Plover: Masking-Friendly Hash-and-Sign Lattice Signatures

Muhammed F. Esgin[1], Thomas Espitau[2], Guilhem Niot[2]*, Thomas Prest[2],
Amin Sakzad[1], and Ron Steinfeld[1]

[1] Monash University
muhammed.esgin@monash.edu, amin.sakzad@monash.edu,
ron.steinfeld@monash.edu
[2] PQShield
thomas@espitau.com, guilhem@gniot.fr, thomas.prest@pqshield.com

Abstract. We introduce a toolkit for transforming lattice-based hash-and-sign signature schemes into masking-friendly signatures secure in the t-probing model. Until now, efficiently masking lattice-based hash-and-sign schemes has been an open problem, with unsuccessful attempts such as Mitaka. A first breakthrough was made in 2023 with the NIST PQC submission Raccoon, although it was not formally proven.

Our main conceptual contribution is to realize that the same principles underlying Raccoon are very generic, and to find a systematic way to apply them within the hash-and-sign paradigm. Our main technical contribution is to formalize, prove, instantiate and implement a hash-and-sign scheme based on these techniques. Our toolkit includes noise flooding to mitigate statistical leaks, and an extended Strong Non-Interfering probing security (SNIu) property to handle masked gadgets with unshared inputs.

We showcase the efficiency of our techniques in a signature scheme, Plover-RLWE, based on (hint) Ring-LWE. It is the *first* lattice-based masked hash-and-sign scheme with quasi-linear complexity $O(d \log d)$ in the number of shares d. Our performances are competitive with the state-of-the-art masking-friendly signature, the Fiat-Shamir scheme Raccoon.

[*] Part of this project was conducted while Guilhem Niot was a student at EPFL and ENS Lyon, interning at PQShield.

Table of Contents

1 Introduction

Post-quantum cryptography is currently one of the most dynamic fields of cryptography, with numerous standardization processes launched in the last decade. The most publicized is arguably the NIST PQC standardization process, which recently selected [1] four schemes for standardization: Kyber [47], Dilithium [36], Falcon [45] and SPHINCS$^+$ [26].

Despite their strong mathematical foundations at an algorithmic level, recent years have witnessed the introduction of various side-channel attacks against the soon-to-be-standardized schemes: see for example the numerous power-analysis attacks against ML-DSA (Dilithium) [30,20,37,9], FN-DSA (Falcon) [31,24,49] or SLH-DSA (SPHINCS$^+$) [29]. This motivates us to consider exploring sound countermeasures allowing secure real-life implementations of mathematically well-founded cryptographic approaches.

Masking post-quantum schemes. In general, the most robust countermeasure against side-channel attacks is masking [27]. It consists of splitting sensitive information in d shares (concretely: $x = x_0 + \cdots + x_{d-1}$), and performing secure computation using MPC-based techniques. Masking offers a *trade-off*: while it increases computational efficiency by causing the running time to increase polynomially in d, it also exponentially escalates the cost of a side-channel attack with the number of shares d, see [16,38,28].

Unfortunately, masking incurs a significant computational overhead on the future NIST standards. For example, the lattice-based signature Dilithium relies on sampling elements in a small subset $S \subsetneq \mathbb{Z}_q$ of the native ring $\mathbb{Z}_q$, and testing membership to a second subset $S' \subsetneq \mathbb{Z}_q$. The best-known approaches for performing these operations in a masked setting rely on *mask conversions* [23]. These operations are extremely expensive, and despite several improvements in the last few years [10,13,14], still constitute the efficiency bottlenecks of existing masked implementations of Dilithium, see Coron et al. [15] and Azouaoui et al. [4], and of many other lattice-based schemes, see the works of Coron et al. on Kyber [13], and of Coron et al. on NTRU [14].

Falcon, based on the hash-and-sign paradigm, is even more challenging to mask. The main reason is the widespread use of floating-point arithmetic; even simple operations such as masked addition or multiplication are highly nontrivial to mask. Another reason is a reliance on discrete Gaussian distributions with secret centers and standard deviations, which also need to be masked. Even without considering masking, these traits make Falcon difficult to implement and to deploy on constrained devices.

More recent Hash-and-Sign schemes, such as Mitaka [18], Robin and Eagle [48], also share both of these undesirable traits. Mitaka proposed novel techniques in an attempt to make it efficiently maskable; however, Prest [44] showed that these techniques were insecure and exhibited a practical key-recovery attack in the t-probing model against Mitaka. As of today, it remains an open problem to build hash-and-sign lattice signatures that can be masked efficiently.

1.1 Our Solution

In this work, we describe a general toolkit for converting hash-and-sign schemes into their masking-friendly variants. The main idea is deceptively simple: instead of using trapdoor sampling to generate a signature that leaks no information about the secret key, using noise that is sufficiently large to hide the secret on its own. While similar ideas were described in the Fiat-Shamir setting by Raccoon [41], we show here that the underlying principles and techniques are much more generic. In our case, we replace the canonical choice of Gaussian distribution—which only depends on the (public) lattice and not on the short secret key—with sums of uniform distributions. This allows us to remove all the complications inherent to the sampler, as we now do not need a sampler more complicated than a uniform one. Then since all the remaining operations are linear in the underlying field, we can simply mask all the values in arithmetic form and follow the usual flow of the algorithm.

The security of the scheme in this approach now relies on the hint variant of the underlying problem (namely, Ring-LWE) as the correlation between the signature and the secret can be exploited when collecting sufficiently many signatures. To showcase the versatility of our toolkit, we propose two possible instantiations of our transform: starting from the recent Eagle proposal of [48], we construct a masking-friendly hash-and-sign signature, Plover, based on the hardness of Hint-RLWE. To provide a high-level view, we describe the transformation in Fig. 1a and Fig. 1b. Differences between the two blueprints are highlighted. Operations that need to be masked in the context of side channels are indicated with comments: Easy when standard fast techniques apply to mask, or Hard otherwise. We replace the two Gaussian samples (Eagle L. 3 and 6) by the noise flooding (Plover L. 7, the mask being generated by the gadget AddRepNoise at L.3) in masked form. The final signature $\mathbf{z}$ is eventually unmasked.

```
Eagle.Sign(sk, msg) → sig
In: A signing key sk, a message msg.
Out: A signature sig of msg under sk.
1: salt ← $\{0,1\}^{320}$
2: $\mathbf{u} := H(\mathsf{msg}, \mathsf{salt})$
3: $\mathbf{p} \leftarrow D_{\mathcal{R}^\ell, \sqrt{s^2\mathbf{I} - r^2\mathbf{T}\mathbf{T}^*}}$      ▷ Hard
4: $\mathbf{c} := \mathbf{u} - \mathbf{A} \cdot \mathbf{p}$      ▷ Easy
5: Decompose $\mathbf{c}$ as $\mathbf{c} = \beta \cdot \mathbf{c}_1 + \mathbf{c}_2$
6: $\mathbf{y} \leftarrow D_{\lfloor q/\beta \rceil \cdot \mathcal{R}^\ell + \mathbf{c}_1, r}$      ▷ Hard
7: $\mathbf{z} := \mathbf{p} + \mathbf{T} \cdot \mathbf{y}$      ▷ Easy
8: sig := (salt, $\mathbf{z}$)
9: return sig
```

(a) Blueprint for Eagle [48].

```
Plover.Sign(sk, msg) → sig
In: A signing key sk, a message msg.
Out: A signature sig of msg under sk.
1: salt ← $\{0,1\}^{2\kappa}$
2: $\mathbf{u} := H(\mathsf{msg}, \mathsf{salt}, \mathsf{vk})$
3: $[\![\mathbf{p}]\!] \leftarrow \mathsf{AddRepNoise}(\mathcal{R}_q^\ell, d, \mathcal{D}, \mathsf{rep})$
4: $\mathbf{c} := \mathbf{u} - \mathsf{Unmask}(\mathbf{A} \cdot [\![\mathbf{p}]\!])$
5: Decompose $\mathbf{c}$ as $\mathbf{c} = \beta \cdot \mathbf{c}_1 + \mathbf{c}_2$
6: [This step is removed]
7: $\mathbf{z} := \mathsf{Unmask}([\![\mathbf{p}]\!] + [\![\mathbf{T}]\!] \cdot \mathbf{c}_1)$
8: sig := (salt, $\mathbf{z}$)
9: return sig
```

(b) Blueprint for Plover.

Fig. 1: High-level comparison between Eagle [48] and our scheme Plover. In both schemes, the signing key is a pair of matrices sk $= (\mathbf{T}, \mathbf{A})$, the verification key is vk $= \mathbf{T}$, and we have $\mathbf{A} \cdot \mathbf{T} = \beta \cdot \mathbf{I}_k$.
The verification procedure is also identical: in each case, we check that $\mathbf{z}$ and $\mathbf{c}_2 := \mathbf{A} \cdot \mathbf{z} - \mathbf{u}$ are sufficiently short.

We also provide a similar approach using an NTRU-based signature. Our analyses reveal that the NTRU-based approach, at the cost of introducing a stronger assumption, sees its keygen becoming slower but signature and verification get faster. However, as the signature size is slightly bigger and the techniques are similar, we choose to present only the RLWE variant here and describe the NTRU one in the supplementary materials.

1.2 Technical Overview

The main ingredients we introduce in this toolkit are the following:

1. *Noise flooding.* The main tool is the so-called *noise flooding* introduced by Goldwasser et al. [22]: we "flood" the sensitive values with enough noise so that the statistical leak becomes marginal. In contrast, other hash-then-sign lattice signatures use trapdoor sampling make the output distribution statistically independent of the signing key. However, marginal does not mean nonexistent and we need to quantify this leakage.

   To achieve this in a *tight* manner, we leverage the recent reduction of Kim et al. [32], which transitions Hint-MLWE to MLWE, providing a solid understanding of the leakage. Noise flooding has recently proved useful in the NIST submission Raccoon [41] to analyze its leakage and optimize parameters. Here also, the tightness of this reduction allows to reduce the relative size of the noise while preserving security, compared to, e.g., using standard Rényi arguments.
2. *SNI with unmasked inputs.* To get a scheme which is *provably* secure in the t-probing model, we need to extend the usual definition of t Strong Non-Interfering (SNI) Gadgets to allow the attacker to know "for free" up to t *unshared inputs* of the gadgets (we call this extended property t-SNIu). This is somehow the "dual" of the (S)NI with public outputs notion (NIo) introduced by Barthe et al. [6].

   In particular, we formally show that the AddRepNoise gadget, introduced by Raccoon [41] to sample small secrets as a sum of small unshared inputs, satisfies our t-SNIu definition and hence enjoys t probing security. This fills a provable security gap left in [41], where the t-probing security of AddRepNoise was only argued informally. Our new model is also sufficient to handle the unmasking present in our signature proposal. We prove the security for the t-probing EUF-CMA notion borrowed from [6].
3. *Masked inversion.* As a natural byproduct of the NTRU-based instantiation, we propose a novel way to perform inversion in masked form. Our proposal combines the NTT representation with Montgomery's trick [40] to speed up masked inversion. It is to the best of our knowledge the first time Montgomery's trick has been used for masking lattice cryptography. Our technique offers an improved asymptotic complexity over previous proposals from [14, Section 4.3 and 5]. Due to page limitation, more details are given in Appendix A.2.

Advantages and limitations The first and main design principle of our toolkit is of course its amenability to masking. In effect, we can mask at order $d-1$ with an overhead of only $O(d \log d)$. This allows masking of Plover at high orders with a small impact on efficiency. High masking orders introduce a new efficiency bottleneck in memory consumption, due to the storage requirements for highly masked polynomials. Second, our proposal Plover relies on (variants of) lattice assumptions that are well-understood (NTRU, LWE), or at least are classically reducible from standard assumptions (Hint-LWE). We emphasize that the simplicity allowed in the design leads to implementation portability. In particular, our scheme enjoys good versatility in its parameter choices—allowing numerous tradeoffs between module sizes, noise, and modulus–enabling target development on various device types. For example, our error distributions can be based on sums of uniform distributions; this makes implementation straightforward across a wide range of platforms. Ultimately, since Plover is a hash-and-sign signature, it does not require masked implementations of symmetric cryptographic components, such as SHA-3/SHAKE. The number of distinct masking gadgets is relatively small, which results in simpler and easier-to-verify firmware and hardware.

As expected, our efficient masking approach comes at the cost of larger parameter sizes (mainly because of the large modulus required) compared to the regular design of hash-and-sign schemes using Gaussian distributions and very small modulus. Additionally, the security is now *query dependant*: as it is the case for Raccoon or most threshold schemes, we can only tolerate a certain number (NIST recommendation being 2^{64}) of queries to the signing oracle with the same private key.

2 Preliminaries

2.1 Notations

Sets, functions and distributions. For an integer $N > 0$, we note $[N] = \{0, \dots, N-1\}$. To denote the assign operation, we use $y := f(x)$ when f is a deterministic and $y \leftarrow f(x)$ when randomized. When S is a finite set, we note $\mathcal{U}(S)$ the uniform distribution over S, and shorthand $x \xleftarrow{\$} S$ for $x \leftarrow \mathcal{U}(S)$.

Given a distribution $\mathcal{D}$ of support included in an additive group $\mathbb{G}$, we note $[T] \cdot \mathcal{D}$ the convolution of T identical copies of $\mathcal{D}$. For $c \in \mathbb{G}$, we may also note $\mathcal{D} + c$ the translation of the support of $\mathcal{D}$ by c. Finally, the notation $\mathcal{P} \overset{s}{\sim} \mathcal{Q}$ indicates that the two distributions are statistically indistinguishable.

Linear algebra. Throughout the work, for a fixed power-of-two n, we note $\mathcal{K} = \mathbb{Q}[x]/(x^n + 1)$ and $\mathcal{R} = \mathbb{Z}[x]/(x^n + 1)$ the associated cyclotomic field and cyclotomic ring. We also note $\mathcal{R}_q = \mathcal{R}/(q\mathcal{R})$. Given $\mathbf{x} \in \mathcal{K}^\ell$, we abusively note $\|\mathbf{x}\|$ the Euclidean norm of the $(n\,\ell)$-dimensional vector of the coefficients of $\mathbf{x}$. By default, vectors are treated as *column* vectors unless specified otherwise.

Rounding. Let $\beta \in \mathbb{N}, \beta \geqslant 2$ be a power-of-two. Any integer $x \in \mathbb{Z}$ can be decomposed uniquely as $x = \beta \cdot x_1 + x_2$, where $x_2 \in \{-\beta/2, \dots, \beta/2 - 1\}$. In this case, $|x_1| \leqslant \left\lceil \frac{x}{\beta} \right\rceil$, where $\lceil \cdot \rfloor$ denote rounding up to the nearest integer. For odd q, we note $\mathsf{Decompose}_\beta : \mathbb{Z}_q \to \mathbb{Z} \times \mathbb{Z}$ the function which takes as input $x \in \mathbb{Z}_q$, takes its unique representative in $\bar{x} \in \{-(q-1)/2, \dots, (q-1)/2\}$, and decomposes $\bar{x} = \beta \cdot x_1 + x_2$ as described above and outputs (x_1, x_2). We extend $\mathsf{Decompose}_\beta$ to polynomials in $\mathbb{Z}_q[x]$, by applying the function to each of its coefficients. For $c \xleftarrow{\$} \mathbb{Z}_q$ and $(c_1, c_2) := \mathsf{Decompose}_\beta(c)$, we have $|c_1| \leqslant \left\lceil \frac{q-1}{2\beta} \right\rceil$, $\mathbb{E}[c_1] = 0$ and $\mathbb{E}[c_1^2] \leqslant \frac{M^2-1}{12}$ for $M = 2\left\lceil \frac{q-1}{2\beta} \right\rceil + 1$.

2.2 Distributions

Definition 1 (Discrete Gaussians). *Given a positive definite $\Sigma \in \mathbb{R}^{m\times m}$, we note $\rho_{\sqrt{\Sigma}}$ the Gaussian function defined over $\mathbb{R}^m$ as*

$$\rho_{\sqrt{\Sigma}}(\mathbf{x}) = \exp\left(-\frac{\mathbf{x}^t \cdot \Sigma^{-1} \cdot \mathbf{x}}{2}\right).$$

We may note $\rho_{\sqrt{\Sigma},\mathbf{c}}(\mathbf{x}) = \rho_{\sqrt{\Sigma}}(\mathbf{x}-\mathbf{c})$. When Σ is of the form $\sigma \cdot \mathbf{I}_m$, where $\sigma \in \mathcal{K}^{++}$ and $\mathbf{I}_m$ is the identity matrix, we note $\rho_{\sigma,\mathbf{c}}$ as shorthand for $\rho_{\sqrt{\Sigma},\mathbf{c}}$.

For any countable set $S \subset \mathcal{K}^m$, we note $\rho_{\sqrt{\Sigma},\mathbf{c}}(S) = \sum_{\mathbf{x}\in\mathcal{K}^m} \rho_{\sqrt{\Sigma},\mathbf{c}}(\mathbf{x})$ whenever this sum converges. Finally, when $\rho_{\sqrt{\Sigma},\mathbf{c}}(S)$ converges, the discrete Gaussian distribution $D_{S,\mathbf{c},\sqrt{\Sigma}}$ is defined over S by its probability distribution function:

$$D_{S,\sqrt{\Sigma},\mathbf{c}}(\mathbf{x}) = \frac{\rho_{\sqrt{\Sigma},\mathbf{c}}(\mathbf{x})}{\rho_{\sqrt{\Sigma},\mathbf{c}}(S)}. \tag{1}$$

Definition 2 (Sum of uniforms). *We note $SU(u,T) := [T]\cdot\mathcal{U}(\{-2^{u-1},\ldots,2^{u-1}-1\})$. In other words, $SU(u,T)$ is the distribution of the sum $X = \sum_{i\in[T]} X_i$, where each X_i is sampled uniformly in the set $\{-2^{u-1},\ldots,2^{u-1}-1\}$.*

2.3 Hardness Assumptions

In a will of unification and clarification, we choose to present the lattice problems used in this work in their Hint-variants, that is to say with some additional statistical information on the secret values. Of course, not adding any hint recovers the plain problems—here being RLWE, and NTRU in the appendix. The Hint-RLWE problem was introduced recently in [32] and reduces (in an almost dimension-preserving way) from RLWE.

Definition 3 (Hint-RLWE). *Let q, Q be integers, $\mathcal{D}_{\mathsf{sk}}, \mathcal{D}_{\mathsf{pert}}$ be probability distributions over $\mathcal{R}_q^2$, and $\mathcal{C}$ be a distribution over $\mathcal{R}_q$. The advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hint\text{-}RLWE}}(\kappa)$ of an adversary $\mathcal{A}$ against the Hint Ring Learning with Errors problem $\mathsf{Hint\text{-}RLWE}_{q,Q,\mathcal{D}_{\mathsf{sk}},\mathcal{D}_{\mathsf{pert}},\mathcal{C}}$ is defined as:*

$$\left|\Pr\left[1 \leftarrow \mathcal{A}\left(a, \left[a\ 1\right]\cdot \mathbf{s}, (c_i,\mathbf{z}_i)_{i\in[Q]}\right)\right] - \Pr\left[1 \leftarrow \mathcal{A}\left(a, u, (c_i,\mathbf{z}_i)_{i\in[Q]}\right)\right]\right|,$$

where $(a,u) \overset{\$}{\leftarrow} \mathcal{R}_q^2$, $\mathbf{s} \leftarrow \mathcal{D}_{\mathsf{sk}}$ and for $i \in [Q]$: $c_i \leftarrow \mathcal{C}$, $\mathbf{r}_i \leftarrow \mathcal{D}_{\mathsf{pert}}$, and $\mathbf{z}_i = c_i \cdot \mathbf{s} + \mathbf{r}_i$. The $\mathsf{Hint\text{-}RLWE}_{q,Q,\mathcal{D}_{\mathsf{sk}},\mathcal{D}_{\mathsf{pert}},\mathcal{C}}$ assumption states that any efficient adversary $\mathcal{A}$ has a negligible advantage. We may write $\mathsf{Hint\text{-}RLWE}_{q,Q,\sigma_{\mathbf{s}},\sigma_{\mathbf{r}},\mathcal{C}}$ as a shorthand when $\mathcal{D}_{\mathsf{sk}} = D_{\sigma_{\mathbf{s}}}$ and $\mathcal{D}_{\mathsf{pert}} = D_{\sigma_{\mathbf{r}}}$ are the Gaussian distributions of parameters $\sigma_{\mathbf{s}}$ and $\sigma_{\mathbf{r}}$, respectively. When $Q = 0$, we recover the classical RLWE *problem:* $\mathsf{RLWE}_{q,\mathcal{D}_{\mathsf{sk}}} = \mathsf{Hint\text{-}RLWE}_{q,Q=0,\mathcal{D}_{\mathsf{sk}},\mathcal{D}_{\mathsf{pert}},\mathcal{C}}$.

