Revisiting Higher-Order Masked Comparison for Lattice-Based Cryptography: Algorithms and Bit-Sliced Implementations

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

Abstract—Masked comparison is one of the most expensive operations in side-channel secure implementations of lattice-based post-quantum cryptography, especially for higher masking orders. First, we introduce two new masked comparison algorithms, which improve the arithmetic comparison of D'Anvers et al. (2021) and the hybrid comparison method of Coron et al. (2021) respectively. We then look into implementation-specific optimizations, and show that small specific adaptations can have a significant impact on the overall performance. Finally, we implement various state-of-the-art comparison algorithms and benchmark them on the same platform (ARM-Cortex M4) to allow a fair comparison between them. We improve on the arithmetic comparison of D'Anvers et al. with a factor $\approx 20\%$ by using Galois Field multiplications and the hybrid comparison of Coron et al. with a factor $\approx 25\%$ by streamlining the design. Our implementation-specific improvements allow a speedup of a straightforward comparison implementation of $\approx 33\%$. We discuss the differences between the various algorithms and provide the implementations and a testing framework to ease future research.

Index Terms—Post-quantum cryptography, lattice-based cryptography, side-channel protection, masking

1 Introduction

Current standards for public-key cryptography, such as RSA or ECC, are under threat of quantum computers. In response, the cryptographic community started work on replacement algorithms that are secure in the presence of large-scale quantum computers. Such quantum computer resisting algorithms are known under the term post-quantum cryptography. In 2016, the National Institute of Standards and Technology (NIST) started a standardization process to find a new post-quantum encryption and digital signature standard [3]. At the moment we are in the final stage of this process, with 4 encryption finalists and 3 signature finalists. Out of these finalists, 3 encryption schemes (Kyber [4], Saber [5] and NTRU [6]) and 2 signature schemes (Dilithium [7] and Falcon [8]) are from the family of lattice-based cryptographic schemes. In this paper we will specifically focus on lattice-based schemes.

When deploying the future standard, one has to take into account the possibility of side-channel attacks. Side-channel attacks are attacks that use information leakage as a result of computation, such as timing, power consumption or electromagnetic radiation. These leakages give an adversary extra information that could be used to break the cryptographic primitive with smaller effort compared to breaking the underlying mathematics.

Similar to other cryptographic families, lattice-based encryption schemes are vulnerable to side-channel attacks. This has been shown in [9], [10], [11] for timing attacks or in [12], [13], [14], [15], [16], [17], [18] for power consumption and electromagnetic radiation attacks. These attacks highlight the importance of protection mechanisms against side-channel attacks. In their latest update [19], NIST specifically highlights side-channel protection of post-quantum cryptographic primitives as an important challenge.

One popular method to protect against side-channel attacks is masking. Masking has been introduced by Chari et al. [20] and provides a framework to harden cryptographic implementations against side-channel leakage. The main idea of masking is to split sensitive values into S shares, so that an adversary that has access to at most $t < S$ shares does not learn any sensitive information. The parameter t denotes the order of the masking, and is typically equal to $S - 1$. The terminology around masking has been extended by Barthe et al. [21], introducing Non-Inference (NI) and Strong Non-Inference (SNI) to allow easier composition of masked building blocks, typically called gadgets.

Masked implementations of encryption standardization candidates were presented for Saber by Van Beirendonck et al. [22] for first order, and later by Coron et al. [2] for higher masking orders. A masked Kyber implementation for generic masking orders was introduced by Bos et al. [23]. Fritzmann et al. [24] optimized a masked implementation of Saber and Kyber using instruction set extensions.

• *The authors are with imec-COSIC KU Leuven, 3001, Leuven, Belgium. E-mail: {janpieter.danvers, michiel.vanbeirendonck, Ingrid.Verbauwhede}@esat.kuleuven.be.*

Manuscript received 15 March 2022; revised 8 June 2022; accepted 10 July 2022. Date of publication 8 August 2022; date of current version 13 January 2023.
This work was supported in part by CyberSecurity Research Flanders with reference number VR20192203, the Research Council KU Leuven (C16/15/058), the Horizon 2020 ERC Advanced Grant (101020005 Belfort) and SRC grant 2909.001. Michiel Van Beirendonck is funded by an FWO PhD fellowship strategic basic research. Jan-Pieter D'Anvers is funded by FWO (Research Foundation – Flanders) as junior post-doctoral fellow (contract number 133185/1238822N LV).
(Corresponding author: Jan-Pieter D'Anvers.)
Recommended for acceptance by Simha Sethumadhavan and Srini Devadas Guest Editors.
Digital Object Identifier no. 10.1109/TC.2022.3197074

For the signature candidates, Dilithium was masked by Migliore et al. [25].

Looking at the cost of the masking the various building blocks, one can see that there are different bottlenecks between masked and unmasked implementations. Unmasked implementations are typically dominated by the polynomial multiplication and the generation of the public matrix. For masked implementations, the most expensive building block is an equality check/comparison operation between the input ciphertext array and a re-encrypted ciphertext array. In this paper, we specifically look at different methods to securely implement this comparison. A complicating factor is that the input ciphertext is compressed for both Kyber and Saber, which will have an effect on which methods can be used in practice.

One observation that one can make is that there is a clear difference between first order and higher order masking, in that there are specific methods that can be used to speed-up first-order masking that do not scale to higher orders. For the comparison, one can use the first-order method of Oder et al. [26]. Their idea is to implement a check to see if a masked array is zero by hashing both shares separately and comparing only the hashed values in the end. A small change to their method, necessary for security has been discussed in [27]. The compression can be performed efficiently using table based A2B conversion [28], [29], specifically developed for first order masking.

For higher orders, several techniques have been developed, which follow the same pattern: first, a preprocessing on the arithmetically masked array, second, a conversion from the arithmetic to the Boolean masking domain, third, a postprocessing, and finally a comparison on the final Boolean masked values. The difference between the various methods lies in the preprocessing and postprocessing steps.

Barthe et al. [30] solved the masked comparison challenge by switching from the arithmetic masking domain to a Boolean masked representation, and then performing the comparison using masked bitwise operations. Bache et al. [31] showed a method to compress the number of array coefficients that needs to be compared by taking a random sum. Bhasin et al. [27] showed a security problem in this method, and adapted the idea to get around the security problems. The drawback of this method is that it only works for specific cases, i.e., prime moduli without compression of the ciphertext. D'Anvers et al. [1] later showed how to implement this method for both prime and power-of-two moduli with compression.

A different approach was taken by Bos et al. [23], who instead of compressing the masked ciphertext, leave it uncompressed and perform two masked checks to see if it is within the required range, i.e., a high- and low-end check. Removing the compression here comes at a cost of two (cheaper) checks per coefficient. Coron et al. [2] introduced several new ideas to more efficiently perform this range check.

1.1 Contributions

Our contributions are threefold: first we introduce an improved version of the comparison method of [1]. Instead of working with arithmetic multiplications modulo some big power-of-two, we propose to work in a Galois field, which saves us a conversion from the Boolean to the arithmetic masking domain and significantly reduces the cost of the comparison operation. We also develop a streamlined version of the Kyber-specific compression of Coron et al. [2]. Both our algorithms outperform the comparisons they are based on.

Second, we discuss specific implementation details such as bitslicing, and changing the Boolean representation after A2B conversion. We show that these implementation changes have a significant impact in reducing the cost of the algorithms.

In the third and final part of the paper we compare the state-of-the-art comparison methods. We implement several algorithms using the same underlying A2B conversion implementation and on the same target platform. We then perform the benchmarking on both Saber and Kyber. By doing this we aim to make an fair and practically useful comparison between the various comparison methods available. We will make our optimized implementations of these algorithms available at https://github.com/KULeuven-COSIC/Revisiting-Masked-Comparison.

2 Preliminaries

2.1 Notation

We denote with $\lfloor \cdot \rfloor$ flooring a number to the nearest lower integer, and with $\lceil \cdot \rfloor$ rounding, with ties rounded upwards. $\lfloor x \rceil_{q \to p}$ is a shorthand for modulus switching and rounding an input $x \in \mathbb{Z}_q$ to an output in $\mathbb{Z}_p$, i.e, $\lfloor x \rceil_{q \to p} = \lfloor \frac{p}{q} x \rceil$. Similarly, $\lceil \cdot \rfloor_{q \to p} = \left\lceil \frac{p}{q} \cdot \right\rfloor$. These operations are extended for vectors, polynomials or vectors of polynomials coefficient-wise. As we will see in Section 2.2, these operations are also extended for masked variables by applying them sharewise. Let $x \gg b$ denote bitwise shifting x to the right with b positions, which is equal to $floor(x/2^b)$. For an array or a polynomial x, denote with $x[i]$ the i^{th} coefficient of x.

Let $x \xleftarrow{\$} \chi$ denote sampling x according to a distribution χ, and let $x \xleftarrow{r} \chi$ denote a pseudorandom sampling based on a seed r. Let $\mathcal{U}(S)$ denote the uniform distribution over a set S.

