# Raccoon: A Masking-Friendly Signature Proven in the Probing Model

Rafaël del Pino[1](✉), Shuichi Katsumata[1,2], Thomas Prest[1], and Mélissa Rossi[3]

[1] PQShield, Oxford, UK
{rafael.delpino,shuichi.katsumata,thomas.prest}@pqshield.com
[2] AIST, Warrendale, USA
[3] ANSSI, Paris, France
melissa.rossi@ssi.gouv.fr

**Abstract.** This paper presents Raccoon, a lattice-based signature scheme submitted to the NIST 2022 call for additional post-quantum signatures. Raccoon has the specificity of always being masked. Concretely, all sensitive intermediate values are shared into $d$ parts. The main design rationale of Raccoon is to be easy to mask at high orders, and this dictated most of its design choices, such as the introduction of new algorithmic techniques for sampling small errors. As a result, Raccoon achieves a masking overhead $O(d \log d)$ that compares favourably with the overheads $O(d^2 \log q)$ observed when masking standard lattice signatures.

In addition, we formally prove the security of Raccoon in the $t$-probing model: an attacker is able to probe $t \leq d - 1$ shares during each execution of the main algorithms (key generation, signing, verification). While for most cryptographic schemes, the black-box $t$-probing security can be studied in isolation, in Raccoon this analysis is performed jointly.

To that end, a bridge must be made between the black-box game-based EUF-CMA proof and the usual simulation proofs of the ISW model (CRYPTO 2003). We formalize an end-to-end masking proof by deploying the probing EUF-CMA introduced by Barthe et al. (Eurocrypt 2018) and exhibiting the simulators of the non-interference properties (Barthe et al. CCS 2016). The proof is divided into three novel parts:

- a simulation proof in the ISW model that allows to propagate the dependancy to a restricted number of inputs and random coins,
- a game-based proof showing that the security of Raccoon with probes can be reduced to an instance of Raccoon with smaller parameters,
- a parameter study to ensure that the smaller instance is secure, using a robust generalization of the Rényi divergence.

While we apply our techniques to Raccoon, we expect that the algorithmic and proof techniques we introduce will be helpful for the design and analysis of future masking-friendly schemes.

**Keywords:** Raccoon signature · $t$-probing model · side-channel attacks

L. Reyzin and D. Stebila (Eds.): CRYPTO 2024, LNCS 14920, pp. 409–444, 2024.
https://doi.org/10.1007/978-3-031-68376-3_13

## 1 Introduction

In the past decade, post-quantum cryptography has reached quickly grown from a mostly theoretical field to one with sufficient maturity to be deployed on a wide scale. This is epitomized by NIST's standardization in 2020 of the hash-based signatures XMSS and LMS, as well as its announcement in 2022 of the future standardization of the lattice-based KEM Kyber, the lattice-based signatures Dilithium and Falcon, and the hash-based signature SPHINCS+. Whilst the efficiency profiles and black-box security of these schemes are well-understood, resistance against side-channel attacks remains a weak spot.

*Side-Channel Attacks.* In a side-channel attack (SCA), an attacker can learn information about the physical execution of an algorithm, such as its running time or its effect on the power consumption, electromagnetic or acoustic emission of the device running it. This information can then be leveraged to recover sensitive information, for example, cryptographic keys.

SCAs can be devastating against cryptographic implementations, and post-quantum schemes are no exception. See Sect. 1.3 for references of concrete SCAs against Dilithium.

*Masking.* The main 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 provides a *trade-off* between efficiency and SCA resistance: the computational efficiency of the implementation is reduced by a polynomial factor in $d$, but the cost of a side-channel attack is expected to grow exponentially [19,28].

Unfortunately, lattice-based signatures contain subroutines that are extremely expensive to mask, such as (a) sampling from a small set, (b) bit-decomposition, and (c) rejection sampling. Currently, the best known ways to perform these operations is to rely on mask conversions [13,26], which convert between arithmetic and boolean masking. This typically incurs an overhead $O(d^2 \log q)$ [14] or $O(2^{d/2})$ [11], and quickly becomes the efficiency bottleneck. As an illustration, the only publicly available *masked* implementation of Dilithium [12] is 53 (resp. 200) times slower than unmasked Dilithium for $d = 2$ (resp. $d = 4$).

*Masking-Friendly Schemes.* In order to overcome these limitations, a natural research direction is to design lattice-based signatures that are naturally amenable to masking. However, this is easier said than done. The few designs that exist have either been shown insecure or lack a formal security proof, see Sect. 1.3 for a more detailed discussion. Thus having a masking-friendly signature with a formal proof has been an elusive goal.

### 1.1 Our Contributions

We propose Raccoon, a masking-friendly signature, and provide a formal security proof in the $t$-probing model [27]. While Raccoon is inspired from the similarly named scheme from [17], we have heavily modified its design in order to make it more efficient and provable secure under standard assumptions. The design presented in this paper is exactly the same as the one submitted to the NIST on-ramp standardization campaign [16].

**Blueprint.** Raccoon is based on the "Lyubashevsky signature without aborts" blueprint, also found in works on threshold signatures [1], and which we recall below. Assume the public key $\mathsf{vk}$ is a Learning With Errors (LWE) sample $(\mathbf{A}, \mathbf{t} = [\,\mathbf{A}\ \mathbf{I}\,] \cdot \mathbf{s})$, where $\mathbf{s}$ is a small vector, $\mathbf{I}$ is the identity matrix and $\mathbf{A}$ is a uniform matrix (precise definitions will be provided later in the paper). Signing proceeds as follows:

(S1) Sample $\mathbf{r}$, compute a commitment $\mathbf{w} = \left[\mathbf{A}\ \mathbf{I}\right] \cdot \mathbf{r}$;
(S2) Compute a challenge $c = H(\mathbf{w}, \mathsf{vk}, \mathsf{msg})$;
(S3) Compute a response $\mathbf{z} = \mathbf{s} \cdot c + \mathbf{r}$.

The verification procedure checks that $H(\mathbf{A} \cdot \mathbf{z} - \mathbf{t} \cdot c, \mathsf{vk}, \mathsf{msg}) = c$ and that $\mathbf{z}$ is short. Using a Rényi divergence argument, we can argue security if the modulus $q$ grows as the square root of the number of queries $Q_s$, that is $q = \Omega(\sqrt{Q_s})$. By eliminating the need for rejection sampling, this sidesteps the issue of masking it. In addition, unlike in Dilithium, the security argument does not rely on bit-decomposition. This eliminates the need to *mask* bit-dropping, which we now employ purely for efficiency reasons. We note that our final modulus has 49 bits, which is larger than the standard precision (32-bit or less) on many embedded platforms. We mitigate this by taking $q = q_1 \cdot q_2$, where $q_1$ and $q_2$ are 24-bit and 25-bit NTT-friendly prime moduli.

We note that rejection sampling in Dilithium requires a smaller modulus $q = \Omega(\dim(\mathbf{s}))$, in practice $\log q \approx 23$ in Dilithium. Our design choice entails a trade-off between compactness (Dilithium) and ease of masking (Raccoon).

**The Problem with Gaussians.** Standard Rényi divergence arguments as in [1] require $\mathbf{r}$ to be sampled from a discrete Gaussian distribution. However, Gaussians are notoriously difficult to generate in a way that is robust to SCA. The most common method for sampling Gaussians in a constant-time manner is via probability distribution tables (PDT), see for example FrodoKEM [35] or Falcon [38]. For signatures, the PDT would require a precision $p \approx \log(Q_s)$, for example Falcon takes $p = 72$. Masking this step would incur a prohibitive overhead $O(d^2 \log q)$. Similarly, all other existing sampling methods (see e.g. "Related works" in [25]) comprise at least one step that is expensive to mask. We *could* use Gaussians, and from a purely theoretic perspective the security proof would go through, but from a practical point of view this would show little relevance to the real-world issues that masking is trying to solve in the first place.

**Sums of Uniforms.** Our solution is to pick a distribution that has Gaussian-style properties, but is easier to sample securely on embedded devices. As it turns out, sampling $\mathbf{r}$ as a sum of uniform variates (over a small set) produces remarkably Gaussian-like distributions, which is unsurprising and a straightforward consequence of the central limit theorem. Unfortunately, standard Rényi divergence arguments fail for these distributions since they have finite support.

We resolve this analytical issue by introducing the *smooth Rényi divergence*, a more robust generalization of the Rényi divergence that is able to provide cryptographically useful statements about sums of uniform distributions. We define it as a simple combination of the statistical distance and the Rényi divergence. This generalisation achieves the best of both worlds: the robustness of the statistical distance and the power of the Rényi divergence.

**Probing-resilient sampling via AddRepNoise.** Now that we have identified a suitable distribution (that is, sum of uniforms) for $\mathbf{r}$, the final step is to sample it in a way that is resilient to $t$-probing adversaries. A naive approach would be to sample in parallel each share $\mathbf{r}_i$ of $[\![\mathbf{r}]\!]$ as the sum of $\mathsf{rep}$ small uniform variates, so that $\mathbf{r}$ is the sum of $d \cdot \mathsf{rep}$ small uniform variates. However, a probing adversary is allowed to probe $t \leq d-1$ individual shares $\mathbf{r}_i$. This would reduce the standard deviation of the conditional distribution of $\mathbf{r}$ by a factor $\sqrt{d}$, and lead to worse parameters.

We resolve this by proposing a new algorithm, called AddRepNoise , which interleaves (a) parallel generation of individual noises and (b) refreshing the masked vector, and repeats this $\mathsf{rep}$ times. We can formally prove that a $t$-probing adversary only learns $t$ individual uniform variates, so that the standard deviation of $\mathbf{r}$ conditioned to these variates is the sum of $d \cdot \mathsf{rep} - d + 1$ uniform variates, which allows to prove security with a minimal loss in tightness.

## 1.2 Overview of the Security Proof

We recall that a high-level description of Raccoon is given in Sect. 1.1. Now, in a masked form, the secret is shared as $\mathbf{s} = \sum_{i \in [d]} \mathbf{s}_i$ where the coefficients of the vectors $\mathbf{s}_i$ are sampled in a short interval. This is a deliberate choice of Raccoon that allows good sampling performance.

At first sight, if the $\mathbf{s}_i$ are safely manipulated in the signature algorithm and never recombined, the masking security seems guaranteed as the exact value of $\mathbf{s}$ cannot be recombined. However, if an adversary probes $d-1$ shares of $\mathbf{s}_i$, say $\{\mathbf{s}_0, \cdots, \mathbf{s}_{d-2}\}$, he can compute $\mathsf{vk}' = \mathsf{vk} - \begin{bmatrix}\mathbf{A}\ \mathbf{I}\end{bmatrix} \sum_{i=0}^{d-2} \cdot \mathbf{s}_i = \begin{bmatrix}\mathbf{A}\ \mathbf{I}\end{bmatrix} \mathbf{s}_{d-1}$. Key recovery is significantly easier as the updated secret is now from a narrower distribution. Hence, while the exact value of $\mathbf{s}$ is inaccessible, the knowledge of the probes combined with the knowledge of the public key can lead to a simpler key recovery. This aspect makes a link between two families of proofs that are typically separated in other works: the black-box game-based EUF-CMA proofs and the simulation proofs of masking. The former quantifies the advantage of a black-box attacker and provides a security statement conditioned to the hardness of well-defined mathematical problems (like LWE). The latter provides a statistical statement showing that any probing attacker limited to $d-1$ probes have no statistical advantage to recover the sensitive information (Fig. 1).

To prove the security of Raccoon, it is important to link these two notions. For that, we detail and formalize the probing security from a game-base perspective, i.e. with well-defined simulators and reuse the notion of probing EUF-CMA

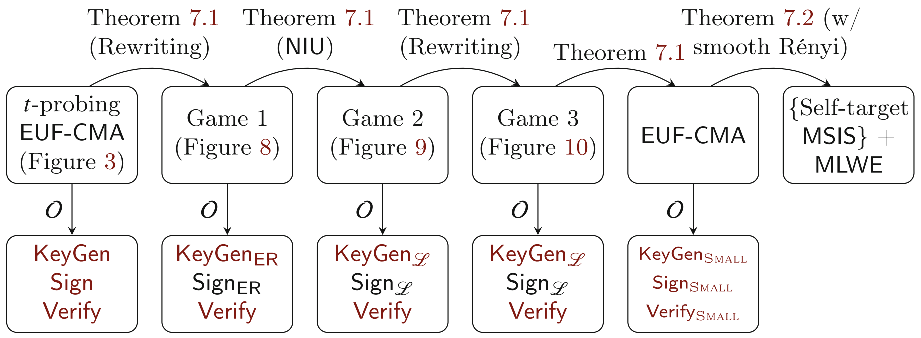

**Fig. 1.** Proof overview. Jump 1 consists in moving randomness to inputs as per Definition 5.2. Jump 2 uses Lemma 5.2 to move all probes to inputs. Jump 3 is a simple rewriting step. Jump 4 is a black-box reduction to a simpler unmasked signature Small Raccoon. Jump 5 is the security proof of Small Racoon. $\mathcal{O}$ denote access to an oracle to the corresponding algorithm.

provided in [5]. Such a notion has been defined but it was not formally used in a game-based proof before. The main contribution of this paper is the proof of the probing EUF-CMA security of Raccoon. It will consist in several steps.

