAdd}}(n,\kappa) \\
&\quad + 7\kappa\lambda n^2 - 7\kappa n^2 + 7\kappa\lambda n + \kappa\lambda - 5\kappa n - \kappa + n.
\end{aligned}
$$

It is noticeable that $\texttt{SecSampler}_2$ requires only one conversion of λ bits, while $\texttt{SecSampler}_1$ consists of 2κ conversion of one bit. This can lead to significant advantages for the former approach assuming small κ as shown in the case study.

Regarding the randomness complexity, we observe a similar trend:

$$
\begin{aligned}
R_{\texttt{SecSampler}_1}(n,\kappa) &= 2\kappa R_{\texttt{B2A}_q}(n,1), \\
R_{\texttt{SecSampler}_2}(n,\kappa) &= R_{\texttt{SecBitAdd}}(n,\kappa) \\
&\quad + R_{\texttt{SecBitSub}}(n,\kappa) + R_{\texttt{SecConstAdd}}(n,\kappa) + R_{\texttt{B2A}_q}(n,\lambda) \\
&= R_{\texttt{B2A}_q}(n,\lambda) + R_{\texttt{SecConstAdd}}(n,\kappa) + \kappa\lambda n^2 - \kappa n^2 - \kappa\lambda n + \kappa n
\end{aligned}
$$

5.4 Case Study: NewHope

To concretely evaluate the performance of our proposed sampling algorithms, we conduct a case study using the parameters of the NIST submission NewHope.

We set the length of the bit-vectors to $\kappa = 8$ and the prime to $q = 12289$. The same prime can be found in multiple NIST submissions, like Kyber and HILA5. The parameter κ is usually different though and for Kyber set to 3, 4, or 5 and for HILA5 set to 16. Both sampling approaches are evaluated with the proposed $\texttt{B2A}_q$ conversions $\texttt{SecB2A}_q$ and `SecArithBoolModp` (quadratic). The latter is instantiated with $k' = \lceil \log_2 2q \rceil = 15$ as discussed in the previous section.

We implement all variants on a 32-bit ARM Cortex-M4F microcontroller embedded in an STM32F4 DISCOVERY board with 1 Mbyte of flash memory, 192 kbyte of RAM, a floating-point unit (FPU), and a true random number generator (TRNG). The TRNG needs 40 cycles of a 48 MHz clock to generate a random 32-bit value. The sampling of a true random value runs simultaneously to other computations of the microcontroller. Assuming a sufficient amount of clock cycles between two calls to the TRNG, a sample will be generated without any wait cycles and therefore the average TRNG call will be much faster than 40 cycles at 48 MHz. The maximum clock frequency of the microcontroller is 168 MHz. We use the `CYCCNT` register of the data watchpoint and trace unit of the microcontroller to measure the performance of our implementation.

To prevent timing leakage, our implementations have a secret-independent running time. In particular, we refrained from using conditional statements or instructions with varying execution time in critical parts of our implementation. We furthermore disabled the data and instruction cache of our target microcontroller. For the implementation of the $\texttt{B2A}_q$ conversions, we need uniform random numbers mod $q = 12289$. The TRNG of our development board outputs 32 uniform random bits. To sample a uniform random number mod q, we split the 32-bit output of the TRNG into two 14-bit vectors and drop the remaining four bits. We then check whether the first 14-bit vector is smaller than q or not. If yes, we accept and return the value. If not we also check the second value. If this check fails again, we get more true random 32-bit vectors from the TRNG until we find a valid sample. This means that there is a $\frac{q}{2^{14}} = 0.75$ probability of a sample being accepted and a 0.25 probability of a sample being rejected. Such a sampling approach does not have a constant run time, but it still does not introduce a timing leakage, as the time required to generate a valid sample is completely independent from the value of the generated sample.

The results of our implementation can be seen in Table 5, where the comparison for $n = 2$ also includes the masked binomial sampler from Oder et al. [25][8]. It is noticeable that $\texttt{SecB2A}_q$ outperforms `SecArithBoolModp` (quadratic) for every order. As discussed before, this huge speed-up is due to the specific parameters of the case study. Indeed the bit sizes of the input $k = 1, 5$ do not fulfill the requirement of $2^k > 2q$, which thwarts its performance since we need

[8] In contrast to [25], our cycle counts do not include the generation of the input bit vectors. Therefore, our 3,757 cycles for one sample do not match the 6 million cycles for 1024 coefficients reported in [25]. However, as the generation of the input samples is a constant overhead that is independent from the sampling algorithm or the $\texttt{B2A}_q$ conversion, we decided to exclude it from our measurements.