The spectral norm $s_1(\mathbf{M})$ of a matrix $\mathbf{M}$ is defined as the value $\max_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{M}\,\mathbf{x}\|}{\|\mathbf{x}\|}$. We recall that if a matrix is symmetric, then its spectral norm is also its largest eigenvalue. Given a polynomial $c \in \mathcal{R}$, we may abusively use the term "spectral norm $s_1(c)$ of c" when referring to the spectral norm of the anti-circulant matrix $\mathcal{M}(c)$ associated to c. Finally, if $c(x) = \sum_{0 \leqslant i < n} c_i \, x^i$, then the Hermitian adjoint of c, which we denote by c^*, is defined as $c^*(x) = c_0 - \sum_{0 < i < n} c_{n-i} \, x^i$. Note that $\mathcal{M}(c)^t = \mathcal{M}(c^*)$.

Theorem 1 (Hardness of Hint-RLWE, adapted from [32]). *Let $\mathcal{C}$ be a distribution over $\mathcal{R}$, and let B_{HRLWE} be a real number such that $s_1(D) \leqslant B_{\mathsf{HRLWE}}$ with overwhelming probability, where $D = \sum_Q c_i c_i^*$. Let $\sigma, \sigma_{\mathsf{sk}}, \sigma_{\mathsf{pert}} > 0$ such that $\frac{1}{\sigma^2} = 2\left(\frac{1}{\sigma_{\mathsf{sk}}^2} + \frac{B_{\mathsf{HRLWE}}}{\sigma_{\mathsf{pert}}^2}\right)$. If $\sigma \geqslant \sqrt{2}\eta_\varepsilon(\mathbb{Z}^n)$ for $0 < \varepsilon \leqslant 1/2$, where $\eta_\varepsilon(\mathbb{Z}^n)$ is the smoothing parameter of $\mathbb{Z}^n$, then there exists an efficient reduction from $\mathsf{RLWE}_{q,\sigma}$ to $\mathsf{Hint\text{-}RLWE}_{q,Q,\sigma_{\mathsf{sk}},\sigma_{\mathsf{pert}},\mathcal{C}}$ that reduces the advantage by at most 4ε.*

For our scheme, concrete bounds for B_{HRLWE} will be given in Lemma 2. Finally, we recall the Ring-SIS (RSIS) assumption.

Definition 4 (RSIS). *Let ℓ, q be integers and $\beta > 0$ be a real number. The advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{RSIS}}(\kappa)$ of an adversary $\mathcal{A}$ against the Ring Short Integer Solutions problem $\mathsf{RSIS}_{q,\ell,\beta}$ is defined as:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{RSIS}}(\kappa) = \Pr\left[\mathbf{a} \xleftarrow{\$} \mathcal{R}_q^\ell,\ \mathbf{z} \leftarrow \mathcal{A}(\mathbf{a})\ :\ 0 < \|\mathbf{z}\| \leqslant \beta\ \wedge \left[1\ \mathbf{a}^\top\right]\mathbf{z} = \mathbf{0} \bmod q\right].$$

The $\mathsf{RSIS}_{q,\ell,\beta}$ assumption states that any efficient adversary $\mathcal{A}$ has a negligible advantage.

2.4 Masking

Definition 5. *Let R be a finite commutative ring and $d \geqslant 1$ be an integer. Given $x \in R$, a d-sharing of x is a d-tuple $(x_i)_{i \in [d]}$ such that $\sum_{i \in [d]} x_i = x$. We denote by $[\![x]\!]_d$ any valid d-sharing of x; when d is clear from context, we may omit it and simply write $[\![x]\!]$. A probabilistic encoding of x is a distribution over encodings of x.*

- A d-shared circuit C is a randomized circuit working on d-shared variables. More specifically, a d-shared circuit takes a set of n input sharings $(x_{1,i})_{i \in [d]}, \ldots, (x_{n,i})_{i \in [d]}$ and computes a set of m output sharings $(y_{1,i})_{i \in [d]}, \ldots, (y_{m,i})_{i \in [d]}$ such that $(y_1, \ldots, y_m) = f(x_1, \ldots, x_n)$ for some deterministic function f. The quantity $(d-1)$ is then referred to as the *masking order*.
- A probe on C or an intermediate variable of C refers to a wire index (for some given indexing of C's wires).
- An evaluation of C on input $(x_{1,i})_{i \in [d]}, \ldots, (x_{n,i})_{i \in [d]}$ under a set of probes P refers to the distribution of the tuple of wires pointed by the probes in P when the circuit is evaluated on $(x_{1,i})_{i \in [d]}, \ldots, (x_{n,i})_{i \in [d]}$, which is denoted by $C((x_{1,i})_{i \in [d]}, \ldots, (x_{n,i})_{i \in [d]})_P$.

In the following, we focus on a special kind of shared circuits which are composed of gadgets. A (u, v)-gadget is a randomized shared circuit as a building block of a shared circuit that performs a given operation on its u input sharings and produces v output sharings.

2.5 Probing Model

The most commonly used leakage model is the probing model, introduced by Ishai, Sahai and Wagner in 2003 [27]. Informally, it states that during the evaluation of a circuit C, at most t wires (chosen by the adversary) leak the value they carry. The circuit C is said to be t-probing secure if the exact values of any set of t probes do not reveal any information about its inputs.

Definition 6 (t-probing security). *A randomized shared arithmetic circuit C equipped with an encoding $\mathcal{E}$ is t-probing secure if there exists a probabilistic simulator $\mathcal{S}$ which, for any input $x \in \mathbb{K}^\ell$ and every set of probes P such that $|P| \leqslant t$, satisfies $\mathcal{S}(C, P) = C(\mathcal{E}(x))_P$.*

Since the computation of distributions is expensive, the security proof relies on stronger simulation-based properties, introduced by Barthe et al. [5], to demonstrate the independence of the leaking wires from the input secrets. Informally, the idea is to perfectly simulate each possible set of probes with the smallest set of shares for each input. We recall the formal definitions of t-non-interference and t-strong non-interference hereafter. These provide a framework for the composition of building blocks, which makes the security analysis easier when masking entire schemes, as is the case here.

Definition 7 (t-non-interference). *A randomized shared arithmetic circuit C equipped with an encoding $\mathcal{E}$ is t-non-interferent (or t-NI) if there exists a deterministic simulator $\mathcal{S}_1$ and a probabilistic simulator $\mathcal{S}_2$, such that, for any input $x \in \mathbb{K}^\ell$, for every set of probes P of size t,*

$$\begin{aligned} &(\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_\ell) \leftarrow \mathcal{S}_1(C, P) \qquad \textit{with} \ \ |\mathcal{I}_1|, |\mathcal{I}_2|, \ldots, |\mathcal{I}_\ell| \leqslant t \\ \textit{and} \quad &\mathcal{S}_2((x_{1,i})_{i \in \mathcal{I}_1}, (x_{2,i})_{i \in \mathcal{I}_2}, \ldots, (x_{\ell,i})_{i \in \mathcal{I}_\ell}) = C(\mathcal{E}(x))_P. \end{aligned}$$

If the input sharing is uniform, a t-non-interferent randomized arithmetic circuit C is also t-probing secure. One step further, the strong non-interference benefits from stopping the propagation of the probes between the outputs and the input shares and additionally trivially implies t-NI.

We now introduce the notion of t-strong non-interference with unshared input values (t-SNIu). The new notion is very much similar to that of t-SNI of Barthe et al. [6] with a special additional *unshared* input values x' along with the usual shared input values x. In addition, there will be no unshared outputs in t-SNIu, hence the interface with other gadgets is with the shared inputs only as with the original definition.

Definition 8 (t-strong non-interference with unshared input values). *A randomized shared arithmetic circuit C equipped with an encoding $\mathcal{E}$ is t-strong non-interferent with unshared input values (or t-SNIu) if there exists a deterministic simulator $\mathcal{S}_1$ and a probabilistic simulator $\mathcal{S}_2$, such that, for any shared inputs $x \in \mathbb{K}^\ell$ and unshared input values $x' \in \mathbb{K}^{\ell'}$, for every set of probes P of size t whose P_1 target internal variables and $P_2 = P \backslash P_1$ target the output shares,*

$$(\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_\ell, \mathcal{I}') \leftarrow \mathcal{S}_1(C, P) \qquad \textit{with} \ \ |\mathcal{I}_1|, |\mathcal{I}_2|, \ldots, |\mathcal{I}_\ell|, |\mathcal{I}'| \leqslant |P_1|$$

$$\textit{and}\quad \mathcal{S}_2((x_{1,i})_{i\in\mathcal{I}_1},(x_{2,i})_{i\in\mathcal{I}_2},\ldots,(x_{\ell,i})_{i\in\mathcal{I}_\ell},(x'_i)_{i\in\mathcal{I}'}) = C(\mathcal{E}(x,x'))_P.$$

We remark that for usual gadgets with no unshared inputs, our above definition of t-SNIu reduces to the usual t-SNI notion. Looking ahead, we will model our AddRepNoise gadget's internal small random values as unshared inputs to the AddRepNoise gadget.

Common Operations Arithmetic masking, which we use in this paper, is compatible with simpler arithmetic performed in time $O(d^2)$ and is shown to be t-SNI by Barthe et al. [5, Proposition 2].

A t-SNI refresh gadget (Refresh), given in Algorithm 1, with complexity $O(d \log d)$ has been proposed by Battistello et al. [7]. Its complexity has been improved by a factor 2 by Mathieu-Mahias [39], which also proves that it is t-SNI in [39, Section 2.2]. We use this improved variant as a building block of our schemes. For completeness, it is reproduced in Algorithms 1 and 3.

Finally, a secure decoding algorithm Unmask is described in Algorithm 2. It is shown by Barthe et al. [6] to be t-NIo [6, Definition 7] .

Refresh and Unmask take as a (subscript) parameter a finite abelian group $\mathbb{G}$. When $\mathbb{G}$ is clear from context, we may drop the subscript for concision.

```
Algorithm 1 Refresh_G([[x]]) → [[x]]'
Require: A d-sharing [[x]] of x ∈ G
Ensure: A fresh d-sharing [[x]] of x
[[z]] ←$ ZeroEncoding(G, d)
return [[x]]' := [[x]] + [[z]]
```

```
Algorithm 2 Unmask_G([[x]]) → x
Require: A d-sharing [[x]] = (x_i)_{i∈[d]} of x ∈ G
Ensure: The clear value x ∈ G
[[x]] ← Refresh([[x]])
return x := Σ_{i∈[d]} x_i
```

```
Algorithm 3 ZeroEncoding(G, d) → [[z]]_d
Require: A power-of-two integer d, a finite abelian group G
Ensure: Uniform d-sharing [[z]] ∈ G^d of 0 ∈ G
if d = 1 then
    return [[z]]_1 := (0)
[[z_1]]_{d/2} ← ZeroEncoding(G, d/2)
[[z_2]]_{d/2} ← ZeroEncoding(G, d/2)
[[r]]_{d/2} ←$ G^{d/2}
[[z_1]]_{d/2} := [[z_1]]_{d/2} + [[r]]_{d/2}
[[z_2]]_{d/2} := [[z_2]]_{d/2} − [[r]]_{d/2}
return [[z]]_d := ([[z_1]]_{d/2} ∥ [[z_2]]_{d/2})
    ▷ (u ∥ v) denote shares concatenation.
```

AddRepNoise The AddRepNoise procedure (Algorithm 4) is one of the key building blocks of our scheme. It is an adaptation of the eponymous procedure from the Raccoon signature scheme [41].

We prove that the AddRepNoise gadget satisfies the SNI with both shared and *unshared* inputs notion (t-SNIu), as defined in Sec. 2.1. In particular, there exists a simulator that can simulate $\leqslant t$ probed variables using $\leqslant t$ unshared input values and $\leqslant t$ shared input values. The underlying intuition (see Sec. 4.2 in [41] for

Algorithm 4 AddRepNoise$(\mathbb{G}, d, \mathcal{D}^{\mathsf{ind}}, \mathsf{rep}) \rightarrow [\![v]\!]$

```
Require: A finite Abelian group $\mathbb{G}$, the number of shares $d$, a noise distribution $\mathcal{D}^{\mathsf{ind}}$,
    a repetition count parameter $\mathsf{rep}$
Ensure: A masked element $[\![v]\!] \in \mathbb{G}^d$ such that $v \sim [d \cdot \mathsf{rep}] \cdot \mathcal{D}^{\mathsf{ind}}$
1: $[\![v]\!] = (v_j)_{j\in[d]} := (0_{\mathbb{G}})^d$                    ▷ $[\![v]\!] \in \mathbb{G}^d$
2: for $i \in [\mathsf{rep}]$ do
3:     for $j \in [d]$ do
4:         $r_{i,j} \leftarrow \mathcal{D}^{\mathsf{ind}}$
5:         $v_j := v_j + r_{i,j}$
6:     $[\![v]\!] \leftarrow$ Refresh$([\![v]\!])$                    ▷ Refresh $[\![v]\!]$ on each repeat
7: return $[\![v]\!]$
```

an informal discussion) is that the t-SNI property of the Refresh gadget inserted between the rep MaskedAdd gadgets effectively isolates the MaskedAdd gadgets and prevents the adversary from combining two probes in different MaskedAdd gadgets to learn information about more than two unshared inputs, i.e. t probes only reveal $\leqslant t$ unshared inputs. The formal statement is given in Lemma 1.

Lemma 1 **(AddRepNoise probing security).** *Gadget AddRepNoise is t-SNIu, considering that AddRepNoise has no shared inputs, and that it takes as* unshared *input the values $(r_{i,j})_{i,j}$.*

Proof. The AddRepNoise consists of rep repeats (over $i \in [\mathsf{rep}]$) of the following Add-Refresh subgadget: a MaskedAdd gadget (line 5) that adds sharewise the d unshared inputs $(r_{i,j})_{j\in[d]}$ to the internal sharing $[\![v]\!]$, followed by a Refresh$([\![v]\!])$ gadget (line 6). For $i \in [\mathsf{rep}]$, we note:

1. $t_{1,R}^{(i)}$ the number of probed internal variables (not including outputs);
2. $t_{2,R}^{(i)}$ the number of simulated or probed output variables in i'th Refresh;
3. $t_{A}^{(i)}$ the total number of probed variables in the i'th MaskedAdd gadget (i.e. including probed inputs and probed outputs that are not probed as inputs of Refresh).

We construct a simulator for the t probed observation in AddRepNoise by composing the outputs of the $[\mathsf{rep}]$ simulators for probed observations in the Add-Refresh subgadgets, proceeding from output to input. For $i = \mathsf{rep} - 1$ down to 0, the simulator for the i'th Add-Refresh subgadget works as follows.

The Refresh gadget is t-SNI according to [5]. Therefore, there exists a simulator $\mathcal{S}_R^{(i)}$ that can simulate $t_{1,R}^{(i)} + t_{2,R}^{(i)} \leqslant t$ variables using $t_{in,R}^{(i)} \leqslant t_{1,R}^{(i)}$ input shared values $[\![v]\!]$ of the i'th Refresh gadget. The latter is also equal to the number of outputs of MaskedAdd gadget that need to be simulated to input to $\mathcal{S}_R^{(i)}$.

Since the ith MaskedAdd gadget performs addition sharewise, we can now construct a simulator $\mathcal{S}_A^{(i)}$ that simulates the required $\leqslant t_A^{(i)} + t_{in,R}^{(i)} \leqslant t_A^{(i)} + t_{1,R}^{(i)}$ variables in the i'th MaskedAdd gadget using $t_{in,A}^{(i)} \leqslant t_A^{(i)} + t_{1,R}^{(i)}$ additions and the corresponding summands: $t_{in,A}^{(i)}$ input shares of the first MaskedAdd gadget in $[\![v]\!]$ and $t_{in,A}^{(i)}$ unshared inputs $r_{i,j}$.

Over all $i \in [\mathsf{rep}]$, the composed simulator $\mathcal{S}$ for AddRepNoise can simulate all t probed observations in AddRepNoise using a total of $t_{in,ARN,u} \leqslant \sum_{i\in[\mathsf{rep}]} t^{(i)}_{in,A} \leqslant \sum_{i\in[\mathsf{rep}]} t^{(i)}_{A} + t^{(i)}_{1,R} \leqslant t$ unshared input values $r_{i,j}$ of AddRepNoise, where $t_{in,ARN,u} \leqslant t$ since the above $\sum_{i\in[\mathsf{rep}]} t^{(i)}_{A} + t^{(i)}_{1,R}$ variables are distinct probed variables in AddRepNoise. □

3 Plover-RLWE: Our RLWE-based Maskable Signature

This section presents a maskable hash-and-sign signature scheme based on RLWE. It leverages the compact lattice gadget from Yu et al. [48], and its mostly linear operations to construct a maskable scheme relying on noise flooding, i.e. Gaussian sampling is replaced by a large noise provably hiding a secret value. We describe the unmasked scheme in Section 3.1, and the masked scheme in Section 3.3. We introduce additional notations.

- ExpandA : $\{0,1\}^\kappa \to \mathcal{R}_q$ deterministically maps a uniform seed seed to a uniformly pseudo-random element $a \in \mathcal{R}_q$.
- $H : \{0,1\}^* \times \{0,1\}^{2\kappa} \times \mathcal{V} \to \mathcal{R}_q$ is a collision-resistant hash function mapping a tuple $(\mathsf{msg}, \mathsf{salt}, \mathsf{vk})$ to an element $u \in \mathcal{R}_q$. We note that H is parameterized by a salt salt for the security proof of Gentry et al. [21] to go through, and by the verification key vk.

3.1 Description of Unmasked Plover-RLWE

Parameters. We sample RLWE trapdoors from a distribution $\mathcal{D}_{\mathsf{sk}}$, and noise in the signature from a distribution $\mathcal{D}_{\mathsf{pert}}$. Additionally, we introduce an integer parameter β; it is used as a divider in the signature generation to decompose challenges in low/high order bits via $\mathsf{Decompose}_\beta$. Despite its name, we do not require that β divides q; that was only required by the Gaussian sampler of [48].

Key generation. The key generation samples a public polynomial a, derived from a seed. The second part of the public key is essentially an RLWE sample shifted by β. A description of the key generation is given in Algorithm 5.