1. **Non-uniform masks and sNIU:** First, one needs to handle the sensitive small uniforms that are deviating from the classical ISW model [27] and other masking proof techniques [4]. For that, all the small uniforms will not be considered as a sharing of a secret value but as several random coins provided in input. The notion of sNIU introduced in [20] (detailed later on in the paper) will come handy. That way, we will be able to prove the masking security of the key generation and signature algorithm when the small uniforms are provided as inputs in Sect. 6.
2. **Reduction from t-probing EUF-CMA to standard EUF-CMA:** Next, we will use this probe simulation property offered by the NIU model (cf. Lemma 5.2) as part of a game based proof in the probing-EUF-CMA security model. Through a sequence of games, we prove that the probing-EUF-CMA security of Raccoon reduces on the black-box-EUF-CMA of a different version of Raccoon with smaller noise distributions, called small Raccoon. This reduction lets us include the probing adversary in the attack and reduce to a standard (non probing) EUF-CMA adversary. This proof is presented in Sect. 7.
3. **Unforgeability and smooth Rényi divergence:** Finally, the proof concludes with the black-box security of small Raccoon. Such a proof is close to existing EUF-CMA proofs of signatures following the Fiat–Shamir with aborts framework with a significative difference. To allow a complete end-to-end proof, we avoid any heuristic assessments and introduce the notion of smooth Rényi divergence for obtaining provable and tighter parameters. This proof is presented in Sect. 7.3.

In Sect. 8, we instantiate the parameters to valid our proof and confirm that the current NIST submission is secure.

### 1.3 Related Works

**SCA Against Dilithium.** Several side-channel attacks against post-quantum schemes have been published. For concision, we only mention those related to Dilithium, which shares similarities with Raccoon. Since its initial publication, a string of increasingly practical side-channel attacks have been proposed against unprotected implementations of Dilithium: see for example [8,9,22,29,32,39,40].

**Masking Lattice Schemes.** The formal study of masking lattice-based signatures has been initiated by Barthe et al. [5], which studied the GLP signature. Since then, BLISS [6] and qTESLA [23] have also been studied from a masking perspective. Masked implementations of Dilithium have been proposed in [3,12,34].

**Masking-Friendly Signatures.** A few masking-friendly signatures have been proposed in the past two years.

- *Mitaka.* Espitau et al. [21] proposed the Mitaka scheme, a masking-friendly variant of Falcon. A flaw in the security proof of Mitaka, as well as a practical attack in the $t$-probing model, was later demonstrated by Prest [37].
- *IEEE SP Raccoon.* At IEEE S&P 2023, del Pino et al. [17] presented a lattice-based masking-friendly signature, also called Raccoon. Our scheme is a conceptual descendent of the scheme from [17], with significant improvements. While both versions of Raccoon are Fiat-Shamir lattice-based signatures, the security proof of [17] relies on several heuristic arguments, and the scheme itself is less compact than ours due to the use of a variant of *uniform secret* LWR. In comparison, our design is more streamlined, more compact, relies on standard assumptions and has a formal security proof.
- *Plover.* Since the original publication of Raccoon as a NIST candidate [16], Esgin et al. [20] have proposed Plover, a signature scheme heavily inspired from our scheme, including the use of AddRepNoise. The key insight of Plover is to realize that our techniques are not limited to Fiat-Shamir signatures, and can also be applied in a hash-then-sign setting. Conversely, [20] introduced the NIU notion, a useful abstraction that we re-use in our analysis.

## 2 Preliminaries

We provide the minimal set of preparation. We refer the readers to the full version for more details. First, let us prepare some notations. We note $\mathbb{N}$ the set of non-negative integers, including zero. Given $n \in \mathbb{N}$, we denote by $[n]$ the set $\{0, 1, \ldots, n-1\}$. Let $f : X \to Y$ be a function, and $x \in X$. When $f$ is deterministic, we use the notation $y := f(x)$ to indicate that we assign the output of $f(x)$ to $y$. When $f$ is randomized, we instead use the notation $y \leftarrow f(x)$. From a programming viewpoint, both of these notations indicate an assignment of the result to the variable on the left. Given a probability distribution $\mathcal{D}$ over $Y$, we note $y \leftarrow \mathcal{D}$ to express that $y \in Y$ is sampled from $\mathcal{D}$.

### 2.1 Hardness Assumptions

The security of Raccoon is based on the Module Learning with Errors (MLWE) and Module Short Integer Solutions (MSIS) assumptions. More precisely, we rely on the Self Target MSIS (SelfTargetMSIS) problem, a variant of the MSIS problem, where the problem is defined relative to some hash function modeled as a random oracle. This assumption also underlies the security of Dilithium.

### 2.2 Masking Preliminaries

We consider all operations and variables used in algorithms to be over the scalar ring $\mathcal{R}_q$ (i.e. we consider that basic operations are done directly on polynomials in $\mathcal{R}_q$), this entails that we consider that probes leak full polynomials in $\mathcal{R}_q$ and not bits or even coefficients (leading to a stronger attacker model). An algorithm is defined as a sequence of gadget calls, each gadget being a sequence of (probabilistic or deterministic) assignments of expressions to local variables.

**Well-Formed Gadgets.** We say a gadget is well-formed if it is written in SSA (single static assignment) form, i.e. if its scalar variables appear at most once on the left-hand side of an assignment, and if all assignments are three-address code instructions, i.e. of the form $a = b * c$ with $*$ an operator. These restrictions ensure that all intermediate values are captured by local variables at some point in the code. An algorithm is well formed if in all gadget calls $\mathbf{b} = G(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ the variables $\mathbf{b}, \mathbf{x}_1, \ldots, \mathbf{x}_k$ are pairwise disjoint. While some algorithms we provide are not well formed (e.g., Algorithms 1 and 2), it is clear that this can be easily remedied by indexing variables and adding new local variables.

We use the notation $[\![\mathbf{x}]\!] = (\mathbf{x}_i)_{i \in [d]}$ to denote a tuple of $d$ values in $\mathcal{R}_q$, which implicitly defines the value $\mathbf{x} = \sum_0^{d-1} \mathbf{x}_i \in \mathcal{R}_q$. This notation is used to express that the secret value $\mathbf{x}$ is shared as $d$ additive shares as the encoding $[\![\mathbf{x}]\!]$.

**Variables' Values and Names.** We will distinguish variables (designated by a binary string representing their name) from the values they take (in the scalar ring $\mathcal{R}_q$), all objects pertaining to variables (singular variables, vectors, sets, etc.) will have a name with a bar (e.g. $\bar{x} \in \{0,1\}^*$, $\bar{\mathcal{V}} \subset \{0,1\}^*$), while the corresponding value will not (e.g. $x \in \mathcal{R}_q$).

For a gadget $G$ we define the local variables of $G$ as $\bar{\mathcal{V}}_G \subset \{0,1\}^*$ (noted $\bar{\mathcal{V}}$ when the gadget is clear from the context), since all variables are assigned only once we can match the position of a variable with its name. For a program $P$ with input scalar variables $(\bar{a}_1, \ldots, \bar{a}_N)$ that calls the gadgets $G_1, \ldots, G_k$, (with $N, k \in \mathbb{N}$), we will consider the set of variables $\bar{\mathcal{V}}_P = \{\bar{a}_1, \ldots, \bar{a}_N\} \uplus \bar{\mathcal{V}}_{G_1} \uplus \ldots \uplus \bar{\mathcal{V}}_{G_k}$ (where the local variables of $G_i$ are additionally labelled with $i$ to differentiate between gadgets and $\uplus$ is the disjoint union). Note that since all gadgets are written in three-address code SSA form, all intermediate computations and output variables are at some point stored locally in a uniquely defined local variable $\bar{v} \in \bar{\mathcal{V}}_P$. We thus define the set of all possible probes as the set $\bar{\mathcal{V}}_P$ of all local variables as well as the input variables.

*Remark 2.1.* We will consider that a program $P$ always outputs all unmasked values it computes even if they are not explicitly returned by $P$.

**Definition 2.1 (Probes).** *For a well-formed program $P$ with variables $\bar{\mathcal{V}}_P$ and input variables $\bar{a}_1, \ldots, \bar{a}_N$, a set of probes is a set $\bar{\mathscr{I}} \subset \bar{\mathcal{V}}_P$. For any set $\bar{\mathscr{I}} \subset \bar{\mathcal{V}}_P$ and any scalars $\mathcal{X} = (a_1, \ldots, a_N)$ we will denote as $\mathsf{ExecObs}(P, \bar{\mathscr{I}}, \mathcal{X})$ the joint distribution of the (masked and unmasked) outputs of $P(a_1, \ldots, a_N)$ and of all the values taken by the variables in $\bar{\mathscr{I}}$. In particular for*

$$(out_{\mathrm{masked}}, out_{\mathrm{unmasked}}, \mathscr{L}) \leftarrow \mathsf{ExecObs}(P, \bar{\mathscr{I}}, \mathcal{X}),$$

*$out_{\mathrm{masked}}$ (resp. $out_{\mathrm{unmasked}}$) is the masked (resp. unmasked) output of $P(a_1, \ldots, a_N)$ for some internal random coins and $\mathscr{L}$ is the value taken by the variables in $\bar{\mathscr{I}}$ for these random coins.*

**Probing Model.** We recall standard non-interference results from [4].

**Definition 2.2 (Perfect simulatability, reformulation of [4]).** *Let $\bar{\mathscr{I}}$ be a set of probes of a gadget $\mathbf{G}$ with input shares $\bar{\mathcal{X}}$. We say that the PPT simulator $(\mathsf{SimIn}, \mathsf{Simout})$ perfectly simulates the probes $\bar{\mathscr{I}}$ if and only if for any input values $\mathcal{X}$, $\mathsf{SimIn}(\mathbf{G}, \bar{\mathscr{I}})$ outputs a subset $\bar{\mathcal{X}}' \subset \mathcal{X}$ of the input variables of $\mathbf{G}$, and $\mathsf{SimOut}(\mathbf{G}, \mathcal{X}')$ (where $\mathcal{X}'$ is the values taken by $\mathcal{X}$ at indices $\bar{\mathcal{X}}'$) outputs a tuple of values such that the marginal distribution of $\mathscr{L}$, for $(out_{\mathrm{masked}}, out_{\mathrm{unmasked}}, \mathscr{L}) \leftarrow \mathsf{ExecObs}(P, \bar{\mathscr{I}}, \mathcal{X})$, and $\mathsf{SimOut}(\mathbf{G}, \mathcal{X}')$ are identical.*

**Definition 2.3 (Non Interference [4]).** *A gadget is said $(d-1)$-non-interfering (written $(d-1)$-$\mathsf{NI}$ for short) iff any set of probes $\bar{\mathscr{I}}$ such that $|\bar{\mathscr{I}}| \leq d-1$ can be perfectly simulated (See Definition 2.2) by a simulator $(\mathsf{SimIn}, \mathsf{SimOut})$ such that $\mathsf{SimIn}(\mathbf{G}, \bar{\mathscr{I}})$ outputs a set $\bar{\mathcal{X}}'$ of at most $d-1$ shares of each input.*

**Definition 2.4 (Strong Non Interference [4]).** *A gadget is said $(d-1)$-strongly-non-interfering (written $(d-1)$-$\mathsf{sNI}$ for short) iff any set $\bar{\mathscr{I}}$ of at most $d-1 = d_{\mathrm{int}} + d_{\mathrm{out}}$ probes, where $d_{\mathrm{int}}$ are made on internal data and $d_{\mathrm{out}}$ are made on the outputs, can be perfectly simulated by a simulator $(\mathsf{SimIn}, \mathsf{SimOut})$ such that $\mathsf{SimIn}(\mathbf{G}, \bar{\mathscr{I}})$ outputs a set $\bar{\mathcal{X}}'$ of at most $d_{\mathrm{int}}$ shares of each input.*

**Lemma 2.1 (Composability of $\mathsf{NI}$ and $\mathsf{sNI}$ gadgets [5]).** *A well-formed algorithm is $\mathsf{NI}$ if all of its gadgets are $\mathsf{NI}$ or $\mathsf{sNI}$ and each sharing is used at most once as input of a non-$\mathsf{sNI}$ gadget. Moreover, a well-formed algorithm is $\mathsf{sNI}$ if it is $\mathsf{NI}$ and its output sharings are issued from a $\mathsf{sNI}$ gadget.*

Lastly, in this paper, the masking order is fixed at $d-1$ where $d$ is the number of shares. For simplicity, we omit the $d-1$ when referring to $\mathsf{NI}$/$\mathsf{sNI}$ properties.

## 2.3 Sum of Uniforms

Given a distribution $\mathcal{D}$ of support included in an additive group, we note $[T] \cdot \mathcal{D}$ the convolution of $T$ identical copies of $\mathcal{D}$; in other words, $[T] \cdot \mathcal{D}$ is the distribution of the sum of $T$ independent random variables, each being sampled from $\mathcal{D}$. Given integers $u, T > 0$, and if we note $\mathcal{U}(S)$ the uniform distribution over a finite $S$, we note:

$$\mathrm{SU}(u, T) = [T] \cdot \mathcal{U}(\{-2^{u-1}, \ldots, 2^{u-1} - 1\}).$$

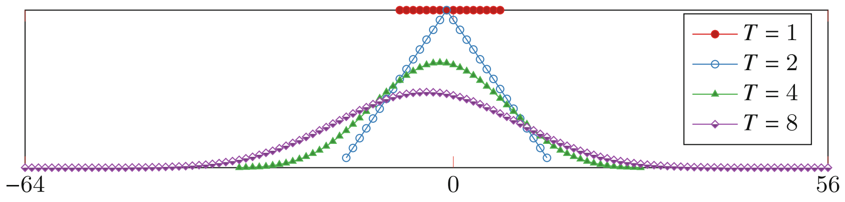

**Fig. 2.** The distribution SU(4, $T$), for $T \in \{1, 2, 4, 8\}$

The acronym SU stands for "sum of uniforms". This class of distributions is illustrated in Fig. 2. This distribution is highly desirable for our purposes, since for $T \geq 4$ it verifies statistical properties in the same way as Gaussians do. However, unlike Gaussians, they are straightforward to sample in constant-time and without requiring tables or elaborate mathematical machinery. This makes them adequate for Raccoon. Finally, we note RSU($u$, 1) the distribution over $\mathcal{R}$ obtained by sampling each integer coefficient of $a \in \mathcal{R}$ according to SU($u$, 1), and outputting $a$. More details about sums of uniforms can be found the full version of this paper.