Table 5: Cycle counts for masked sampling of one binomial distributed coefficient using the B2A conversions (**A**) $\texttt{SecB2A}_q$ and (**C**) SecBoolArithModp (quadratic). We excluded the generation of the input bit vectors for better comparison as this is a constant overhead for all approaches.

	n	2	3	4	5
	[25]	3,637	-	-	-
$\texttt{SecSampler}_1$	(**C**)	271,423	638,315	1,076,155	1,758,184
	(**A**)	6,145	13,913	24,397	37,880
$\texttt{SecSampler}_2$	(**C**)	17,564	40,977	68,914	112,402
	(**A**)	2,649	5,573	9,462	14,587

to instantiate it with a larger k'. This is especially problematic for $\texttt{SecSampler}_1$, which requires conversions with $k = 1$. Overall, our $\texttt{SecSampler}_2$ offers a significant improvement over $\texttt{SecSampler}_1$ for both conversion algorithms and it even outperforms the approach from Oder et al. [25], that is highly optimized for $n = 2$. Applying a similar degree of optimization to our proposed sampler for this special case would help to decrease the number of cycles even further.

6 Conclusions

In this work, we initially presented two new conversion techniques to transform Boolean shares to arithmetic shares that work with arbitrary moduli and orders. While the first provides the best asymptotic complexity of $\texttt{B2A}_q$ algorithm with $\mathcal{O}(n^2 \log k)$, the second proposal offers a significant performance improvement for relevant bit sizes. It can even be applied in symmetric cryptography as for certain number of shares (e.g., $n \geq 11$ for $k = 32$), it outperforms previous work that is optimized for power-of-two moduli. Using these conversions as basis, we further constructed masked binomial sampling algorithms. To evaluate them, we developed implementations for a popular microcontroller platform to obtain realistic performance measurements. Thereby, we show that the combination of $\texttt{SecB2A}_q$ with our bitsliced sampler outperforms previous work and leads to the currently most-efficient masked binomial sampling algorithm for the considered parameters. Our work helps to better understand the overhead cost for masking of post-quantum cryptography and, thus, is an important contribution for the evaluation of these schemes in the ongoing NIST standardization process.

Acknowledgement

The authors are grateful to the AsiaCrypt2018 reviewers for useful comments and feedback. The research in this work was supported in part by the European Unions Horizon 2020 program under project number 644729 SAFEcrypto and

724725 SWORD, by the VeriSec project 16KIS0634 from the Federal Ministry of Education and Research (BMBF) and by H2020 project PROMETHEUS, grant agreement ID 780701.