2.2 Masking

Masking is a technique to protect implementations of cryptographic algorithms against side-channel attacks. The main idea is to split sensitive values into S shares so that an adversary only learns sensitive information if he has access to at least $t+1$ shares, where typically $t+1 = S$. For a sensitive value x we will denote that it is masked with $x^{(\cdot)}$, where $x = x^{(\cdot)}$. The notation $x^{(i)}$ specifically denotes the i^{th} share of the masked $x^{(\cdot)}$.

There are various methods to accomplish a sharing, and we will specifically utilize two: Boolean masking and arithmetic masking. In Boolean masking, a sensitive value is masked by XOR'ing it with uniformly random strings such that ${x_B}^{(\cdot)} = \bigoplus_{i=0}^{S-1} {x_B}^{(i)}$. For arithmetic masking, one chooses a masking modulus q and after which masking is performed by subtracting uniformly random strings such that ${x_A}^{(\cdot)} = \sum_{i=0}^{S-1} {x_A}^{(i)} \bmod q$. Arithmetic masking is typically used when performing arithmetic operations on the shares, as linear operations (addition, multiplication with a constant) are efficient under this masking. Boolean masking is

typically used when performing Boolean operations on data. For more information on masking we refer to [21], [32].

2.3 Lattice-Based Encryption

In this paper we will specifically look at the comparison operation that happens at the end of the decapsulation if compiled using the Fujisaki-Okamoto transformation. To give some context we introduce lattice-based encryption in this section, and will explain the Fujisaki-Okamoto (FO) transformation in the next section. We focus on a general algorithm that can be used to describe both Saber and Kyber.

Algorithm 1. PKE.KeyGen

1 $\mathrm{sd}_A \xleftarrow{\$} \{0,1\}^{256}$;
2 $A \xleftarrow{\mathrm{sd}_A} \mathcal{U}(R_q^{k\times k})$;
3 $(s,e) \xleftarrow{\$} \chi(R_q^{k\times 1}) \times \chi(R_q^{k\times 1})$;
4 $t \leftarrow \lfloor A \cdot s + e \rceil_{q\to q_2}$;
5 **return** $pk := (\mathrm{sd}_A, t), sk := s$;

Algorithm 2. PKE.Enc

Input: $pk = (\mathrm{sd}_A, t)$
Input: $m \in \mathcal{M}$
Input: $r \xleftarrow{\$} \{0,1\}^{256}$
1 $A \xleftarrow{\mathrm{sd}_A} \mathcal{U}(R_q^{k\times k})$;
2 $(r, e_1, e_2) \xleftarrow{r} \chi(R_q^{k\times 1}) \times \chi(R_q^{k\times 1}) \times \chi(R_q^{1\times 1})$;
3 $u \leftarrow A \cdot r + e_1$;
4 $v \leftarrow (\frac{q}{q_2} \cdot t) \cdot r + e_2 + \lceil \frac{q}{2} \rfloor \cdot m$;
5 $u_c \leftarrow \lfloor u \rceil_{q\to p}$;
6 $v_c \leftarrow \lfloor v \rceil_{q\to T}$;
7 **return** $c := (u_c, v_c)$;

Algorithm 3. PKE.Dec

Input: $sk = s$
Input: $c = (u_c, v_c)$
1 $u \leftarrow \lfloor u_c \rceil_{p\to q}$;
2 $v \leftarrow \lfloor v_c \rceil_{T\to q}$;
3 $m \leftarrow \lfloor v - s^T \cdot u \rceil_{q\to 2}$;
4 **return** m;

Algorithms 1, 2, and 3 depict a lattice-based encryption and decryption procedure. It works on vectors of ring elements R_q^k, with $R_q = \mathbb{Z}_q[X]/(X^n+1)$. In both Saber and Kyber, $n = 256$ and k has a value between 2 and 4 depending on the security level. The main difference between the two is that Saber works with a power of two modulus $q = 2^{13}$ while Kyber works with a prime $q = 3329$. Both algorithms compress the ciphertext from modulus q to lower moduli p and T for transmission of the ciphertext (and the public key in case of Saber). The values of p and T differ between the various versions of Kyber and Saber. Both are chosen to be powers-of-two, with $p = 2^{10}$ or 2^{11} while T has a smaller value typically around $T = 2^4$. The modulus q_2 is the public key compression modulus, which equals 2^{10} for Saber, but $q_2 = q$ for Kyber as it has no public key compression. The distribution $\chi(R_q^k)$ returns vectors with small coefficients that are drawn from a binomial distribution. For more information we refer to the original publications of Kyber [33] and Saber [34].

2.4 Fujisaki-Okamoto Transformation

The encryption scheme described in Section 2.3 only provides security from passive adversaries (IND-CPA). To achieve active security (IND-CCA) one can use a generic transformation such as a post-quantum version of the Fujisaki-Okamoto transformation [35], [36]. The main idea of such a transformation is to make the encryption deterministic based on a random seed, which is then transmitted as the message. During decapsulation, the ciphertext is decrypted into the random seed, which allows the ciphertext to be recomputed. The re-encrypted ciphertext is then compared with the input ciphertext and the procedure is aborted if both ciphertexts are not the same.

Algorithms 4, 5, and 6 give a more detailed look into the Fujisaki-Okamoto transformation, where the functions $\mathcal{G}$ and $\mathcal{H}$ are cryptographic hash functions and where KDF is a key derivation function. We will denote variables computed during re-encryption with an accent, to clearly distinguish from the input ciphertext.

Algorithm 4. KEM.KeyGen

1 $z \xleftarrow{\$} \{0,1\}^{256}$;
2 $(pk, sk') = \text{PKE.KeyGen}()$;
3 $sk = (sk' || pk || \mathcal{H}(pk) || z)$;
4 **return** pk, sk;

Algorithm 5. KEM.Encaps

Input: Public key of KEM pk
1 $m \xleftarrow{\$} \{0,1\}^{256}$;
2 $m \leftarrow \mathcal{H}(m)$;
3 $(\bar{K}, r) = \mathcal{G}(m || \mathcal{H}(pk))$;
4 $c = \text{PKE.Enc}(pk, m, r)$;
5 $K = \text{KDF}(\bar{K} || \mathcal{H}(c))$;
6 **return** c, K;

In this paper we will specifically look at the comparison in line 5 of Algorithm 6. The input ciphertext is a publicly known value, and thus not sensitive to leakage. The re-encrypted ciphertext is sensitive and should be masked. As an example, an attacker that could see (part of) the re-encrypted ciphertext could mount a chosen-ciphertext side-channel attack comparable to the attack described in [10], where side-channel information of the re-encrypted ciphertext can be used to determine if a ciphertext failed to decrypt.

This re-encrypted ciphertext has initially coefficients modulo q, but is compressed in lines 5 and 6 of Algorithm 2 before the comparison. The comparison operation we investigate in this paper includes the compression as an integral part of the algorithm. The re-encrypted ciphertext (before compression) is typically arithmetically masked. We will also ignore the ring structure of the ciphertext, and consider a polynomial in $R = \mathbb{Z}_q[X]/(X^n+1)$ as a vector in $\mathbb{Z}^n$ and a vector of polynomials in R^k as a vector in $\mathbb{Z}^{kn}$. This is reasonable as we don't use any property of the ring in the comparison operation.

3 Comparison Methods

On a high level, a comparison algorithm can be constructed by subtracting the input ciphertext from the re-encrypted ciphertext and performing a bitwise OR on all bits representing the result of the subtraction. However, in practice, there are some obstacles that need to be overcome to do this.

First, the re-encrypted ciphertext is typically arithmetically masked, which works well for the subtraction of both ciphertexts, but is ill-suited for the subsequent bitwise OR operation. Therefore, one typically wants to perform an arithmetic to Boolean (A2B) conversion on the data between the subtraction and the bitwise OR.

Second, the input and re-encrypted ciphertext are not in the same domain, as the input ciphertext is compressed. Moreover, the A2B conversion is not straightforward when working with prime moduli q.

We will first discuss these issues, and then give an overview of three state-of-the-art comparison techniques.

3.1 A2B and Compression

In this section we will discuss the subtraction of both ciphertexts and subsequent A2B conversion. We will first tackle the case of Saber, i.e., power-of-two q, and then talk about Kyber, i.e., prime q. While the power-of-two technique is relatively straightforward, the necessary adaptations to make this technique work for prime moduli were introduced by Fritzmann et al. [24].

Looking at the first ciphertext component u_c, we want to compute $\Delta u^{(\cdot)} = A2B(u_c'^{(\cdot)} - u_c)$ from the input ciphertext u_c and the re-encrypted uncompressed ciphertext $u'^{(\cdot)}$.