Algorithm 5 $\mathsf{Plover\text{-}RLWE.Keygen}(1^\kappa) \to (\mathsf{vk}, \mathsf{sk})$

```
Require: The ring $\mathcal{R}_q$, a divider $\beta$, a distribution $\mathcal{D}_{\mathsf{sk}}$ over $\mathcal{R}^2$
Ensure: A verification key $\mathsf{vk} = (\mathsf{seed}, b) \in \{0,1\}^\kappa \times \mathcal{R}_q$, a signing key $\mathsf{sk} = (s, e) \in \mathcal{R}^2$
seed $\xleftarrow{\$} \{0,1\}^\kappa$
$a := \mathsf{ExpandA}(\mathsf{seed})$        ▷ ExpandA maps a seed to an element in $\mathcal{R}$
$(s, e) \leftarrow \mathcal{D}_{\mathsf{sk}}$
$b := \beta - (as + e) \bmod q$
return $\mathsf{vk} := (\mathsf{seed}, b), := (\mathsf{vk}, s, e)$
```

Signing procedure. The signature generation is described in Algorithm 6. It first hashes the given message msg to a target polynomial u. It then uses its trapdoor to find a short pre-image $\mathbf{z} = (z_1, z_2, z_3)$ such that $\mathbf{A} \cdot \mathbf{z} := z_1 + a\, z_2 + b\, z_3 = u - c_2 \bmod q$ for a small c_2 and $\mathbf{A} := \begin{bmatrix} 1 & a & b \end{bmatrix}$. In order to prevent leaking the trapdoor, a noise vector $\mathbf{p}$ is sampled and added to the pre-image $\mathbf{z}$. As in [48], the actual signature is (z_2, z_3), since $z_1 + c_2 = u - a\, z_2 - b\, z_3$ can be recovered in the verification procedure. Additionally, c_1 is public and does not require to be hidden by noise. Signature size is then dominated by sending z_2.

Ahead of Section 3.3, we note that, except for Line 5, all operations in Algorithm 6 either (i) are linear functions of sensitive data ($\mathbf{T}$ and $\mathbf{p}$), and can therefore be masked with overhead $\tilde{O}(d)$, or (ii) can be performed unmasked.

Algorithm 6 Plover-RLWE.Sign(msg, sk) → sig

```
Require: A message msg, the secret key sk = ((seed, $b$), $s$, $e$), a bound $B_2 > 0$
Ensure: A signature (salt, $z_2$, $z_3$)
1: $a :=$ ExpandA(seed)
2: salt $\xleftarrow{\$} \{0,1\}^{2\kappa}$
3: $u := H(\mathsf{msg}, \mathsf{salt}, \mathsf{vk})$
4: $\mathbf{A} := \begin{bmatrix} 1 & a & b \end{bmatrix}$, $\mathbf{T} := \begin{bmatrix} e \\ s \\ 1 \end{bmatrix}$
5: $\mathbf{p} \leftarrow \mathcal{D}_{\mathsf{pert}} \times \{0\}$          ▷ Recall $\mathcal{D}_{\mathsf{pert}}$ is over $\mathcal{R}_q^2$
6: $c := u - \mathbf{A} \cdot \mathbf{p}$
7: $(c_1, c_2) :=$ Decompose$_\beta(c)$          ▷ $c = \beta \cdot c_1 + c_2$
8: $\mathbf{z} \leftarrow \mathbf{p} + \mathbf{T} \cdot c_1$          ▷ $\mathbf{z} = (z_1, z_2, z_3)$ and $z_3 = c_1$
9: return sig := (salt, $z_2$, $z_3$), $\mathsf{aux}_{\mathsf{sig}} = c_2$
          ▷ $\mathsf{aux}_{\mathsf{sig}}$ used in security proof, but not in verification.
```

Verification. The verification first recovers $z_1' := u - a\, z_2 - b\, z_3$ (equal to $z_1 + c_2$), followed by checking the shortness of (z_1', z_2, z_3). A formal description is given in Algorithm 7. Using notations from Algorithm 6, correctness follows from:

$$\mathbf{A}\,\mathbf{z} = \mathbf{A}\,\mathbf{p} + \mathbf{A}\,\mathbf{T}\,c_1 = (u - c) + \beta \cdot c_1 = u - c_2$$

Algorithm 7 Plover-RLWE.Verify(vk, msg, sig) → **accept** or **reject**

```
Require: sig = (salt, $z_2$, $z_3$), msg, vk = (seed, $b$), and a bound $B_2 > 0$
Ensure: Accept or reject.
$a :=$ ExpandA(seed),
$u := H(\mathsf{msg}, \mathsf{salt}, \mathsf{vk})$
$z_1' := (u - a\, z_2 - b\, z_3) \bmod q$          ▷ $z_1' = z_1 + c_2$
accept if { $\|(z_1', z_2, z_3)\| \leqslant B_2$ and $\|z_3\|_\infty \leqslant q/(2\beta) + 1/2$ }, else reject
```

To provide a more modular exposition to our algorithms and security proofs, we next prove the EUF-CMA security of our unmasked signature proposal. Later in Section 3.4, we will reduce the t-probing security of our *masked* construction from the EUF-CMA security of the *unmasked* construction. To facilitate the latter reduction, we show the EUF-CMA security of the unmasked construction even when the signing oracle outputs the auxiliary signature information $\mathsf{aux}_{\mathsf{sig}} = c_2$ (see Algorithm 6) along with the signature sig.

3.2 EUF-CMA Security of Unmasked Plover-RLWE

For the Hint-RLWE reduction in Theorem 2, we introduce Definition 9. Note that in the definition, if β divides q, then c_1 and c_2 are independent and uniformly random in their supports but this is not necessary for our reduction.

Definition 9 (Distributions for Hint-RLWE). *Let (c_1, c_2) be sampled from the joint distribution induced by sampling c uniformly at random from R_q and setting $(c_1, c_2) := \mathsf{Decompose}_\beta(c)$. Then:*

- *We let $\mathcal{C}_1$ denote the marginal distribution of c_1.*
- *For a fixed c_1', we let $\mathcal{C}_2^{|c_1'}$ denote the conditional distribution of c_2 conditioned on the event $c_1 = c_1'$.*

Before we move into the formal security statement, we emphasize that the security of unmasked Plover reduces to the standard RLWE and RSIS problems when the distributions $\mathcal{D}_{\mathsf{sk}}, \mathcal{D}_{\mathsf{pert}}$ are chosen to be discrete Gaussians (with appropriate parameter). This is due to the fact that Hint-RLWE reduces to RLWE as proven in [33], see also Theorem 1.

Theorem 2. *The* Plover-RLWE *scheme is* EUF-CMA *secure in the random oracle model if* $\mathsf{RLWE}_{q,\mathcal{U}([-B_2/\sqrt{2n},B_2/\sqrt{2n}]^n)^2}$, $\mathsf{Hint\text{-}RLWE}_{q,Q_{\mathsf{Sign}},\mathcal{D}_{\mathsf{sk}},\mathcal{D}_{\mathsf{pert}},\mathcal{C}_1}$ *and* $\mathsf{RSIS}_{q,2,2B_2}$ *assumptions hold. Formally, let $\mathcal{A}$ be an adversary against the* EUF-CMA *security game making at most Q_{Sign} signing queries and at most Q_H random oracle queries. Denote an adversary $\mathcal{H}$'s advantage against* $\mathsf{Hint\text{-}RLWE}_{q,Q_{\mathsf{Sign}},\mathcal{D}_{\mathsf{sk}},\mathcal{D}_{\mathsf{pert}},\mathcal{C}_1}$ *by* $\mathsf{Adv}_{\mathcal{H}}^{\mathsf{Hint\text{-}RLWE}}(\kappa)$*, and an adversary $\mathcal{D}$'s advantage against* $\mathsf{RLWE}_{q,\mathcal{U}([-B_2/\sqrt{2n},B_2/\sqrt{2n}]^n)^2}$ *by* $\mathsf{Adv}_{\mathcal{D}}^{\mathsf{RLWE}}(\kappa)$*. Then, there exists an adversary $\mathcal{B}$ running in time $T_{\mathcal{B}} \approx T_{\mathcal{H}} \approx T_{\mathcal{D}} \approx T_{\mathcal{A}}$ against* $\mathsf{RSIS}_{q,2,2B_2}$ *with advantage* $\mathsf{Adv}_{\mathcal{B}}^{\mathsf{RSIS}}(\kappa)$ *such that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}} \leqslant p_c + Q_{\mathsf{Sign}}\, Q_H/2^{2\kappa} + \mathsf{Adv}_{\mathcal{H}}^{\mathsf{Hint\text{-}RLWE}}(\kappa) + Q_H \cdot \mathsf{Adv}_{\mathcal{D}}^{\mathsf{RLWE}}(\kappa) + \mathsf{Adv}_{\mathcal{B}}^{\mathsf{RSIS}}$$

for some $p_c \leqslant 2^{-n\cdot\left(2\log_2(2B_2/\sqrt{2n})-\log_2(q)\right)}$.

Proof. We prove the security of the above scheme with intermediary hybrid games, starting from the EUF-CMA game against our signature scheme in the ROM and then finally arriving at a game where we can build an adversary $\mathcal{B}$ against $\mathsf{RSIS}_{q,2,2B_2}$. Let $\mathcal{A}$ be an adversary against the EUF-CMA security game.

Game_0. This is the original $\mathsf{EUF\text{-}CMA}$ security game. A key pair $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Plover\text{-}RLWE.Keygen}(1^\kappa)$ is generated and $\mathcal{A}$ is given vk. $\mathcal{A}$ gets access to a signing oracle $\mathsf{OSign}(\mathsf{msg})$ that on input a message msg (chosen by $\mathcal{A}$) outputs a signature, along with the auxiliary signature information $(\mathsf{sig}, \mathsf{aux}_{\mathsf{sig}}) \leftarrow \mathsf{Plover\text{-}RLWE.Sign}(\mathsf{msg}, \mathsf{sk})$ and adds $(\mathsf{msg}, \mathsf{sig}, \mathsf{aux}_{\mathsf{sig}})$ to a table $\mathcal{T}_s$. The calls to the random oracle H are stored in a table $\mathcal{T}_H$ and those to OSign are stored in a table $\mathcal{T}_s$.

Game_1. Given a message msg, we replace the signing oracle OSign as follows :

1. Sample $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{2\kappa}$. Abort if an entry matching the $(\mathsf{msg}, \mathsf{salt}, \mathsf{vk})$ tuple exists in $\mathcal{T}_H$ (Abort I).
2. Sample $u' \xleftarrow{\$} \mathcal{R}_q$ and decompose it as $(c_1, c_2) := \mathsf{Decompose}_\beta(u')$ (i.e., $u' = \beta \cdot c_1 + c_2$).
3. Sample $\mathbf{p} \leftarrow \mathcal{D}_{\mathsf{pert}} \times \{0\}$.
4. Compute $\mathbf{z}' := \mathbf{p} + \mathbf{T} \cdot c_1 + \begin{bmatrix} c_2 & 0 & 0 \end{bmatrix}$. Program the random oracle H such that $H(\mathsf{msg}, \mathsf{salt}, \mathsf{vk}) := \mathbf{A}\mathbf{z}'$. Store in $\mathcal{T}_H$ the entry $((\mathsf{msg}, \mathsf{salt}), \mathbf{z}')$.
5. Return $\mathsf{sig} := (\mathsf{salt}, z'_2, z'_3)$ and $\mathsf{aux}_{\mathsf{sig}} := c_2$, where $\mathbf{z}' = (z'_1, z'_2, z'_3)$, and store $(\mathsf{msg}, \mathsf{sig}, \mathsf{aux}_{\mathsf{sig}})$ in $\mathcal{T}_s$.

Observe that Abort I happens with probability at most $Q_{\mathsf{Sign}} Q_H / 2^{2\kappa}$. If it does not, then the view of $\mathcal{A}$ in Game_1 is distributed identically to their view in Game_0. Indeed, in Game_1, the value u output by H for signed values is still uniform in $\mathcal{R}_q$ and independent of $\mathbf{p}$. This is due to the fact that $u := \mathbf{A}\mathbf{z}' = \mathbf{A}\mathbf{p} + \mathbf{A}\mathbf{T} \cdot c_1 + c_2 = \mathbf{A}\mathbf{p} + \beta c_1 + c_2 = \mathbf{A}\mathbf{p} + u'$ and u' is uniform in $\mathcal{R}_q$ and independent of $\mathbf{p}$. Hence, there is an advantage loss only if Abort I occurs; that is,

$$\left| \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_0} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_1} \right| \leqslant Q_{\mathsf{Sign}} Q_H / 2^{2\kappa}.$$

Game_2. In this game, we make a single change over Game_1 and replace $b = \beta - (as + e)$ by $b = \beta - b'$ where b' is a uniformly random polynomial in $\mathcal{R}_q$. This means that b also follows uniform distribution over $\mathcal{R}_q$.
We can observe that this reduces to $\mathsf{Hint\text{-}RLWE}$ problem with Q_{Sign} hints. In particular, given $\mathsf{Hint\text{-}RLWE}$ instance $(a, b', \{c_{1,i}, (h_{1,i}, h_{2,i})\}_{i \in [Q_{\mathsf{Sign}}]})$ with $c_{1,i} \leftarrow \mathcal{C}_1$, and $h_{1,i} := p_{1,i} + e \cdot c_{1,i}$ and $h_{2,i} := p_{2,i} + s \cdot c_{1,i}$, adversary $\mathcal{H}$ runs $\mathcal{A}$ with verification key (a, b') and simulates the view of $\mathcal{A}$ as in Game_1, computing the values of $z'_{1,i}, z'_{2,i}$ in step 4 of the i'th query to OSign in Game_1 using the hints $h_{1,i}, h_{2,i}$ as follows: $z'_{1,i} = h_{1,i} + c_{2,i}$, $z'_{2,i} = h_{2,i}$, with $c_{2,i}$ sampled from the conditional distribution $\mathcal{C}_2^{|c_{1,i}}$. At the end of the game, $\mathcal{H}$ returns 1 if $\mathcal{A}$ wins the game, and 0 otherwise. Observe that if b' in the $\mathsf{Hint\text{-}RLWE}$ instance is from the real RLWE (resp. uniform in R_q) distribution, then $\mathcal{H}$ simulates to $\mathcal{A}$ its view in Game_1 (resp. Game_2), so $\mathcal{H}$'s advantage is lower bounded as

$$\left| \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_1} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_2} \right| \leqslant \mathsf{Adv}_{\mathcal{H}}^{\mathsf{Hint\text{-}RLWE}}(\kappa).$$

Game_3. In this game, we replace the random oracle H as follows. If an entry has not been queried before, H returns $\mathbf{A}\mathbf{z}$ where $\mathbf{z} \xleftarrow{\$} \{0\} \times \left([-B_2/\sqrt{2n}, B_2/\sqrt{2n}]^n\right)^2$ (observe that $\|\mathbf{z}\| \leqslant B_2$). We store in $\mathcal{T}_H$ the entry $((\mathsf{msg}, \mathsf{salt}), \mathbf{z})$ for an input query $(\mathsf{msg}, \mathsf{salt}, \mathsf{vk})$. Note that the result of $\mathbf{A}\mathbf{z}$ is indistinguishable from a uniformly random value in $\mathcal{R}_q$ by the $\mathsf{RLWE}_{q,\mathcal{U}([-B_2/\sqrt{2n},B_2/\sqrt{2n}]^n)^2}$ assumption. Hence, we have

$$\left|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_2} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_3}\right| \leqslant Q_H \cdot \mathsf{Adv}_{\mathcal{D}}^{\mathsf{RLWE}}(\kappa).$$

Game_4. Let $\mathsf{sig}^* := (\mathsf{salt}^*, z_2^*, z_3^*) \notin \mathcal{T}_s$ be the forged signature output by $\mathcal{A}$ for a message msg^*. Define $z_1^* := u - az_2^* - bc_1^*$ and $\mathbf{z}^* := (z_1^*, z_2^*, z_3^*)$. Without loss of generality, we assume that the pair $(\mathsf{msg}^*, \mathsf{salt}^*)$ has been queried to the random oracle H. From $\mathcal{T}_H$, we retrieve $\widehat{\mathbf{z}} = (\widehat{z}_1, \widehat{z}_2, \widehat{z}_3)$ corresponding to $(\mathsf{msg}^*, \mathsf{salt}^*)$. If $\widehat{\mathbf{z}} = \mathbf{z}^*$, then we abort (Abort II).

- **Case 1:** Suppose $H(\mathsf{msg}^*, \mathsf{salt}^*, \mathsf{vk})$ was called by the signing oracle OSign. Then, since $\mathsf{sig}^* := (\mathsf{salt}^*, z_2^*, z_3^*) \notin \mathcal{T}_s$, we must have $(\mathsf{salt}^*, z_2^*, z_3^*) \neq (\mathsf{salt}^*, \widehat{z}_2, \widehat{z}_3)$, which implies $\widehat{\mathbf{z}} \neq \mathbf{z}^*$. Hence, Abort II never happens in this case.
- **Case 2:** Suppose $H(\mathsf{msg}^*, \mathsf{salt}^*, \mathsf{vk})$ was queried directly to H. Then, since the first entry of $\widehat{\mathbf{z}}$ (resp. $\mathbf{z}^*$) is uniquely determined by the remaining entries of $\widehat{\mathbf{z}}$ (resp. $\mathbf{z}^*$), Abort II happens with a probability

  $$p_c := \max_u \Pr[(z_2^*, z_3^*) = (\widehat{z}_2, \widehat{z}_3) \mid H(\mathsf{msg}^*, \mathsf{salt}^*, \mathsf{vk}) = u = \mathbf{A}\widehat{\mathbf{z}}] \leqslant 2^{-H_\infty((\widehat{z}_2, \widehat{z}_3)|u)}$$

  Since $H_\infty((\widehat{z}_2, \widehat{z}_3)) \geqslant 2n \log_2(2B_2/\sqrt{2n})$ and $H_\infty(u) \leqslant n \log_2(q)$, we have:

  $$\begin{aligned} H_\infty((\widehat{z}_2, \widehat{z}_3)|u) &\geqslant H_\infty((\widehat{z}_2, \widehat{z}_3, u)) - H_\infty(u) \\ &\geqslant H_\infty((\widehat{z}_2, \widehat{z}_3)) - H_\infty(u) \\ &\geqslant n \cdot (2\log_2(2B_2/\sqrt{2n}) - \log_2(q)) \end{aligned}$$

Hence, we get

$$\left|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_3} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_4}\right| \leqslant p_c \quad \text{for} \quad p_c \leqslant 2^{-n\cdot(2\log_2(2B_2/\sqrt{2n}) - \log_2(q))}.$$

Observe from the verification algorithm (Alg. 7) that $\mathbf{A}\mathbf{z}^* = u = H(\mathsf{msg}, \mathsf{salt}, \mathsf{vk})$ and $\|\mathbf{z}^*\| \leqslant B_2$. Also, by the construction of H, $u = \mathbf{A}\widehat{\mathbf{z}}$ with $\|\widehat{\mathbf{z}}\| \leqslant B_2$ (see Game_3). Consequently, if Abort II does not happen, we can construct an adversary $\mathcal{B}$ that solves the $\mathsf{RSIS}_{q,2,2B_2}$ problem for $\mathbf{A}$ since $\mathbf{A}(\widehat{\mathbf{z}} - \mathbf{z}^*) = 0 \bmod q$ for $\widehat{\mathbf{z}} - \mathbf{z}^* \neq \mathbf{0}$ where $\mathbf{A} = \begin{bmatrix}1 & a & b\end{bmatrix}$ for a random a (modelling $\mathsf{ExpandA}$ as a random oracle) and random b (as discussed in Game_2). More concretely, let $\mathbf{A} = \begin{bmatrix}1 & a & b\end{bmatrix}$ be the challenge RSIS vector given to $\mathcal{B}$ where $a, b \xleftarrow{\$} \mathcal{R}_q$. The adversary $\mathcal{B}$ samples $\mathsf{seed} \xleftarrow{\$} \{0,1\}^\kappa$ and provides $\mathsf{vk} = (\mathsf{seed}, b)$ to $\mathcal{A}$ against

Game_4 and programs $\mathsf{ExpandA}(\mathsf{seed}) = a$ (modelling $\mathsf{ExpandA}$ as a random oracle). Note that the distribution of (seed, b) matches perfectly the distribution of vk produced in Game_4 due to the change of b in Game_2. Since OSign is run using only with publicly computable values in Game_4, $\mathcal{B}$ simulates OSign queries as in Game_4. $\mathcal{B}$ also simulates the queries to H as in Game_4 and stores the corresponding tables $\mathcal{T}_H$ and $\mathcal{T}_s$. As discussed above, provided that Abort II does not happen, $\mathcal{B}$ can use $\mathcal{A}$'s output forgery to create an $\mathsf{RSIS}_{q,2,2B_2}$ solution. Hence, $\left|\mathsf{Adv}_{\mathcal{B}}^{\mathsf{RSIS}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_4}\right| \leqslant p_c$ and $T_{\mathcal{B}} \approx T_{\mathcal{A}}$. As a result, we get

$$\begin{aligned}\left|\mathsf{Adv}_{\mathcal{B}}^{\mathsf{RSIS}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}\right| &= \left|\mathsf{Adv}_{\mathcal{B}}^{\mathsf{RSIS}} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_4} + \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_4} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}\right| \\ &\leqslant p_c + Q_{\mathsf{Sign}} Q_H / 2^{2\kappa} + \mathsf{Adv}_{\mathcal{H}}^{\mathsf{Hint\text{-}RLWE}}(\kappa) + Q_H \cdot \mathsf{Adv}_{\mathcal{D}}^{\mathsf{RLWE}}(\kappa).\end{aligned}$$

This concludes the proof. □

3.3 Description of Masked Plover-RLWE

This section describes our main construction, the masked $\mathsf{Plover\text{-}RLWE}$. $\mathcal{D}_{\mathsf{sk}}$ and $\mathcal{D}_{\mathsf{pert}}$ are respectively replaced by sums of distributions $[d\,\mathsf{rep}_{\mathsf{sk}}] \cdot \mathcal{D}_{\mathsf{sk}}^{\mathsf{ind}}$ and $[d\,\mathsf{rep}_{\mathsf{pert}}] \cdot \mathcal{D}_{\mathsf{pert}}^{\mathsf{ind}}$ to enable the masking, where $\mathsf{rep}_{\mathsf{sk}}$ and $\mathsf{rep}_{\mathsf{pert}}$ are newly introduced parameters.