## 3 The Raccoon Signature Scheme

In this section, we present our masking-friendly signature scheme called Raccoon. We describe the key generation (Algorithm 1), signing (Algorithm 2) and verification (Algorithm 3). Key generation and signing are always performed in a masked manner; when $d = 1$, the algorithmic descriptions remain valid but the algorithms are, in effect, unmasked.

We reference relevant variables and parameters in Table 1.

### 3.1 Key Generation

Masked key generation process is described by Algorithm 1. At a high-level, KeyGen generates $d$-sharings ($[\![\mathbf{s}]\!], [\![\mathbf{e}]\!]$) of small errors ($\mathbf{s}, \mathbf{e}$), computes the verification key as an LWE sample ($\mathbf{A}, \mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$), and rounds $\mathbf{t}$ for efficiency. A key technique is that $[\![\mathbf{s}]\!], [\![\mathbf{e}]\!]$ are generated in Lines 4 and 6 using our novel algorithm AddRepNoise (Algorithm 5).

**Table 1.** Overview of parameters used in the Raccoon signature.

| Parameter | Explanation |
|---|---|
| $(\mathcal{R}_q, n)$ | Polynomial ring $\mathcal{R}_q = \mathbb{Z}[X]/(q, X^n + 1)$ |
| $(k, \ell)$ | Dimension of public matrix $\mathbf{A} \in \mathcal{R}_q^{k\times\ell}$ |
| $d$ | Number of shares used, corresponding to a masking order $d - 1$ |
| $\mathrm{RSU}(a, b)$ | Sum of $a$ polynomials with coefficients uniform in $\{-2^{u-1}, \dots, 2^{u-1} - 1\}$ |
| $u_{\mathrm{t}}, u_{\mathrm{w}}$ rep | Parameter and repetition rate used for the sum of uniform in the secret/signature $\mathbf{s} \leftarrow \mathrm{RSU}^{\ell}(u_{\mathrm{t}}, \mathsf{rep})$, $\mathbf{r} \leftarrow \mathrm{RSU}^{\ell}(u_{\mathrm{w}}, \mathsf{rep})$ |
| $\nu_{\mathrm{t}}$ | Amount of bit dropping performed on verification key |
| $\nu_{\mathrm{w}}$ | Amount of bit dropping performed on (aggregated) commitment |
| $(q_{\mathrm{t}}, q_{\mathrm{w}})$ | Rounded moduli satisfying $(q_{\mathrm{t}}, q_{\mathrm{w}}) := (\lfloor q/2^{\nu_{\mathrm{t}}} \rfloor, \lfloor q/2^{\nu_{\mathrm{w}}} \rfloor)$ |
| $(C, \omega)$ | Challenge set $\{c \in \mathcal{R}_q \mid \|c\|_\infty = 1 \wedge \|c\|_1 = \omega\}$ s.t. $\lvert C\rvert \geq 2^{2\kappa}$ |
| $(B_2, B_\infty)$ | Two-norm and infinity-norm bounds on the signature |

**Algorithm 1.** KeyGen$(\emptyset) \rightarrow (\mathsf{vk}, \mathsf{sk})$

```
Output: Keypair vk, sk
1: $\mathsf{seed} \leftarrow \{0,1\}^\kappa$    ▷ $\kappa$-bit random seed for $\mathbf{A}$.
2: $\mathbf{A} := \mathsf{ExpandA}(\mathsf{seed})$    ▷ Similar to ExpandA in Dilithium. $\mathbf{A} \in \mathcal{R}_q^{k\times\ell}$.
3: $[\![\mathbf{s}]\!] \leftarrow \ell \times \mathsf{ZeroEncoding}(d)$    ▷ Masked zero vector $[\![\mathbf{s}]\!] \in (\mathcal{R}_q^{\ell})^d$. Algorithm 8.
4: $[\![\mathbf{s}]\!] \leftarrow \mathsf{AddRepNoise}([\![\mathbf{s}]\!], u_{\mathrm{t}}, \mathsf{rep})$    ▷ Generate the secret distribution. Algorithm 5.
5: $[\![\mathbf{t}]\!] := \mathbf{A} \cdot [\![\mathbf{s}]\!]$    ▷ Compute masked product $[\![\mathbf{t}]\!] \in (\mathcal{R}_q^{k})^d$.
6: $[\![\mathbf{t}]\!] \leftarrow \mathsf{AddRepNoise}([\![\mathbf{t}]\!], u_{\mathrm{t}}, \mathsf{rep})$    ▷ Add masked noise to $[\![\mathbf{t}]\!]$. Algorithm 5.
7: $\mathbf{t} := \mathsf{Decode}([\![\mathbf{t}]\!])$    ▷ Collapse $\mathbf{t} \in \mathcal{R}_q^{k}$. Algorithm 6.
8: $\mathbf{t} := \lfloor \mathbf{t} \rceil_{\nu_{\mathrm{t}}}$    ▷ Rounding and right-shift to modulus $q_{\mathrm{t}} = \lfloor q/2^{\nu_{\mathrm{t}}} \rfloor$.
9: return $(\mathsf{vk} := (\mathsf{seed}, \mathbf{t}), \mathsf{sk} := (\mathsf{vk}, [\![\mathbf{s}]\!]))$    ▷ Return serialized key pair.
```

## 3.2 Signing Procedure

The masked signing process is described by Algorithm 2. This signing procedure is similar to the "Lyubashevsky's Signature Without Aborts" in [1]. Again, the use of AddRepNoise is crucial in this procedure. The challenge computation is divided in two parts, first a $2\kappa$ bitstring is computed using the hash function ChalHash, then this bitstring is mapped to a ternary polynomial with fixed hamming weight using ChalPoly. As in previous works this distinction is made for ease of implementation and storage.

## 3.3 Verification Procedure

Algorithm 3 describes the signature verification process. Signature verification is not masked, and its parameters are independent of the number of shares $d$ used when creating the signature. As is usual in lattice signatures, verification performs a bound check and an equality check.

It is easy to check that the equation of line 7 verifies by construction when the signature algorithm is run honestly, we will fix the bounds $B_\infty$ and $B_2$ such that honest signatures verify with overwhelming probability (this is necessary for the reduction of Sect. 7.2 to go through).

**Algorithm 2.** Sign($[\![\mathsf{sk}]\!]$, msg) $\rightarrow$ sig

```
Input: Secret signing key $\mathsf{sk} = (\mathsf{vk}, [\![\mathbf{s}]\!])$, message to be signed $\mathsf{msg} \in \{0,1\}^*$.
Output: Signature $\mathsf{sig} = (c_{\mathsf{hash}}, \mathbf{h}, \mathbf{z})$ of msg under sk.
 1: $\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk}) \| \mathsf{msg})$   ▷ Bind vk with msg to form $\mu \in \{0,1\}^{2\kappa}$.
 2: $\mathbf{A} := \mathsf{ExpandA}(\mathsf{seed})$   ▷ Similar to ExpandA in Dilithium. $\mathbf{A} \in \mathcal{R}_q^{k \times \ell}$.
 3: $[\![\mathbf{r}]\!] \leftarrow \ell \times \mathsf{ZeroEncoding}(d)$   ▷ Masked zero vector $[\![\mathbf{r}]\!] \in (\mathcal{R}_q^\ell)^d$. Algorithm 8.
 4: $[\![\mathbf{r}]\!] \leftarrow \mathsf{AddRepNoise}([\![\mathbf{r}]\!], u_{\mathsf{w}}, \mathsf{rep})$   ▷ Add masked noise to $[\![\mathbf{r}]\!]$. Algorithm 5.
 5: $[\![\mathbf{w}]\!] := \mathbf{A} \cdot [\![\mathbf{r}]\!]$   ▷ Compute masked product $[\![\mathbf{w}]\!] \in (\mathcal{R}_q^k)^d$.
 6: $[\![\mathbf{w}]\!] \leftarrow \mathsf{AddRepNoise}([\![\mathbf{w}]\!], u_{\mathsf{w}}, \mathsf{rep})$   ▷ Add masked noise to $[\![\mathbf{w}]\!]$. Algorithm 5.
 7: $\mathbf{w} := \mathsf{Decode}([\![\mathbf{w}]\!])$   ▷ Collapse LWE commitment $\mathbf{w}$. Algorithm 6.
 8: $\mathbf{w} := \lfloor \mathbf{w} \rceil_{\nu_{\mathsf{w}}}$   ▷ Rounding and right-shift to modulus $q_{\mathsf{w}} = \lfloor q/2^{\nu_{\mathsf{w}}} \rfloor$.
 9: $c_{\mathsf{hash}} := \mathsf{ChalHash}(\mathbf{w}, \mu)$   ▷ Map $\mathbf{w}$ and $\mu$ to $c_{\mathsf{hash}} \in \{0,1\}^{2\kappa}$.
10: $c_{\mathsf{poly}} := \mathsf{ChalPoly}(c_{\mathsf{hash}})$   ▷ Map $c_{\mathsf{hash}}$ to $c_{\mathsf{poly}} \in C$.
11: $[\![\mathbf{s}]\!] \leftarrow \mathsf{Refresh}([\![\mathbf{s}]\!])$   ▷ Refresh $[\![\mathbf{s}]\!]$ before re-use. Algorithm 7.
12: $[\![\mathbf{r}]\!] \leftarrow \mathsf{Refresh}([\![\mathbf{r}]\!])$   ▷ Refresh $[\![\mathbf{r}]\!]$ before re-use. Algorithm 7.
13: $[\![\mathbf{z}]\!] := c_{\mathsf{poly}} \cdot [\![\mathbf{s}]\!] + [\![\mathbf{r}]\!]$   ▷ Masked response $[\![\mathbf{z}]\!] \in (\mathcal{R}_q^\ell)^d$.
14: $[\![\mathbf{z}]\!] \leftarrow \mathsf{Refresh}([\![\mathbf{z}]\!])$   ▷ Refresh $[\![\mathbf{z}]\!]$ before collapsing it. Algorithm 7.
15: $\mathbf{z} := \mathsf{Decode}([\![\mathbf{z}]\!])$   ▷ Collapse into response $\mathbf{z} \in \mathcal{R}_q^\ell$. Algorithm 6.
16: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - 2^{\nu_t} \cdot c_{\mathsf{poly}} \cdot \mathbf{t}$   ▷ "Noisy" LWE commitment.
17: $\mathbf{h} := \mathbf{w} - \lfloor \mathbf{y} \rceil_{\nu_{\mathsf{w}}}$   ▷ Compute hint $\mathbf{h} \in \mathcal{R}_{q_w}^k$. Subtraction mod $q_{\mathsf{w}}$.
18: $\mathsf{sig} := (c_{\mathsf{hash}}, \mathbf{h}, \mathbf{z})$
19: if $\{\mathsf{CheckBounds}(\mathsf{sig}) = \mathsf{FAIL}\}$ goto Line 3   ▷ Sanity check on the signature.
    Algorithm 4.
20: return sig   ▷ Return encoded signature triplet.
```

**Algorithm 3.** Verify(sig, msg, vk) $\rightarrow$ {OK or FAIL}

```
Input: Signature $\mathsf{sig} = (c_{\mathsf{hash}}, \mathbf{h}, \mathbf{z})$, message $\mathsf{msg} \in \{0,1\}^*$, public key $\mathsf{vk} = (\mathsf{seed}, \mathbf{t})$.
Output: Signature validity: OK (accept) or FAIL (reject).
 1: if $\mathsf{CheckBounds}(\mathsf{sig}) = \mathsf{FAIL}$ return FAIL   ▷ Norms check. Algorithm 4.
 2: $\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk}) \| \mathsf{msg})$ ; $\mathbf{A} := \mathsf{ExpandA}(\mathsf{seed})$
 3: $c_{\mathsf{poly}} := \mathsf{ChalPoly}(c_{\mathsf{hash}})$   ▷ Map $c_{\mathsf{hash}}$ to $c_{\mathsf{poly}} \in C$.
 4: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - 2^{\nu_t} \cdot c_{\mathsf{poly}} \cdot \mathbf{t}$   ▷ Scale $\mathbf{t}$ from $\mathbb{Z}_{q_t}$ to $\mathbb{Z}_q$ and recompute the commitment.
 5: $\mathbf{w}' := \lfloor \mathbf{y} \rceil_{\nu_{\mathsf{w}}} + \mathbf{h}$   ▷ Adjust the LWE commitment with hint (mod $q_{\mathsf{w}}$).
 6: $c'_{\mathsf{hash}} := \mathsf{ChalHash}(\mathbf{w}', \mu)$   ▷ Recompute $c'_{\mathsf{hash}} \in \{0,1\}^{2\kappa}$.
 7: if $c_{\mathsf{hash}} \neq c'_{\mathsf{hash}}$ return FAIL   ▷ Check commitment.
 8: return OK   ▷ Signature is accepted.
```

## 3.4 Helper Algorithms

The following are algorithms used within our key generation (Algorithm 1), signing (Algorithm 2) and verification (Algorithm 3). The algorithm AddRepNoise (Algorithm 5) is the most interesting one, which we come back later when discussing probing security.

**Checking Bounds.** The function CheckBounds (Algorithm 4) is used to check the norm bounds and encoding soundness of signatures by both the verification function (Algorithm 3), but also by the signing function (Algorithm 2). Note that unlike rejection, CheckBounds is used to enforce the zero-knowledge property, and therefore it does need to be masked. Rather, it detects signatures that are a bit too large. Note that CheckBounds could be removed entirely at the cost of a slight increase in signature size (and therefore a slight decrease in security).