References

1. E. Alkim, R. Avanzi, J. Bos, L. Ducas, A. de la Piedra, T. Pöppelmann, P. Schwabe, and D. Stebila. NewHope Algorithm Specifications and Supporting Documentation. https://newhopecrypto.org/data/NewHope_2017_12_21.pdf. Accessed: 2018-05-09.
2. E. Alkim, P. Jakubeit, and P. Schwabe. NewHope on ARM Cortex-M. In *SPACE*, 2016.
3. R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS-kyber. Technical report, National Institute of Standards and Technology, 2017. available at https://pq-crystals.org/kyber/data/kyber-specification.pdf.
4. G. Barthe, S. Belaïd, F. Dupressoir, P. Fouque, and B. Grégoire. Compositional Verification of Higher-Order Masking: Application to a Verifying Masking Compiler. *IACR Cryptology ePrint Archive*, 2015:506, 2015.
5. G. Barthe, S. Belaïd, F. Dupressoir, P. Fouque, B. Grégoire, P. Strub, and R. Zucchini. Strong Non-Interference and Type-Directed Higher-Order Masking. In *ACM CCS 2016*, pages 116–129. ACM, 2016.
6. G. Barthe, S. Belaïd, T. Espitau, P. Fouque, B. Grégoire, M. Rossi, and M. Tibouchi. Masking the GLP Lattice-Based Signature Scheme at Any Order. In *EUROCRYPT (2)*. Springer, 2018.
7. L. Bettale, J. Coron, and R. Zeitoun. Improved High-Order Conversion From Boolean to Arithmetic Masking. *TCHES*, 2018, 2018.
8. A. Biryukov, D. Dinu, Y. L. Corre, and A. Udovenko. Optimal first-order boolean masking for embedded iot devices. In *CARDIS*. Springer, 2017.
9. S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO*. Springer, 1999.
10. C. Chen, T. Eisenbarth, I. von Maurich, and R. Steinwandt. Differential Power Analysis of a McEliece Cryptosystem. In *ACNS*. Springer, 2015.
11. C. Chen, T. Eisenbarth, I. von Maurich, and R. Steinwandt. Masking Large Keys in Hardware: A Masked Implementation of McEliece. In *SAC*. Springer, 2015.
12. J. Coron. High-Order Conversion from Boolean to Arithmetic Masking. In *TCHES*. Springer, 2017.
13. J. Coron, J. Großschädl, M. Tibouchi, and P. K. Vadnala. Conversion from Arithmetic to Boolean Masking with Logarithmic Complexity. In *FSE*. Springer, 2015.
14. J. Coron, J. Großschädl, and P. K. Vadnala. Secure Conversion between Boolean and Arithmetic Masking of Any Order. In *CHES*. Springer, 2014.
15. J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren. SABER: Mod-LWR based KEM. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions.
16. B. Debraize. Efficient and provably secure methods for switching from arithmetic to boolean masking. In *CHES*. Springer, 2012.
17. J. Ding, T. Takagi, X. Gao, and Y. Wang. Dingkeyexchange. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions.

18. T. Eisenbarth, T. Kasper, A. Moradi, C. Paar, M. Salmasizadeh, and M. T. M. Shalmani. On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoqCode Hopping Scheme. In *CRYPTO*. Springer, 2008.
19. L. Goubin. A Sound Method for Switching between Boolean and Arithmetic Masking. In *CHES*. Springer, 2001.
20. T. Güneysu, V. Lyubashevsky, and T. Pöppelmann. Practical Lattice-Based Cryptography: A Signature Scheme for Embedded Systems. In *CHES*. Springer, 2012.
21. M. Hutter and M. Tunstall. Constant-time higher-order boolean-to-arithmetic masking. *IACR Cryptology ePrint Archive*, 2016:1023, 2016.
22. Y. Ishai, A. Sahai, and D. A. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO*. Springer, 2003.
23. M. Karroumi, B. Richard, and M. Joye. Addition with blinded operands. In *COSADE*. Springer, 2014.
24. P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO*. Springer, 1996.
25. T. Oder, T. Schneider, T. Pöppelmann, and T. Güneysu. Practical CCA2-Secure and Masked Ring-LWE Implementation. *TCHES*, 2018, 2018.
26. N. I. of Standards and Technology. Post-Quantum Cryptography - Round 1 Submissions. https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions. Accessed: 2018-12-10.
27. N. I. of Standards and Technology. Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf. Accessed: 2018-05-10.
28. O. Reparaz, S. S. Roy, R. de Clercq, F. Vercauteren, and I. Verbauwhede. Masking ring-LWE. *J. Cryptographic Engineering*, 6(2):139–153, 2016.
29. M.-J. O. Saarinen. HILA5. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions.
30. T. Schneider, A. Moradi, and T. Güneysu. Arithmetic addition over Boolean masking - towards first- and second-order resistance in hardware. In *ACNS*. Springer, 2015.
31. N. P. Smart, M. R. Albrecht, Y. Lindell, E. Orsini, V. Osheter, K. G. Paterson, and G. Peer. LIMA-1.1: A PQC encryption scheme. Technical report, National Institute of Standards and Technology, 2017. available at https://lima-pq.github.io/files/lima-pq.pdf.
32. F. Standaert, N. Veyrat-Charvillon, E. Oswald, B. Gierlichs, M. Medwed, M. Kasper, and S. Mangard. The World Is Not Enough: Another Look on Second-Order DPA. In *ASIACRYPT*. Springer, 2010.
33. Y. Won and D. Han. Efficient conversion method from arithmetic to boolean masking in constrained devices. In *COSADE*. Springer, 2017.