Key generation. The key generation generates d-sharings small secrets $(\llbracket s \rrbracket, \llbracket e \rrbracket)$ and the corresponding RLWE sample $b = a \cdot s + e$. As in Raccoon [41], a key technique is the use of $\mathsf{AddRepNoise}$ for the generation of the small errors which ensures that a t-probing adversary learns limited information about (s, e).

Algorithm 8 $\mathsf{Plover\text{-}RLWE.MaskKeygen}(1^\kappa) \to (\mathsf{vk}, \mathsf{sk})$

```
Require: The ring $\mathcal{R}$, a modulus $q$
Ensure: A public key $(\mathsf{seed}, b) \in \{0,1\}^\kappa \times \mathcal{R}$, a private key $(s, e) \in \mathcal{R}^2$
$\mathsf{seed} \xleftarrow{\$} \{0,1\}^\kappa$
$a := \mathsf{ExpandA}(\mathsf{seed})$    ▷ Map a seed to an element in $\mathcal{R}$
$\llbracket (s, e) \rrbracket \leftarrow \mathsf{AddRepNoise}\left(\mathcal{R}_q^2, d, \mathcal{D}_{\mathsf{sk}}^{\mathsf{ind}}, \mathsf{rep}_{\mathsf{sk}}\right)$    ▷ Samples $s, e$ from $\mathcal{D}_{\mathsf{sk}}$
$\llbracket b \rrbracket := \beta - (a \cdot \llbracket s \rrbracket + \llbracket e \rrbracket)$
$b := \mathsf{Unmask}(\llbracket b \rrbracket)$
return $(\mathsf{vk} := (\mathsf{seed}, b), \mathsf{sk} := (\mathsf{vk}, \llbracket s \rrbracket))$
```

Signature procedure. The signature procedure is adapted to remove the computation of z_1 and save on masking. It recovers $z_1' = z_1 + c_2$ from unmasked values as done in the verification Algorithm 7 from unmasked values. This also allows to drop e from the private key and significantly reduces its size. A formal description is given in Algorithm 9.

Algorithm 9 $\mathsf{Plover\text{-}RLWE.MaskSign}(\mathsf{msg}, \mathsf{sk}) \to \mathsf{sig}$

```
Require: A message $\mathsf{msg}$, the secret key $\mathsf{sk} = ((\mathsf{seed}, b), [\![s]\!])$
Ensure: A signature $(\mathsf{salt}, z_2, z_3, \mathsf{msg})$
$\mathsf{salt} \xleftarrow{\$} \{0,1\}^{2\kappa}$
$u := H(\mathsf{msg}, \mathsf{salt}, \mathsf{vk})$
$a := \mathsf{ExpandA}(\mathsf{seed})$
$[\![\mathbf{p}]\!] \leftarrow \mathsf{AddRepNoise}(\mathcal{R}_q^2, d, \mathcal{D}_{\mathsf{pert}}^{\mathsf{ind}}, \mathsf{rep}_{\mathsf{pert}})$    ▷ $\mathbf{p} = (p_1, p_2) \in \mathcal{D}_{\mathsf{pert}}^2$
$[\![\mathbf{w}]\!] \leftarrow [\![p_1]\!] + a \cdot [\![p_2]\!]$
$w := \mathsf{Unmask}([\![w]\!])$
$c := u - w$
$(c_1, c_2) := \mathsf{Decompose}_\beta(c)$    ▷ $c = \beta \cdot c_1 + c_2$
$[\![s]\!] \leftarrow \mathsf{Refresh}([\![s]\!])$    ▷ Refresh $[\![s]\!]$ before re-use
$[\![z_2]\!] := [\![p_2]\!] + c_1 \cdot [\![s]\!]$
$z_2 := \mathsf{Unmask}([\![z_2]\!])$
$z_3 := c_1$
return $\mathsf{sig} := (\mathsf{salt}, z_2, z_3)$, $\mathsf{aux}_{\mathsf{sig}} = c_2$
    ▷ $\mathsf{aux}_{\mathsf{sig}}$ is used in security proof, but not in verification.
```

Verification. The verification first recovers $z_1' := u - az_2 - bz_3 = z_1 + c_2$. It then checks the shortness of (z_1', z_2, z_3).A formal description is given in Algorithm 7.

3.4 Security of Masked Plover-RLWE

We now turn to the security of the masked version of Plover, in the t-probing model. Contrary to proofs for less efficient masking techniques, which have no security loss even in the presence of the probes, we propose a fine-grained result where we quantify precisely the loss induced by the probes and show how the security of this leaky scheme corresponds to the security of the leak-free unmasked Plover, but with slightly smaller secret key parameters and slightly larger verification norm bound.

Theorem 3. *The masked* Plover-RLWE *scheme with parameters* $(d, \mathcal{D}_{\mathsf{sk}}^{\mathsf{ind}}, \mathsf{rep}_{\mathsf{sk}}, \mathcal{D}_{\mathsf{pert}}^{\mathsf{ind}}, \mathsf{rep}_{\mathsf{pert}}, B_2)$ *is t-probing* EUF-CMA *secure in the random oracle model if the unmasked* Plover-RLWE *scheme with parameters* $(\mathcal{D}_{\mathsf{sk}}, \mathcal{D}_{\mathsf{pert}}, B_2')$ *is* EUF-CMA *secure in the random oracle model, with*

$$\begin{cases} \mathcal{D}_{\mathsf{sk}} & := [d\,\mathsf{rep}_{\mathsf{sk}} - t] \cdot \mathcal{D}_{\mathsf{sk}}^{\mathsf{ind}}, \\ \mathcal{D}_{\mathsf{pert}} & := [d\,\mathsf{rep}_{\mathsf{pert}} - t] \cdot \mathcal{D}_{\mathsf{pert}}^{\mathsf{ind}} \\ B_2' & := B_2 + t \cdot (B_{\mathsf{pert}} + n(q/(2\beta) + 1/2)\, B_{\mathsf{sk}}), \end{cases} \tag{2}$$

where B_{pert} and B_{sk} denote upper bounds on the ℓ_2 norm of samples from $\mathcal{D}_{\mathsf{pert}}^{\mathsf{ind}}$ and $\mathcal{D}_{\mathsf{sk}}^{\mathsf{ind}}$, respectively. B_2' is the norm bound used by the unmasked Plover-RLWE.

Formally, let $\mathcal{A}$ denote an adversary against the t-probing EUF-CMA *security game against masked* Plover-RLWE *making at most Q_{Sign} signing queries and at most Q_H random oracle queries and advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{pr\text{-}EUF\text{-}CMA}}$. Then, there exists an adversary $\mathcal{A}'$ against* EUF-CMA *security of unmasked* Plover-RLWE*, running* *in time $T_{\mathcal{A}'} \approx T_{\mathcal{A}}$ and making $Q'_{\mathsf{Sign}} = Q_{\mathsf{Sign}}$ sign queries and $Q_{H'} = Q_H$ random oracle queries with advantage $\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A}'}$ such that:*

$$\mathsf{Adv}^{\mathsf{pr\text{-}EUF\text{-}CMA}}_{\mathcal{A}} \leqslant \mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A}'} + Q_{\mathsf{Sign}} Q_H / 2^{2\kappa}.$$

Proof. We describe the reduction with several hybrid games starting from the t-probing $\mathsf{EUF\text{-}CMA}$ game played with adversary $\mathcal{A}$ against the masked signature with random oracle H and ending with a game where we can build an adversary $\mathcal{A}'$ against the $\mathsf{EUF\text{-}CMA}$ security for the unmasked signature with a random oracle H'. In this and the following games we let S_i denote the event that $\mathcal{A}$ wins the t-probing $\mathsf{EUF\text{-}CMA}$ game.

Game_0. This corresponds to the t-probing $\mathsf{EUF\text{-}CMA}$ unforgeability game [6] played with adversary $\mathcal{A}$. At the beginning of the game, $\mathcal{A}$ outputs a key gen. probing set P_{KG} of size $\leqslant t$, then a masked key generation oracle OKG runs $\mathsf{MaskKeygen}(1^\kappa)$ to output $(\mathsf{vk} := (\mathsf{seed}, b), \mathsf{sk} := (\mathsf{vk}, [\![s]\!]))$ and $\mathcal{A}$ is given $(\mathsf{vk}, \mathcal{L}_{\mathrm{KG}})$, where $\mathcal{L}_{\mathrm{KG}} = \mathsf{MaskKeygen}_{P_{\mathrm{KG}}}$ denotes the observed values of the t probed variables during the execution of $\mathsf{Plover\text{-}RLWE.MaskKeygen}$ with oracle access to Algorithms 8 and 9 with adversary $\mathcal{A}$. In addition, the adversary is allowed to probe and learn the values of t variables during each execution of Algorithms 8 and 9.

The adversary gets access to a (masked) signing oracle $\mathsf{OSign}(m, P_S)$, where m is a message and P_S is a signing probing set of size at most t. The oracle returns $(\mathsf{sig}, \mathcal{L}_S)$ where $\mathsf{sig} \leftarrow \mathsf{Plover\text{-}RLWE.MaskSign}(m, \mathsf{sk})$ and $\mathcal{L}_S$ is the observed values of the t probed variables during the execution of $\mathsf{Plover\text{-}RLWE.MaskSign}$. Before each such OSign query, the $\mathsf{Refresh}([\![s]\!])$ gadget is called by the challenger to refresh the secret key shares (this challenger-run gadget is not probed by $\mathcal{A}$). The adversary can also query the random oracle H for the masked scheme. In this game, queries to the masked random oracle H are answered using an internal random oracle H' (not accessible directly to $\mathcal{A}$). The oracles in this game are similar to those in Fig. 2 but *without* the highlighted lines that are introduced in the following game. The adversary wins the game if it outputs a valid forgery message/signature pair $(\mathsf{msg}^*, \mathsf{sig}^*)$, where msg^* has not been queried to OSign.

Game_1 *(Fig. 2).* In this game, we change the computation of the probed observations $(\mathcal{L}_{\mathrm{KG}}, \mathcal{L}_S)$ given to $\mathcal{A}$, from the actual values to the values simulated by probabilistic polynomial time algorithms $\mathsf{SimKG}(P_{\mathrm{KG}}, \mathsf{aux}_{\mathrm{KG}})$ and $\mathsf{SimSig}(P_S, \mathsf{aux}_{\mathrm{MS}})$, respectively. The simulation algorithms simulate the probed values using auxiliary information $\mathsf{aux}_{\mathrm{KG}}$ (resp. $\mathsf{aux}_{\mathrm{MS}}$) consisting of public values and certain leaked internal values as indicated in the highlighted lines of Fig. 2. The main idea (see Section 4.2 of [41] for a similar proof) is that the internal t-probed observations in all the gadgets except $\mathsf{AddRepNoise}$ can be simulated without the secret shared inputs, whereas by SNI with unshared inputs property of $\mathsf{AddRepNoise}$ in Lemma 1, only $\leqslant t$ *unshared* inputs (captured by the auxiliary values $(\check{s}_i, \check{e}_i)_{i \in [t]}$ and $(\check{\mathbf{p}}_i)_{i \in [t]}$ in the masked key generation and signing algorithms, respectively) suffice to simulate its t-probed observations. Note that Game_1 writes z_1' as $z_1' = p_1 + c_1\, e + c_2$ instead of $z_1' = u - a\, z_2 - b\, z_3$; this is a purely syntactic change, as the two expressions are equal and we assume that the secret key includes the error e.

We construct the simulators SimKG and SimSig by composing the outputs of the simulators for each gadget, going from the last gadget to the first gadget, similar to the analysis in [6]. In the following description, we use the following notations: For the i'th gadget in SimKG (resp.SimSig), we let t_i denote the number of probed variables in this i'th gadget and by aux_i the auxiliary (leaked) information needed to simulate the internal view of the i'th gadget. Simulator SimKG for the probed observations $\mathcal{L}_{\mathrm{KG}}$ works as follows:

1. The Unmask($[\![b]\!]$) gadget (gadget 3) in Plover-RLWE.MaskKeygen is t-NIo (by Lemma 8 in [6]) with public output b. Hence, the probed observations in Unmask can be simulated by SimKG using $\leqslant t_3$ input shares in $[\![b]\!]$ and the auxiliary information $\mathsf{aux}_3 := b$.
2. The multiplication gadget $a \cdot [\![s]\!] + [\![e]\!]$ (gadget 2) in Plover-RLWE.MaskKeygen is computed share-wise and therefore is t-NI. Hence, the probed observations in this gadget can be simulated by SimKG using $\leqslant t_2 + t_3$ input shares in $[\![s]\!], [\![e]\!]$.
3. The AddRepNoise gadget in Plover-RLWE.MaskKeygen is t-SNIu with $d \cdot \mathsf{rep}$ unshared inputs $(r_{i,j})_{i \in [\mathsf{rep}], j \in [d]} := ((\widehat{s}_k, \widehat{e}_k)_{k \in [d \cdot \mathsf{rep} - t]}, (\check{s}_k, \check{e}_k)_{k \in [t]})$ by Lemma 1.

   Hence, the probed observations in AddRepNoise can be simulated by SimKG using $\leqslant t_1 + t_2 + t_3 \leqslant t$ leaked unshared inputs $(\check{s}_k, \check{e}_k)_{k \in [t]}$ (i.e. the set of safe (unleaked) unshared inputs of AddRepNoise are denoted by $(\widehat{s}_k, \widehat{e}_k)_{k \in [d \cdot \mathsf{rep} - t]}$).

Overall, SimKG can simulate the probed observations in P_{KG} using auxiliary information $\mathsf{aux}_{\mathrm{KG}} := (\mathsf{vk}, (\check{s}_i, \check{e}_i)_{i \in [t]})$, as shown in Fig. 2.

Similarly, simulator SimSig for the probed observations $\mathcal{L}_S$ works as follows:

1. The Unmask($[\![z_2]\!]$) gadget (gadget 6) in Plover-RLWE.MaskSign is t-NIo (by Lemma 8 in [6]) with public output z_2. Hence, the probed observations in Unmask can be simulated by SimSig using $\leqslant t_6$ input shares in $[\![z_2]\!]$ and the auxiliary information $\mathsf{aux}_6 := z_2$.
2. The multiplication gadget $[\![p_2]\!] + c_1 \cdot [\![s]\!]$ (gadget 5) in Plover-RLWE.MaskSign is t-NI. Hence, the probed observations in this gadget can be simulated by SimSig using $\leqslant t_5 + t_6 \leqslant t$ input shares in $[\![p_2]\!], [\![s]\!]$.
3. The Refresh($[\![s]\!]$) gadget (gadget 4) in Plover-RLWE.MaskSign is t-SNI (by [39]). Hence, the probed observations in this gadget can be simulated by SimSig using $\leqslant t_4 \leqslant t$ input shares in $[\![s]\!]$ (note that those t_4 input shares in $[\![s]\!]$ can be simulated by SimSig as independent uniformly random shares due to the Refresh($[\![s]\!]$) called by the challenger before each OSign call).
4. The Unmask($[\![w]\!]$) gadget (gadget 3) in Plover-RLWE.MaskSign is t-NIo (by Lemma 8 in [6]) with public output w. Hence, the probed observations in Unmask can be simulated by SimSig using $\leqslant t_3$ input shares in $[\![w]\!]$ and the auxiliary information $\mathsf{aux}_3 := w$.

5. The multiplication gadget $[\![p_1]\!] + a \cdot [\![p_2]\!]$ (gadget 5) in Plover-RLWE.MaskSign is t-NI. Hence, the probed observations in this gadget can be simulated by SimSig using $\leqslant t_2 + t_3 \leqslant t$ input shares in $[\![p_1]\!], [\![p_2]\!]$.
6. The AddRepNoise gadget (gadget 1) in Plover-RLWE.MaskSign is t-SNI with $d \cdot \mathsf{rep}$ unshared inputs $(\widehat{\mathbf{p}}_k)_{k \in [d \cdot \mathsf{rep} - t]}, (\check{\mathbf{p}}_k)_{k \in [t]})$ by Lemma 4. Hence, the probed observations in AddRepNoise can be simulated by SimSig using $\leqslant t_1 \leqslant t$ leaked unshared inputs $(\check{\mathbf{p}}_k)_{k \in [t]})$ (i.e. the set of safe (unleaked) unshared inputs of AddRepNoise are denoted by $(\widehat{\mathbf{p}}_k)_{k \in [d \cdot \mathsf{rep} - t]})$.