First we look at the case of power-of-two q, p. To efficiently compute $\Delta u^{(\cdot)}$ we want to compute the arithmetic operations in the arithmetic domain, while computing the flooring operation in the Boolean domain. To this end we rewrite the equation as:

$$A2B(u_c'^{(\cdot)} - u_c) = A2B\left(\left\lfloor \frac{p}{q} \cdot u'^{(\cdot)} \right\rfloor - u_c\right) \quad (1)$$

$$= A2B\left(\left\lfloor \frac{p}{q} \cdot u'^{(\cdot)} - u_c \right\rfloor\right) \quad (2)$$

$$= A2B\left(\left\lfloor \frac{p}{q} \cdot u'^{(\cdot)} - u_c + \frac{1}{2} \right\rfloor\right) \quad (3)$$

$$= A2B\left(\left\lfloor \frac{p}{q} \cdot \left(u'^{(\cdot)} - \frac{q}{p} \cdot u_c + \frac{q}{2p}\right)\right\rfloor\right) \quad (4)$$

$$= A2B\left(u'^{(\cdot)} - \frac{q}{p} \cdot u_c + \frac{q}{2p}\right) \gg \log_2\left(\frac{p}{q}\right). \quad (5)$$

For prime q, the step from Equations (4) to (5) is not straightforward for two reasons: first, $\log_2(\frac{p}{q})$ is not an integer, which would mean we have to shift with a fractional number which makes no sense, and second, the term in the A2B conversion has an infinite fractional representation.

Fritzmann et al. [24] noticed that only a limited precision is needed in the fractional representation. Given a number of bits needed for the required precision τ, they rewrite the expression above as:

$$A2B(u_c'^{(\cdot)} - u_c) = A2B\left(\left\lfloor \frac{p}{q} \cdot u'^{(\cdot)} - u_c + \frac{1}{2} \right\rfloor\right) \quad (6)$$

$$= A2B\left(\left\lfloor \frac{1}{2^\tau} \lfloor 2^\tau \left(\frac{p}{q} \cdot u'^{(\cdot)} - u_c + \frac{1}{2}\right) \rfloor \right\rfloor\right) \quad (7)$$

$$= A2B\left(\left\lfloor 2^\tau \cdot \left(\frac{p}{q} \cdot u'^{(\cdot)} - u_c + \frac{1}{2}\right) \right\rfloor\right) \gg \tau, \quad (8)$$

where we can get to Equation (7) if τ is large enough to avoid any error due to the flooring operation, as proven in [24]. Note that the flooring operation, the multiplications and the shift operation are performed independently on each share. In practice we need τ to be an integer bigger than $\log_2(S) - \log_2(\frac{\lceil q/2 \rceil}{q} - 0.5)$, which is 13 for Kyber if $S = 3$.

Algorithm 6. KEM.Decaps

Input: Ciphertext of KEM c
Input: Secret key of KEM sk
Extract $(sk'||pk||\mathcal{H}(pk)||z)$ from sk;
$m' = \text{PKE.Dec}(sk', c)$;
$(\bar{K}', r') = \mathcal{G}(m'||\mathcal{H}(pk))$;
$c' = \text{PKE.Enc}(pk, m', r')$;
if $c = c'$ **then**
  $K = \text{KDF}(\bar{K}'||\mathcal{H}(c))$;
else
  $K = \text{KDF}(z||\mathcal{H}(c))$;
end
return K;

Similar derivations can be performed to calculate $\Delta v^{(\cdot)} = A2B(v_c'^{(\cdot)} - v_c)$, where one only needs to replace u with v and the modulus p with T. To simplify the algorithms presented in the rest paper, we will define a function $\texttt{precalc}_{q\to p}(u'^{(\cdot)}, u_c)$ that calculates $\Delta u^{(\cdot)} = A2B(u_c'^{(\cdot)} - u_c)$ from $u'^{(\cdot)}$ and u_c as described above. Similarly, we define $\texttt{precalc}_{q\to T}(v'^{(\cdot)}, v_c)$ as the function that calculates $\Delta v^{(\cdot)} = A2B(v_c'^{(\cdot)} - v_c)$ from $v'^{(\cdot)}$ and v_c.

3.2 Simple Method

The simplest method to perform the comparison would be to perform the preprocessing as described above. This would result in a Boolean masked array of coefficients, of which should be checked if it equals zero. Then one can do the zero check by performing a bitwise masked OR operation, which can easily be obtained from a masked AND [37] operation combined with masked NOT operations, the latter operation only requiring a bitwise negation of one share. This description can be seen as a variant of the comparison method as used by Barthe et al. [30] to mask the GLP signature scheme. The resulting algorithm is given in Algorithm 7.

Algorithm 7. Simple

Input: Input ciphertext: u_c, v_c
Input: Re-encrypted ciphertext: $u'^{(\cdot)}, v'^{(\cdot)}$
$\Delta u^{(\cdot)} = \texttt{precalc}_{q\to p}(u'^{(\cdot)}, u_c)$;
$\Delta v^{(\cdot)} = \texttt{precalc}_{q\to T}(v'^{(\cdot)}, v_c)$;
$result = \texttt{OR}(\Delta u^{(\cdot)} \mid \Delta v^{(\cdot)})$;
return $result$;

3.3 Arithmetic Comparison

The masked OR operation in the simple approach needs to be calculated on kn coefficients of $\log_2(p)$ bits and n coefficients of $\log_2(T)$ bits. To reduce the number of masked OR operations, D'Anvers et al. [1] propose a technique to reduce the $k(n+1)$ coefficients that need to be checked into one (bigger) coefficient by summing them together. This technique is inspired by the random sum method of Bache et al. [31]. However, to avoid chosen ciphertext attacks where adaptations in one coefficient are offset with an inverse adaptation in another coefficient, all coefficients are first multiplied with a random number before summation. As this random number is the same for all shares of a coefficient, and due to distributivity (i.e, $\sum_i R \cdot x[i] = R \cdot \sum_i x[i]$), it can be proven that the resulting sum equals zero if all masked coefficients are zero.

One drawback of this method is that there is a small collision probability in which an incorrect input ciphertext is wrongly accepted. This collision probability equals 2^{-s} with s a security parameter related to the bit-size of R, and can not be influenced by an adversary. As such, it is not possible to increase the probability of obtaining a failure using for example failure boosting [38]. In many adversarial models, the adversary is limited in the number of queries Q he can perform, and the parameter s should be chosen such that an adversary can not reasonably find collisions, or: $2^s \geq Q$.

For a more detailed description of the algorithm and more in-depth security analysis we refer the interested reader to the original publication [1].

Algorithm 8. ARITHMETIC

Input: Input ciphertext: u_c, v_c
Input: Re-encrypted ciphertext: $u'^{(\cdot)}, v'^{(\cdot)}$

$\Delta u^{(\cdot)} = \texttt{precalc}_{q\to p}(u'^{(\cdot)}, u_c)$;
$\Delta v^{(\cdot)} = \texttt{precalc}_{q\to T}(v'^{(\cdot)}, v_c)$;
$b^{(\cdot)} = B2A_{p\cdot 2^{s-1}}(\Delta u^{(\cdot)}) \mid B2A_{p\cdot 2^{s-1}}(\Delta v^{(\cdot)})$;
$E^{(\cdot)} = 0$;
for $i = 1$ **to** $(k+1)n$ **do**
  $r \xleftarrow{\$} \mathcal{U}(\{0,1\}^s)$;
  $E^{(\cdot)} \mathrel{+}= r \cdot b^{(\cdot)}[i] \bmod p \cdot 2^{s-1}$;
end
$result = \texttt{OR}(E^{(\cdot)})$;
return $result$;

3.4 Hybrid Comparison

Coron et al. [2] introduce a hybrid method to perform the comparison. They first build several subfunctions and combine them into one comparison algorithm aimed at prime moduli q, as used in Kyber. These subfunctions include two new tests to check the zeroness of a polynomial and 'decompress-and-multiply', a method to process a masked ciphertext without performing compression, by instead decompressing the nonsensitive input ciphertext u_c. In this section we give an high-level overview of their comparison algorithm, which is given in Algorithm 9. For more details we refer to [2], [24] and [27].

The idea of the hybrid method is that the first and second ciphertext parts are processed using different approaches. The reason is that the first part of the ciphertext u only undergoes a small compression, while the second part v typically undergoes stronger compression. The decompress-and-multiply technique is only efficient for small compression, and is therefore only used for u, while v is processed in a more traditional approach. We will first look into the processing of the first part of the ciphertext u, then discuss the second part v and finally the postprocessing to combine both parts.