A Further Algorithms

Algorithm 16 `SecAddModp` [6]

Input: $\mathbf{x} = (x_i)_{1 \le i \le n} \in \mathbb{F}_{2^k}$, $\mathbf{y} = (y_i)_{1 \le i \le n} \in \mathbb{F}_{2^k}$ such that $\bigoplus_i x_i = x$, $\bigoplus_i y_i = y$
Output: $\mathbf{z} = (z_i)_{1 \le i \le n} \in \mathbb{F}_{2^k}$ with $2^k > 2q$ such that $\bigoplus_i z_i = x + y \mod q$

$(p_i)_{1 \le i \le n} \leftarrow (2^k - p, 0, \ldots, 0)$
$\mathbf{s} \leftarrow \texttt{SecAdd}(\mathbf{x}, \mathbf{y})$
$\mathbf{s}' \leftarrow \texttt{SecAdd}(\mathbf{s}, \mathbf{p})$
$\mathbf{b} \leftarrow \mathbf{s} >> (k-1)$
$\mathbf{c} \leftarrow \texttt{RefreshXOR}(\mathbf{b}, 1)$
$\mathbf{z} \leftarrow \texttt{SecAnd}(\mathbf{s}, \tilde{\mathbf{c}})$
$\mathbf{c} \leftarrow \texttt{RefreshXOR}(\mathbf{b}, 1)$
$\mathbf{z} \leftarrow \mathbf{z} \oplus \texttt{SecAnd}(\mathbf{s}', \neg\tilde{\mathbf{c}})$

Algorithm 17 `SecAdd` [6]

Input: $\mathbf{x} = (x_i)_{1 \le i \le n} \in \mathbb{F}_{2^k}$, $\mathbf{y} = (y_i)_{1 \le i \le n} \in \mathbb{F}_{2^k}$ such that $\bigoplus_i x_i = x$, $\bigoplus_i y_i = y$
Output: $\mathbf{z} = (z_i)_{1 \le i \le n} \in \mathbb{F}_{2^k}$ such that $\bigoplus_i z_i = x + y \mod 2^k$

$\mathbf{p} \leftarrow \mathbf{x} \oplus \mathbf{y}$
$\mathbf{g} \leftarrow \texttt{SecAnd}(\mathbf{x}, \mathbf{y})$
for $j = 1$ to $W = \lceil \log_2(k-1) \rceil - 1$ **do**
  pow $\leftarrow 2^{j-1}$
  $\mathbf{a} \leftarrow \mathbf{g} << (\text{pow})$
  $\mathbf{a} \leftarrow \texttt{SecAnd}(\mathbf{a}, \mathbf{p})$
  $\mathbf{g} \leftarrow \mathbf{g} \oplus \mathbf{a}$
  $\mathbf{a}' \leftarrow \mathbf{p} << (\text{pow})$
  $\mathbf{a}' \leftarrow \texttt{RefreshXOR}(\mathbf{a}', k)$
  $\mathbf{p} \leftarrow \texttt{SecAnd}(\mathbf{p}, \mathbf{a}')$
end for
$\mathbf{a} \leftarrow \mathbf{g} << (2^W)$
$\mathbf{a} \leftarrow \texttt{SecAnd}(\mathbf{a}, \mathbf{p})$
$\mathbf{g} \leftarrow \mathbf{g} \oplus \mathbf{a}$
$\mathbf{z} \leftarrow \mathbf{x} \oplus \mathbf{y} \oplus (\mathbf{g} << 1)$

Algorithm 18 `SecAnd` [14]

Input: $\mathbf{x} = (x_i)_{1 \leq i \leq n} \in \mathbb{F}_{2^k}$, $\mathbf{y} = (y_i)_{1 \leq i \leq n} \in \mathbb{F}_{2^k}$ such that $\bigoplus_i x_i = x$, $\bigoplus_i y_i = y$
Output: $\mathbf{z} = (z_i)_{1 \leq i \leq n} \in \mathbb{F}_{2^k}$ such that $\bigoplus_i z_i = x \wedge y$

$\mathbf{z} \leftarrow \mathbf{x} \wedge \mathbf{y}$
for $i = 1$ to $n - 1$ **do**
  for $j = 1 + i$ to n **do**
    $r_{i,j} \xleftarrow{\$} \mathbb{F}_{2^k}$
    $r_{j,i} \leftarrow (x_i \wedge y_j) \oplus r_{i,j}$
    $r_{j,i} \leftarrow r_{j,i} \oplus (x_j \wedge y_i)$
    $z_i \leftarrow z_i \oplus r_{i,j}$
    $z_j \leftarrow z_j \oplus r_{j,i}$
  end for
end for