Overall, SimSig can simulate the probed observations in P_S using auxiliary information $\mathsf{aux}_{\mathrm{MS}} := (\mathsf{msg}, \mathsf{vk}, (\check{\mathbf{p}}_i)_{i \in [t]}, \mathsf{sig}, \mathsf{aux}_{\mathsf{sig}})$, as shown in Fig. 2. (note that $\mathsf{aux}_3 = w$ can be computed from $\mathsf{aux}_{\mathrm{MS}}$ since $w = u - c$, $u = H(\mathsf{msg}, \mathsf{salt}, \mathsf{vk})$ with salt taken from sig, and c computed from c_1 in sig and c_2 in $\mathsf{aux}_{\mathsf{sig}}$).

```
$\mathsf{OKG}(1^\kappa, P_{\mathrm{KG}}) \to (\mathsf{vk}, \mathsf{sk}, \mathcal{L}_{\mathrm{KG}})$
$\mathsf{seed} \xleftarrow{\$} \{0,1\}^\kappa$
$a := \mathsf{ExpandA}(\mathsf{seed})$
$(\widehat{s}, \widehat{e}) \leftarrow [d\,\mathsf{rep}_{\mathsf{sk}} - t] \cdot \mathcal{D}^{\mathsf{ind}}_{\mathsf{sk}}$    ▷ Safe
$(\check{s}_i, \check{e}_i)_{i \in [t]} \leftarrow (\mathcal{D}^{\mathsf{ind}}_{\mathsf{sk}})^t$    ▷ Leaked
$(s, e) := (\widehat{s} + \sum_{i \in [t]} \check{s}, \widehat{e} + \sum_{i \in [t]} \check{e}_i)$
$b := \beta - (a\,s + e)$
$\mathsf{vk} := (\mathsf{seed}, b)$
$\mathsf{sk} := (\mathsf{vk}, s)$
$\mathsf{aux}_{\mathrm{KG}} := (\mathsf{vk}, (\check{s}_i, \check{e}_i)_{i \in [t]})$
$\mathcal{L}_{\mathrm{KG}} \leftarrow \mathsf{SimKG}(P_{\mathrm{KG}}, \mathsf{aux}_{\mathrm{KG}})$
return $\mathsf{vk}, \mathsf{sk}, \mathcal{L}_{\mathrm{KG}}$
```

```
$H(\mathsf{msg}, \mathsf{salt}, \mathsf{vk}) \to u$
$u := H'(\mathsf{msg}, \mathsf{salt}, \mathsf{vk})$
return $u$
```

```
$\mathsf{OSign}(\mathsf{msg}, \mathsf{sk}, P_S) \to (\mathsf{sig}, \mathcal{L}_S)$
$\mathsf{salt} \xleftarrow{\$} \{0,1\}^{2\kappa}$
$u := H(\mathsf{msg}, \mathsf{salt}, \mathsf{vk})$
$a := \mathsf{ExpandA}(\mathsf{seed})$
$\widehat{\mathbf{p}} \leftarrow [d\,\mathsf{rep}_{\mathsf{pert}} - t] \cdot \mathcal{D}^{\mathsf{ind}}_{\mathsf{pert}}$    ▷ Safe
$(\check{\mathbf{p}}_i)_{i \in [t]} \leftarrow (\mathcal{D}^{\mathsf{ind}}_{\mathsf{pert}})^t$    ▷ Leaked
$\mathbf{p} := \widehat{\mathbf{p}} + \sum_{i \in [t]} \check{\mathbf{p}}_i$
$w := [1 \ a] \cdot \mathbf{p}$
$c := u - w$
$(c_1, c_2) := \mathsf{Decompose}_\beta(c)$
$z_2 := p_2 + c_1\,s$
$z_3 := c_1$
$z_1' := p_1 + c_1\,e + c_2$
$\mathsf{sig} := (\mathsf{salt}, z_2, z_3)$
$\mathsf{aux}_{\mathsf{sig}} := c_2$
$\mathsf{aux}_{\mathrm{MS}} := (\mathsf{msg}, \mathsf{vk}, (\check{\mathbf{p}}_i)_{i \in [t]}, \mathsf{sig}, \mathsf{aux}_{\mathsf{sig}})$
$\mathcal{L}_S \leftarrow \mathsf{SimSig}(P_S, \mathsf{aux}_{\mathrm{MS}})$
return $(\mathsf{sig}, \mathcal{L}_S)$
```

Fig. 2: Algorithms in Game_1

Since the view of $\mathcal{A}$ is perfectly simulated in this game as in the previous game, we have $\Pr[S_1] = \Pr[S_0]$.

Game_2 *(Fig. 3).* In this game, we re-arrange the computation in OKG to first compute a 'safe' verification key $\widehat{b} := \beta - (a\widehat{s} + \widehat{e})$ using the 'safe' part $(\widehat{s}, \widehat{e})$ of the secret key, and only later sample the 'leaked' part $(\check{s}, \check{e}) := \sum_{i \in [t]} (\check{s}_i, \check{e}_i)$ of the secret key and use this leaked secret and $\widehat{b}$ to compute the full verification key $b := \widehat{b} - (a\check{s} + \check{e})$. The above change to OKG is just a re-ordering of the computation and thus does not change the view of $\mathcal{A}$.

In this game, we also similarly re-arrange the computation in OSign to first compute a 'safe' part of the signature $\widehat{\mathsf{sig}}$ with $\widehat{z}_2 = \widehat{p}_2 + \widehat{c}_1\widehat{s}$, using the 'safe' perturbation part $\widehat{p}_2$ and 'safe secret key part $\widehat{s}$, and later compute the full signature sig from the $\widehat{z}_2$ by adding the 'leaked' signature part to get $z_2 = \widehat{z}_2 + \sum_{i\in[t]} \check{p}_{i,2} + c_1 \widehat{\sum}_{i\in[t]\check{s}_i} = (\widehat{p}_2 + \check{p}_2) + \widehat{c}_1\widehat{s} + c_1\check{s} = (\widehat{p}_2 + \check{p}_2) + c_1(\widehat{s} + \check{s})$, where the last equality holds if $\widehat{c} = c$. Hence, for this re-arranged computation to preserve the correctness of the final signature (in particular z_2) as in the previous game (and thus preserve $\mathcal{A}$'s view), we need to ensure that $\widehat{c} := \widehat{u} - \widehat{w}$ in the top 'safe' part of the computation, is equal to $c := u - w$ used in the bottom 'leaked' part of the computation. To achieve this, we use the random oracle H' (not directly accessible to $\mathcal{A}$) to compute $\widehat{u} := H'(\mathsf{msg}, \mathsf{salt}, \mathsf{vk})$ in the 'safe' part of the computation, and we change the simulation of the random oracle H accessible to $\mathcal{A}$ by programming H so that $u = H(\mathsf{msg}, \mathsf{salt}, \mathsf{vk}) := \widehat{u} + \begin{bmatrix}1 & a\end{bmatrix} \cdot \check{\mathbf{p}}$, where $\check{\mathbf{p}}$ is sampled by the simulation and stored in the table $\mathcal{T}_H$ for H. Defining $\check{w} := \begin{bmatrix}1 & a\end{bmatrix} \cdot \check{\mathbf{p}}$, we have $c = u - w = (\widehat{u} + \check{w}) - (\widehat{w} + \check{w}) = \widehat{u} - \widehat{w} = \widehat{c}$, as required.

Since $\widehat{u} := H'(\mathsf{msg}, \mathsf{salt}, \mathsf{vk})$ is uniformly random in R_q and independent of $\begin{bmatrix}1 & a\end{bmatrix} \cdot \check{\mathbf{p}}$, the simulation of H is identical to the previous game from $\mathcal{A}$'s view, except if an abort happens in OSign line 18 (we say then that the event B_2 occurs). However, since salt is uniformly random in $\{0,1\}^{2\kappa}$ for each sign query, the event B_2 occurs with negligible probability $\Pr[B_2] \leqslant Q_{\mathsf{Sign}} Q_H / 2^{2\kappa}$. Therefore, overall we have $\Pr[S_2] \geqslant \Pr[S_1] - \Pr[B_2] \geqslant \Pr[S_1] - Q_{\mathsf{Sign}} Q_H / 2^{2\kappa}$.

We now construct an adversary $\mathcal{A}'$ against the $\mathsf{EUF\text{-}CMA}$ of the unmasked signature scheme Sign with random oracle H', secret key distribution $\mathcal{D}_{\mathsf{sk}} := [d\,\mathsf{rep}_{\mathsf{sk}} - t] \cdot \mathcal{D}_{\mathsf{sk}}^{\mathsf{ind}}$, and perturbation distribution $\mathcal{D}_{\mathsf{pert}} := [d\,\mathsf{rep}_{\mathsf{pert}} - t] \cdot \mathcal{D}_{\mathsf{pert}}^{\mathsf{ind}}$ that simulates view of $\mathcal{A}$ in Game_2, such that $\mathcal{A}'$ wins its game with probability $\geqslant \Pr[S_2]$. The challenger for $\mathcal{A}'$ generates a challenge key pair $(\widehat{\mathsf{vk}}, \widehat{\mathsf{sk}})$ by running lines 1-6 of OKG in Game_2 (this corresponds exactly to the key gen. algorithm for the unmasked scheme) and runs $\mathcal{A}'$ on input vk'. Then $\mathcal{A}'$ runs as follows.

1. It first runs $\mathcal{A}$ to get P_{KG} and then runs lines 7-12 of OKG in Game_2 to get $(\mathsf{vk}, \mathsf{sk}, \mathcal{L}_{\mathrm{KG}})$ and runs $\mathcal{A}$ on input $(\mathsf{vk}, \mathcal{L}_{\mathrm{KG}})$.
2. Similarly, to respond to each OSign query (msg, P_S) of $\mathcal{A}$, $\mathcal{A}'$ calls its Sign algorithm on input msg (this corresponds to running lines 1-11 of OKG in Game_2), and using the returned $\widehat{\mathsf{sig}}$ and $\widehat{\mathsf{aux}}_{\mathsf{sig}}$, $\mathcal{A}'$ runs lines 12-26 of OSign in Game_2 to compute and return $(\mathsf{sig}, \mathcal{L}_S)$ to $\mathcal{A}$ (note that $\widehat{w} = \widehat{u} - \widehat{c}$ is computed by $\mathcal{A}'$ from $\widehat{u} = H'(\mathsf{msg}, \mathsf{salt}, \widehat{\mathsf{vk}})$ and $\widehat{c}$ obtained from c_1 in $\widehat{\mathsf{sig}}$ and c_2 in $\widehat{\mathsf{aux}}_{\mathsf{sig}}$).
3. $\mathcal{A}'$ also runs the H simulator in Game_2 to respond to $\mathcal{A}$'s H queries, where H' is the random oracle provided to $\mathcal{A}'$ by its challenger.

Consequently, the view of $\mathcal{A}$ is perfectly simulated as in Game_2, so with probability $\Pr[S_2]$, $\mathcal{A}$ outputs a valid forgery $(\mathsf{msg}^*, \mathsf{sig}^* = (\mathsf{salt}^*, z_2^*, c_1^*))$ such that

```
$\mathsf{OKG}(1^\kappa) \to (\mathsf{vk}, \mathsf{sk}, \mathcal{L}_{\mathrm{KG}})$
$\mathsf{seed} \xleftarrow{\$} \{0,1\}^\kappa$
$a := \mathsf{ExpandA}(\mathsf{seed})$
$(\widehat{s}, \widehat{e}) \leftarrow [d\,\mathsf{rep}_{\mathsf{sk}} - t] \cdot \mathcal{D}_{\mathsf{sk}}^{\mathsf{ind}}$    $\triangleright$ Safe
$\widehat{b} := \beta - (a\,\widehat{s} + \widehat{e})$
$\widehat{\mathsf{vk}} := (a, \widehat{b})$
$\widehat{\mathsf{sk}} := (\widehat{\mathsf{vk}}, \widehat{s})$
$(\check{s}_i, \check{e}_i)_{i\in[t]} \leftarrow \left(\mathcal{D}_{\mathsf{sk}}^{\mathsf{ind}}\right)^t$    $\triangleright$ Leaked
$b := \widehat{b} - (a \sum_{i\in[t]} \check{s} + \sum_{i\in[t]} \check{e}_i)$
$(s, e) := (\widehat{s} + \sum_{i\in[t]} \check{s}, \widehat{e} + \sum_{i\in[t]} \check{e}_i)$
$\mathsf{vk} := (a, b)$
$\mathsf{sk} := (\mathsf{vk}, s)$
$\mathsf{aux}_{\mathrm{KG}} := (\mathsf{vk}, (\check{s}_i, \check{e}_i)_{i\in[t]})$
$\mathcal{L}_{\mathrm{KG}} := \mathsf{SimKG}(P_{\mathrm{KG}}, \mathsf{aux}_{\mathrm{KG}})$
**return** $\mathsf{vk}, \mathsf{sk}, \mathcal{L}_{\mathrm{KG}}$
```

```
$H(\mathsf{msg}, \mathsf{salt}, \mathsf{vk}) \to u$
$\widehat{u} := H'(\mathsf{msg}, \mathsf{salt}, \widehat{\mathsf{vk}})$
**if** $\exists(\check{\mathbf{p}}, u) : ((\mathsf{msg}, \mathsf{salt}, \mathsf{vk}), u, \check{\mathbf{p}}) \in \mathcal{T}_H$
    **then** **return** $\widehat{u} + [1\ a] \cdot \check{\mathbf{p}}$
$\check{\mathbf{p}} \leftarrow [t] \cdot \mathcal{D}_{\mathsf{pert}}^{\mathsf{ind}}$
$u := \widehat{u} + [1\ a] \cdot \check{\mathbf{p}}$
Add $((\mathsf{msg}, \mathsf{salt}, \mathsf{vk}), u, \check{\mathbf{p}})$ to $\mathcal{T}_H$
**return** $u$
```

```
$\mathsf{OSign}(\mathsf{msg}, \mathsf{sk}, P_S) \to (\mathsf{sig}, \mathcal{L}_S)$
$\mathsf{salt} \xleftarrow{\$} \{0,1\}^{2\kappa}$
$\widehat{u} := H'(\mathsf{msg}, \mathsf{salt}, \widehat{\mathsf{vk}})$
$a := \mathsf{ExpandA}(\mathsf{seed})$
$\widehat{\mathbf{p}} \leftarrow [d\,\mathsf{rep}_{\mathsf{pert}} - t] \cdot \mathcal{D}_{\mathsf{pert}}^{\mathsf{ind}}$    $\triangleright$ Safe
$\widehat{w} \leftarrow [1\ a] \cdot \widehat{\mathbf{p}}$
$\widehat{c} \leftarrow \widehat{u} - \widehat{w}$
$(\widehat{c}_1, \widehat{c}_2) := \mathsf{Decompose}_\beta(\widehat{c})$
$\widehat{z}_2 := \widehat{p}_2 + \widehat{c}_1\,\widehat{s}$
$\widehat{z}_3 := \widehat{c}_1$
$\widehat{z}_1' := \widehat{p}_1 + \widehat{c}_1\,\widehat{e} + \widehat{c}_2$
$\widehat{\mathsf{sig}} := (\mathsf{salt}, \widehat{z}_2, \widehat{z}_3)$
$\widehat{\mathsf{aux}_{\mathrm{sig}}} := \widehat{c}_2$
$(\check{\mathbf{p}}_i)_{i\in[t]} \leftarrow \left(\mathcal{D}_{\mathsf{pert}}^{\mathsf{ind}}\right)^t$    $\triangleright$ Leaked
$\check{\mathbf{p}} = \sum_{i\in[t]} \check{\mathbf{p}}_i$
$u := \widehat{u} + [1\ a] \cdot \check{\mathbf{p}}$
$w := \widehat{w} + [1\ a] \cdot \check{\mathbf{p}}$
$(c_1, c_2) := \mathsf{Decompose}_\beta(u - w)$
**if** $((\mathsf{msg}, \mathsf{salt}, \mathsf{vk}), u, \check{\mathbf{p}}) \in \mathcal{T}_H$ **then**
    **return** $\perp$    $\triangleright$ Abort
Add $((\mathsf{msg}, \mathsf{salt}, \mathsf{vk}), u, \check{\mathbf{p}})$ to $\mathcal{T}_H$
$z_2 := \widehat{z}_2 + \sum_{i\in[t]} \check{p}_{i,2} + c_1 \widehat{\sum}_{i\in[t]} \check{s}_i$
$z_3 := c_1$
$z_1' := \widehat{z}_1' + \sum_{i\in[t]} \check{p}_{i,1} + c_1 \widehat{\sum}_{i\in[t]} \check{e}_i$
$\mathsf{sig} := (\mathsf{salt}, z_2, z_3)$
$\mathsf{aux}_{\mathrm{sig}} := c_2$
$\mathsf{aux}_{\mathrm{MS}} := (\mathsf{vk}, (\check{\mathbf{p}}_i)_{i\in[t]}, \mathsf{sig}, \mathsf{aux}_{\mathrm{sig}})$
$\mathcal{L}_S := \mathsf{SimSig}(P_S, \mathsf{aux}_{\mathrm{MS}})$
**return** $(\mathsf{sig}, \mathcal{L}_S)$
```

Fig. 3: Algorithms in Game_2

$\|(z_1'^*, z_2^*, z_3^*)\| \leqslant B_2$, and $\|c_1^*\|_\infty \leqslant q/(2\beta) + 1/2$ and $z_1'^* + az_2^* + bc_1^* = u^* = H(\mathsf{msg}^*, \mathsf{salt}^*, \mathsf{vk})$ where msg^* has not been queried by $\mathcal{A}$ to OSign. Then, $\mathcal{A}'$

computes $(\check{z}_1'^*, \check{z}_2^*) = \check{\mathbf{p}} + c_1^*(\breve{e}, \check{s})$ with $(\check{s}, \breve{e}) = (\sum_{i\in[t]} \check{s}_i)$ and returns its forgery $(\mathsf{msg}^*, \widehat{\mathsf{sig}}^* = (\mathsf{salt}^*, \hat{z}_2^*, c_1^*))$, where $(\hat{z}_1'^*, \hat{z}_2^*) := (z_1'^*, z_2^*) - (\check{z}_1'^*, \check{z}_2^*)$.

Note that, defining $\breve{w}^* := \begin{bmatrix}1 & a\end{bmatrix} \cdot \check{\mathbf{p}}^*$ and $\check{b} := a\check{s} + \breve{e}$ (where $\check{\mathbf{p}}^*$ is obtained from $\mathcal{T}_H$ entry for the forgery H-query $(\mathsf{msg}^*, \mathsf{salt}^*, \mathsf{vk})$), we have $\check{z}_1'^* + a\check{z}_2^* - \check{b}c_1^* = \breve{w}^*$ and so forgery $\widehat{\mathsf{sig}}^*$ satisfies the unmasked scheme validity relation $\hat{z}_1'^* + a\hat{z}_2^* + \hat{b}c_1^* = (z_1'^* + az_2^* + bc_1^*) - (\check{z}_1'^* + a\check{z}_2^* - \check{b}c_1^*) = u^* - \breve{w}^* = H'(\mathsf{msg}^*, \mathsf{salt}^*, \widehat{\mathsf{vk}})$, as required. Also, $\|(\hat{z}_1'^*, \hat{z}_2^*, c_1^*)\| \leqslant \|(z_1'^*, z_2^*, c_1^*)\| + \|(\check{z}_1'^*, \check{z}_2^*, 0)\| \leqslant B_2 + t \cdot (B_{\mathsf{pert}} + n\frac{q}{2\eta}B_{\mathsf{sk}}) := B_2'$, since $\|(\check{z}_1'^*, \check{z}_2^*)\| \leqslant \|\check{\mathbf{p}}\| + n\|c_1^*\|_\infty \|(\breve{e}, \check{s})\| \leqslant (tB_{\mathsf{pert}} + n(q/(2\beta) + 1/2)tB_{\mathsf{sk}})$. Finally, msg^* has not been queried by $\mathcal{A}'$ to its unmasked signing oracle. It follows that $\mathcal{A}'$ wins with probability $\geqslant \Pr[S_2] \geqslant \Pr[S_0] - Q_{\mathsf{Sign}} Q_H / 2^{2\kappa}$. This concludes the proof. □

3.5 Cryptanalysis and Parameter Selection

Now that the security of our scheme is formally proven in unmasked form for general distributions $\mathcal{D}_{\mathsf{sk}}, \mathcal{D}_{\mathsf{pert}}$ and the security of the masked form reduces to its unmasked form, we wish to demonstrate concrete parameter selection for masked $\mathsf{Plover\text{-}RLWE}$. We evaluate the concrete security of our scheme against RSIS for forgery, and against $\mathsf{Hint\text{-}RLWE}$ for key-indistinguishability using the reduction from $\mathsf{Hint\text{-}RLWE}$ to RLWE (Theorem 1) and standard evaluation heuristics.