Algorithm 9. HYBRID METHOD

Input: Input ciphertext: u_c, v_c
Input: Re-encrypted ciphertext: $u'^{(\cdot)}, v'^{(\cdot)}$
// Adapted procedure for u
1 **for** $i = 1$ **to** kn **do**
2 $\Delta u_\times{}^{(\cdot)}[i] = 1$;
3 **for** $u[i][j]$ **in** $\texttt{Decompress}(u_c[i])$ **then**
4 $\Delta u_\times{}^{(\cdot)}[i] \mathrel{\times}= (u[i][j] - u'^{(\cdot)}[i])$;
5 **end**
6 **end**
// Normal procedure for v
7 $\Delta v^{(\cdot)} = \texttt{precalc}_{q\to T}(v'^{(\cdot)}, v_c)$;
8 $res_v^{B(\cdot)} = \texttt{OR}(\Delta v^{(\cdot)})$;
9 $res_v^{A(\cdot)} = B2A_q(res_v^{B(\cdot)})$;
10 $b^{(\cdot)} = \Delta u_\times{}^{(\cdot)} | res_v^{A(\cdot)}$;
// Compression
11 $E^{(\cdot)} = 0$;
12 **for** $j = 1$ **to** l_2 **do**
13 **for** $i = 1$ **to** $kn + 1$ **do**
14 $r \xleftarrow{\$} \mathcal{U}([0, q))$;
15 $E^{(\cdot)}[j] \mathrel{+}= r \cdot b^{(\cdot)}[i] \bmod q$;
16 **end**
17 **end**
// Final comparison
18 $result = \texttt{PolyZeroTest}(E^{(\cdot)})$;
19 **return** $result$;

To process the first part of the ciphertext, instead of compressing the masked coefficients of $u'^{(\cdot)}$, the public ciphertext u_c is decompressed. For each coefficient $u_c[i]$, this results in multiple possible decompressed values $u[i][j]$. For each of these possible decompressed values we subtract $u[i][j]$ from the masked recomputed ciphertext $u'^{(\cdot)}[i]$. The result of this subtraction should equal zero for one j (the one corresponding to the original decompressed value of $u_c[i]$). We then perform a masked multiply on all these values $\Delta u_\times{}^{(\cdot)} = \prod_j (u[i][j] - u'^{(\cdot)}[i])$, which results in a masked zero if and only if the decompressed ciphertext equals the recomputed ciphertext. These steps are given in line 1 to 6 of Algorithm 9.

Meanwhile, the second part of the ciphertext undergoes the simple comparison procedure from Section 3.2 in line 7-8 of Algorithm 9. This results in a Boolean masked bit representing the result of the comparison of v_c and $v'^{(\cdot)}$. This bit is then converted to arithmetic masking modulo q and added to the processed first ciphertext part.

The result of the above algorithm is a vector in $\mathbb{Z}_q^{nk+1}$ that needs to be equal to zero. This vector fulfills the condition to use the ReduceComparison technique of Bhasin et al. [27], which reduces the number of coefficients that need to be checked for zeroness. This reduction is performed in line

11-17. The algorithm is then finished by performing a zero check on the resulting polynomial.

As is the case in the ReduceComparison technique, this algorithm also has a probability of accepting invalid ciphertexts. This probability is upper bounded by q^{-l_2}, with q the modulus and l_2 the number of coefficients after compression. As before, the adversary can not increase this collision probability as it is entirely dependent on internal values of r.

4 New Comparison Algorithms

In the previous section we detailed three state-of-the art comparison algorithms. In this section we first improve on the arithmetic comparison technique, and then present a simplified version of the hybrid comparison technique. We will show in Section 6 that both techniques outperform their original algorithms.

4.1 Galois Field Compression

We first describe an improved version of the arithmetic compression method described in Section 3.3. The main difference between both algorithms is that the multiplication is changed from an arithmetic multiplication modulo $p \cdot 2^{s-1}$ to a multiplication in a Galois field with characteristic 2. The main advantage of this approach is that addition in a Galois Field of characteristic 2 is an XOR of the inputs, which works well on a Boolean representation. Therefore, multiplication and addition can be natively perfomed on Boolean masked shares, eliminating the need for the expensive $B2A_{p\cdot 2^{s-1}}$ conversion.

More precisely, for the multiplication operation we represent the inputs as polynomials with binary coefficients in $\mathbb{Z}_2[X]$ and perform a polynomial multiplication, after which a reduction modulo an irreducible polynomial f is executed. We represent this multiplication operation with the $\odot$ symbol.

It is possible to avoid the reduction step of this multiplication to reduce complexity of the algorithm. The downside is an increase in the number of coefficients that need to be processed in the OR operation. In section Section 3.3 we will see that the OR operation cost is negligible compared to the rest of the algorithm, and as such we implement the Galois field multiplication without the reduction. That is, as a multiplication between binary polynomials.

The Galois field comparison method can be found in Algorithm 10.

Algorithm 10. Galois Field

Input: Input ciphertext: u_c, v_c
Input: Re-encrypted ciphertext: $u'^{(\cdot)}, v'^{(\cdot)}$
$\Delta u^{(\cdot)} = \texttt{precalc}_{q\to p}(u'^{(\cdot)}, u_c)$;
$\Delta v^{(\cdot)} = \texttt{precalc}_{q\to T}(v'^{(\cdot)}, v_c)$;
$b^{(\cdot)} = \Delta u^{(\cdot)} \mid \Delta v^{(\cdot)}$;
$E^{(\cdot)} = 0$;
for $i = 1$ **to** $(k+1)n$ **do**
  $r \xleftarrow{\$} \mathcal{U}(\{0,1\}^s)$;
  $E^{(\cdot)} \oplus= r \odot b^{(\cdot)}[i]$;
end
$result = \texttt{OR}(E^{(\cdot)})$;
return $result$;

Theorem 1 (Correctness and Security of Algorithm 10). *The Galois field compression method of Algorithm 10 returns 1 upon input of a valid ciphertext $(u_c, v_c) = (\lceil u'^{(\cdot)} \rfloor_{q\leftarrow p}, (\lceil v'^{(\cdot)} \rfloor_{q\leftarrow T})$ and 0 with probability at least $1 - 2^{-s}$ if this condition is not fulfilled.*

Proof. This proof largely follows the analogous proof of [1], with the difference that some sums are replaced with XOR operations, and some arithmetic multiplications with GF multiplications. We first derive the value of $E^{(\cdot)}$ at the end of the algorithm.

$$E^{(\cdot)} = \bigoplus_{k=0}^{S-1} E^{(k)} \tag{9}$$

$$= \bigoplus_{k=0}^{S-1} \left(\bigoplus_{i=0}^{(k+1)n-1} (r_i \odot b^{(k)}[i]) \right) \tag{10}$$

$$= \bigoplus_{i=0}^{(k+1)n-1} \left(\bigoplus_{k=0}^{S-1} (r_i \odot b^{(k)}[i]) \right) \tag{11}$$

$$= \bigoplus_{i=0}^{(k+1)n-1} r_i \odot \left(\bigoplus_{k=0}^{S-1} b^{(k)}[i] \right) \tag{12}$$

$$= \begin{pmatrix} \bigoplus_{i=0}^{kn-1} r_i \odot \left(\bigoplus_{k=0}^{S-1} \Delta u^{(k)}[i] \right) \\ \oplus \bigoplus_{i=0}^{n-1} r_{kn+i} \odot \left(\bigoplus_{k=0}^{S-1} \Delta v^{(k)}[i] \right) \end{pmatrix}, \tag{13}$$

which by definition of $\texttt{precalc}_{q\to p}$ and $\texttt{precalc}_{q\to T}$ equals:

$$E^{(\cdot)} = \begin{pmatrix} \bigoplus_{i=0}^{kn-1} r_i \odot \left(\sum_{k=0}^{S-1} \left\lceil u'^{(k)} \right\rfloor_{q\to p} - u_c \right) \\ \oplus \bigoplus_{i=0}^{n-1} r_{kn+i} \odot \left(\sum_{k=0}^{S-1} \left\lceil v'^{(k)} \right\rfloor_{q\to T} - v_c \right) \end{pmatrix}, \tag{14}$$

We will further denote the terms $(\sum_{k=0}^{S-1} \left\lceil u'^{(k)} \right\rfloor_{q\to p} - u_c)$ and $(\sum_{k=0}^{S-1} \left\lceil v'^{(k)} \right\rfloor_{q\to T} - v_c)$ with β_i and γ_i respectively, for conciseness. This gives the following simplified expression for $E^{(\cdot)}$:

$$E^{(\cdot)} = \begin{pmatrix} \bigoplus_{i=0}^{kn-1} r_i \odot \beta_i \\ \oplus \bigoplus_{i=0}^{n-1} r_{kn+i} \odot \gamma_i \end{pmatrix}, \tag{15}$$

Correctness. If the input ciphertext (u_c, v_c) matches the recomputed compressed ciphertext $(\lceil u'^{(\cdot)} \rfloor_{q\to p}, \lceil v'^{(\cdot)} \rfloor_{q\to T})$, then all β_i and γ_i are zero and thus $E^{(\cdot)}$ is zero. This proves the first statement.