Algorithm 19 `FullXOR` [14]

Input: $\mathbf{x} = (x_i)_{1 \leq i \leq n} \in \mathbb{F}_{2^k}$ such that $\bigoplus_i x_i = x$
Output: x

$\mathbf{y} \leftarrow$ `FullRefreshXOR`$(\mathbf{x}, k)$
$x \leftarrow y_1$
for $i = 2$ to n **do**
  $x \leftarrow x \oplus y_i$
end for

Algorithm 20 `RefreshXOR` [6]

Input: $\mathbf{x} = (x_i)_{1 \leq i \leq n} \in \mathbb{F}_{2^{k_1}}$ such that $\bigoplus_i x_i = x$, bit size k_2
Output: $\mathbf{y} = (y_i)_{1 \leq i \leq n} \in \mathbb{F}_{2^{k_2}}$ such that $\bigoplus_i y_i = x$

$\mathbf{y} \leftarrow \mathbf{x}$
for $i = 1$ to $n - 1$ **do**
  for $j = 1 + i$ to n **do**
    $r \xleftarrow{\$} \mathbb{F}_{2^{k_2}}$
    $y_i \leftarrow y_i \oplus r$
    $y_j \leftarrow y_j \oplus r$
  end for
end for

Algorithm 21 `FullRefreshXOR` [6]

Input: $\mathbf{x} = (x_i)_{1 \leq i \leq n} \in \mathbb{F}_{2^{k_1}}$ such that $\bigoplus_i x_i = x$, bit size k_2
Output: $\mathbf{y} = (y_i)_{1 \leq i \leq n} \in \mathbb{F}_{2^{k_2}}$ such that $\bigoplus_i y_i = x$

$\mathbf{y} \leftarrow \mathbf{x}$
for $i = 1$ to n **do**
  for $j = 2$ to n **do**
    $r \xleftarrow{\$} \mathbb{F}_{2^{k_2}}$
    $y_1 \leftarrow y_1 \oplus r$
    $y_j \leftarrow y_j \oplus r$
  end for
end for

Algorithm 22 `SecMul` (based on `SecAnd`)

Input: $\mathbf{A} = (A_i)_{1 \le i \le n} \in \mathbb{F}_q$, $\mathbf{B} = (B_i)_{1 \le i \le n} \in \mathbb{F}_q$ such that $\sum_i A_i = x \mod q$, $\sum_i B_i \leftarrow y \mod q$
Output: $\mathbf{C} = (C_i)_{1 \le i \le n} \in \mathbb{F}_q$ such that $\sum_i C_i = x + y \mod q$

$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} \mod q$
for $i = 1$ to $n - 1$ **do**
  for $j = 1 + i$ to n **do**
    $R_{i,j} \xleftarrow{\$} \mathbb{F}_q$
    $R_{j,i} \leftarrow (A_i \cdot B_j) + R_{i,j} \mod q$
    $R_{j,i} \leftarrow R_{j,i} + (A_j \cdot B_i) \mod q$
    $C_i \leftarrow C_i - R_{i,j} \mod q$
    $C_j \leftarrow C_j + R_{j,i} \mod q$
  end for
end for

Algorithm 23 Masked Binomial Sampler [25]

Input: $\mathbf{x} = (x_i)_{1 \le i \le 2} \in \mathbb{F}_{2^\kappa}$, $\mathbf{y} = (y_i)_{1 \le i \le 2} \in \mathbb{F}_{2^\kappa}$ such that $\bigoplus_i x_i = x$, $\bigoplus_i y_i = y$
Output: $\mathbf{A} = (A_i)_{1 \le i \le 2} \in \mathbb{F}_q$ such that $\sum_i A_i = \mathtt{HW}(x) - \mathtt{HW}(y) \mod q$