Optimizations. For our implementation, we use these standard optimizations:

- **Norm check.** We add a norm check in $\mathsf{MaskSign}$ against B_2, allowing to reject with low probability some large signatures, and making forgery harder. Note that this is *not* rejection sampling, and it can be done unmasked.
- **Bit-dropping.** We can drop the ν least significant bits of b. More formally, let us note $(b_1, b_2) = \mathsf{Decompose}_{\{2^\nu\}}(b)$ where ν is the number of bits dropped in each coefficient of b. We can set $2^\nu \cdot b_1$ as a public key.
  As long as $\nu = O\left(\log\left(\frac{\sigma_{\mathsf{pert}}^2}{q\sqrt{n}}\right)\right)$, we can show that breaking inhomogeneous RSIS for $\begin{bmatrix}1 & a & 2^\nu \cdot b_1\end{bmatrix}$ implies breaking it for $\begin{bmatrix}1 & a & b\end{bmatrix}$ with comparable parameters. This reduces the size of vk, while preserving the security reduction.

Forgery Attacks and Practical RSIS Security Let $\sigma_{\mathsf{sk}}, \sigma_{\mathsf{pert}}$ denote the standard deviation of the (unmasked) secret key and perturbation, respectively. In a legitimate signature:

$$\begin{aligned}\mathbb{E}\left[\|\mathbf{z}'\|^2\right] &= \mathbb{E}\left[\|p_1 + e\cdot c_1 + c_2 + b_2 \cdot c_1\|^2\right] + \mathbb{E}\left[\|p_2 + s\cdot c_1\|^2\right] + \mathbb{E}\left[\|c_1\|^2\right] \\ &\approx n\left(2\,\sigma_{\mathsf{pert}}^2 + \frac{\beta^2}{12} + \frac{q^2\,n}{6\,\beta^2}\,\sigma_{\mathsf{sk}}^2 + n\frac{2^{2\nu}}{12}\frac{q^2}{12\beta^2}\right)\end{aligned}$$

Based on this analysis, we set $B_2 = 1.2\sqrt{n\left(2\,\sigma_{\mathsf{pert}}^2 + \frac{\beta^2}{12} + \frac{q^2\,n}{6\,\beta^2}\,\sigma_{\mathsf{sk}}^2 + n\frac{2^{2\nu}}{12}\frac{q^2}{12\beta^2}\right)}$. The "slack" factor 1.2 allows an extremely large number of generated signatures to satisfy $\|\mathbf{z}'\| \leqslant B_2$, which means that the restart rate will be very low.

Solving Inhomogeneous RSIS. To forge a message, an adversary must either break the collision resistance of H or solve the equation:

$$\left(\begin{bmatrix}1 \; a \; \beta \cdot b_1\end{bmatrix} \cdot \mathbf{z}' = u\right) \wedge \left(\|\mathbf{z}'\| \leqslant B_2\right) \tag{3}$$

Note that $\begin{bmatrix}1 \; a \; \beta \cdot b_1\end{bmatrix} \cdot \mathbf{z}' = \begin{bmatrix}1 \; a \; b\end{bmatrix} \cdot \mathbf{z}''$, where $\mathbf{z}'' = \mathbf{z}' - (z_3 \cdot b_2, 0, 0)$, and that $\|z_3 \cdot b_2\| \leqslant \|c_1\|_1 \cdot \|b_2\| \leqslant n^{3/2} \cdot \frac{q\, 2^{\nu-2}}{\beta}$. Then Eq. (3) is an instance of the inhomogeneous RSIS problem, with a bound $B_{\mathsf{RSIS}} = B_2 + n^{3/2} \cdot \frac{q\, 2^{\nu-2}}{\beta}$.

We estimate its hardness based on Chuengsatiansup et al. [12] and Espitau and Kirchner [19]. Under the geometric series assumption, [19, Theorem 3.3] states that Eq. (3) can be solved in $\mathsf{poly}(n)$ calls to a CVP oracle in dimension B_{BKZ}, as long as:

$$B_{\mathsf{RSIS}} \leqslant \left(\delta_{B_{\mathsf{RSIS}}}^{3n}\, q^{1/3}\right), \quad \text{where} \quad \delta_{B_{\mathsf{RSIS}}} = \left(\frac{(\pi \cdot B_{\mathsf{BKZ}})^{1/B_{\mathsf{BKZ}}} \cdot B_{\mathsf{BKZ}}}{2\pi e}\right)^{1/(2(B_{\mathsf{BKZ}}-1))}. \tag{4}$$

This attack has been optimized in [12] by omitting $x \leqslant n$ of the first columns of $\mathbf{A}$ (when considered as a $n \times 3\,n$ matrix). The dimension is reduced by x, however, the co-volume of the lattice is increased to $q^{\frac{n}{3n-x}}$. This strengthens Eq. (4) to the more stringent condition $B_{\mathsf{RSIS}} \leqslant \min_{x \leqslant n}\left(\delta_{B_{\mathsf{RSIS}}}^{3n-x} q^{\frac{n}{3n-x}}\right)$.

Key-Indistinguishability and Hint-RLWE In order to apply Theorem 1, we need quantitative bounds on B_{HRLWE}. These are given in Lemma 2, which is a minor adaptation of [42, Lemma B.2]. A proof is provided in Appendix B for completeness.

Lemma 2. *For* $j \in [Q_{\mathsf{Sign}}]$*, let* $c^{[j]} \leftarrow \mathcal{C}_1$*, where* $\mathcal{C}_1$ *is defined as in Definition 9. Let* $D = \sum_{j \in [Q_{\mathsf{Sign}}]} c^{[j]}\,(c^{[j]})^*$*. Let* $M = 2\left\lceil \frac{q-1}{2\beta} \right\rceil + 1$*. We then have* $\Pr\left[s_1(D) \geqslant B_{\mathsf{HRLWE}}\right] \leqslant 2^{-\kappa}$*, where* $B_{\mathsf{HRLWE}} = \frac{Q_{\mathsf{Sign}}\, n\, M^2}{12}\left(1 + \frac{O(\kappa\, n \log n)}{\sqrt{Q_{\mathsf{Sign}}}}\right)$*. Specifically, when* $Q_{\mathsf{Sign}} = \omega(\kappa\, n \log n)^2$*, then* $s_1(D)$ *is equivalent to* $\frac{Q_{\mathsf{Sign}}\, n\, M^2}{12}$.

Advantage against $\mathsf{Hint\text{-}RLWE}$. An adversary breaking the key-indistinguishability of vk is also able to break $\mathsf{Hint\text{-}RLWE}_{q, Q_{\mathsf{Sign}}, \widehat{\mathcal{D}_{\mathsf{sk}}}, \widehat{\mathcal{D}_{\mathsf{pert}}}, \mathcal{C}}$. In the Gaussian case, $\mathcal{D}_{\mathsf{sk}} \overset{s}{\sim} D_{\widehat{\sigma}_{\mathsf{sk}}}$ and $\mathcal{D}_{\mathsf{pert}} \overset{s}{\sim} D_{\widehat{\sigma}_{\mathsf{pert}}}$, where $\frac{\widehat{\sigma}_{\mathsf{sk}}}{\sigma_{\mathsf{sk,ind}}} = \frac{\widehat{\sigma}_{\mathsf{pert}}}{\sigma_{\mathsf{pert,ind}}} = \sqrt{d\, \mathsf{rep} - t}$.

Theorem 1 and Lemma 2 state that such an adversary is also able to break $\mathsf{RLWE}_{q, D_{\sigma_{\mathsf{red}}}}$, where $\frac{1}{\sigma_{\mathsf{red}}^2} = 2\left(\frac{1}{\widehat{\sigma}_{\mathsf{sk}}^2} + \frac{B_{\mathsf{HRLWE}}}{\widehat{\sigma}_{\mathsf{pert}}^2}\right)$ and B_{HRLWE} is as in Lemma 2. For the parameters we choose in practice, this entails: $\frac{\sigma_{\mathsf{red}}}{\widehat{\sigma}_{\mathsf{pert}}} \approx \frac{\beta}{q}\sqrt{\frac{6}{n\, Q_{\mathsf{Sign}}}}$ Estimating the concrete hardness of RLWE is well-documented. We rely on the lattice estimator [3], an open-source tool available at `https://github.com/malb/lattice-estimator`.

Parameter Selection Despite the many variables involved, parameter selection is fairly straightforward. We set $\beta = \Theta(\sigma_{\mathsf{pert}})$, $\nu = \Theta\left(\log\left(\frac{\sigma_{\mathsf{pert}}^2}{q\sqrt{n}}\right)\right)$ and $\sigma_{\mathsf{sk}} = o\left(\frac{\beta\,\sigma_{\mathsf{pert}}}{q\sqrt{n}}\right)$. This guarantees efficiency while ensuring that $B_{\mathsf{RSIS}} = O(\sigma_{\mathsf{pert}}\sqrt{n})$.

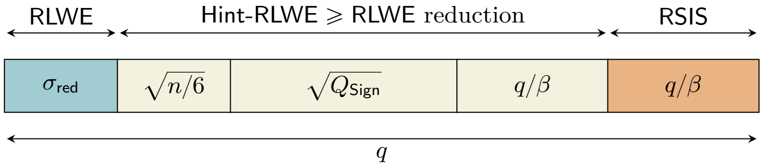

Fig. 4: Illustration of the constraints on q (in log scale): RSIS and RLWE must be hard, and the Hint-RLWE ⩾ RLWE reduction must be non-vacuous.

These parameters also guarantee an efficient reduction in Theorem 1. We estimate the number of queries, Q_{Sign}, by increasing it for as long as the RLWE instance entailed by the reduction of Theorem 1 and Lemma 2 remains secure according to the state-of-the-art. Q'_{Sign} corresponds to the number of queries allowed when the condition $\sigma \geqslant \sqrt{2}\eta_\varepsilon(\mathbb{Z}^n)$ is dropped in Theorem 1.

$\lceil\log q\rceil$	35	37	39	41	43	45	47	49
$\log\beta$	31	33	35	37	38	39	40	41
$\log\sigma_{\mathsf{pert}}$	30	32	34	36	37	38	39	40
$\log\sigma_{\mathsf{sk}}$	21	23	25	27	27	27	27	27
ν	15	17	19	21	21	21	21	21
Q_{Sign}	2^{40}	2^{44}	2^{48}	2^{52}	2^{52}	2^{50}	2^{48}	2^{46}
Q'_{Sign}	2^{46}	2^{48}	2^{52}	2^{54}	2^{52}	2^{50}	2^{48}	2^{46}
$\lvert\mathsf{vk}\rvert$	5136	5136	5136	5136	5648	6160	6672	7184
$\lvert\mathsf{sig}\rvert$	11488	12198	12908	13617	13972	14327	14682	15037

Table 1: Parameter sets for $\kappa = 128$. All parameter sets feature $n = 2048$.

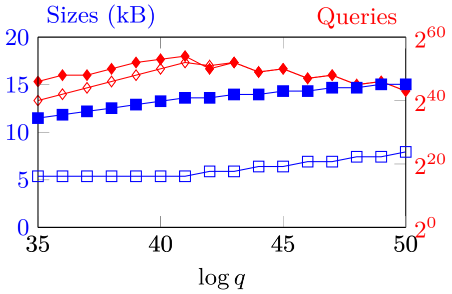

Fig. 5: Number of signing queries (conservative: −◇−, standard: −◆−) and bytesizes (|vk|: −□−, |sig|: −■−) as functions of q. Parameter sets as in Section 3.5.

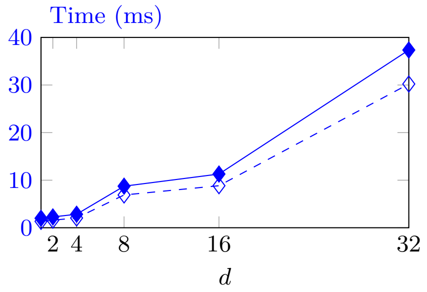

Fig. 6: Timings of Plover-RLWE (Keygen: -◇-, Sign: −◆−) as functions of d. Parameter set from Section 3.5 with $\lceil\log q\rceil = 41$, and concrete parameters from Table 2.

3.6 Implementation

We provide both a Python and a C reference implementation for Plover-RLWE, available at https://github.com/GuilhemN/masksign-plover. They are designed to match the high-level pseudo-code from Subsection 3.3 and allows one to read a concrete implementation of each of the functions we introduced. The Python implementation aims for simplicity and is not constant-time, while the C implementation is constant-time and uses optimization techniques. We include scripts for parameters selection under the folder *params* based on the lattice estimator [3].

These reference implementations re-use several components of Raccoon reference implementations [41] for the NTT, Montgomery modular reduction, and randomness generators. They are portable and can target various masking orders $d-1$. Note however that they suffer from the same issues as Raccoon reference implementations. Specifically, a deterministic portable code written in a high-level language cannot realistically be considered to be fully resistant to side-channel attacks, and notably due to the use of the *randombytes* function defined by NIST, which represents an abstract RBG (Random Bit Generator), but is only suitable to ease reproducibility and generation of test vectors. Additionally, our reference implementations are severely limited in their key management as the NIST API does not allow for a refresh of the secret key, which is required for t-probing security. We argue that these implementations still provide evidence that Plover-RLWE is easy to mask at high masking orders.

General Implementation Characteristics Plover-RLWE has building blocks resembling those of Raccoon [41], as well as a modulus q of same magnitude and format (product of two Solinas primes). In particular we reuse part of their codebase and of their implementation tricks. The C reference implementation uses Montgomery modular reduction to implement efficient constant-time modular operations. In this case, the table of pre-computed roots of unity is multiplied by the Montgomery factor r in order to remove half of the Montgomery reductions that would otherwise be required by the NTT transform algorithm. For sampling random shares, we reuse the placeholder generator based on 127-bit LFSR packaged with Raccoon's code. This generator is completely deterministic and is only provided for evaluation purposes.

Signature Encoding. We encode low-order bits using binary encoding, and high-order bits using Huffman/unary-type encoding. This encoding is similar to the ones in Falcon and Raccoon. We chose this technique over ANS encoding – although ANS could compress signatures further – as the latter proved hard to implement securely in NIST Call for Additional Digital Signature Schemes, with vulnerabilities discovered in the HuFu and HAETAE proposals[3].

[3] See https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/Hq-wRFDbIaU.

Mask Compression Technique. Our implementation uses the mask compression technique introduced in [46,41] in order to reduce the size of the stored secret key, which contains the masked polynomial $[\![s]\!]$. A masked polynomial at order d can be compressed into one polynomial and $d-1$ seeds which can be later expanded into full polynomial masking shares. We refer to [41, Algorithms 14 and 15] and [46] for a detailed specification of this technique.

This technique could also be used to to drastically reduce the memory requirements of Plover-RLWE. Our masking gadgets can be adapted to do runtime computations on compressed masked polynomials to limit the impact of a larger d on memory requirements. For reference, Raccoon [41, section 3.3.2] reduced memory usage for a masking order $d = 32$ by a factor of 15 using this technique.

Hardware. Plover-RLWE could be implemented on hardware in a similar manner to Raccoon. Several versions of Raccoon were implemented on FPGA architecture, one is reported in [43]. These implementations contain a RISC-V controller, a Keccak accelerator, and a lattice unit with direct memory access via a 64-bit interface, using hard-coded support for Raccoon's arithmetic modulus q. Plover-RLWE can share a large part of these implementations.

As for Raccoon [41, section 3.3.1] the usage of SHAKE as hash function in the implementation of ExpandA and AddRepNoise can be highly optimized in hardware, and the hardware XOF (eXtendable-Output Functions) sampler can implement a full Keccak round and produce output at a very high rate.

Performance. We evaluated the performance of Plover-RLWE on a Ryzen Pro 7 5850U (16CPU threads at 3GHz), boost disabled, and running Manjaro 22.1. The results are provided in Table 2 and Fig. 6. The reference implementation instantiates the parameter set from Table 1 such that $\lceil \log q \rceil = 41$, as it is optimal for the number of possible queries for $n = 2048$ and $\kappa = 128$. Other parameter sets perform very similarly since – performance-wise – only the encoding differs between them. The implementation packages parameters for d shares, $d \in \{1, 2, 4, 8, 16, 32\}$, and the distributions $\mathcal{D}_{\mathsf{sk}}$ and $\mathcal{D}_{\mathsf{pert}}$ are sums of uniforms $\mathrm{SU}(u_{\mathsf{sk}}, d \cdot \mathsf{rep}_{\mathsf{sk}})$ and $\mathrm{SU}(u_{\mathsf{pert}}, d \cdot \mathsf{rep}_{\mathsf{pert}})$ with $\mathsf{rep} := \mathsf{rep}_{\mathsf{sk}} = \mathsf{rep}_{\mathsf{pert}} \in \{2, 4, 8\}$ a function of d. u_{sk} and u_{pert} are chosen as to achieve a standard deviation close to σ_{sk} and σ_{pert}. Plover-RLWE has performance very similar to Raccoon; in particular, we observe a (quasi-)linear increase in the execution times and stack usage of our functions with d, which makes the use of a high masking order practical. For instance, Plover-RLWE masked with a number of shares $d = 8$ still performs better than Dilithium masked with $d = 2$ [41, Table 6].

References

1. Alagic, G., Apon, D., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Miller, C., Moody, D., Peralta, R., Perlner, R., Robinson, A., Smith-Tone, D., Liu, Y.K.: NISTIR 8413 – Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process (2022), https://doi.org/10.6028/NIST.IR.8413

Variant	**Parameters**			**Keygen**			**Sign**			**Verify**		
$\kappa - d$	rep	u_{sk}	u_{pert}	ms	Mclk	stack	ms	Mclk	stack	ms	Mclk	stack
128-1	8	27	36	1.341	2.546	49312	1.989	3.788	164128	0.432	0.820	32864
128-2	4	27	36	1.595	3.030	114848	2.272	4.316	246048	=	=	=
128-4	2	27	36	2.045	3.885	213184	2.835	5.386	410016	=	=	=
128-8	4	26	35	6.887	13.083	409856	8.732	16.588	737760	=	=	=
128-16	2	26	35	8.832	16.782	803200	11.288	21.460	1393248	=	=	=
128-32	4	25	34	30.213	57.404	1589888	37.350	70.959	2704224	=	=	=

Table 2: Performance of the Plover-RLWE reference implementation for different masking orders on our reference platform. Across all parameter sets, we have $(\kappa, n, \lceil \log q \rceil, \log \beta, \nu) = (128, 2048, 41, 37, 21)$, and we set $\mathsf{rep}_{\mathsf{sk}} = \mathsf{rep}_{\mathsf{pert}} = \mathsf{rep}$.