**Algorithm 4.** CheckBounds(sig) $\to$ {OK or FAIL}

```
Input: Signature sig = $(c_{\mathsf{hash}}, \mathbf{h}, \mathbf{z})$.
Output: Format validity check OK or FAIL.
1: if $(\|(\mathbf{z}, 2^{\nu_w} \cdot \mathbf{h})\|_\infty > B_\infty)$ or $(\|(\mathbf{z}, 2^{\nu_w} \cdot \mathbf{h})\|_2 > B_2)$ return FAIL else return OK
```

**Error Distributions.** AddRepNoise (Algorithm 5) implements the Sum of Uniforms (SU) distribution $\mathrm{SU}(u, d \cdot \mathsf{rep})$ (Sect. 2.3) in a masked implementation. AddRepNoise interleaves noise additions and refresh operations; more precisely, for each (masked) coefficient $[\![a]\!]$ of $[\![\mathbf{v}]\!]$, small uniform noise is added to each share of $[\![a]\!]$, then $[\![a]\!]$ is refreshed, and this operation is repeated rep times. The security properties of AddRepNoise is analyzed in Sect. 6.2.

**Algorithm 5.** AddRepNoise($[\![\mathbf{v}]\!], u, \mathsf{rep}$) $\to$ $[\![\mathbf{v}]\!]$

```
Input: Masked vector $[\![\mathbf{v}]\!] = (\mathbf{v}_j)_{j \in [d]} = (v_{i,j})_{i \in [\mathsf{len}(\mathbf{v})], j \in [d]}$.
Input: Bit size (distribution parameter) $u$.
Input: Global repetition count parameter rep.
Output: Updated $[\![\mathbf{v}]\!]$ with $\mathrm{SU}(u, d \cdot \mathsf{rep})$ distribution added to each coefficient of $\mathbf{v}$.
1: for $i \in [\mathsf{len}(\mathbf{v})]$ do                          ▷ Vector index.
2:     for $i_{\mathsf{rep}} \in [\mathsf{rep}]$ do                     ▷ Repetition index.
3:         for $j \in [d]$ do                                           ▷ Share index.
4:             $\rho \leftarrow \mathrm{RSU}(u, 1)$                     ▷ uniform sample of $u$ bits
5:             $v_{i,j} \leftarrow v_{i,j} + \rho$                      ▷ Add small uniform to the polynomial.
6:         $[\![\mathbf{v}_i]\!] \leftarrow$ Refresh($[\![\mathbf{v}_i]\!]$)   ▷ Refresh polynomial on each repeat.
7: return $[\![\mathbf{v}]\!]$
```

**Challenge Computation.** As in Dilithium, the challenge computation is split in two subroutines: ChalHash computes a hash digest, and ChalPoly expands it into a challenge polynomial $c_{\mathsf{poly}}$ that is (pseudo-randomly) uniform in the set $C = \{c \in \mathcal{R}, \|c\|_1 = \omega\}$. These functions do not need to be masked.

**Refresh and Decoding Gadgets.** Lastly, we recall some useful gadgets. Refresh (Algorithm 7) generates a fresh $d$-sharing of a value in $\mathcal{R}_q$, or "refresh" the $d$-sharing. This operation is important for security against $t$-probing adversaries. Refresh uses ZeroEncoding (Algorithm 8) as a subroutine. Both algorithms perform $O(d \log d)$ basic operations over $\mathcal{R}_q$ and require $O(d \log(d) \log(q))$ bits of entropy. While we present ZeroEncoding as a recursive algorithm, one can see that it can be computed in-place and its memory requirement is $O(d)$. Remark

**Algorithm 6.** Decode($[\![x]\!]$) $\rightarrow x$

```
Input: $d$-sharing $[\![x]\!] = (x_i)_i$ of $x \in \mathcal{R}_q$
Output: The clear value $x \in \mathcal{R}_q$
1: $[\![x]\!] \leftarrow$ Refresh($[\![x]\!]$)
2: return $x := \sum_{i \in [d]} x_i$
```

**Algorithm 7.** Refresh($[\![x]\!]$) $\rightarrow [\![x]\!]'$

```
Input: A $d$-sharing $[\![x]\!]$ of $x \in \mathcal{R}_q$
Output: A fresh $d$-sharing $[\![x]\!]$ of $x$
1: $[\![z]\!] \leftarrow$ ZeroEncoding($d$)
2: return $[\![x]\!]' := [\![x]\!] + [\![z]\!]$
```

**Algorithm 8.** ZeroEncoding($d$) $\rightarrow [\![z]\!]_d$

```
Input: A power-of-two integer $d$, a ring $\mathcal{R}_q$
Output: A uniform $d$-sharing $[\![z]\!] \in \mathcal{R}_q^d$ of $0 \in \mathcal{R}_q$
1: if $d = 1$ then
2:     return $[\![z]\!]_1 := (0)$        ▷ There is only one way to encode zero into 1 share.
3: $[\![z_1]\!]_{d/2} \leftarrow$ ZeroEncoding($d/2$)        ▷ Recursively obtain left side.
4: $[\![z_2]\!]_{d/2} \leftarrow$ ZeroEncoding($d/2$)        ▷ Recursively obtain right side.
5: $[\![r]\!]_{d/2} \xleftarrow{M} \mathcal{R}_q^{d/2}$        ▷ Sampled using a Mask Random Generator (MRG).
6: $[\![z_1]\!]_{d/2} := [\![z_1]\!]_{d/2} + [\![r]\!]_{d/2}$        ▷ Add to the left side.
7: $[\![z_2]\!]_{d/2} := [\![z_2]\!]_{d/2} - [\![r]\!]_{d/2}$        ▷ Subtract from the right side.
8: return $[\![z]\!]_d := ([\![z_1]\!]_{d/2} \parallel [\![z_2]\!]_{d/2})$        ▷ Concatenate the two.
```

that our ZeroEncoding algorithm entails that the number of shares $d$ is a power of 2, as the rest of our algorithms are agnostic to this property we could use a ZeroEncoding that produces a more fine-grained number of shares to obtain different parameters (e.g. by using Algorithm 8 and collapsing some of the shares).

We describe in Algorithm 6 a Decode gadget that takes $[\![\mathbf{x}]\!] = (\mathbf{x}_i)_{i \in [t+1]}$ as input, refreshes it with Algorithm 7, then computes the sum $\mathbf{x}_0 + \cdots + \mathbf{x}_{d-1} \bmod q$. When the decoding gadget is already preceded by a refresh gadget, one of them may be omitted. Decode is similar to the algorithm “FullAdd” from [5, Alg. 16].

## 4 Smooth Rényi Divergence and Useful Bounds

Raccoon's core design choice is using the sum of uniforms distributions as opposed to the discrete Gaussian distributions. From a practical point of view, the sum of uniforms distribution is a much simpler distribution to mask and implement. On the other hand, from a theoretical point of view, it poses more challenges, as there are far fewer established statistical guarantees usable in cryptography. Notably, since the sum of uniforms distribution only has finite support, a standard proof technique used in lattice-based cryptography relying on the Rényi divergence breaks down. To this end, we generalize the Rényi divergence and prepare useful statistical bounds on the sum of uniforms distribution.

## 4.1 Smooth Rényi Divergence

The usual Rényi divergence is undefined for distributions $P, Q$ of supports not included in one another. For example, this happens when $P = \mathrm{SU}(u, T)$ and $Q = P + a$, for any $a \neq 0$. The *smooth* Rényi divergence (Definition 4.1) addresses these limitations by combining the statistical distance and the Rényi divergence. The statistical distance component captures problematic sets (typically, distribution tails), while the Rényi divergence component benefits from the same efficiency as the usual Rényi divergence over unproblematic parts of the supports.

**Definition 4.1 (Smooth Rényi divergence).** *Let $\epsilon \geq 0$ and $1 < \alpha < \infty$. Let $P, Q$ be two distributions of countable supports* $\mathrm{Supp}(P) \subseteq \mathrm{Supp}(Q) = X$. *The smooth Rényi divergence of parameters $(\alpha, \epsilon)$ between $P$ and $Q$ is defined as:*

$$R_\alpha^\epsilon(P; Q) = \min_{\substack{\Delta_{\mathrm{SD}}(P'; P) \leq \epsilon \\ \Delta_{\mathrm{SD}}(Q'; Q) \leq \epsilon}} R_\alpha(P'; Q'), \tag{1}$$

*where $\Delta_{\mathrm{SD}}$ and $R_\alpha$ denote the statistical distance and the Rényi divergence, respectively:*

$$\Delta_{\mathrm{SD}}(P; Q) = \frac{1}{2} \sum_{x \in X} |P(x) - Q(x)|, \qquad R_\alpha(P; Q) = \left( \sum_{x \in X} \frac{P(x)^\alpha}{Q(x)^{\alpha-1}} \right)^{\frac{1}{\alpha-1}}.$$

While [18] has also provided a definition of smooth Rényi divergence, we argue that our definition is more natural. Indeed, it satisfies variations of properties that are expected from classical Rényi divergences. These are listed in Lemma 4.1.

*Tools for Smooth Rényi Divergence.* We review some basic properties of the smooth Rényi divergence.

**Lemma 4.1.** *The smooth Rényi divergence satisfies the following properties.*

1. ***Data processing inequality.*** *Let $P, Q$ be two distributions, let $\epsilon \geq 0$, and $g$ be a randomized function over (a superset of)* $\mathrm{Supp}(P) \cup \mathrm{Supp}(Q)$.
$$R_\alpha^\epsilon(g(P); g(Q)) \leq R_\alpha^\epsilon(P; Q). \tag{2}$$
2. ***Probability preservation.*** *For any event $E \subseteq$* $\mathrm{Supp}(Q)$*:*
$$P(E) \leq (Q(E) + \epsilon)^{(\alpha-1)/\alpha} \cdot R_\alpha^\epsilon(P; Q) + \epsilon. \tag{3}$$
3. ***Tensorization.*** *Let $(P_i)_{i \in I}, (Q_i)_{i \in I}$ be two finite families of distributions, let $\epsilon_i \geq 0$ for $i \in I$, and let $\epsilon = \sum_{i \in I} \epsilon_i$.*
$$R_\alpha^\epsilon\left( \prod_{i \in I} P_i; \prod_{i \in I} Q_i \right) \leq \prod_{i \in I} R_\alpha^{\epsilon_i}(P_i; Q_i). \tag{4}$$

*Proof.* We recall that $\Delta_{\mathrm{SD}}$ and $(R_\alpha^\alpha - 1)$ can be cast as $f$-divergences, following Csiszár's terminology [15]. Item 1 follows from the data processing inequality for $f$-divergences. Item 2 is a special case of Item 1 . Finally, Item 3 follows from tensorization properties of the statistical distance and the Rényi divergence. □

## 4.2 Useful Bounds on Sum of Uniforms

We bound the smooth Rényi divergence between two sums of uniforms, centered at either 0 or a small offset. This will be a key lemma establishing the hardness of standard EUF-CMA security of the small Raccoon (cf. Section 7.3). Due to page limitation, the proof is provided in the full version of this paper.

**Lemma 4.2.** *Let $T, u, N \in \mathbb{N}$ and $c \in \mathbb{Z}$ such that $T \geq 4$ and $N = 2^u$. Let $P = \mathrm{SU}(u, T)$ and $Q$ the distributions corresponding to shifting the support of $P$ by $c$. Let $\alpha \geq 2$ and $\tau > 0, \epsilon > 0$ be such that:*

*1. $\alpha\,|c| \leq \tau = o(N/(T-1))$ ;*
*2. $\epsilon = \frac{(\tau+T)^T}{N^T\,T!}$.*

*Then:*

$$R_\alpha^\epsilon(P; Q) \leq \left(1 + \frac{\alpha(\alpha-1)}{2}\left(\frac{Tc}{N}\right)^2 + \frac{2}{T!}\left(\frac{T\alpha c}{N}\right)^2 + \epsilon + O\left(\left(\frac{T\alpha c}{N}\right)^3\right)\right)^{1/(\alpha-1)} \tag{5}$$

*Gap with Practice.* In practice, Lemma 4.2 is a bit sub-optimal. Let us note $\sigma^2 = \frac{T(N^2-1)}{12}$ the variance of $P$ and $Tc = o(N)$, which follows from Item 1 above. We also use the notation $a \lesssim b$ for $a \leq b + o(b)$. Then, Lemma 4.2 essentially tells us that $\log R_\alpha^\epsilon(P; Q) \lesssim \frac{\alpha}{2}\left(\frac{Tc}{N}\right)^2 \sim \frac{\alpha\, c^2\, T^3}{24\,\sigma^2}$. In comparison, [1, Lemma 2.28] tells that if $P$ is instead a Gaussian of parameter $\sigma$, then $\log R_\alpha(P; Q) \leq \frac{\alpha\, c^2}{2\,\sigma^2}$. Thus there is a gap $O(T^3)$ between Lemma 4.2 and [1, Lemma 2.28].

One could assume that this gap is caused by a fundamental difference between Gaussians and sums of uniforms. However we performed extensive experiments and found that this gap does not exist in practice, i.e., it seems to be an artifact of our proof. For this reason, we put forward the following conjecture which ignores this gap and which we use when setting our concrete parameters. Due to page limitation, we expand upon Conjecture 4.1 in the full version.