$(A_i)_{1 \le i \le 2} \leftarrow 0$
for $j = 1$ to k **do**
  $A_1 \leftarrow A_1 + (x_1^{(j)} - y_1^{(j)})$
  $A_2 \leftarrow A_2 + (x_2^{(j)} - y_2^{(j)})$
  $(w_1, z_1, r) \xleftarrow{\$} \mathbb{F}_q^3$
  $w_2 \leftarrow x_1^{(j)} - w_1$
  $z_2 \leftarrow x_2^{(j)} - z_1$
  $A_1 = A_1 - 2((((r + w_1 z_1) + w_1 z_2) + w_2 z_1) + w_2 z_2)$
  $w_2 = y_1^{(j)} - w_1$
  $z_2 = y_2^{(j)} - z_1$
  $A_2 = A_2 + 2((((r + w_1 z_1) + w_1 z_2) + w_2 z_1) + w_2 z_2)$
end for

B Estimation: Number of Operations

In the following, we derive the number of operations for the Boolean-to-arithmetic conversion of [6] and our new conversions.

B.1 `SecBoolArithModp` (cubic)

– `SecAnd` [12]:

$$T_{\texttt{SecAnd}}(n) = \frac{7n^2}{2} + \frac{5n}{2}.$$

– `RefreshXOR`:

$$T_{\texttt{RefreshXOR}}(n) = \frac{3n(n-1)}{2}.$$

– `SecAdd`:

$$\begin{aligned} T_{\texttt{SecAdd}}(n,k) &= 6n + 2T_{\texttt{SecAnd}}(n) + (\lceil \log_2(k-1) \rceil - 1) \cdot (3n + 2T_{\texttt{SecAnd}}(n) + T_{\texttt{RefreshXOR}}(n)) \\ &= 11n + 7n^2 + \frac{n(\lceil \log_2(k-1) \rceil - 1)(17n+13)}{2}. \end{aligned}$$

– `SecAddModp`:

$$\begin{aligned} T_{\texttt{SecAddModp}}(n,k) &= 2T_{\texttt{SecAdd}}(n,k) + 2T_{\texttt{RefreshXOR}}(n) + 2T_{\texttt{SecAnd}}(n) + 2n \\ &= 29n + 2(\lceil \log_2(k-1) \rceil - 1)(8n + \frac{3n(n-1)}{2} + 7n^2) + 3n(n-1) + 21n^2. \end{aligned}$$

– `SecArithBoolModp` (cubic):

$$\begin{aligned} T_{\texttt{SecA2BCube}}(n,k) &= n \cdot (T_{\texttt{RefreshXOR}}(n) + T_{\texttt{SecAddModp}}(n,k)) \\ &= n(29n + 2(\lceil \log_2(k-1) \rceil - 1)(8n + \frac{3n(n-1)}{2} + 7n^2) + \frac{9n(n-1)}{2} + 21n^2). \end{aligned}$$

– `SecBoolArithModp` (cubic):

$$\begin{aligned} & T_{\texttt{SecB2ACube}}(n,k) \\ &= T_{\texttt{SecA2BCube}}(n,k) + T_{\texttt{SecAddModp}}(n,k) + 3n^2 - 3 \\ &= 13n + 13n\lceil \log_2(k-1) \rceil + \frac{43n^2}{2} + \frac{17n^3}{2} + 30n^2\lceil \log_2(k-1) \rceil + 17n^3\lceil \log_2(k-1) \rceil - 3. \end{aligned}$$

B.2 `SecBoolArithModp` (quadratic)

– `SecArithBoolModp` (quadratic):

$$\begin{aligned} & T_{\texttt{SecA2BQuad}}(n,k) \\ &= T_{\texttt{SecA2BQuad}}(\lfloor n/2 \rfloor, k) + T_{\texttt{SecA2BQuad}}(\lceil n/2 \rceil, k) + 2T_{\texttt{RefreshXOR}}(n) + T_{\texttt{SecAddModp}}(n,k) \\ &= 27 \cdot 2^{2m+1} - 27 \cdot 2^{m+1} + 23m \cdot 2^m + 2^m(2\lceil \log_2(k-1) \rceil - 2)(\frac{13m}{2} + 12 \cdot 2^m - 17). \end{aligned}$$