2. Albrecht, M.R., Bai, S., Ducas, L.: A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 153–178. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53018-4_6
3. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. **9**(3), 169–203 (2015), http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml
4. Azouaoui, M., Bronchain, O., Cassiers, G., Hoffmann, C., Kuzovkova, Y., Renes, J., Schneider, T., Schönauer, M., Standaert, F., van Vredendaal, C.: Protecting dilithium against leakage revisited sensitivity analysis and improved implementations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(4), 58–79 (2023). https://doi.org/10.46586/tches.v2023.i4.58-79
5. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.A., Grégoire, B., Strub, P.Y., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 116–129. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978427
6. Barthe, G., Belaïd, S., Espitau, T., Fouque, P.A., Grégoire, B., Rossi, M., Tibouchi, M.: Masking the GLP lattice-based signature scheme at any order. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 354–384. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78375-8_12
7. Battistello, A., Coron, J.S., Prouff, E., Zeitoun, R.: Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 23–39. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53140-2_2
8. Bernstein, D.J., Brumley, B.B., Chen, M.S., Tuveri, N.: OpenSSLNTRU: Faster post-quantum TLS key exchange. In: Butler, K.R.B., Thomas, K. (eds.) USENIX Security 2022. pp. 845–862. USENIX Association (Aug 2022)
9. Berzati, A., Viera, A.C., Chartouny, M., Madec, S., Vergnaud, D., Vigilant, D.: Exploiting intermediate value leakage in dilithium: A template-based approach. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(4), 188–210 (2023). https://doi.org/10.46586/tches.v2023.i4.188-210

10. Bronchain, O., Cassiers, G.: Bitslicing arithmetic/boolean masking conversions for fun and profit with application to lattice-based KEMs. IACR TCHES **2022**(4), 553–588 (2022). https://doi.org/10.46586/tches.v2022.i4.553-588
11. Cheon, J.H., Jeong, J., Lee, C.: An algorithm for ntru problems and cryptanalysis of the ggh multilinear map without a low-level encoding of zero. LMS Journal of Computation and Mathematics **19**(A), 255–266 (2016). https://doi.org/10.1112/S1461157016000371
12. Chuengsatiansup, C., Prest, T., Stehlé, D., Wallet, A., Xagawa, K.: ModFalcon: Compact signatures based on module-NTRU lattices. In: Sun, H.M., Shieh, S.P., Gu, G., Ateniese, G. (eds.) ASIACCS 20. pp. 853–866. ACM Press (Oct 2020). https://doi.org/10.1145/3320269.3384758
13. Coron, J., Gérard, F., Montoya, S., Zeitoun, R.: High-order polynomial comparison and masking lattice-based encryption. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(1), 153–192 (2023). https://doi.org/10.46586/tches.v2023.i1.153-192
14. Coron, J., Gérard, F., Trannoy, M., Zeitoun, R.: High-order masking of NTRU. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(2), 180–211 (2023). https://doi.org/10.46586/tches.v2023.i2.180-211
15. Coron, J., Gérard, F., Trannoy, M., Zeitoun, R.: Improved gadgets for the high-order masking of dilithium. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(4), 110–145 (2023). https://doi.org/10.46586/tches.v2023.i4.110-145
16. Duc, A., Faust, S., Standaert, F.X.: Making masking security proofs concrete (or how to evaluate the security of any leaking device), extended version. Journal of Cryptology **32**(4), 1263–1297 (Oct 2019). https://doi.org/10.1007/s00145-018-9277-0
17. Ducas, L., van Woerden, W.P.J.: NTRU fatigue: How stretched is overstretched? In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part IV. LNCS, vol. 13093, pp. 3–32. Springer, Heidelberg (Dec 2021). https://doi.org/10.1007/978-3-030-92068-5_1
18. Espitau, T., Fouque, P.A., Gérard, F., Rossi, M., Takahashi, A., Tibouchi, M., Wallet, A., Yu, Y.: Mitaka: A simpler, parallelizable, maskable variant of falcon. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 222–253. Springer, Heidelberg (May / Jun 2022). https://doi.org/10.1007/978-3-031-07082-2_9
19. Espitau, T., Kirchner, P.: The nearest-colattice algorithm. Cryptology ePrint Archive, Report 2020/694 (2020), https://eprint.iacr.org/2020/694
20. Fournaris, A.P., Dimopoulos, C., Koufopavlou, O.G.: Profiling Dilithium Digital Signature Traces for Correlation Differential Side Channel Attacks. In: Orailoglu, A., Jung, M., Reichenbach, M. (eds.) Embedded Computer Systems: Architectures, Modeling, and Simulation - 20th International Conference, SAMOS 2020, Samos, Greece, July 5-9, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12471, pp. 281–294. Springer (2020). https://doi.org/10.1007/978-3-030-60939-9_19, https://doi.org/10.1007/978-3-030-60939-9_19
21. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC. pp. 197–206. ACM Press (May 2008). https://doi.org/10.1145/1374376.1374407
22. Goldwasser, S., Kalai, Y.T., Peikert, C., Vaikuntanathan, V.: Robustness of the learning with errors assumption. In: Yao, A.C. (ed.) Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceed-

ings. pp. 230–240. Tsinghua University Press (2010), http://conference.iiis.tsinghua.edu.cn/ICS2010/content/papers/19.html

23. Goubin, L.: A sound method for switching between Boolean and arithmetic masking. In: Koç, Çetin Kaya., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 3–15. Springer, Heidelberg (May 2001). https://doi.org/10.1007/3-540-44709-1_2
24. Guerreau, M., Martinelli, A., Ricosset, T., Rossi, M.: The hidden parallelepiped is back again: Power analysis attacks on falcon. IACR TCHES **2022**(3), 141–164 (2022). https://doi.org/10.46586/tches.v2022.i3.141-164
25. Hough, P., Sandsbråten, C., Silde, T.: Concrete ntru security and advances in practical lattice-based electronic voting. Cryptology ePrint Archive, Paper 2023/933 (2023), https://eprint.iacr.org/2023/933, https://eprint.iacr.org/2023/933
26. Hülsing, A., Bernstein, D.J., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.L., Kampanakis, P., Kölbl, S., Lange, T., Lauridsen, M.M., Mendel, F., Niederhagen, R., Rechberger, C., Rijneveld, J., Schwabe, P., Aumasson, J.P., Westerbaan, B., Beullens, W.: SPHINCS$^+$. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
27. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_27
28. Ito, A., Ueno, R., Homma, N.: On the success rate of side-channel attacks on masked implementations: Information-theoretical bounds and their practical usage. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 1521–1535. ACM Press (Nov 2022). https://doi.org/10.1145/3548606.3560579
29. Kannwischer, M.J., Genêt, A., Butin, D., Krämer, J., Buchmann, J.: Differential power analysis of XMSS and SPHINCS. In: Fan, J., Gierlichs, B. (eds.) COSADE 2018. LNCS, vol. 10815, pp. 168–188. Springer, Heidelberg (Apr 2018). https://doi.org/10.1007/978-3-319-89641-0_10
30. Karabulut, E., Alkim, E., Aysu, A.: Single-Trace Side-Channel Attacks on ω-Small Polynomial Sampling: With Applications to NTRU, NTRU Prime, and CRYSTALS-DILITHIUM. In: IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2021, Tysons Corner, VA, USA, December 12-15, 2021. pp. 35–45. IEEE (2021). https://doi.org/10.1109/HOST49136.2021.9702284
31. Karabulut, E., Aysu, A.: FALCON Down: Breaking FALCON Post-Quantum Signature Scheme through Side-Channel Attacks. In: 58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021. pp. 691–696. IEEE (2021). https://doi.org/10.1109/DAC18074.2021.9586131, https://doi.org/10.1109/DAC18074.2021.9586131
32. Kim, D., Lee, D., Seo, J., Song, Y.: Toward practical lattice-based proof of knowledge from hint-MLWE. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part V. LNCS, vol. 14085, pp. 549–580. Springer, Heidelberg (Aug 2023). https://doi.org/10.1007/978-3-031-38554-4_18
33. Kim, M., Lee, D., Seo, J., Song, Y.: Accelerating HE operations from key decomposition technique. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part IV. LNCS, vol. 14084, pp. 70–92. Springer, Heidelberg (Aug 2023). https://doi.org/10.1007/978-3-031-38551-3_3

34. Kirchner, P., Fouque, P.A.: Revisiting lattice attacks on overstretched NTRU parameters. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I. LNCS, vol. 10210, pp. 3–26. Springer, Heidelberg (Apr / May 2017). https://doi.org/10.1007/978-3-319-56620-7_1
35. Krausz, M., Land, G., Richter-Brockmann, J., Güneysu, T.: Efficiently masking polynomial inversion at arbitrary order. In: Cheon, J.H., Johansson, T. (eds.) Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022, Virtual Event, September 28-30, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13512, pp. 309–326. Springer (2022). https://doi.org/10.1007/978-3-031-17234-2_15, https://doi.org/10.1007/978-3-031-17234-2_15
36. Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehlé, D., Bai, S.: CRYSTALS-DILITHIUM. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
37. Marzougui, S., Ulitzsch, V., Tibouchi, M., Seifert, J.P.: Profiling side-channel attacks on Dilithium: A small bit-fiddling leak breaks it all. Cryptology ePrint Archive, Report 2022/106 (2022), https://eprint.iacr.org/2022/106
38. Masure, L., Rioul, O., Standaert, F.: A nearly tight proof of duc et al.'s conjectured security bound for masked implementations. In: Buhan, I., Schneider, T. (eds.) Smart Card Research and Advanced Applications - 21st International Conference, CARDIS 2022, Birmingham, UK, November 7-9, 2022, Revised Selected Papers. Lecture Notes in Computer Science, vol. 13820, pp. 69–81. Springer (2022). https://doi.org/10.1007/978-3-031-25319-5_4
39. Mathieu-Mahias, A.: Securisation of implementations of cryptographic algorithms in the context of embedded systems. Theses, Université Paris-Saclay (Dec 2021), https://theses.hal.science/tel-03537322
40. Montgomery, P.L.: Speeding the pollard and elliptic curve methods of factorization. Mathematics of Computation **48**(177), 243–264 (1987). https://doi.org/10.1090/s0025-5718-1987-0866113-7
41. del Pino, R., Espitau, T., Katsumata, S., Maller, M., Mouhartem, F., Prest, T., Rossi, M., Saarinen, M.J.: Raccoon, A Side-Channel Secure Signature Scheme. Tech. rep., National Institute of Standards and Technology (2023), available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures
42. del Pino, R., Katsumata, S., Maller, M., Mouhartem, F., Prest, T., Saarinen, M.J.: Threshold raccoon: Practical threshold signatures from standard lattice assumptions. Cryptology ePrint Archive, Paper 2024/184 (2024), https://eprint.iacr.org/2024/184, https://eprint.iacr.org/2024/184
43. del Pino, R., Prest, T., Rossi, M., Saarinen, M.O.: High-order masking of lattice signatures in quasilinear time. In: 44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023. pp. 1168–1185. IEEE (2023). https://doi.org/10.1109/SP46215.2023.10179342
44. Prest, T.: A key-recovery attack against mitaka in the t-probing model. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part I. LNCS, vol. 13940, pp. 205–220. Springer, Heidelberg (May 2023). https://doi.org/10.1007/978-3-031-31368-4_8
45. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: FALCON. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022

46. Saarinen, M.J.O., Rossi, M.: Mask compression: High-order masking on memory-constrained devices. Cryptology ePrint Archive, Paper 2023/1117 (2023), https://eprint.iacr.org/2023/1117
47. Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Seiler, G., Stehlé, D., Ding, J.: CRYSTALS-KYBER. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
48. Yu, Y., Jia, H., Wang, X.: Compact lattice gadget and its applications to hash-and-sign signatures. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part V. LNCS, vol. 14085, pp. 390–420. Springer, Heidelberg (Aug 2023). https://doi.org/10.1007/978-3-031-38554-4_13
49. Zhang, S., Lin, X., Yu, Y., Wang, W.: Improved power analysis attacks on falcon. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part IV. LNCS, vol. 14007, pp. 565–595. Springer, Heidelberg (Apr 2023). https://doi.org/10.1007/978-3-031-30634-1_19

A An NTRU-based Maskable Hash-and-Sign Scheme

This section additionally introduces Plover-NTRU, which consists in applying our transform to Robin [48].

A.1 Additional definitions

Definition 10 (Hint-NTRU). *Let q, Q be integers, $\mathcal{D}_{\mathsf{sk}}, \mathcal{D}_{\mathsf{pert}}$ be probability distributions over $\mathcal{R}_q \times \mathcal{R}_q^\times$ and $\mathcal{R}_q^2$ respectively, and $\mathcal{C}$ be a set over $\mathcal{R}_q$. The advantage $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hint\text{-}NTRU}}(\kappa)$ of an adversary $\mathcal{A}$ against the Hint NTRU problem $\mathsf{Hint\text{-}NTRU}_{q,Q,\mathcal{D}_{\mathsf{sk}},\mathcal{D}_{\mathsf{pert}},\mathcal{C}}$ is defined as:*

$$\left|\Pr\left[1 \leftarrow \mathcal{A}\left(g/f, (c_i, \mathbf{z}_i,)_{i\in[Q]}\right)\right] - \Pr\left[1 \leftarrow \mathcal{A}\left(u, (c_i, \mathbf{z}_i)_{i\in[Q]}\right)\right]\right|,$$

where $u \xleftarrow{\$} \mathcal{R}_q$, $\mathbf{s} = (f, g) \leftarrow \mathcal{D}_{\mathsf{sk}}$ and for $i \in [Q]$: $c_i \leftarrow \mathcal{C}$, $\mathbf{r}_i \leftarrow \mathcal{D}_{\mathsf{pert}}$, and $\mathbf{z}_i = c_i \cdot \mathbf{s} + \mathbf{r}_i$. The $\mathsf{Hint\text{-}NTRU}_{q,Q,\mathcal{D}_{\mathsf{sk}},\mathcal{D}_{\mathsf{pert}},\mathcal{C}}$ assumption states that any efficient adversary $\mathcal{A}$ has negligible advantage. We may write $\mathsf{Hint\text{-}NTRU}_{q,Q,\sigma_{\mathbf{s}},\sigma_{\mathbf{r}},\mathcal{C}}$ as a shorthand when $\mathcal{D}_{\mathsf{sk}}$ and $\mathcal{D}_{\mathsf{pert}}$ are the Gaussian distributions of standard deviation $\sigma_{\mathbf{r}}$ and $\sigma_{\mathbf{r}}$, respectively. When $Q = 0$, we recover the classical NTRU problem: $\mathsf{NTRU}_{q,\mathcal{D}_{\mathsf{sk}}} = \mathsf{Hint\text{-}NTRU}_{q,Q=0,\mathcal{D}_{\mathsf{sk}},\mathcal{D}_{\mathsf{pert}},\mathcal{C}}$.

Algorithm 10 Plover-NTRU.Keygen$(1^\kappa) \to \mathsf{vk}, \mathsf{sk}$

```
Require: The ring $\mathcal{R}$, a modulus $q$
Ensure: A public key $h \in \mathcal{R}_q$, a private key $(f, g) \in \mathcal{R}^2$
$(f, g) \leftarrow \mathcal{D}_{\mathsf{sk}}$
$h := (\beta - f)/g \bmod q$
return $\mathsf{vk} := h, \mathsf{sk} := (\mathsf{vk}, g)$
```

Algorithm 11 Plover-NTRU.Sign$(\mathsf{msg}, \mathsf{vk}, \mathsf{sk}) \to \mathsf{sig}$

```
Require: A message $\mathsf{msg}$, the keypair $(h, (f, g))$, a divider $\beta$, a bound $B_2 > 0$
Ensure: A signature $(\mathsf{salt}, z)$
$\mathsf{salt} \xleftarrow{\$} \{0,1\}^{2\kappa}$
$u := H(\mathsf{msg}, \mathsf{salt}, \mathsf{vk})$
$\mathbf{A} := \begin{bmatrix} 1 & h \end{bmatrix}$, $\mathbf{T} := \begin{bmatrix} f \\ g \end{bmatrix}$
$\mathbf{p} \leftarrow \mathcal{D}_{\mathsf{pert}}$                          ▷ $\mathbf{p} \in \mathcal{R}^2$
$c := u - \mathbf{A}\,\mathbf{p}$
$(c_1, c_2) := \mathsf{Decompose}_\beta(c)$               ▷ $c = \beta \cdot c_1 + c_2$
$z_2 := p_2 + g \cdot c_1$
$z_1' := (u - h z_2) \bmod q$                  ▷ $z_1' = z_1 + c_2 = p_1 + f \cdot c_1 + c_2$
if $\|(z_1', z_2)\| \geqslant B_2$ then           ▷ Approximation of $\mathbf{z} = \mathbf{p} + \mathbf{T} \cdot c_1$
    restart
return $\mathsf{sig} := (\mathsf{salt}, z_2)$
```

Algorithm 12 Plover-NTRU.Verify(vk, msg, sig) → **accept** or **reject**

```
Require: A signature $(\mathsf{salt}, z_2)$ of a message $\mathsf{msg}$, the public key $h$, a bound $B_2 > 0$
Ensure: accept or reject.
$u := H(\mathsf{msg}, \mathsf{salt}, \mathsf{vk})$
$z_1' := (u - hz_2) \bmod q$
accept if $\{ \|(z_1', z_2)\| \leqslant B_2 \}$, else reject
```

A.2 Masked key generation

Plover-NTRU key generation requires inverting the masked polynomial $[\![g]\!]$. We introduce a novel efficient algorithm PseudoInverse to perform this masked operation in Section A.5.

Algorithm 13 Plover-NTRU.MaskKeygen(∅) → (vk, sk)

```
Require: The ring $\mathcal{R}$
Ensure: A public key $h \in \mathcal{R}$, a private key $(f, g) \in \mathcal{R}^2$
$e := 0$
$[\![f]\!] \leftarrow \mathsf{AddRepNoise}\left(\mathcal{R}_q, d, \mathcal{D}_{\mathsf{sk}}^{\mathsf{ind}}, \mathsf{rep}_{\mathsf{sk}}\right)$
while $(e = 0)$ do
    $[\![g]\!] \leftarrow \mathsf{AddRepNoise}\left(\mathcal{R}_q, d, \mathcal{D}_{\mathsf{sk}}^{\mathsf{ind}}, \mathsf{rep}_{\mathsf{sk}}\right)$
    $(e, [\![g^\times]\!]) \leftarrow \mathsf{PseudoInverse}([\![g]\!])$    ▷ If $g$ is invertible, compute $[\![g^\times]\!]$ such that $g^\times = g^{-1}$ and $e = 1$, else $e = 0$
$[\![h]\!] \leftarrow (\beta - [\![f]\!]) \cdot [\![g^\times]\!]$
$h := \mathsf{Unmask}([\![h]\!])$
return $(\mathsf{vk} := h, \mathsf{sk} := (\mathsf{vk}, [\![g]\!]))$
```

A.3 Masked signing

We describe the masked signing procedure of Plover-NTRU in Algorithm 14.

A.4 Cryptanalysis and parameter selection

Forgery and RSIS Let $\sigma_{\mathsf{sk}}, \sigma_{\mathsf{pert}}$ denote the standard deviation of the (unmasked) secret key and perturbation, respectively. In a legitimate signature:

$$\begin{aligned}\mathbb{E}\left[\|\mathbf{z}'\|^2\right] &= \mathbb{E}\left[\|z_1'\|^2\right] + \mathbb{E}\left[\|z_2\|^2\right] \\ &= \mathbb{E}\left[\|p_1 + g \cdot c_1\|^2\right] + \mathbb{E}\left[\|p_2 + f \cdot c_1 + c_2\|^2\right] \\ &\approx n\left(2\,\sigma_{\mathsf{pert}}^2 + \frac{\beta^2}{12} + \frac{q^2\, n}{6\,\beta^2}\,\sigma_{\mathsf{sk}}^2\right)\end{aligned}$$

We can therefore select B_2 as in Section 3.5. The rest of our analysis is also identical to Section 3.5, except that we need to solve RSIS for a 1 × 2 NTRU matrix $\left[1\ h\right]$, instead of a 1 × 3 matrix $\left[1\ a\ b\right]$. This makes the RSIS problem much harder in the NTRU case.