*Conjecture 4.1.* Under the conditions of Lemma 4.2, we have

$$R_\alpha^\epsilon(P; Q) \lesssim \exp\left(\frac{C_{\text{Rényi}} \cdot \alpha \cdot c^2\,(1 + \frac{2}{\alpha-1})}{T \cdot N^2}\right) \tag{6}$$

for a constant $C_{\text{Rényi}} \approx 6$. Therefore, for any $M$-dimensional vector $\mathbf{c}$, $\mathcal{P} = P^M$ and $\mathcal{Q} = \mathbf{c} + Q^M$, and further assuming $\alpha = \omega_{\mathsf{asymp}}(1)$ and $T = o(\alpha|c_i|)$ for all the $i$-th ($i \in [M]$) entry of $\mathbf{c}$, we have:

$$R_\alpha^\epsilon(\mathcal{P}; \mathcal{Q}) \lesssim \exp\left(\frac{C_{\text{Rényi}} \cdot \alpha \cdot \|\mathbf{c}\|_2^2}{T \cdot N^2}\right), \tag{7}$$

$$\text{where} \quad \epsilon \approx \frac{\alpha^T\,\|\mathbf{c}\|_T^T}{N^T\,T!} \lesssim \frac{1}{\sqrt{2\,\pi\,T}}\left(\frac{\alpha\, e\,\|\mathbf{c}\|_2}{N\,T}\right)^T \tag{8}$$

and where $\|\mathbf{c}\|_T \leq \|\mathbf{c}\|_2$ is the $L_T$ norm.

## 5 Enhancing NI/sNI for Probing EUF-CMA Security

We first formally define NI security against a probing adversary, the security model in which Raccoon will later be prove in. We then argue that existing probing tools/models discussed in Sect. 2.2 are insufficient to prove EUF-CMA security and prepare useful tools that may be of independent interest. Our tools build on the recent techniques developed by [20] (cf. Sect. 1.3).

### 5.1 EUF-CMA Security in the Probing Model

We use the definition of [5] that captures the fact that no PPT adversary with access to less than $d-1$ probes on KeyGen and Sign should be able to break EUF-CMA security (i.e., unforgeability). Below, our definition slightly deviates from theirs as we rely on more generalized (and formal) notion of probes captured by the function ExecObs (cf. Definition 2.1).

**Definition 5.1.** *Let $d \geq 1$ an integer, $Q_s$ be a fixed maximum amount of signature queries. A signature scheme* (KeyGen, Sign, Verify) *with an efficient signing key update algorithm* KeyUpdate *is* EUF-CMA*-secure in the* $(d-1)$*-probing model if any probabilistic polynomial time adversary has a negligible probability of winning the game presented in Fig. 3.*

As in [5], we assume a KeyUpdate algorithm that refreshes the secret key between signature queries and cannot be probed by the attacker. This is performed to avoid attackers probing more than $d-1$ shares of the secret across different signature queries. See [5, Remark 3] for more details.

*Remark 5.1 (Standard* EUF-CMA *security).* We note that Definition 5.1 incorporates the standard notion of standard EUF-CMA (i.e., 0-probing). For this, we define KeyUpdate to be the identify function; the restriction that the adversary can only query an empty set for the set of probes is enforced by the winning condition.

### 5.2 Insufficiency of the NI/sNI Models

At first glance, all subroutines of Raccoon can be proven composable in the NI model. However, careful consideration shows that the NI model does not capture security when the intermediate values are not uniformly distributed and biased with the knowledge of the public output. Indeed, for example in the KeyGen, the combined knowledge of some shares of $[\![\mathbf{s}]\!]$ and of the public key vk allows one to decrease the key-recovery security (decreasing the standard deviation of the short vector in a lattice) as presented in the technical overview in Sect. 1.

The gist of the problem when taking the output of an algorithm into account comes from the fact that the NI model proves that there exists a simulator that can simulate any set of probes from a subset of the input shared secrets of the

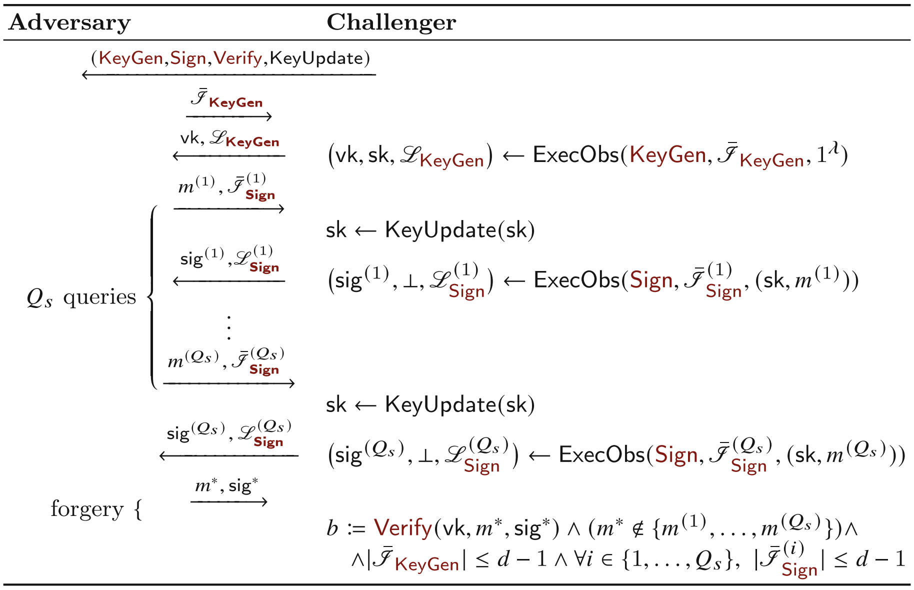
Adversary
Challenger
$\xleftarrow{(\mathsf{KeyGen},\mathsf{Sign},\mathsf{Verify},\mathsf{KeyUpdate})}$
$\xrightarrow{\bar{\mathcal{J}}_{\mathsf{KeyGen}}}$
$\xleftarrow{\mathsf{vk},\mathscr{L}_{\mathsf{KeyGen}}}$
$(\mathsf{vk},\mathsf{sk},\mathscr{L}_{\mathsf{KeyGen}}) \leftarrow \mathsf{ExecObs}(\mathsf{KeyGen},\bar{\mathcal{J}}_{\mathsf{KeyGen}},1^\lambda)$
$Q_s$ queries
$\xrightarrow{m^{(1)},\bar{\mathcal{J}}^{(1)}_{\mathsf{Sign}}}$
$\mathsf{sk} \leftarrow \mathsf{KeyUpdate}(\mathsf{sk})$
$\xleftarrow{\mathsf{sig}^{(1)},\mathscr{L}^{(1)}_{\mathsf{Sign}}}$
$(\mathsf{sig}^{(1)},\perp,\mathscr{L}^{(1)}_{\mathsf{Sign}}) \leftarrow \mathsf{ExecObs}(\mathsf{Sign},\bar{\mathcal{J}}^{(1)}_{\mathsf{Sign}},(\mathsf{sk},m^{(1)}))$
$\vdots$
$\xrightarrow{m^{(Q_s)},\bar{\mathcal{J}}^{(Q_s)}_{\mathsf{Sign}}}$
$\mathsf{sk} \leftarrow \mathsf{KeyUpdate}(\mathsf{sk})$
$\xleftarrow{\mathsf{sig}^{(Q_s)},\mathscr{L}^{(Q_s)}_{\mathsf{Sign}}}$
$(\mathsf{sig}^{(Q_s)},\perp,\mathscr{L}^{(Q_s)}_{\mathsf{Sign}}) \leftarrow \mathsf{ExecObs}(\mathsf{Sign},\bar{\mathcal{J}}^{(Q_s)}_{\mathsf{Sign}},(\mathsf{sk},m^{(Q_s)}))$
forgery {
$\xrightarrow{m^*,\mathsf{sig}^*}$
$b := \mathsf{Verify}(\mathsf{vk},m^*,\mathsf{sig}^*) \wedge (m^* \notin \{m^{(1)},\ldots,m^{(Q_s)}\})\wedge$
$\wedge|\bar{\mathcal{J}}_{\mathsf{KeyGen}}| \le d-1 \wedge \forall i \in \{1,\ldots,Q_s\},\ |\bar{\mathcal{J}}^{(i)}_{\mathsf{Sign}}| \le d-1$

**Fig. 3.** EUF-CMA security game in the $d-1$-probing model. See Definition 2.1 for the definition of ExecObs.

algorithm. However, the aforementioned property does not entail that the distribution of the probes can be simulated when taking into account the output. This is clearly apparent in Definition 2.2 where the definition requires $\mathsf{SimOut}(\mathbf{G},\mathcal{X}')$ and $\mathscr{L}$ to be identically distributed, but not $(out_{\text{unmasked}},\mathsf{SimOut}(\mathbf{G},\mathcal{X}'))$ and $(out_{\text{unmasked}},\mathscr{L})$.

```
non-NIU(([[v]])
1: for $j \in [d]$ do
2:     $\rho_j \leftarrow \mathrm{RSU}(u,1)$
3:     $v'_j \leftarrow v_j + \rho_j$
```

```
NIU(([[v]], $(\rho_i)_{i\in[d]}$)
1: for $j \in [d]$ do
2:     $v'_j \leftarrow v_j + \rho_j$
```

**Fig. 4.** Example of an algorithm without unshared inputs (left), and its equivalent where randomnesses are explicitly passed as unshared inputs (right).

To see that the marginal distributions being identical is insufficient we give a simple example in Fig. 4: both algorithms are trivially NI since any probe $\bar{\rho}_j$ or $\bar{v}'_j$ can be simulated by sampling a small uniform and outputting it or adding it to the corresponding input $v_j$. However, if we consider the unmasked value

$w$ as a public output, a simulator taking as input shares of $[\![\mathbf{v}]\!]$ cannot output probes that are correlated to $w$. For example, in gadget $\mathsf{non\text{-}NIU}$, consider the set of probes $\bar{\mathscr{I}} = \{\bar{v}'_1\}$ which corresponds to the sum of $v_1$ and $\rho_1$. A simulator $(\mathsf{SimIn}, \mathsf{SimOut})$ can perfectly simulate $\bar{\mathscr{I}}$ by setting $\mathsf{SimIn}(\mathsf{non\text{-}NIU}, \bar{\mathscr{I}}) := \bar{v}_1$, and $\mathsf{SimOut}(\mathsf{non\text{-}NIU}, v_1) := v_1 + \mathrm{RSU}(u, 1)$. Then the variable $\mathscr{L} = v'_1$ being probed has the same distribution as $\mathsf{SimOut}(\mathsf{non\text{-}NIU}, v_1)$. However the distribution of $(\mathscr{L}, out_{\text{unmasked}}) = (v_1 + \rho_1, v + \rho_1 + \ldots + \rho_d)$ is clearly not the same as that of $(\mathsf{SimOut}(\mathsf{non\text{-}NIU}, v_1), out_{\text{unmasked}}) = (v_1 + \mathrm{RSU}(u, 1), v + \rho_1 + \ldots + \rho_d)$.

## 5.3 NI/sNI with Unshared Inputs

To be able to handle cases where the values being probed are correlated with the public output we will modify the relevant gadgets and consider that any correlated random variables will be considered as inputs. We will formalize this idea with a model named Non-Interference with Unshared Inputs (NIU) (see Definitions 5.2 and 5.3 below), in which we will consider a variant of the algorithm where all random values that can affect the distribution of the output will be considered as inputs of the algorithm. While this model is stronger than the NI model, as it can be used to prove security even in the presence of leakage (see Lemma 5.2), we note that once an algorithm $P$ has been modified to have its relevant randomness moved to inputs, the difference with the NI model becomes mostly syntactical since the new inputs of the algorithm and gadgets can be considered as just an additional shared secret input.

As an example, see the algorithm NIU in Fig. 4 where we parse the random samples $\rho_i$ as inputs rather than local variables. NIU thus takes two tuples of $d$ values as input, and can as before be proven NI (where we artificially consider the tuple $(\rho_i)_{i\in[d]}$ as a shared input). However this time the NI proof does entail that the joint distribution of the probes and the output is identical to that of the simulator and output, because the output is a deterministic function of the input. Using the same set of probes $\bar{\mathscr{I}} = \bar{v}'_1$ as before, this time the simulator needs to use two input values to simulate the probe: $\mathsf{SimIn}(\mathsf{NIU}, \bar{\mathscr{I}}) := \{\bar{v}_1, \bar{\rho}_1\}$ , however since each input variable is in a different shared input this simulator fits the definition of 2-NI in Definition 2.3, and we can set $\mathsf{SimOut}(\mathsf{NIU}, \{v_1, \rho_1\}) := v_1 + \rho_1$ . It is obvious that in this case $(\mathscr{L}, out_{\text{unmasked}}) = (v_1 + \rho_1, v + \rho_1 + \ldots + \rho_d) = (\mathsf{SimOut}(\mathsf{NIU}, \{v_1, \rho_1\}), out_{\text{unmasked}})$ .

We will now first formalize the $(d-1)$-NIU notion, introduced in [20], in Definitions 5.2 and 5.3. Using the formalism of Sect. 2.2 we can then state and prove composition properties in Lemma 5.1, which are straightforward though never made explicit in [20]. Finally we can prove the core simulatability property of Lemma 5.2 which shows that when passing appropriate random variables as input NIU is sufficient to simulate the joint distribution of the probes and outputs of an algorithm. While this property was implicitly used in [20], it was actually never proven.