– SecBoolArithModp (quadratic):

$$\begin{aligned}
&T_{\texttt{SecB2AQuad}}(n,k)\\
&= T_{\texttt{SecA2BQuad}}(n,k) + T_{\texttt{SecAddModp}}(n,k) + 3n^2 - 3\\
&= 29n - 27\cdot 2^{m+1} + 27\cdot 2^{2m+1} + 23m\cdot 2^m + 3n(n-1) + 24n^2\\
&+ 2^m(2\lceil\log_2(k-1)\rceil - 2)(\frac{13m}{2} + 17\cdot 2^m - 17) + n(\lceil\log_2(k-1)\rceil - 1)(17n+13) - 3.
\end{aligned}$$

B.3 SecB2A$_q$

– RefreshADD:

$$T_{\texttt{RefreshADD}}(n) = \frac{3n(n-1)}{2}.$$

– B2A$^{(n)}_{q-Bit}$:

$$T_{\texttt{B2AqBit}}(n) = 6n - 3.$$

– SecB2A$_{q-Bit}$:

$$\begin{aligned}
T_{\texttt{SecB2AqBit}}(n) &= T_{\texttt{RefreshADD}}(n) + \sum_{j=2}^{n} T_{\texttt{B2AqBit}}(j)\\
&= \frac{9n^2}{2} - \frac{3n}{2} - 3.
\end{aligned}$$

– SecB2A$_q$:

$$\begin{aligned}
T_{\texttt{SecB2Aq}}(n,k) &= 2n + T_{\texttt{SecB2AqBit}}(n) + \sum_{j=2}^{k} \left(T_{\texttt{SecB2AqBit}}(n) + 4n\right)\\
&= \frac{9kn^2}{2} + \frac{5kn}{2} - 2n - 3k.
\end{aligned}$$

C Estimation: Randomness Complexity

In the following, we derive the randomness complexity (i.e., number of required random bits) for the Boolean-to-arithmetic conversion of [6] and our new algorithms. We denote the bit size of the input encoding as k_1 and approximate the bit size of the prime as $k_2 = \lceil\log_2 q\rceil$. Additionally, we estimate the number of RNG calls for the conversion of [14].

C.1 SecBoolArithModp (cubic)

– SecAnd [12]:

$$R_{\texttt{SecAnd}}(n,k_1) = k_1 \cdot \frac{n^2-n}{2}.$$

– RefreshXOR:

$$R_{\texttt{RefreshXOR}}(n,k_1) = k_1 \cdot \frac{n^2-n}{2}.$$

– `FullRefreshXOR`:

$$R_{\texttt{FullRefreshXOR}}(n, k_1) = k_1 \cdot n \cdot (n-1).$$

– `SecAdd`:

$$\begin{aligned} R_{\texttt{SecAdd}}(n, k_1) &= 2R_{\texttt{SecAnd}}(n, k_1) + (\lceil \log_2(k_1 - 1) \rceil - 1) \cdot (2R_{\texttt{SecAnd}}(n, k_1) + R_{\texttt{RefreshXOR}}(n, k_1)) \\ &= \frac{k_1 n (3\lceil \log_2(k_1 - 1) \rceil - 1)(n-1)}{2}. \end{aligned}$$

– `SecAddModp`:

$$\begin{aligned} R_{\texttt{SecAddModp}}(n, k_1) &= 2R_{\texttt{SecAdd}}(n, k_1) + 2R_{\texttt{RefreshXOR}}(n, k_1) + 2R_{\texttt{SecAnd}}(n, k_1) \\ &= \frac{k_1 n (3\lceil \log_2(k_1 - 1) \rceil + 1)(n-1)}{.} \end{aligned}$$

– `SecArithBoolModp` (cubic):

$$\begin{aligned} R_{\texttt{SecA2BCube}}(n, k_1) &= n \cdot (R_{\texttt{RefreshXOR}}(n, k_1) + R_{\texttt{SecAddModp}}(n, k_1)) \\ &= \frac{3k_1 n^2 (2\lceil \log_2(k_1 - 1) \rceil + 1)(n-1)}{2}. \end{aligned}$$

– `SecBoolArithModp` (cubic):

$$\begin{aligned} & R_{\texttt{SecB2ACube}}(n, k_1) \\ &= k_2 \cdot (n-1) + R_{\texttt{SecA2BCube}}(n, k_1) + R_{\texttt{SecAddModp}}(n, k_1) + R_{\texttt{FullRefreshXOR}}(n, k_1) \\ &= \frac{(n-1)(2k_2 + 4k_1 n + 3k_1 n^2 + 6k_1 n^2 \lceil \log_2(k_1 - 1) \rceil + 6k_1 n \lceil \log_2(k_1 - 1) \rceil)}{2}. \end{aligned}$$