Algorithm 14 $\mathsf{Plover\text{-}NTRU.MaskSign}(\mathsf{msg}, \mathsf{vk}, \mathsf{sk}) \rightarrow \mathsf{sig}$

```
Require: A message msg, the keypair $(h, [\![g]\!])$
Ensure: A signature $(\mathsf{salt}, z)$
salt $\xleftarrow{\$} \{0,1\}^{2\kappa}$
$u := H(\mathsf{msg}, \mathsf{salt}, \mathsf{vk})$
$[\![\mathbf{p}]\!] \leftarrow \mathsf{AddRepNoise}(\mathcal{R}_q^2, d, \mathcal{D}_{\mathsf{pert}}^{\mathsf{ind}}, \mathsf{rep}_{\mathsf{pert}})$        ▷ $\mathbf{p} = (p_1, p_2)$
$[\![w]\!] := [1\ h] \cdot [\![\mathbf{p}]\!]$
$w := \mathsf{Unmask}([\![w]\!])$
$c := u - w$
$(c_1, c_2) := \mathsf{Decompose}_\beta(c)$        ▷ $c = \beta \cdot c_1 + c_2$
$[\![g]\!] \leftarrow \mathsf{Refresh}([\![g]\!])$        ▷ Refresh $[\![g]\!]$ before re-use
$[\![z_2]\!] := [\![p_2]\!] + [\![g]\!] \cdot c_1$
$z_2 := \mathsf{Unmask}([\![z_2]\!])$
$z_1' := (u - hz_2) \bmod q$        ▷ $z_1' = z_1 + c_2 = p_1 + f \cdot c_1 + c_2$
if $\|(z_1', z_2)\| \geqslant B_2$ then
    restart
return sig := $(\mathsf{salt}, z_2)$
```

Key-indistinguishability and Hint-NTRU An adversary breaking the key-indistinguishability of vk is also able to break $\mathsf{Hint\text{-}RLWE}_{q,Q_{\mathsf{Sign}},\widehat{\mathcal{D}_{\mathsf{sk}}},\widehat{\mathcal{D}_{\mathsf{pert}}},\mathcal{C}}$. In the Gaussian case, $\widehat{\mathcal{D}_{\mathsf{sk}}} \overset{s}{\sim} D_{\widehat{\sigma}_{\mathsf{sk}}}$ and $\widehat{\mathcal{D}_{\mathsf{pert}}} \overset{s}{\sim} D_{\widehat{\sigma}_{\mathsf{pert}}}$, where $\frac{\widehat{\sigma}_{\mathsf{sk}}}{\sigma_{\mathsf{sk,ind}}} = \sqrt{d\,\mathsf{rep}_{\mathsf{sk}} - t}$ and $= \frac{\widehat{\sigma}_{\mathsf{pert}}}{\sigma_{\mathsf{pert,ind}}} = \sqrt{d\,\mathsf{rep}_{\mathsf{pert}} - t}$.

In the Hint-NTRU case, we don't have a reduction similar to Theorem 1. We assume that an adversary against Hint-NTRU is able to break the indistinguishability of an inhomogeneous NTRU instance:

$$\frac{f_{\mathsf{red}}}{g_{\mathsf{red}}} \quad \text{where} \quad (f, g) \sim D_{\mathcal{R},\mathcal{R}^\times,\sigma_{\mathsf{red}},(f_0,g_0)} \tag{5}$$

We assume that inhomogeneous NTRU is as hard as homogeneous NTRU for the same parameters $(q, n, \sigma_{\mathsf{red}})$.

When q is large, the class of attacks known as overstretched NTRU attacks [2,11,34] becomes relevant. According to the most recent analyses [17,25], these attack become relevant when $q > q_{\mathsf{fatigue}}$, where $q_{\mathsf{fatigue}} = 0.0058 \cdot \sigma_{\mathsf{red}}^2 \cdot n^{2.484}$. We rely on the script by van Woerden (https://github.com/WvanWoerden/NTRUFatigue) to precisely estimate the impact of overstretched NTRU attacks on our scheme.

Parameter selection Selected parameters for different values of $\lceil \log q \rceil$ are summarized in Table 3.

A.5 Implementation

Plover-NTRU shares a significant portion of its implementation code with Plover-RLWE. We also provide both a C and Python reference implementations for Plover-NTRU,

$\lceil \log q \rceil$	35	37	39	41	43	45	47	49
$\log \beta$	31	33	35	37	38	38	39	40
$\log \sigma_{\mathsf{pert}}$	30	32	34	36	37	38	39	40
$\log \sigma_{\mathsf{sk}}$	21	23	25	27	27	27	27	27
Q_{Sign}	37	39	42	44	42	40	38	36
$\lvert \mathsf{vk} \rvert$	8960	9472	9984	10496	11008	11520	12032	12544
$\lvert \mathsf{sig} \rvert$	11488	12198	12908	13617	13972	14327	14682	15037

Table 3: Parameter sets of Plover-NTRU for $\kappa = 128$. All parameter sets feature $n = 2048$, and B_2 defined as in Appendix A.4.

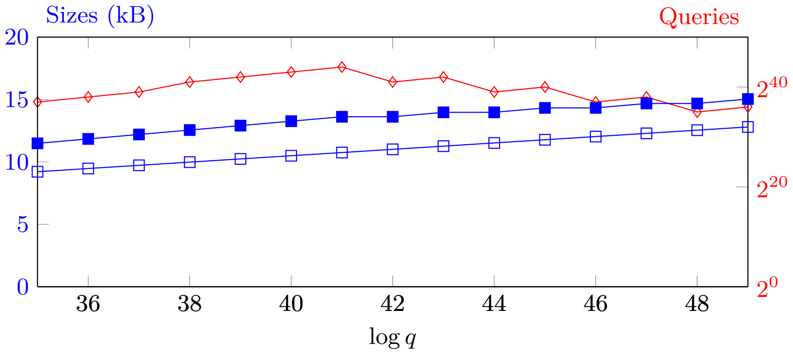

Fig. 7: Number of signing queries (—◇—) and bytesizes ($\lvert \mathsf{vk} \rvert$: —□—, $\lvert \mathsf{sig} \rvert$: —■—) as functions of q. Parameter sets as in Appendix A.4.

available at https://github.com/GuilhemN/masksign-plover. Polynomial multiplications and modular reduction are implemented in the same way as Raccoon and Plover-RLWE.

Plover-NTRU additionally packages the ISW masked multiplication algorithm [5]. This algorithm is used in the key generation Algorithm 13. As an extra optimization of our scheme, we present a novel optimization technique with PseudoInverse to compute the inverse of a masked polynomial efficiently when the underlying ring supports number theoretic transform (NTT).

Masking complexity. The usage of masked multiplications in the key generation increases its masking complexity to $O(d^2)$ in the masking order d. The signature procedure however keeps a quasi-linear masking complexity $O(d \log d)$. However, the performance of Plover-NTRU remains competitive with Plover-RLWE, notably due to the optimization of the polynomial inversion introduced in Section A.5.

Plover-NTRU has the same general characteristics, and randomness implementation as Plover-RLWE, which are detailed in Section 3.6. Encoding techniques are also analogous to Plover-RLWE.

Efficient masked polynomial inversion. We recall that the number theoretic transform (NTT) is a linear operation and can thus be performed in masked form in linear overhead $O(d)$, for a total cost $O(d\, n \log n)$. We also recall that for $x \in \mathbb{Z}_q$, $x^{\varphi(q)} \in \{0, 1\}$, and $x^{\varphi(q)} = 1$ if and only if x is invertible. $x^{\varphi(q)}$ can be computed in $\lfloor \log q \rfloor$ squarings and at most as many products. Note that $\varphi(q)$ is public so the exponentiation algorithm can safely leak information about the exponent.

Algorithm 15 PseudoInverse

```
Require: $[\![f]\!] \in \mathcal{R}_q^d$
Ensure: $e \in \{0,1\}, [\![f^\times]\!]\mathcal{R}_q^d$ such that, if $e = 1$, then $f^\times = f^{-1}$
$[\![\mathbf{a}]\!] := \mathsf{NTT}([\![f]\!])$                              ▷ $\mathbf{a} = (a_i)_{i\in[n]} \in \mathbb{Z}_q^n$
$[\![b_{-1}]\!] := [\![0]\!]$
for $i \in \{0, \ldots, n-1\}$ do
    $[\![b_i]\!] \leftarrow [\![a_i]\!] \cdot [\![b_{i-1}]\!]$
$[\![t_{n-1}]\!] \leftarrow [\![b_{n-1}]\!]^{\varphi(q)-1}$              ▷ Done via the square-and-multiply algorithm
$[\![b_{n-1}]\!] \leftarrow \mathsf{Refresh}_{\mathbb{Z}_q}([\![b_{n-1}]\!])$      ▷ Refresh before re-use
$[\![e]\!] := [\![t_{n-1}]\!] \cdot [\![b_{n-1}]\!]$
$e \leftarrow \mathsf{Unmask}([\![e]\!])$          ▷ $e \in \{0,1\}$, and $e = 0$ if and only if $f$ is not invertible
$[\![\mathbf{a}]\!] \leftarrow \mathsf{Refresh}_{\mathbb{Z}_q^n}([\![\mathbf{a}]\!])$             ▷ Refresh $[\![\mathbf{a}]\!]$ before re-use
for $(i = n-1, \ldots, 0)$ do
    $[\![c_i]\!] \leftarrow [\![t_i]\!] \cdot [\![b_{i-1}]\!]$                        ▷ $c_i = a_i^{-1}$
    $[\![t_i]\!] \leftarrow \mathsf{Refresh}_{\mathbb{Z}_q}([\![t_i]\!])$                 ▷ Refresh before re-use
    $[\![t_{i-1}]\!] \leftarrow [\![t_i]\!] \cdot [\![a_i]\!]$                      ▷ $t_i = \left(\prod_{j\leqslant i} a_i\right)^{-1}$
$[\![\mathbf{c}]\!] := ([\![c_i]\!])_{i\in[n]}$                              ▷ $\mathbf{c} \in \mathbb{Z}_q^n$
$[\![f^\times]\!] := \mathsf{NTT}^{-1}([\![\mathbf{c}]\!])$
return $e$, $[\![f^\times]\!]$
```

Complexity. We count the complexity in the number of arithmetic operations (additions, multiplications) over $\mathbb{Z}$. Two NTTs: $O(d\, n \log n)$. Computing the Montgomery "chains" and e: $3n + 1$ multiplications, hence $O(n\, d^2)$. Computing the exponentiation: $O(d^2 \log q)$. The total complexity is:

$$O\left(d\left(n \log n + n\, d + d\, \log q\right)\right) \tag{6}$$

Use cases. Polynomial inversion is a useful operation in many lattice-based schemes based on NTRU and its variants. It is especially useful during key generation of these schemes when the operation f/g is performed in masked form.

Related works To the best of our knowledge, our work is the first time Montgomery's trick has been used for masking lattice-based cryptography.

Masked inversion. Algorithms for performing masked inversion have been proposed in [14, Sections 4.3 and 5] and [35]. The complexity of these algorithms is at least in the order of magnitude of d^2 multiplications in $\mathcal{R}_q$. Since known methods for multiplying two elements in $\mathcal{R}_q$ cost at least $O(n \log n)$ arithmetic operations in $\mathbb{Z}_q$, the running time of these algorithms is at least $O(d^2 n \log n)$, which is higher than the complexity we claim in Eq. (6).

Montgomery's trick. Integral to the efficiency of our method is the use of the so-called Montgomery's trick, which allows to perform n inversions in $\mathbb{Z}_q$ for the cost of $3n$ multiplications in $\mathbb{Z}_q$ and one inversion in $\mathbb{Z}_q$. This algorithm has been described in [40]. It has been used in cryptographic contexts.

[8] uses Montgomery's trick for inverting several values in $f_1, \ldots, f_m \in \mathcal{R}_q$ at the same time, whereas our method already provides benefits when inverting one single value $f \in \mathcal{R}_q$. We note that the work of [8] does not mention masking, and many of their subroutines (for example, the subroutine `isInvertible` described in [8, Section 3.1.1]) do not immediately seem easy to mask efficiently.

Performance. We performed a performance evaluation of Plover-NTRU on a 2022 Lenovo Thinkpad P14s equipped with a Ryzen Pro 7 5850U (16CPU threads at 3GHz), boost disabled, and running Manjaro 22.1. We provide as for Plover-RLWE results in milliseconds, millions of cycles, and the peak memory usage of each function in bytes. Concrete parameters are also defined from the optimal set in the number of queries $\lceil \log q \rceil = 41$ from Table 3, which leads to the same concrete parameters used for Plover-RLWE evaluation given in Table 1.

Results are summarized in Table 4, and plotted in Figure 8 for Keygen and Sign. The performance of the key generation degrades quadratically with d – as expected, but they remain practical even at order $d = 32$ with one key generation taking about 1/4 second on our evaluation machine. Furthermore, the signature generation is slightly faster than the one of Plover-RLWE, and its verification is even about 30% faster, which makes Plover-NTRU very competitive with both Raccoon and Plover-RLWE for applications where performance of signature generation and verification are more important than the efficiency of the key generation.

Variant	**Keygen**			**Sign**			**Verify**		
$\kappa - d$	ms	Mclk	stack	ms	Mclk	stack	ms	Mclk	stack
128-1	1.302	2.479	65712	1.829	3.473	180512	0.281	0.534	65616
128-2	1.791	3.415	131488	2.118	4.023	262432	=	=	=
128-4	3.759	7.141	262752	2.658	5.051	426400	=	=	=
128-8	16.260	30.898	525184	8.502	16.151	754144	=	=	=
128-16	54.571	103.759	1050080	10.914	20.734	1409632	=	=	=
128-32	248.506	471.994	2099296	36.725	69.775	2720608	=	=	=

Table 4: Performance of the Plover-NTRU reference implementation on an AMD PC. Units: ms = milliseconds, Mclk = millions of clock cycles, stack = stack usage in bytes.

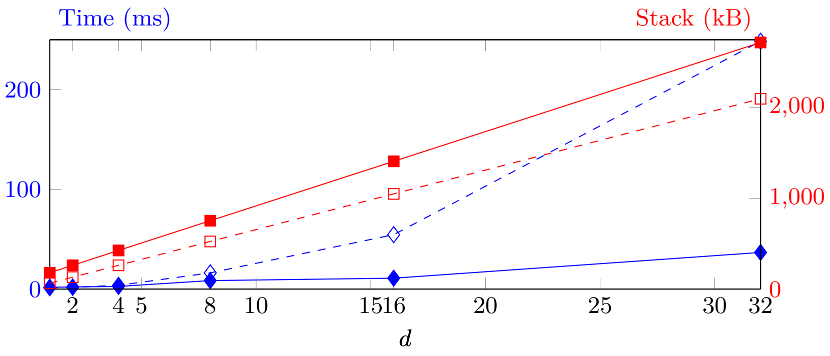

Fig. 8: Timing of Plover-NTRU (Keygen: -◇-, Sign: ─◆─) and memory usage (Keygen: -□-, Sign: ─■─) as functions of d. Parameter set from Section 3.5 such that $\lceil \log q \rceil = 41$, with concrete parameters from Table 1.

B Proof of Lemma 2

Proof. Recall that for $c \leftarrow \mathbb{Z}_q$ and $(c_1, c_2) \coloneqq \mathsf{Decompose}_\beta(c)$, we have $|c_1| \leqslant \left\lceil \frac{q-1}{2\beta} \right\rceil$, $\mathbb{E}[c_1] = 0$ and $\mathbb{E}[c_1^2] \leqslant \frac{M^2-1}{12}$ for $M = 2\left\lceil \frac{q-1}{2\beta} \right\rceil + 1$. First, let us consider a single $c \leftarrow \mathcal{C}_1$. Let $d(x) = c\,c^* = \sum_{0 \leqslant i < n} d_i\, x^i$. For each $k \in \{0, \ldots, n-1\}$, d_k can be expressed explicitly as:

$$d_k = \sum_{0 \leqslant i < n-k} c_i\, c_{i+k} \quad - \sum_{n-k \leqslant i < n} c_i\, c_{i+k-n}. \tag{7}$$

From Eq. (7), it is clear that:

1. $d_0 = \|c\|_2^2$
2. If $k \neq 0$, d_k is a random variable satisfying:

$$\|d_k\| \leqslant \|c\|_2^2 \tag{8}$$
$$\mathbb{E}[d_k] = 0. \tag{9}$$

While Eq. (8) is immediate from Eq. (7), Eq. (9) is a bit more subtle. The mapping $(k, i) \in \{0, \ldots, n-1\}^2 \mapsto (i + k \bmod n)$ is a group action. As such, its orbits form a partition of $\{0, \ldots, n-1\}$, and we note that each orbit has an even number of elements. Each orbit can be written as $\{i_0 + k\,x \bmod n | x \in n/\gcd(k,n)\}$, where i_0 is (say) the smallest element in this orbit. We define the function $\delta_k : \{0, \ldots, n-1\} \to \{-1, 1\}$ as follows: for each $0 \leqslant i < n$, we determine its orbit, write $i = i_0 + k\,x$ in this orbit, and set $\delta_k(i) = (-1)^x$. Now consider the mapping $\varphi_k : \mathcal{C}_1 \to \mathcal{C}_1$ defined as

$$\varphi_k : \left(c = \sum_{0 \leqslant i < n} c_i\, x^i \right) \longrightarrow \left(c' = \sum_{0 \leqslant i < n} \delta_k(i)\, c_i\, x^i \right). \tag{10}$$

φ_k maps $c \in \mathrm{Supp}(\mathcal{C}_1)$ to a new $c' \in \mathrm{Supp}(\mathcal{C}_1)$ such that $\mathcal{C}_1(c) = \mathcal{C}_1(c')$, and one can check from Eqs. (7) and (10) that $(c\,c^*)_k = -(c'\,c'^*)_k$. Since φ_k is an involution over its support, this implies that $\mathbb{E}[d_k] = 0$.

Let us note $d^{[j]} = c^{[j]}\,(c^{[j]})^*$. For $k \neq 0$, we can bound the sum $D_k = \sum_{j \in [Q_{\mathsf{Sign}}]} d_k^{[j]}$ by combining Hoeffding's inequality with Eqs. (8) and (9):

$$\begin{aligned}|D_k| &\leqslant \max_{c \leftarrow \mathcal{C}_1} \|c\|^2 \sqrt{2Q_{\mathsf{Sign}}\,((\kappa+1)\log(2) + \log(n))} \\ &\leqslant n \left\lceil \frac{q-1}{2\beta} \right\rceil^2 \sqrt{2Q_{\mathsf{Sign}}\,((\kappa+1)\log(2) + \log(n))}\end{aligned}$$

except with probability at most $2^{-\kappa}/n$. From the union bound, the above inequality is true for all $k \neq 0$, except with probability $\leqslant 2^{-\kappa}$. We can now bound the spectral norm of D. Since D is self-adjoint, $s_1(D)$ is the largest eigenvalue of D, that is $D(\zeta)$ for some primitive root of unity $\zeta, |\zeta| = 1$. By applying Hoeffding's inequality on $\sum_{Q_{\mathsf{Sign}}} \left\|c^{[j]}\right\|^2$, with probability $\geqslant 1 - 2^{-\kappa}$, we have:

$$\begin{aligned}s_1(D) = D(\zeta) &\leqslant D_0 + \sum_{k \neq 0} |D_k| \\ &\leqslant \sum_{Q_{\mathsf{Sign}}} \left\|c^{[j]}\right\|^2 + (n-1)n \left\lceil \frac{q-1}{2\beta} \right\rceil^2 \sqrt{2Q_{\mathsf{Sign}}\,((\kappa+1)\log(2) + \log(n))} \\ &\leqslant \frac{Q_{\mathsf{Sign}}\,n\,M^2}{12} + \sqrt{\kappa \cdot Q_{\mathsf{Sign}} \cdot n} \left\lceil \frac{q-1}{2\beta} \right\rceil \\ &+ (n-1)n \left\lceil \frac{q-1}{2\beta} \right\rceil^2 \sqrt{2Q_{\mathsf{Sign}}\,((\kappa+1)\log(2) + \log(n))}\end{aligned}$$

Setting $Q_{\mathsf{Sign}} = \omega(\kappa^2 \log^2 n)$ concludes the proof. □

C Differences from the Proceedings Version

In this full version, we include additional material to present:

- Plover-NTRU, our NTRU based version of Plover in Appendix A. We recall that it achieves slightly larger sizes than the RLWE-based approach used for Plover-RLWE, and has a complexity increasing in $O(d^2)$ with the masking order d. It remains of interest as an additional proof of the amenability of our framework.
- A masked inversion technique coined PseudoInverse in Appendix A.5. We leverage it in Plover-NTRU, and we believe it is of general interest as it asymptotically outperforms previous masked polynomial inversion algorithms.