Security. If the input ciphertext does not match the recomputed ciphertext, there is at least one β_i or γ_i that does not equal zero. Without loss of generality, we will assume that β_0 is a nonzero coefficient. We can then separate this coefficient from the equation:

$$E^{(\cdot)} = \begin{pmatrix} r_0 \odot \beta_0 \\ \oplus \bigoplus_{i=1}^{kn-1} r_i \odot \beta_i \\ \oplus \bigoplus_{i=0}^{n-1} r_{kn+i} \odot \gamma_i \end{pmatrix}, \tag{16}$$

and simplify this equation into:

$$E^{(\cdot)} = (r_0 \odot X) \oplus Y, \tag{17}$$

by taking:

$$X = \beta_0 \quad \text{and} \quad Y = \left(\bigoplus_{i=1}^{kn-1} r_i \odot \beta_i\right) \oplus \left(\bigoplus_{i=0}^{n-1} r_{kn+i} \odot \gamma_i\right). \tag{18}$$

The adversary is tasked with finding a value X and Y so that $E^{(\cdot)} = (r_0 \odot X) \oplus Y = 0$. A necessary condition for this is that $(r_0 \odot X) \oplus Y \bmod f = 0$, with f an irreducible polynomial of degree 2^s. Which means that the condition can be rewritten as:

$$r_0 = Y \cdot X^{-1} \bmod f. \tag{19}$$

As r_0 is independent of the terms X and Y and is unknown to the adversary, the probability of finding a ciphertext such that this condition is fulfilled is limited to the guessing entropy of r_0, which equals 2^{-s}. This proof can be easily generalized if another value of β_i or γ_i is nonzero. □

Theorem 2 (t-SNI of Algorithm 10). *The Galois field compression method of Algorithm 10 is t-SNI secure.*

Proof. Due to the similarities with the arithmetic comparison method, we can rely on the proof from [1]. To support this claim we will highlight the differences between the Galois field method and the arithmetic comparison method, and then show that they do not change the security proof.

The t-SNI security proof of the arithmetic masking [1] divides the algorithm in 4 types of gadgets. Gadget G_0 and G_1 correspond to the preprocessing in the arithmetic comparison and are exactly the same in the Galois field method. Gadget G_4 corresponds to the final equality test in the arithmetic masking method which is the same as OR operation on line 9. The only difference between both algorithms is thus gadgets G_2 and G_3. Gadget G_2 is no longer needed in the Galois field method as we no longer need to perform B2A conversion.

This leaves us gadget G_3 which is different between both approaches. In the arithmetic masking, gadget G_3 computes an arithmetic random sum, while in the Galois field method, the gadget computes a random sum on binary polynomials. However, both approaches perform computations on each share separately. This property is what is used in the original proof [1] and as it also applies on the Galois field method, the original proof still holds for the Galois field method. □

4.2 Streamlined Hybrid

In this section we introduce an improved version of the hybrid compression technique from [2]. One disadvantage of the hybrid method is that it is complex in comparison to the other comparison methods, due to the various subfunctions used. The aim of the streamlined hybrid method is to simplify the implementation of the hybrid method, while also improving its efficiency.

One of the main speedups of the hybrid comparison method is due to the reduction of the number of coefficients that need to be converted from arithmetic to boolean masking in the A2B step. This is achieved by using the decompress-and-multiply technique from [2] and then perform the comparison reduction from [27]. These steps are only efficient for the first ciphertext part u. In the streamlined hybrid method we still use these techniques as they provide a significant speedup.

After these operations we revert to the standard simple procedure from Algorithm 7. As we will show in Section 5, the A2B and OR operations can be sped up significantly using implementation tricks. This means that while these operations theoretically don't scale as well as some alternatives in [2], they do outperform these functions in practical implementations. Due to their simplicity and efficiency we choose the postprocessing of the simple method over the postprocessing of the hybrid method. Specifically, we convert the remaining coefficients, from both the compressed ciphertext and the second ciphertext part v, to the Boolean domain and perform the OR operation on the Boolean masked coefficients. Algorithm 11 gives a high level overview of our streamlined hybrid method.

Algorithm 11. STREAMLINED HYBRID METHOD

Input: Input ciphertext: u_c, v_c
Input: Re-encrypted ciphertext: $u'^{(\cdot)}, v'^{(\cdot)}$
// Adapted procedure for u
1 $E^{(\cdot)} = 0$;
2 **for** $i = 0$ **to** $kn - 1$ **do**
3 $\quad \Delta u^{(\cdot)}[i] = 1$;
4 $\quad$ **for** $u[i][j]$ **in** Decompress$(u_c[i])$ **do**
5 $\quad\quad \Delta u^{(\cdot)}[i] \mathrel{\times}= (u[i][j] - u'^{(\cdot)}[i])$;
6 $\quad$ **end**
7 $\quad$ **for** $j = 1$ **to** l_2 **do**
8 $\quad\quad r \xleftarrow{\$} \mathcal{U}([0, q))$;
9 $\quad\quad E^{(\cdot)}[j] \mathrel{+}= r \cdot \Delta u^{(\cdot)} \bmod q$;
10 $\quad$ **end**
11 **end**
12 $E_B{}^{(\cdot)} = A2B(\lfloor \frac{2^{\lambda+\tau}}{q} \cdot E^{(\cdot)} + 2^{\tau} - 1 \bmod 2^{\lambda+\tau} \rfloor) \gg \tau$;
// Normal procedure for v
13 $\Delta v^{(\cdot)} = \texttt{precalc}_{q \to T}(v'^{(\cdot)}, v_c)$;
14 $res_v^{B^{(\cdot)}} = \texttt{OR}(\Delta v^{(\cdot)} | E_B{}^{(\cdot)})$;
15 **return** *result*;

Theorem 3 (Correctness and Security of Algorithm 11). *For* $\lambda + \log_2(2^{\tau}/(2^{\tau}-1)) > \log_2(q)$ *and* $\lambda + \tau \geq \log_2(q \cdot (S+1))$, *the streamlined hybrid compression method of Algorithm 11 returns 1 upon input of a valid ciphertext* $(u_c, v_c) = (\lceil u'^{(\cdot)} \rfloor_{q \leftarrow p}, (\lceil v'^{(\cdot)} \rfloor_{q \leftarrow T})$ *and 0 with probability at least* $1 - q^{-l_2}$ *if the above condition is not fulfilled.*

Proof. The ciphertext consists of two parts. The second part v is treated in the same way as the simple method, and thus shares the same characteristics: if $v_c = \lceil v'^{(\cdot)} \rfloor_{q \leftarrow T}$ then $\Delta v^{(\cdot)} = 0$, and if the ciphertexts do not match then $\Delta v^{(\cdot)} \neq 0$.

As such we will focus the proof on the value of $E_B{}^{(\cdot)}$. We will first consider a valid first ciphertext part $u_c = (\lceil u'^{(\cdot)} \rfloor_{q \leftarrow p})$, and then an invalid first ciphertext part where $u_c \neq \lceil u'^{(\cdot)} \rfloor_{q \leftarrow p}$.

Correctness. If $u_c = (\lceil u'^{(\cdot)} \rfloor_{q \leftarrow p})$, then by definition of `Decompress`, for each coefficient i one of the decompressed values $u[i][j]$ equals $u'^{(\cdot)}[i]$, which means $u[i][j] - u'^{(\cdot)}[i] = 0$ for this $u[i][j]$. This also implies that one term of the multiplication is zero for each coefficient and thus $\Delta u^{(\cdot)}$ is the zero vector. If $\Delta u^{(\cdot)}$ is a zero vector, then $E^{(\cdot)}$ is a sum of terms that are all zero, and thus $E^{(\cdot)}$ equals zero.

This leaves the A2B conversion of line 12 of Algorithm 11. While $E^{(\cdot)}$ is the zero vector, the individual shares are not necessarily equal to zero. However, similar to the derivation in [24], we can write:

$$\bigoplus_{k=0}^{S-1} {E_B}^{(k)} = \left\lfloor \frac{1}{2^\tau} \sum_{k=0}^{S-1} \left\lfloor \frac{2^{\lambda+\tau}}{q} \cdot E^{(k)} + 2^\tau - 1 \bmod 2^{\lambda+\tau} \right\rfloor \right\rfloor \quad (20)$$

$$= \left\lfloor \frac{1}{2^\tau} \sum_{k=0}^{S-1} \left\lfloor \frac{2^{\lambda+\tau}}{q} \cdot E^{(k)} \right\rfloor + \frac{2^\tau - 1}{2^\tau} \right\rfloor \bmod 2^\lambda \quad (21)$$

$$= \left\lfloor \frac{1}{2^\tau} \sum_{k=0}^{S-1} \left(\frac{2^{\lambda+\tau}}{q} \cdot E^{(k)} - e_k \right) + \frac{2^\tau - 1}{2^\tau} \right\rfloor \bmod 2^\lambda, \quad (22)$$

where e is a rounding error in [0,1). Now we can use the fact that $E^{(\cdot)}$ equals zero to simplify this expression to:

$$\bigoplus_{k=0}^{S-1} {E_B}^{(k)} = \left\lfloor \sum_{k=0}^{S-1} \left(\frac{-e_k}{2^\tau} \right) + \frac{2^\tau - 1}{2^\tau} \right\rfloor \bmod 2^\lambda, \quad (23)$$

At the upper bound, where all $e_k = 0$, we have:

$$\bigoplus_{k=0}^{S-1} {E_B}^{(k)} = \left\lfloor \frac{2^\tau - 1}{2^\tau} \right\rfloor = 0 \bmod 2^\lambda, \quad (24)$$

while at the lower bound, with $e_k = 1$, this gives

$$\bigoplus_{k=0}^{S-1} {E_B}^{(k)} = \left\lfloor \frac{-S}{2^\tau} + \frac{2^\tau - 1}{2^\tau} \right\rfloor = \left\lfloor \frac{2^\tau - 1 - S}{2^\tau} \right\rfloor \bmod 2^\lambda, \quad (25)$$

which also results in zero as long as $2^\tau - 1 > S$, or $\tau > \log_2(S+1)$.