**Definition 5.2 (Non Interference with Unshared input [20]).** *Let $G$ be a gadget taking two types of inputs:*

1. *shared inputs $\mathcal{X}$, where all elements in $\mathcal{X}$ are $d$-tuples of elements in $\mathcal{R}_q$*
2. *unshared input $\mathcal{Y}$, where all elements in $\mathcal{Y}$ are tuples (not of fixed size) of elements in $\mathcal{R}_q$*

*A gadget* $\mathbf{G}$ *with shared and unshared inputs is said* $(d-1)$*-non-interfering with unshared inputs (written* $(d-1)$-NIU *for short) iff any set of probes* $\bar{\mathscr{I}}$ *such that* $|\bar{\mathscr{I}}| \leq d-1$ *can be perfectly simulated (See Definition 2.2) by a simulator* $(\mathsf{SimIn}, \mathsf{SimOut})$ *such that* $\mathsf{SimIn}(\mathbf{G}, \bar{\mathscr{I}})$ *outputs a set* $\bar{\mathcal{X}}' \bigcup \bar{\mathcal{U}}$ *of at most* $d-1$ *shares of each shared input* $(\bar{\mathcal{X}}')$ *and each unshared input* $(\bar{\mathcal{U}})$*.*

**Definition 5.3 (Strong Non Interference with Unshared input [20]).** *A gadget is said* $(d-1)$*-strongly-non-interfering with unshared inputs(written* $(d-1)$-sNIU *for short) iff any set* $\bar{\mathscr{I}}$ *of at most* $d-1 = d_{\mathrm{int}} + d_{\mathrm{out}}$ *probes where* $d_{\mathrm{int}}$ *are made on internal data and* $d_{\mathrm{out}}$ *are made on the outputs can be simulated as in Definition 5.2 with* $d_{\mathrm{int}}$ *instead of* $d-1$*.*

Since unshared inputs only differ from shared inputs by semantics (the distinction comes mostly from the fact that they do not represent a secret being used by the algorithm but internal randomnesses), one can note that if we ignore this distinction, the definitions of NIU and NI are identical. The interesting property of NIU comes from the fact that first transforming the relevant gadgets (namely AddRepNoise ) to include the randomness as unshared inputs allows NIU to prove a meaningful statement on the joint distribution of the probes and the output. A key property we use to prove EUF-CMA in the probing model.

As argued earlier once the randomness is moved to inputs the definition of NIU becomes identical to the one of NI with the difference that inputs are separated in two sets by whether they are shared or unshared. Since this difference is purely syntactical the composition lemma of NI naturally extends to NIU.

**Lemma 5.1 (Composability of NIU and sNIU gadgets).** *A well-formed algorithm is* NIU *if all of its gadgets are* NIU *or* sNIU *and each sharing and each unshared variable is used at most once as input of a non-*sNIU *gadget. Moreover, a well-formed algorithm is* sNIU *if it is* NIU *and its output sharings are issued from an* sNIU *gadget.*

We now give a core lemma to use NIU. In essence the following lemma states that by passing the relevant randomnesses of a program to inputs, proving NIU becomes sufficient to prove that probes can be simulated even in the presence of outputs.

**Lemma 5.2.** *Let* $P$ *be an algorithm with shared inputs* $\mathcal{X}$ *and unshared inputs* $\mathcal{U}$*. If* $P$ *is* $(d-1)$-NIU*, and the public output of* $P$ *is a deterministic function of* $(\mathcal{X}, \mathcal{U})$*. Then for any input* $\mathcal{X}$ *and any probes* $\bar{\mathscr{I}}$ *(with* $|\bar{\mathscr{I}}| \leq d-1$*), the distribution of* $(out_{\mathrm{unmasked}}, \mathsf{SimOut}(P, (\mathcal{X}', \mathcal{U}')))$ *and* $(out_{\mathrm{unmasked}}, \mathscr{L})$ *over the randomness* $\mathcal{U}$ *and the random coins of* $P$ *and* $\mathsf{SimOut}$ *are identical, where* $(out_{\mathrm{masked}}, out_{\mathrm{unmasked}}, \mathscr{L}) \leftarrow \mathsf{ExecObs}(P, \bar{\mathscr{I}}, \mathcal{X})$ *and* $(\bar{\mathcal{X}}', \bar{\mathcal{U}}') \leftarrow \mathsf{SimIn}(P, \bar{\mathscr{I}})$*.*

*Proof.* We will fix the input $\mathcal{X}$ and not $\mathcal{D}$ the distribution from which $\mathcal{U}$ is sampled. $\mathscr{L}$ and $out_{\mathrm{unmasked}}$ are random variables over the choice of $\mathcal{U}$ and the random coins of $P$ which we will note $rc_P$, and $\mathsf{SimOut}(P,(\mathcal{X}',\mathcal{U}')))$ is a random variable over the choice of $\mathcal{U}$ and the random coins of SimOut which we will note $rc_S$ (SimOut only uses the randomness in $\mathcal{U}' \subset \mathcal{U}$ but we can consider it as a variable of $\mathcal{U}$ since $\mathcal{U}'$ is a marginal of $\mathcal{U}$). First we observe that since the definition of NI and NIU are identical if we simply consider the extra randomness as another input we have that the marginal distributions of $\mathscr{L}$ and $\mathsf{SimOut}(P,(\mathcal{X}',\mathcal{U}'))$ are identical, i.e. for any possible leakage $\Lambda$ we have:

$$\Pr_{\mathcal{U}\leftarrow\mathcal{D},rc_P\leftarrow\{0,1\}^*}[\mathscr{L}(\mathcal{X},\mathcal{U},rc_P)=\Lambda] = \Pr_{\mathcal{U}\leftarrow\mathcal{D},rc_P\leftarrow\{0,1\}^*}[\mathsf{SimOut}(\mathcal{X},\mathcal{U},rc_S)=\Lambda]$$

Since the algorithm $P$ is deterministic when given $(\mathcal{X},\mathcal{U})$, we have that for any possible leakage value $\Lambda$ and output value $\theta$:

$$\begin{aligned}
&\Pr_{\mathcal{U}\leftarrow\mathcal{D},rc_P\leftarrow\{0,1\}^*}[\mathscr{L}(\mathcal{X},\mathcal{U},rc_P)=\Lambda, out_{\text{unmasked}}(\mathcal{X},\mathcal{U})=\theta]\\
&\quad=\sum_{\mathcal{U}\text{ s.t } out_{\text{unmasked}}(\mathcal{X},\mathcal{U})=\theta}\ \Pr_{rc_P\leftarrow\{0,1\}^*}[\mathscr{L}(\mathcal{X},\mathcal{U},rc_P)=\Lambda]\\
&\quad=\sum_{\mathcal{U}\text{ s.t } out_{\text{unmasked}}(\mathcal{X},\mathcal{U})=\theta}\ \Pr_{rc_S\leftarrow\{0,1\}^*}[\mathsf{SimOut}(\mathcal{X},\mathcal{U},rc_S)=\Lambda]\\
&\quad=\Pr_{\mathcal{U}\leftarrow\mathcal{D},rc_S\leftarrow\{0,1\}^*}[\mathsf{SimOut}(\mathcal{X},\mathcal{U},rc_S)=\Lambda, out_{\text{unmasked}}(\mathcal{X},\mathcal{U})=\theta]
\end{aligned}$$

which is the desired result. □

## 6 NIU Property of Raccoon's KeyGen and Sign

Before establishing EUF-CMA security of Raccoon in the probing model, we prove that the KeyGen and Sign algorithms are NIU. Looking ahead, this allows a reduction to simulate the probes $\mathscr{L}_{\mathsf{KeyGen}}$ and $\mathscr{L}^{(i)}_{\mathsf{Sign}}$ in the EUF-CMA security game in the probing model (cf. Fig. 3).

### 6.1 Existing Security Properties

Thanks to the composability of the sNI/NIU models, we can focus on the smaller gadgets comprising the KeyGen and Sign algorithms. Table 2 summarizes the security properties of the gadgets used in Raccoon, where we can rely on prior works to establish the security of every gadget, except for AddRepNoise . We refer to the cited papers for more information about the proofs.

### 6.2 Security Property of the AddRepNoise Gadget

Let us start with an intuition on the role of the Refresh operations in AddRepNoise. When considering unmasked coefficients, AddRepNoise is functionally equivalent to performing $a \leftarrow a+\mathrm{SU}(u,T)$ for each coefficient $a$, for $T = d\cdot\mathsf{rep}$. The internal use of Refresh operations does not affect this behavior but is meant to offer some resilience to probing adversaries.

**Table 2.** Security properties of the known and new gadgets. No security property is necessary for the other unmasked operations (ExpandA, ChalHash, ChalPoly, CheckBounds, Computing the hint $\mathbf{h}$).

| Name | Property | Proof reference |
|---|---|---|
| $\times\mathbf{A}$ and Line 13 of Algorithm 2 | NI | $\mathbb{Z}_q$−linear |
| Refresh (Algorithm 7) | sNI | [7,24,33] |
| ZeroEncoding (Algorithm 8) | sNI | [33] |
| Decode (Algorithm 6) | NI | [5, Alg. 16] |
| AddRepNoise (Algorithm 5) | sNIU | Proved in Section 6.2, Section 6.1 |

Without Refresh, a viable strategy would be to probe individual shares of $[\![a]\!]$ at the start and at the end of AddRepNoise , allowing to learn the sum $b$ of $\mathsf{rep} \cdot (d-1)/2$ small uniform errors. The conditional distribution of the additive noise (conditioned on the $d-1$ probed values) is now $b + \mathrm{SU}(u, T - (d-1) \cdot \mathsf{rep}/2)$. With Refresh, this strategy is not possible anymore but a probing adversary can still probe individual errors, which in the end gives out no more than the sum $b$ of $d-1$ small uniform errors. The conditional distribution of the additive noise (conditioned on the $d-1$ probed values) is now $b + \mathrm{SU}(u, T - (d-1))$, where the adversary learns $b$ but knows nothing about the realization of $\mathrm{SU}(u, T - (d-1))$.

While AddRepNoise performs operations share by share, the underlying distributions are not uniform. The addition of short noise values are added biases the *a posteriori* distribution of the final noise. Hence, one cannot prove that this gadget is probing secure. We resolve this issue by moving the short noise values as random coin inputs of the algorithm, introducing $\mathsf{AddRepNoise}_{\mathsf{ER}}$ in Algorithm 9, an instance of AddRepNoise with explicit randomness (ER) for the small uniforms. Note that the complete set of small uniforms is considered as a single unshared input. We can now formally show in Lemma 6.1 that AddRepNoiseER is sNIU. A similar result was proven in [20] but our proof strategy is different and perhaps a bit more formal. Later, these inputs will be handled in the general composition proof.

**Lemma 6.1.** *The* $\mathsf{AddRepNoise}_{\mathsf{ER}}$ *gadget is (d-1)-*sNIU*.*

*Proof.* We represent AddRepNoiseER as a sequential succession of MiniAddRepNoise and Refresh as presented in Fig. 5. To prove the sNIU property, we exhibit the randomness $\rho_{i,i_{\mathsf{rep}},j}$ in the input. Let us remark that the randomness involved in Refresh (and thus in ZeroEncoding) are not explicited as the algorithm is already proved sNI. Hence, $\mathsf{AddRepNoise}_{\mathsf{ER}}$ is partially derandomized. Our proof proceeds in two steps; we first study the MiniAddRepNoise sub-gadget, then $\mathsf{AddRepNoise}_{\mathsf{ER}}$.

*Step 1:* MiniAddRepNoise. We first show that any probe inside MiniAddRepNoise can be perfectly simulated (see Definition 2.2) with $\rho_{i,i_{\mathsf{rep}},j}$ and the input $\mathbf{v}_j$, where $(i, i_{\mathsf{rep}}, j)$ corresponds to the targeted loop. Indeed, let $p$ be a probe inside MiniAddRepNoise. The description of this probe necessarily includes $(i, i_{\mathsf{rep}}, j)$ to specify the involved loop. The intermediate value targeted by $p$ can be

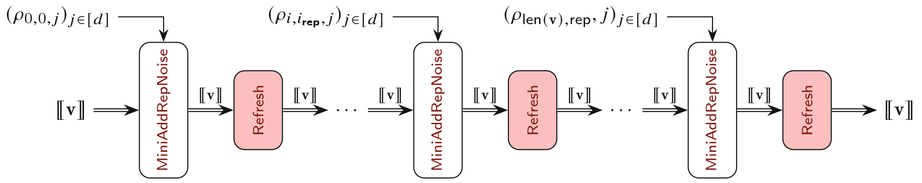
$(\rho_{0,0,j})_{j\in[d]}$
$(\rho_{i,i_{\mathsf{rep}},j})_{j\in[d]}$
$(\rho_{\mathsf{len(v)},\mathsf{rep},j})_{j\in[d]}$
$[\![\mathbf{v}]\!]$
MiniAddRepNoise
Refresh
$\cdots$

**Fig. 5.** Structure of AddRepNoise$_{\mathsf{ER}}$ (using Algorithm 10). A gadget proven sNI is noted gadget . The gadgets with no proven property are noted gadget . Single arrows ( $\longrightarrow$ ) and double arrows ( $\Longrightarrow$ ) represent plain and masked values, respectively.

**Algorithm 9.** AddRepNoise$_{\mathsf{ER}}$($[\![\mathbf{v}]\!], (\rho_{i,i_{\mathsf{rep}},j})$) $\to [\![\mathbf{v}']\!]$, w/ partial explicit randomness

**Input:** Masked vector $[\![\mathbf{v}]\!] = (\mathbf{v}_j)_{j\in[d]} = (v_{i,j})_{i\in[\mathsf{len}(\mathbf{v})],j\in[d]}$.
**Input:** Randomness $(\rho_{i,i_{\mathsf{rep}},j})_{i\in[\mathsf{len}(\mathbf{v})],i_{\mathsf{rep}}\in[\mathsf{rep}],j\in[d]}$
**Output:** Updated $[\![\mathbf{v}]\!]$ with $\mathrm{SU}(u, d\cdot\mathsf{rep})$ distribution added to each coefficient of $\mathbf{v}$.

```
1: for $(i, i_{\mathsf{rep}}) \in [\mathsf{len}(\mathbf{v})] \times [\mathsf{rep}]$ do    ▷ Vector index.
2:     for $i_{\mathsf{rep}} \in [\mathsf{rep}]$ do
3:         $[\![\mathbf{v}_i]\!] \leftarrow$ MiniAddRepNoise($[\![\mathbf{v}_i]\!], (\rho_{i,i_{\mathsf{rep}},j})_{i\in[\mathsf{len}(\mathbf{v})],j\in[d]}$)
4:         $[\![\mathbf{v}_i]\!] \leftarrow$ Refresh($[\![\mathbf{v}_i]\!]$)    ▷ Refresh polynomial on each repeat.
5: return $[\![\mathbf{v}]\!]$
```

**Algorithm 10.** MiniAddRepNoise($[\![\mathbf{v}]\!], i_{\mathsf{rep}}, (\rho_{i,i_{\mathsf{rep}},j})$) $\to [\![\mathbf{v}']\!]$

**Input:** Masked vector $[\![\mathbf{v}']\!]$, index $i_{\mathsf{rep}} \in [\mathsf{rep}]$, randomness $(\rho_{i,i_{\mathsf{rep}},j})_{i\in[\mathsf{len}(\mathbf{v})],j\in[d]}$
**Output:** Updated $[\![\mathbf{v}]\!]$.

```
1: for $j \in [d]$ do
2:     $v'_j \leftarrow v_j + \rho_{i,i_{\mathsf{rep}},j}$
3: return $[\![\mathbf{v}']\!]$
```

1. the randomness $\rho_{i,i_{\mathsf{rep}},j}$,
2. the value $v_j$ or $v'_j$.