C.2 `SecBoolArithModp` (quadratic)

– `SecArithBoolModp` (quadratic):

$$R_{\texttt{SecA2BQuad}}(n, k_1) = R_{\texttt{SecA2BQuad}}(\lfloor n/2 \rfloor, k_1) + R_{\texttt{SecA2BQuad}}(\lceil n/2 \rceil, k_1) + 2R_{\texttt{RefreshXOR}}(n, k_1) + R_{\texttt{SecAddModp}}(n, k-1).$$

If $n = 2^m$, then

$$\begin{aligned} R_{\texttt{SecA2BQuad}}(n, k_1) &= \sum_{j=1}^{m} (2^{m-j} T_{\texttt{RefreshXOR}}(2^j, k_1) + T_{\texttt{SecAddModp}}(2^j, k-1)) \\ &= 27 \cdot 2^{2m+1} - 27 \cdot 2^{m+1} + 23m \cdot 2^m + 2^m (2\lceil \log_2(k-1) \rceil - 2)(\frac{13m}{2} + 17 \cdot 2^m - 17). \end{aligned}$$

– `SecBoolArithModp` (quadratic):

$$\begin{aligned} & R_{\texttt{SecB2AQuad}}(n, k_1) \\ &= k_2 \cdot (n-1) + R_{\texttt{SecA2BQuad}}(n, k_1) + R_{\texttt{SecAddModp}}(n, k_1) + R_{\texttt{FullRefreshXOR}}(n, k_1) \\ &= 27 \cdot 2^{2m+1} - 27 \cdot 2^{m+1} + k_2(n-1) + 23m \cdot 2^m + 5 \cdot k_1 \cdot n \cdot (n-1) \\ &+ 2^m (2\lceil \log_2(k-1) \rceil - 2)(\frac{13m}{2} + 17 \cdot 2^m - 17) + 3k_1 n (\lceil \log_2(k_1 - 1) \rceil - 1)(n-1). \end{aligned}$$

C.3 $\texttt{SecB2A}_q$

– $\texttt{RefreshADD}$:

$$R_{\texttt{RefreshADD}}(n) = k_2 \cdot \frac{n^2 - n}{2}.$$

– $\texttt{B2A}^{(n)}_{q-Bit}$:

$$R_{\texttt{B2AqBit}}(n) = k_2 \cdot (n-1).$$

– $\texttt{SecB2A}_{q-Bit}$:

$$\begin{aligned} R_{\texttt{SecB2AqBit}}(n) &= R_{\texttt{RefreshADD}}(n) + \sum_{j=2}^{n} R_{\texttt{B2AqBit}}(j) \\ &= k_2 \cdot (n^2 - n). \end{aligned}$$

– $\texttt{SecB2A}_q$:

$$\begin{aligned} R_{\texttt{SecB2Aq}}(n, k_1) &= R_{\texttt{SecB2AqBit}}(n) + \sum_{j=2}^{k_1} \left(R_{\texttt{SecB2AqBit}}(n) \right) \\ &= k_2 \cdot k_1 \cdot (n^2 - n). \end{aligned}$$

C.4 CGV14-B2A [14]

– $\texttt{SecAdd}$:

$$R_{\texttt{SecAddGoubin}}(n, k) = k \cdot R_{\texttt{SecAnd}}(n, k).$$

– $\texttt{CGV14} - \texttt{A2B}$:

$$\begin{aligned} R_{\texttt{CGV14}-\texttt{A2B}}(n, k) = n &+ R_{\texttt{CGV14}-\texttt{A2B}}(\lfloor n/2 \rfloor, k) + R_{\texttt{CGV14}-\texttt{A2B}}(\lceil n/2 \rceil, k) \\ &+ R_{\texttt{SecAddGoubin}}(n, k). \end{aligned}$$

– $\texttt{CGV14} - \texttt{B2A}$:

$$R_{\texttt{CGV14}-\texttt{B2A}}(n, k) = (n-1) + R_{\texttt{CGV14}-\texttt{A2B}}(n, k) + R_{\texttt{SecAddGoubin}}(n, k) + R_{\texttt{FullRefreshXOR}}(n, k).$$