We proved that if $u_c = (\lceil u'^{(\cdot)} \rfloor_{q \leftarrow p})$, then $\bigoplus_{k=0}^{S-1} {E_B}^{(\cdot)}$ equals zero, and we know from the proofs of the simple method that if $v_c = \lceil v'^{(\cdot)} \rfloor_{q \leftarrow T}$ then $\Delta v^{(\cdot)} = 0$. As the result is computed as an OR of these values, we proved that a valid ciphertext $(u_c, v_c) = (\lceil u'^{(\cdot)} \rfloor_{q \leftarrow p}, (\lceil v'^{(\cdot)} \rfloor_{q \leftarrow T})$ will return 0.

Security. If the ciphertext is invalid, $(u_c, v_c) \neq (\lceil u'^{(\cdot)} \rfloor_{q \leftarrow p}, (\lceil v'^{(\cdot)} \rfloor_{q \leftarrow T})$, at least one of the coefficients of u_c or v_c is invalid. As v_c is processed using exactly the same procedure as the simple comparison, we know that an invalid coefficient of v will propagate to a nonzero $\Delta v^{(\cdot)}$ and the result of the algorithm will be 1.

The second case is that u_c has at least one invalid coefficient, and without loss of generalization we will assume that this is the first coefficient. This is synonymous to the fact that none of the decompressed values $u[0][j]$ equals the recomputed ciphertext $u'^{(\cdot)}[0]$. As the multiplication to obtain $\Delta u^{(\cdot)}$ is calculated in the field $\mathbb{Z}_q$, and none of the terms are zero, we know that $\Delta u^{(\cdot)}[0] \neq 0$.

Following Theorem 1 of [1], a nonzero input $\Delta u^{(\cdot)}[0] \neq 0$ leads to a nonzero output $E^{(\cdot)}$ with probability $1 - q^{-l_2}$. This means that with this probability, at least one of the terms of $E^{(\cdot)}$ is nonzero.

In the end, our goal is to have at least one coefficient of ${E_B}^{(\cdot)}$ to be nonzero, which would result in a returned value of 1 due to the OR operation. If a coefficient of $E^{(\cdot)}$ is nonzero (and without loss of generalization we assume it is the first coefficient), we have:

$$\bigoplus_{k=0}^{S-1} {E_B}^{(\cdot)}[0] = \left\lfloor \frac{1}{2^\tau} \sum_{k=0}^{S-1} \left(\frac{2^{\lambda+\tau}}{q} \cdot E^{(k)}[0] - e_k \right) + \frac{2^\tau - 1}{2^\tau} \right\rfloor \bmod 2^\lambda \quad (26)$$

$$= \left\lfloor \left(\frac{2^\lambda}{q} \cdot E^{(\cdot)}[0] \right) + \frac{2^\tau - 1 - \sum_{k=0}^{S-1} e_k}{2^\tau} \right\rfloor \bmod 2^\lambda \quad (27)$$

with e_k a value in [0,1) as derived above. Remember that $E^{(\cdot)} \neq 0$. As such ${E_B}^{(\cdot)}$ can only occur due to over- or underflow. The two closest values are $E^{(\cdot)} = 1$ or $E^{(\cdot)} = q - 1$.

First we will look at the possibility of an underflow. The worst case scenario is that $E^{(\cdot)} = 1$ and all $e_k = 1$, which gives:

$$\bigoplus_{k=0}^{S-1} {E_B}^{(\cdot)}[0] = \left\lfloor \frac{2^\lambda}{q} + \frac{2^\tau - 1 - S}{2^\tau} \right\rfloor \bmod 2^\lambda. \quad (28)$$

This does not equal zero as long as $\frac{2^\lambda}{q} + \frac{2^\tau - 1 - S}{2^\tau} \geq 1$, or equivalently $\lambda + \tau \geq \log_2(q \cdot (S+1))$.

For the scenario of an overflow, we have a worst case scenario $E^{(\cdot)} = q - 1$ and $e_k = 0$.

$$\bigoplus_{k=0}^{S-1} {E_B}^{(\cdot)}[0] = \left\lfloor \left(\frac{2^\lambda}{q} \cdot (q-1) \right) + \frac{2^\tau - 1}{2^\tau} \right\rfloor \bmod 2^\lambda \quad (29)$$

which is not zero as long as $(\frac{2^\lambda}{q} \cdot (q-1)) + \frac{2^\tau - 1}{2^\tau} < 2^\lambda$, or $\frac{2^{\lambda+\tau}}{2^\tau - 1} > q$.

In conclusion, a non valid ciphertext will result in at least one nonzero coefficient in $res_v^{B^{(\cdot)}}$ with probability at least $1 - q^{-l_2}$, and thus a result of 1. □

Theorem 4 (t-SNI of Algorithm 11). *The streamlined hybrid method of Algorithm 11 is t-SNI secure.*

Proof. The streamlined hybrid comparison is a combination of the hybrid comparison and the simple comparison. As such we can use the t-SNI security of the gadgets of the hybrid comparison, as proved in [2], to prove t-SNI security of the streamlined hybrid comparison. In this proof we will divide the streamlined hybrid method into gadgets that correspond to gadgets already proven t-SNI secure for the hybrid comparison ([2], [27]) or the simple comparison ([1]).

The streamlined hybrid comparison can be split into 4 gadgets. Gadget G_1 to G_3 correspond to gadgets in the hybrid comparison: Gadget G_1 is the masked multiplication calculated in line 3 to 5 of Algorithm 11. These lines correspond to the secMultList algorithm of [2], and is proven t-SNI secure according to theorem 16 of that paper. Gadget G_2 represents line 7 to 10, which corresponds to the ReduceComparison technique from [27],

TABLE 1
Cycles Counts (x1000) Between the Different Optimization Levels for the Simple Comparison

	Non bitsliced	bitsliced	bitsliced + reinterpretation
Precalc	27	26	26
A2B	56,006 (x18.6)	3,015 (x1)	2,384 (x0.79)
OR	7,690 (x8.4)	909 (x1)	244 (x0.27)
Total	63,723 (x16.1)	3,950 (x1)	2,654 (x0.67)

proven t-SNI secure in Theorem 2 of that paper. Gadget G_3 is the A2B conversion, which should be chosen as a t-SNI secure A2B conversion. The rest of the algorithm, considered gadget G_4, proceeds exactly as the simple comparison method and has therefore the same security guarantees. □

5 Implementation Aspects

To obtain an efficient implementation of the comparison, it is not only important to search for an optimal algorithm, but also to consider implementation aspects. In this section we will first look at the importance of bitslicing the A2B conversion and the OR operation. We will show that bitslicing the A2B conversion gives a significant speedup and is essential to obtain an efficient implementation. Moreover, bitslicing is applied in the comparison implementation by D'Anvers et al. [1] but not in the implementations by Bos et al. [23] and Coron et al. [2]. This makes comparing these results difficult.

Bitslicing typically needs a pre- and postprocessing to correctly align the memory. In the second part of this section we will show that it is not always necessary to perform this postprocessing in our case, due to a reinterpretation of the outputs.

5.1 Bitslicing

Most comparison implementations use the A2B conversion and OR operation of Coron et al. [37]. A first observation is that this A2B conversion and the OR function involve almost exclusively bitwise operations. These operations can be bitsliced on a 32-bit CPU, where 32 inputs are taken as input and the bitwise operations are performed on all 32 inputs at the same time.

Such an implementation requires an pre- and postprocessing to rearrange the inputs in memory. This means that 32 input coefficients are taken in, and re-arranged in memory. In the preprocessing, the first bit of each coefficient is put in the first register, the second bit of each coefficient in the second register, and so on. Bitwise calculations are then performed on each register, digesting 32 coefficients at the same time. At the end of the A2B conversion, the postprocessing restores the output to the 32 coefficients of $\log_2(p)$ or $\log_2(T)$ bits.

Due to this pre- and post-A2B memory realignment, the speedup is not a full factor 32, but as can be seen from Table 1, bitslicing does have a significant impact on the efficiency of the algorithm.

TABLE 2
Cycle Cost (x1000) of All Subfunctions of a Bitsliced A2B

	Saber (2nd order)	Saber (3th order)
pre-A2B memory realignment	16 (26%)	22 (22%)
A2B	32 (52%)	58 (59%)
post-A2B memory realignment	14 (23%)	19 (19%)

5.2 Reinterpretation of the Boolean Masked Bits

After the A2B conversion, the main goal of the previous comparison techniques is to check if all coefficients of the polynomial or vector are zero. This is equivalent to stating that all Boolean masked bits need to be zero. As such, after A2B conversion one can represent the bits at will, for example by representing it as a vector with coefficients in $\mathbb{Z}_{2^{32}}$ instead of coefficients in $\mathbb{Z}_p$ and $\mathbb{Z}_T$.

There are multiple advantage to such a change in representation. For example, in the simple comparison method it is more efficient to perform the OR operation on coefficients of 32 bits due to bitslicing. It is also possible to use such a representation switch in the Galois field method, where line 4 to 8 of Algorithm 10 would work on a vector with coefficients in $\mathbb{Z}_{2^{32}}$ instead of $\mathbb{Z}_p$, which results in fewer coefficients that need to be processed, and thus lower execution time and less randomness consumption.