It is easy to conclude that any of these values can be perfectly simulated from $\rho_{i,i_{\mathsf{rep}},j}$ and the input $\mathbf{v}_j$. The only intermediate value that needs both is $v'_j$ as it needs $\rho_{i,i_{\mathsf{rep}},j}$.

*Step 2:* AddRepNoise$_{\mathsf{ER}}$. Let us now look at the bigger picture. In this proof, we will perform a composition proof by propagating the dependency of the intermediate variables to shares of $\rho_{i,i_{\mathsf{rep}},j}$ and $\mathbf{v}_j$. Let $\bar{\mathscr{I}}$ be the given set of at most $d-1$ probes in AddRepNoise . We decompose $\bar{\mathscr{I}}$ as follows.

– Let $\delta^{i,i_{\mathsf{rep}}}_{\mathsf{MiniAddRepNoise}}$ be the number intermediate variables that are probed inside the MiniAddRepNoise gadget of the loop with indexes $i, i_{\mathsf{rep}}$.
– Let $\delta^{i,i_{\mathsf{rep}}}_{\mathsf{Refresh}}$ be the number intermediate variables that are probed inside the Refresh gadget of the loop with indexes $i, i_{\mathsf{rep}}$.

By definition,

$$\sum_{i=0}^{\mathsf{len}(\mathbf{v})} \sum_{i_{\mathsf{rep}}=0}^{\mathsf{rep}} \left( \delta_{\mathsf{MiniAddRepNoise}}^{i,i_{\mathsf{rep}}} + \delta_{\mathsf{Refresh}}^{i,i_{\mathsf{rep}}} \right) \leq d-1. \tag{9}$$

Going from right to left in Fig. 5, we first consider the last Refresh of the last loop (where $i = \mathsf{len}(\mathbf{v})$ and $i_{\mathsf{rep}} = \mathsf{rep}$). Thanks to the sNI property of the last Refresh algorithm, all the $\delta_{\mathsf{Refresh}}^{\mathsf{len}(\mathbf{v}),\mathsf{rep}}$ probes can be perfectly simulated from $\delta_{\mathsf{Refresh}}^{\mathsf{len}(\mathbf{v}),\mathsf{rep}}$ shares of $\mathbf{v}'$, which is also the output of the last MiniAddRepNoise. So, thanks to the above paragraph about MiniAddRepNoise, all the probes from the last MiniAddRepNoise, can be perfectly simulated from two sets of probes:

- $\bar{\mathscr{I}}_{\mathsf{len}(\mathbf{v}),\mathsf{rep}}$ defined as the description of at most $\delta_{\mathsf{MiniAddRepNoise}}^{\mathsf{len}(\mathbf{v}),\mathsf{rep}} + \delta_{\mathsf{Refresh}}^{\mathsf{len}(\mathbf{v}),\mathsf{rep}}$ values of $\rho_{\mathsf{len}(\mathbf{v}),\mathsf{rep},j}$ (with several different $j$'s),
- $\bar{\mathscr{I}}'_{\mathsf{len}(\mathbf{v}),\mathsf{rep}}$ defined as the set of to at most $\delta_{\mathsf{MiniAddRepNoise}}^{\mathsf{len}(\mathbf{v}),\mathsf{rep}} + \delta_{\mathsf{Refresh}}^{\mathsf{len}(\mathbf{v}),\mathsf{rep}}$ shares of $\mathbf{v}$, the input of the last MiniAddRepNoise.

The set of $\bar{\mathscr{I}}'_{\mathsf{len}(\mathbf{v}),\mathsf{rep}}$ can also be seen as probes of the output of the penultimate Refresh. But, thanks to the sNI property of the penultimate Refresh algorithm, they can be simulated independently from the $\delta_{\mathsf{Refresh}}^{\mathsf{len}(\mathbf{v})-1,\mathsf{rep}-1}$ intermediate variables probed inside the penultimate Refresh algorithm. In conclusion, the $\bar{\mathscr{I}}'_{\mathsf{len}(\mathbf{v}),\mathsf{rep}}$ probes can be simulated from uniform random.

Applying the same reasoning for all the subsequent loops, the set of $\bar{\mathscr{I}}$ probes can be perfectly simulated from

- $\bar{\mathscr{I}}_{i,i_{\mathsf{rep}}}$ defined as the description of at most $\delta_{\mathsf{MiniAddRepNoise}}^{i,i_{\mathsf{rep}}} + \delta_{\mathsf{Refresh}}^{i,i_{\mathsf{rep}}}$ values of $\rho_{i,i_{\mathsf{rep}},j}$ (with several different $j$'s),
- $\bar{\mathscr{I}}'_{0,0}$ defined as the set of to at most $\delta_{\mathsf{MiniAddRepNoise}}^{0,0} + \delta_{\mathsf{Refresh}}^{0,0}$ shares of $\mathbf{v}$, the input of the $\mathsf{AddRepNoise}_{\mathsf{ER}}$.

We define $\bar{\mathcal{U}} = \bar{\mathscr{I}}_{0,0} \cup \cdots \cup \bar{\mathscr{I}}_{\mathsf{len}(\mathbf{v}),\mathsf{rep}}$ and $\bar{\mathcal{X}}' = \bar{\mathscr{I}}'_{0,0}$. Thanks to Eq. (9) and Lemma 2.1, we have shown that $\mathsf{AddRepNoise}_{\mathsf{ER}}$ is (d-1)-sNIU. □

## 6.3 Security Property of KeyGen and Sign

Now that $\mathsf{AddRepNoise}_{\mathsf{ER}}$ is proved, one needs to derive the security of the key generation and signature algorithms with a composition proof. Let us first introduce $\mathsf{KeyGen}_{\mathsf{ER}}$ and $\mathsf{Sign}_{\mathsf{ER}}$, simple modifications of KeyGen and Sign algorithms where the small uniform randomness is provided as input. $\mathsf{KeyGen}_{\mathsf{ER}}$ is formally described in Algorithm 11. Due to space constraints, the formal description of $\mathsf{Sign}_{\mathsf{ER}}$ is deferred to the full version. We provide a proof of Lemma 6.2 for $\mathsf{KeyGen}_{\mathsf{ER}}$. The proof for $\mathsf{Sign}_{\mathsf{ER}}$ proceeds in a similar fashion and is included in the full version of this paper.

**Lemma 6.2.** *The algorithms* $\mathsf{KeyGen}_{\mathsf{ER}}$ *and* $\mathsf{Sign}_{\mathsf{ER}}$ *are* $(d-1)$-NIU.

**Algorithm 11.** $\mathsf{KeyGen}_{\mathsf{ER}}((\rho^{(0)}_{i,i_{\mathsf{rep}},j}),(\rho^{(1)}_{i,i_{\mathsf{rep}},j})) \to (\mathsf{vk},\mathsf{sk})$
▷ KeyGen with explicit randomness for AddRepNoise

```
Input: Randomness $(\rho^{(0)}_{i,i_{\mathsf{rep}},j})_{i\in[\mathsf{len}(\mathbf{v})],i_{\mathsf{rep}}\in[\mathsf{rep}],j\in[d]}$, $(\rho^{(1)}_{i,i_{\mathsf{rep}},j})_{i\in[\mathsf{len}(\mathbf{v})],i_{\mathsf{rep}}\in[\mathsf{rep}],j\in[d]}$
Output: Keypair vk, sk
1: $\mathsf{seed} \leftarrow \{0,1\}^\kappa$; $\mathbf{A} := \mathsf{ExpandA}(\mathsf{seed})$
2: $[\![\mathbf{s}]\!] \leftarrow \ell \times \mathsf{ZeroEncoding}(d)$
3: $[\![\mathbf{s}]\!] \leftarrow \mathsf{AddRepNoise}_{\mathsf{ER}}([\![\mathbf{s}]\!], u_{\mathsf{t}}, \mathsf{rep}, (\rho^{(0)}_{i,i_{\mathsf{rep}},j}))$    ▷ Partially derandomized AddRepNoise.
4: $[\![\mathbf{t}]\!] := \mathbf{A}\cdot[\![\mathbf{s}]\!]$
5: $[\![\mathbf{t}]\!] \leftarrow \mathsf{AddRepNoise}_{\mathsf{ER}}([\![\mathbf{t}]\!], u_{\mathsf{t}}, \mathsf{rep}, (\rho^{(1)}_{i,i_{\mathsf{rep}},j}))$    ▷ Partially derandomized AddRepNoise.
6: $\mathbf{t} := \mathsf{Decode}([\![\mathbf{t}]\!])$
7: $\mathbf{t} := \lfloor \mathbf{t} \rceil_{\nu_{\mathsf{t}}}$
8: return $(\mathsf{vk} := (\mathsf{seed},\mathbf{t}), \mathsf{sk} := (\mathsf{vk},[\![\mathbf{s}]\!]))$
```

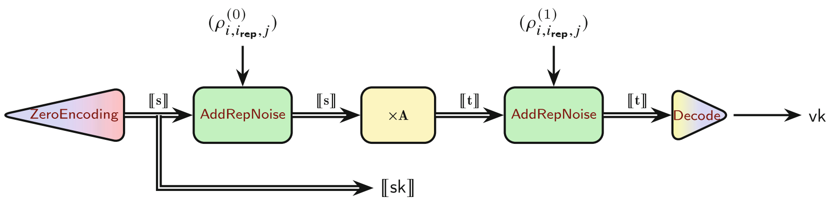

**Fig. 6.** Structure of KeyGen (Algorithm 11). Gadgets proven NI (resp. sNIU) is noted gadget (resp. gadget). Triangular gadgets either start from a masked input and output an unmasked value, or the other way around.

*Proof. (Lemma 6.2).* Let us decompose the key generation as a succession of gadgets. The gadgets may be represented as in Fig. 6. We assume the respective NI/sNI/sNIU properties of each gadget as presented in Table 2.

Recall that given a set $\bar{\mathscr{I}}$ of at most $d-1$ probes inside $\mathsf{KeyGen}_{\mathsf{ER}}$, we aim at proving that they can be perfectly simulated with at most $d-1$ shares of $(\rho^{(0)}_{i,i_{\mathsf{rep}},j})$ and $d-1$ shares of $(\rho^{(1)}_{i,i_{\mathsf{rep}},j})$. In other words we will exhibit two sets $\bar{\mathscr{I}}_0$ of at most $d-1$ values of $(\rho^{(0)}_{i,i_{\mathsf{rep}},j})$, and $\bar{\mathscr{I}}_1$ of at most $d-1$ values of $(\rho^{(1)}_{i,i_{\mathsf{rep}},j})$ which will be enough to perfectly simulate $\bar{\mathscr{I}}$.

Let us decompose the set $\bar{\mathscr{I}}$ of at most $d-1$ probes in $\mathsf{KeyGen}_{\mathsf{ER}}$ among the different gadgets. By convention, to avoid counting certain probes twice (once as output of a gadget and once as input of the subsequent gadget), we do not count the probes on the outputs. For example, if a probe is made on the output of a gadget $\mathbf{G}$, we will consider that it is actually made on the input of the subsequent gadget. We note:

– $\delta_0$ the number of intermediate variables probed in Line 6 (final Decode gadget);

- $\delta_1$ the number of intermediate variables probed in Line 5 (second $\mathsf{AddRepNoise}_{\mathsf{ER}}$);
- $\delta_2$ the number of intermediate variables probed in Line 4 (multiplication with $\mathbf{A}$);
- $\delta_3$ the number of intermediate variables probed in Line 3 (first $\mathsf{AddRepNoise}_{\mathsf{ER}}$);
- $\delta_4$ the number of intermediate variables probed in Line 2 (ZeroEncoding);

We recall that by definition of $\bar{\mathscr{I}}$, $\sum_{i=0}^{4} \delta_i \leq d-1$.

The proof is similar to a standard composition proof. Thanks to the NI property of the Decode gadget, all the $\delta_0$ intermediate variables can be perfectly simulated (see Definition 2.2) with at most $\delta_0$ shares of $[\![\mathbf{t}]\!]$. Since the second $\mathsf{AddRepNoise}_{\mathsf{ER}}$ is $d-1$-sNIU, the $\delta_1+\delta_0$ intermediate variables observed during Decode and the last $\mathsf{AddRepNoise}_{\mathsf{ER}}$ may be perfectly simulated with $\delta_1$ shares of $[\![t]\!]$ (the output of the $\times\mathbf{A}$ operation) and $\delta_1$ shares of $(\rho^{(1)}_{i,i_{\mathsf{rep}}})$. We note $\bar{\mathscr{I}}_1$ this set. Note that $\delta_0$ is discarded as it concerns the output of a sNIU gadget.