Moreover, the representation can be chosen in such a way to avoid any post-A2B memory alignment in the bitsliced A2B function. Remember that the coefficients at the output are aligned in memory as $\log_2(p)$ (or $\log_2(T)$ for v) registers of 32 bits. We can then reinterpret the registers to be $\log_2(p)$ coefficients in $\mathbb{Z}_{2^{32}}$. This means that we reinterpret the 32 coefficients of $\log_2(p)$ bits into $\log_2(p)$ coefficients of 32 bits. The reason this works is that if all 32 coefficients of $\log_2(p)$ bits are zero, then it must also be true that the $\log_2(p)$ coefficients of 32 bits are zero.

Avoiding the post-A2B memory alignment step has a significant impact on the total A2B cost, as can be seen in Table 2, where the postprocessing accounts for 19-23% of the full A2B procedure. Table 1 depicts the impact of a reinterpretation of the coefficients, as described in this section. Our method leads to a speedup of around 23% for the simple comparison. To the best of our knowledge, such a change of representation has not been presented or implemented in previous works.

6 Evaluation

We have implemented and benchmarked the various algorithms described in this paper. Benchmarking was performed on an STM32F407 board with an ARM-Cortex M4F using `arm-none-eabi-gcc` version 9.2.1 with -O3. The system clock was set to 24 Mhz and TRNG clock to 48 Mhz, in accordance to the popular benchmarking framework PQM4 [39]. Randomness is sampled from the on-chip TRNG and its sampling cost is included in the cycle counts.

TABLE 3
Results on Cortex M4

Order		Cycles 2	Cycles 3	Randomness (bytes) 2	Randomness (bytes) 3
Simple$^{\times}$	Saber	2.5M	3.9M	35K	72K
	Kyber	4.4M	7.1M	57K	118K
Simple† (new)	Saber	**1.6M**	**2.6M**	26K	53K
	Kyber	3.1M	5.3M	48K	100K
Arithmetic [1]	Saber	3.4M	6.5M	90K	205K
	Kyber	5.3M	9.7M	111K	251K
GF† (new)	Saber	2.7M	4.0M	26K	49K
	Kyber	4.2M	6.7M	47K	95K
Hybrid [2]	Kyber	3,3M	4.5M	80K	94K
Streamlined hybrid† (new)	Kyber	**2.5M**	**3.6M**	44K	62K

$\times$ *With bitslicing but without reinterpretation of the masked Boolean bits.*
† *With bitslicing and reinterpretation of the masked Boolean bits.*

The Simple, Galois field and streamlined hybrid methods can be optimized using the optimized bitslicing from Section 5.2, which is the case for the numbers in Table 3. The arithmetic method has a security parameters $s = 54$, while the Galois field method has an increased security of $s = 64$, which should be sufficient for cases where an adversary has a limit of 2^{64} queries. The reasoning for the specific s values is that for these values the implementation variables nicely align with 32 bit registers of our microprocessor. For the Galois Field method with reinterpretation of the Boolean masked bits we have 32 bit coefficients and 64 bit randomness r, which results in a 96 bit output $E^{(\cdot)}$. If one would want to increase s, one can select 32 bits coefficients with for example 96 or 128 bit randomness r which would result in respectively 128 and 160 bits $E^{(\cdot)}$ at the output. However, we believe such an increase of s is overkill in most scenarios as discussed in Section 3.3.

The Hybrid method is the original implementation of Coron et al. [2], adapted to allow execution on an ARM platform and with bitsliced A2B conversion to allow fair comparison. Both the hybrid and streamlined hybrid method have a collision probability under 2^{-128}. Note that it would be possible to increase this collision probability to around 2^{-64} without sacrificing security in many situations as discussed above. However, since the cycle cost between both options is minimal, we stick to a similar value as in [2].

We choose not to measure stack memory usage, as these comparison methods can be easily optimized for this if implemented in a full decapsulation operation. The idea of such optimization would be a greedy approach: Immediately after a coefficient of $u'^{(\cdot)}$ or $v'^{(\cdot)}$ is available, as much of the comparison is calculated as possible. Such a greedy approach would lead to a minimal stack usage compared to other functions in the decapsulation, as only the coefficient in current use and a limited number of intermediate variables need to be stored.

In the rest of this section we will compare the different methods to the simple method. We will start with the (streamlined) hybrid comparison, and then move to the arithmetic/GF method.

TABLE 4
Cycle Counts (x1000) for Subfunctions in the Simple, Streamlined Hybrid and Galois Field Comparison Methods for a 3th Order Implementation

	Kyber			Saber	
	Simple	Streamlined hybrid	GF	Simple	GF
Preprocessing	467	2,510	468	27	27
A2B	4,595	1,084	4,595	2,385	2,385
Postprocessing			1,619		1,619
OR	244	43	6	244	6
Total	5,306	3,637	6,688	2,656	4,037

6.1 Hybrid Comparison

The (streamlined) hybrid comparison essentially performs an additional preprocessing step in order to reduce the number of coefficients that need to be A2B converted. As can be seen in Table 4, the preprocessing becomes significantly more expensive, but it is compensated with a larger subsequent reduction in A2B cost. Notably, the hybrid comparison only works for prime moduli schemes, i.e., Kyber, and not for power-of-two q.

Our streamlined hybrid comparison, using the simple method to finish calculations, outperforms the hybrid comparison from [40]. While the initial calculations are the same, the final comparison is significantly faster in the streamlined hybrid comparison. An additional advantage is that the codebase of the streamlined hybrid comparison is less complex as it requires less functions and the complexity of the functions is lower.

6.2 Arithmetic/GF Comparison

Comparing the arithmetic and Galois field comparison methods weighs clearly in favour of the Galois field method. This is mostly due to the elimination of the expensive B2A conversion.

On the other hand, on the ARM-Cortex M4 there is native support for the arithmetic multiplication, while lacking support for the Galois field multiplication. This impacts the cost of the multiply-accumulate operation, which costs 197k cycles for the arithmetic operations, and 1,619k cycles for the Galois field multiply-accumulate (these numbers include randomness sampling, multiplication and addition to obtain $E^{(\cdot)}$).

In a scenario where the Galois field multiplication would have similar hardware support, for example in a hardware implementation or a hardware-software codesign, the Galois Field multiplication would slightly outperform the simple comparison method as can be derived from Table 4. This would come at a slight increase of implementation complexity.

It is possible to combine the streamlined hybrid method with the Galois field method. The streamlined hybrid method focusses on reducing the preprocessing cost, while the Galois field method focusses on the postprocessing cost. It is therefore straightforward to combine both methods, in which the output of the A2B conversion would serve as the interface between both methods.

7 Conclusion

The state-of-the art higher-order masked comparison techniques can be generalized into a common framework which consists of a preprocessing, an A2B conversion, a postprocessing and a final OR operation. In the most simple case, preprocessing is kept to the bare minimum and no postprocessing is performed. Coron et al. [40] introduced a hybrid method, specifically aimed at prime moduli q, to reduce the A2B cost by performing additional preprocessing. We sped up this design with $\approx 25\%$ in our streamlined hybrid algorithm. D'Anvers et al. [1] introduced a technique to speed up the OR operation at an increased postprocessing cost. We improved this method with $\approx 20\%$ by replacing the arithmetic multiplication with a Galois field multiplication. While this method does not outperform the simple comparison method on a microprocessor platform due to the lack of hardware support for the Galois field multiplication, it might be interesting to compare both methods on other platforms where support for the multiplication can be build in.

We also looked into implementation optimizations. We reiterated the importance of bitslicing, and showed that additional speedups are possible when reinterpreting the output of the Boolean masked bits output from the A2B conversion. The latter optimization simplifies our codebase and reduces the cycle count of the simple method with $\approx 33\%$.

Our comparison was performed on an ARM-Cortex M4 microprocessor. Interesting future work could be to make a similar comparison on other platforms, where one can add hardware support for the masked A2B and OR operations, or the multiplications needed in the hybrid and Galois field compression methods.

References

[1] J.-P. D'Anvers, D. Heinz, P. Pessl, M. van Beirendonck, and I. Verbauwhede, "Higher-order masked ciphertext comparison for lattice-based cryptography," *IACR Cryptol. ePrint Arch.*, 2021, Art. no. 1422. [Online]. Available: https://ia.cr/2021/1422

[2] J.-S. Coron, F. Gérard, S. Montoya, and R. Zeitoun, "High-order polynomial comparison and masking lattice-based encryption," *IACR Cryptol. ePrint Arch.*, 2021, Art. no. 1615. [Online]. Available: https://ia.cr/2021/1615

[3] NIST Computer Security Division, "Post-quantum cryptography standardization," 2016. [Online]. Available: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography

[4] P. Schwabe et al., "CRYSTALS-KYBER," National Institute of Standards and Technology, 2020. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions

[5] J.-P. D'Anvers et al., "SABER," National Institute of Standards and Technology, 2020. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions

[6] C. Chen et al., "NTRU," National Institute of Standards and Technology, 2020. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions

[7] V. Lyubashevsky et al., "Crystals-Dilithium," National Institute of Standards and Technology, 2020. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions

[8] T. Prest et al., "FALCON," National Institute of Standards and Technology, 2020. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions

[9] J. H. Silverman and W. Whyte, "Timing attacks on NTRUEncrypt via variation in the number of hash calls," in *Proc. Cryptographers' Track RSA Conf.*, 2007, pp. 208–224.

[10] J.-P. D'Anvers, M. Tiepelt, F. Vercauteren, and I. Verbauwhede, "Timing attacks on error correcting codes in post-quantum schemes," in *Proc. ACM Workshop Theory Implementation Secur. Workshop*, 2019, pp. 2–9.

[11] Q. Guo, T. Johansson, and A. Nilsson, "A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM," in *Proc. Annu. Int. Cryptol. Conf.*, 2020, pp. 359–386.

[12] A. C. Atici, L. Batina, B. Gierlichs, and I. M. R. Verbauwhede, "Power analysis on NTRU implementations for RFIDs: First results," in *Proc. 4th Workshop RFID Secur.*, 2008, pp. 128–139.

[13] A. Wang, X. Zheng, andZ. Wang, "Power analysis attacks and countermeasures on NTRU-based wireless body area networks," *KSII Trans. Internet Inf. Syst.*, vol. 7, no. 5, pp. 1094–1107, 2013.

[14] R. Primas, P. Pessl, and S. Mangard, "Single-trace side-channel attacks on masked lattice-based encryption," in *Proc. Int. Conf. Cryptographic Hardware Embedded Syst.*, 2017, pp. 513–533.

[15] D. Amiet, A. Curiger, L. Leuenberger, and P. Zbinden, "Defeating NewHope with a single trace," in *Proc. 11th Int. Conf. Post-Quantum Cryptography*, 2020, pp. 189–205.

[16] P. Ravi, S. S. Roy, A. Chattopadhyay, and S. Bhasin, "Generic side-channel attacks on CCA-secure lattice-based PKE and KEMs," *IACR IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2020, no. 3, pp. 307–335, 2020. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8592

[17] Z. Xu, O. Pemberton, S. S. Roy, and D. Oswald, "Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of Kyber," *IACR Cryptol. ePrint Arch.*, 2020, Art. no. 912. [Online]. Available: https://eprint.iacr.org/2020/912

[18] R. Ueno, K. Xagawa, Y. Tanaka, A. Ito, J. Takahashi, and N. Homma, "Curse of re-encryption: A generic power/EM analysis on post-quantum KEMs," *IACR Cryptol. ePrint Arch.*, 2021, Art. no. 849. [Online]. Available: https://ia.cr/2021/849

[19] G. Alagic et al., "Status report on the second round of the NIST post-quantum cryptography standardization process," 2020. [Online]. Available: https://csrc.nist.gov/publications/detail/nistir/8309/final

[20] S. Chari, C. S. Jutla, J. R. Rao, and P. Rohatgi, "Towards sound approaches to counteract power-analysis attacks," in *Proc. Annu. Int. Cryptol. Conf.*, 1999, pp. 398–412.

[21] G. Barthe, et al., "Strong non-interference and type-directed higher-order masking," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 116–129.

[22] M. V. Beirendonck, J. D'Anvers, A. Karmakar, J. Balasch, and I. Verbauwhede, "A side-channel-resistant implementation of SABER," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 17, no. 2, pp. 10:1–10:26, 2021.

[23] J. W. Bos, M. Gourjon, J. Renes, T. Schneider, and C. van Vredendaal, "Masking Kyber: First- and higher-order implementations," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2021, no. 4, pp. 173–214, 2021. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/9064

[24] T. Fritzmann et al., "Masked accelerators and instruction set extensions for post-quantum cryptography," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2022, no. 1, pp. 414–460, Nov. 2021. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/9303

[25] V. Migliore, B. Gérard, M. Tibouchi, and P.-A. Fouque, "Masking Dilithium: Efficient implementation and side-channel evaluation," in *Proc. Int. Conf. Appl. Cryptography Netw. Secur.*, 2019, pp. 344–362.

[26] T. Oder, T. Schneider, T. Pöppelmann, and T. Güneysu, "Practical CCA2-secure masked Ring-LWE implementations," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2018, no. 1, pp. 142–174, 2018. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/836

[27] S. Bhasin, J.-P. D'Anvers, D. Heinz, T. Pöppelmann, and M. Van Beirendonck, "Attacking and defending masked polynomial comparison," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2021, no. 3, pp. 334–359, 2021. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8977

[28] B. Debraize, "Efficient and provably secure methods for switching from arithmetic to Boolean masking," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2012, pp. 107–121.

[29] M. Van Beirendonck, J.-P. D'Anvers, and I. Verbauwhede, "Analysis and comparison of table-based arithmetic to boolean masking," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2021, no. 3, pp. 275–297, 2021. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8975

[30] G. Barthe, et al., "Masking the GLP lattice-based signature scheme at any order," in *Proc. Annu. Int. Conf. Theory Appl. Cryptographic Techn.*, 2018, pp. 354–384.

[31] F. Bache, C. Paglialonga, T. Oder, T. Schneider, and T. Güneysu, "High-speed masking for polynomial comparison in lattice-based KEMs," *IACR Trans. Cryptographic Hardware Embedded Syst.*, vol. 2020, no. 3, pp. 483–507, 2020. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8598
[32] Y. Ishai, A. Sahai, and D. Wagner, "Private circuits: Securing hardware against probing attacks," in *Proc. Annu. Int. Cryptol. Conf.*, 2003, pp. 463–481.
[33] J. Bos, et al., "CRYSTALS - Kyber: A CCA-secure module-lattice-based KEM," in *Proc. IEEE Eur. Symp. Secur. Privacy*, 2018, pp. 353–367.
[34] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren, "Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM," in *Proc. Int. Conf. Cryptol. Afr.*, 2018, pp. 282–305.
[35] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in *Proc. Annu. Int. Cryptol. Conf.*, 1999, pp. 537–554.
[36] D. Hofheinz, K. Hövelmanns, and E. Kiltz, "A modular analysis of the Fujisaki-Okamoto transformation," in *Proc. Theory Cryptography Conf.*, 2017, pp. 341–371.
[37] J.-S. Coron, J. Großschädl, and P. K. Vadnala, "Secure conversion between Boolean and arithmetic masking of any order," in *Proc. Int. Workshop Cryptographic Hardware Embedded Syst.*, 2014, pp. 188–205.
[38] J.-P. D'Anvers, Q. Guo, T. Johansson, A. Nilsson, F. Vercauteren, and I. Verbauwhede, "Decryption failure attacks on IND-CCA secure lattice-based schemes," in *Proc. IACR Int. Workshop Public Key Cryptography*, 2019, pp. 565–598.
[39] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, "PQM4: Post-quantum crypto library for the ARM Cortex-M4." [Online]. Available: https://github.com/mupq/pqm4
[40] J.-S. Coron, F. Gérard, S. Montoya, and R. Zeitoun, "High-order table-based conversion algorithms and masking lattice-based encryption," *IACR Cryptol. ePrint Arch.*, 2021, Art. no. 1314. [Online]. Available: https://ia.cr/2021/1314

Jan-Pieter D'Anvers received the MSc and PhD degrees in electrical engineering from KU Leuven, in 2015 and 2021, respectively. He is currently a postdoctoral researcher with the COSIC Research Group, KU Leuven, funded by an FWO (Research Foundation Flanders) postdoctoral grant. His research focuses on the design, security and side-channel security of post-quantum cryptography, and fully homomorphic encryption. He is co-designer of Saber, one of the final candidates in the NIST post-quantum standardization process.

Michiel Van Beirendonck received the BSc and MSc degrees in electrical engineering from KU Leuven, Belgium, in 2017 and 2019, respectively. He is currently working toward the PhD degree with the Research Group COSIC, KU Leuven. During his MSc studies, he spent one year with EPFL, Switzerland, as part of the SEMP exchange program. His research focuses broadly on the implementational challenges of lattice-based cryptography. He has worked extensively on side-channel attacks and countermeasures for post-quantum cryptosystems, as well as hardware acceleration of fully homomorphic encryption schemes.

Ingrid Verbauwhede (Fellow, IEEE) is a professor with the Research Group COSIC, Electrical Engineering Department, KU Leuven. At COSIC, she leads the secure embedded systems and hardware group. She was elected as member of the Royal Flemish Academy of Belgium for science and the arts in 2011. She received the IEEE 2017 Computer Society Technical Achievement Award. She is a recipient of two ERC Advanced Grants, one in 2016 and a second one in 2021. She is a pioneer in the field of efficient and secure implementations of cryptographic algorithms on many different platforms: ASIC, FPGA, embedded, cloud. With her research, she bridges the gaps between electronics, the mathematics of cryptography, and the security of trusted computing. Her group owns and operates an advanced electronic security evaluation lab. She is a fellow of the IACR.