With the same reasoning, all the $\delta_0+\delta_1+\delta_2+\delta_3$ intermediate variables observed after the first $\mathsf{AddRepNoise}_{\mathsf{ER}}$ can be perfectly simulated with at most $\delta_3$ shares of $[\![s]\!]$ (which are also the output of ZeroEncoding) and at most $\delta_3$ shares of $(\rho^{(0)}_{i,i_{\mathsf{rep}}})$. We note $\bar{\mathscr{I}}_0$ this sets. In addition, the $\delta_4$ intermediate variables in the ZeroEncoding gadget may be perfectly simulated from the public parameters as ZeroEncoding is NI and does not take any input.

Putting everything together, we have proved that the distribution of the intermediate variables in $\bar{\mathscr{I}}$ may be perfectly simulated from:

- the set $\bar{\mathscr{I}}_0$ containing at most $\delta_3$ shares of $(\rho^{(0)}_{i,i_{\mathsf{rep}}})$
- the sets $\bar{\mathscr{I}}_1$ containing at most $\delta_1$ shares of $(\rho^{(1)}_{i,i_{\mathsf{rep}}})$

Since $\delta_3+\delta_1 \leq \sum_{i=0}^{4} \delta_i \leq d-1$, we have exhibited a ses $\bar{\mathcal{U}}$ of at most $d-1$ of the unshared input which concludes the proof. □

## 7 EUF-CMA Security of Raccoon in the Probing Model

We are finally ready to prove EUF-CMA security of Raccoon in the probing model. This is done in two steps. We first reduce EUF-CMA security of Raccoon in the probing model to the *standard* EUF-CMA security of *small* Raccoon, formally defined in Fig. 7. We then establish that this small Raccoon is EUF-CMA secure. Technically, the first part relies on the NIU property of KeyGen and Sign (cf. Sect. 6), a purely statistical step claiming that given a small Raccoon key and signature, we can simulate the leakage of Raccoon. The second part relies on the smooth Rényi divergence for the sum of uniform distributions (cf. Sect. 4), and reduces to computational problems.

## 7.1 Description of a Non-masked Small Raccoon

We first formally define a *non-masked* and simplified variant of Raccoon, called *small* Raccoon, depicted in Fig. 7. Notice that there are no more masking or bit-droppings applied. More importantly, it is "small" since the sum of uniform distribution is smaller. We effectively modify the bounds on the signature size to be smaller, using $\bar{B}_\infty$ and $\bar{B}_2$, whose formal definition appears in Theorem 7.1.

```
Algorithm 12 KeyGen_SMALL(∅) → (vk, sk)
Output: Keypair vk, sk
 1: $\mathsf{seed} \leftarrow \{0,1\}^\kappa$
 2: $\mathbf{A} := \mathsf{ExpandA}(\mathsf{seed})$
 3: $(\mathbf{s}, \mathbf{e}) \leftarrow \mathrm{RSU}(u_\mathrm{t}, d(\mathsf{rep}-1)+1)^\ell \times \mathrm{RSU}(u_\mathrm{t}, d(\mathsf{rep}-1)+1)^k$
 4: $\mathbf{t} := \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$          ▷ No rounding of $\mathbf{t} \in \mathcal{R}_q^k$
 5: return $(\mathsf{vk} := (\mathsf{seed}, \mathbf{t}), \mathsf{sk} = (\mathsf{vk}, \mathbf{s}))$

Algorithm 13 Sign_SMALL(sk, msg) → sig
Input: Secret signing key $\mathsf{sk} = (\mathsf{vk}, \mathbf{s})$, message $\mathsf{msg} \in \{0,1\}^*$.
Output: Signature $\mathsf{sig} = (c_\mathsf{hash}, \mathbf{h}, \mathbf{z})$ of msg under sk.
 1: $\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk}) \| \mathsf{msg})$
 2: $(\mathbf{r}, \mathbf{e}') \leftarrow \mathrm{RSU}(u_\mathrm{w}, d(\mathsf{rep}-1)+1)^\ell \times \mathrm{RSU}(u_\mathrm{w}, d(\mathsf{rep}-1)+1)^k$
 3: $\mathbf{w} = \mathbf{A} \cdot \mathbf{r} + \mathbf{e}'$          ▷ No rounding of $\mathbf{w} \in \mathcal{R}_q^k$
 4: $c_\mathsf{hash} := \mathsf{Chal\bar{H}ash}(\mathbf{w}, \mu)$          ▷ $\mathsf{Chal\bar{H}ash}$ redefined to take $\mathbf{w} \in \mathcal{R}_q^k$
 5: $c_\mathsf{poly} := \mathsf{ChalPoly}(c_\mathsf{hash})$
 6: $\mathbf{z} := c_\mathsf{poly} \cdot \mathbf{s} + \mathbf{r}$
 7: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - c_\mathsf{poly} \cdot \mathbf{t}$          ▷ No need to lift $\mathbf{t}$ anymore
 8: $\mathbf{h} := \mathbf{w} - \mathbf{y}$          ▷ Hint $\mathbf{h}$ now defined over $\mathcal{R}_q^k$
 9: $\mathsf{sig} := (c_\mathsf{hash}, \mathbf{z}, \mathbf{h})$
10: if $(\|(\mathbf{z}, \mathbf{h})\|_\infty > \bar{B}_\infty)$ or $(\|(\mathbf{z}, \mathbf{h})\|_2 > \bar{B}_2)$ goto Line 2          ▷ Check smaller bound
11: return sig

Algorithm 14 Verify_SMALL(sig, msg, vk) → {OK or FAIL}
Input: Signature $\mathsf{sig} = (c_\mathsf{hash}, \mathbf{h}, \mathbf{z}) := \mathsf{sig}$.
Output: Signature validity: OK (accept) or FAIL (reject).
 1: if $(\|(\mathbf{z}, \mathbf{h})\|_\infty > \bar{B}_\infty)$ or $(\|(\mathbf{z}, \mathbf{h})\|_2 > \bar{B}_2)$ return FAIL else return OK
 2: $\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk}) \| \mathsf{msg})$; $\mathbf{A} := \mathsf{ExpandA}(\mathsf{seed})$
 3: $c_\mathsf{poly} := \mathsf{ChalPoly}(c_\mathsf{hash})$
 4: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - c_\mathsf{poly} \cdot \mathbf{t}$
 5: $\mathbf{w} := \mathbf{y} + \mathbf{h}$
 6: $c'_\mathsf{hash} := \mathsf{ChalHash}(\mathbf{w}', \mu)$
 7: if $c_\mathsf{hash} \neq c'_\mathsf{hash}$ return FAIL
 8: return OK
```

**Fig. 7.** A non-masked and simplified Raccoon, named *small* Raccoon. While we used the notation from the masked Raccoon for consistency, notice above that $\mathbf{h}$ simply becomes $c_\mathsf{poly} \cdot \mathbf{e} + \mathbf{e}'$ without rounding errors.

### 7.2 EUF-CMA Security of Small Raccoon ⇒ Probing EUF-CMA Security of Raccoon

This consists of the first step. Once the following theorem is established, we only need to prove standard EUF-CMA security of small Raccoon.

**Theorem 7.1.** *Let $B_\infty$ and $B_2$ satisfying:*

- $\bar{B_\infty} \geq B_\infty + \omega \cdot (d-1) \cdot \left(\frac{1}{2} + \frac{2^{3u_t}}{3}\right) \cdot (\kappa + \log(n(k+\ell)) + 2^{\nu_w} + \omega \cdot 2^{\nu_t}$
- $\bar{B}_2 \geq B_2 + \omega \cdot \sqrt{n(k+\ell)} \cdot (d-1) \cdot \left(\frac{1}{2} + \frac{2^{3u_t}}{3}\right) \cdot (\kappa + \log(n(k+\ell)) + 2^{\nu_w} \cdot \sqrt{nk} + \omega \cdot 2^{\nu_t} \cdot \sqrt{nk}$

*Let $Q_H$ and $Q_S$ denote the number of random oracle queries and signing queries performed by $\mathcal{A}$. For any PPT adversary $\mathcal{A}$ against the* EUF-CMA *security on Raccoon in the $(d-1)$-probing model with time $T$ and advantage $\varepsilon$, there exists a PTT adversary $\mathcal{B}$ against the* EUF-CMA *security on small Raccoon (cf. Fig. 7) with time $O(T)$ and advantage:*

$$\mathsf{Adv}_{\mathcal{B}} \geq \mathsf{Adv}_{\mathcal{A}} - 4Q_H Q_S \cdot 2^{-2\kappa} - 2^{-\kappa+1} - \frac{1}{|C|}.$$

We will use a series of hybrids defined below to prove the theorem.

$\mathsf{Hybrid}_0$**:** This hybrid corresponds to real the EUF-CMA security game in the $(d-1)$-probing model (cf. Fig. 3).

$\mathsf{Hybrid}_1$**:** In this hybrid we replace KeyGen with $\mathsf{KeyGen}_{\mathsf{ER}}$ and Sign with $\mathsf{Sign}_{\mathsf{ER}}$, in which all randomnesses are sampled prior to running the algorithm. Since the algorithms are functionally identical the advantage is unchanged.

$\mathsf{Hybrid}_2$**:** This hybrid corresponds to Fig. 8, in which all the probes queried by the adversary during either key generation or signature are mapped to probes that target only the randomness used in the AddRepNoise gadgets. We prove that the values output by these probes can be used to perfectly simulate the output queried by the adversary in Lemma 6.2.

More precisely there is a first PPT simulator $(\mathsf{SimIn}_{\mathsf{KeyGen}}, \mathsf{SimOut}_{\mathsf{KeyGen}})$ such that for any probe set $|\bar{\mathscr{I}}_{\mathsf{KeyGen}}| \leq t$ in $\mathsf{KeyGen}(1^\kappa)$, all probes in $\bar{\mathscr{I}}' := (\bar{\mathscr{I}}'_{\mathrm{s}}, \bar{\mathscr{I}}'_{\mathrm{e}}) := \mathsf{SimIn}_{\mathsf{KeyGen}}(\bar{\mathscr{I}}_{\mathsf{KeyGen}})$ are of the form $\bar{\rho}_{\mathrm{s},i,i_{\mathsf{rep}},j} \in \bar{\mathscr{I}}'_{\mathrm{s}}$ for some $(i, i_{\mathsf{rep}}, j) \in [\ell, \mathsf{rep}, d]$, and $\bar{\rho}_{\mathrm{e},i,i_{\mathsf{rep}},j} \in \bar{\mathscr{I}}'_{\mathrm{e}}$ for some $(i, i_{\mathsf{rep}}, j) \in [k, \mathsf{rep}, d]$ (note that the variable names $\bar{\rho}$ are also indexed by the AddRepNoise gadget to which they belong to ensure unique namings), and $\max(|\bar{\mathscr{I}}'_s|, |\bar{\mathscr{I}}'_e|) \leq d-1$. Using Lemma 5.2 we have that $(\mathsf{vk}, \mathsf{SimOut}(\mathsf{KeyGen}_{\mathsf{ER}}, \bar{\mathscr{I}}'))$ follows the same distribution as $(\mathsf{vk}, \mathscr{L})$, where $(\mathsf{sk}, \mathsf{vk}, \mathscr{L}) \leftarrow \mathsf{ExecObs}(\bar{\mathscr{I}}_{\mathsf{KeyGen}}, \mathsf{KeyGen}_{\mathsf{ER}}, 1^\lambda)$.

Similarly there is a second PPT simulator $(\mathsf{SimIn}_{\mathsf{Sign}}, \mathsf{SimOut}_{\mathsf{Sign}})$ such that for any message $\mathsf{msg}$, masked secret key $[\![\mathsf{sk}]\!]$, and probe set $|\bar{\mathscr{I}}_{\mathsf{Sign}}| \leq t$ in $\mathsf{Sign}([\![\mathsf{sk}]\!], \mathsf{msg})$, all probes in $\bar{\mathscr{I}}' := (\bar{\mathscr{I}}'_{\mathrm{r}}, \bar{\mathscr{I}}'_{\mathrm{e'}}, \bar{\mathscr{I}}'_{\mathsf{sk}}) := \mathsf{SimIn}_{\mathsf{Sign}}(\bar{\mathscr{I}}_{\mathsf{Sign}})$ are of the form $\bar{\rho}_{\mathrm{r},i,i_{\mathsf{rep}},j} \in \bar{\mathscr{I}}'_{\mathrm{r}}$ for some $(i, i_{\mathsf{rep}}, j) \in [\ell, \mathsf{rep}, d]$, $\bar{\rho}_{\mathrm{e'},i,i_{\mathsf{rep}},j} \in$

| **Adversary** | | **Challenger** |
|---|---|---|
| | $\xleftarrow{(\mathsf{KeyGen},\mathsf{Sign},\mathsf{Verify},\mathsf{KeyUpdate})}$ | |
| | $\xrightarrow{\bar{\mathscr{I}}_{\mathsf{KeyGen}}}$ | |
| | | $\bar{\mathscr{I}}'_{\mathsf{KeyGen}} \leftarrow \mathsf{SimIn}_{\mathsf{KeyGen}}(\bar{\mathscr{I}}_{\mathsf{KeyGen}})$ |
| | | $(\mathsf{vk},\mathsf{sk},\mathscr{L}'_{\mathsf{KeyGen}}) \leftarrow \mathsf{ExecObs}(\bar{\mathscr{I}}'_{\mathsf{KeyGen}}, \mathsf{KeyGen}, 1^{\lambda})$ |
| | $\xleftarrow{\mathsf{vk}, \mathscr{L}_{\mathsf{KeyGen}}}$ | $\mathscr{L}_{\mathsf{KeyGen}} \leftarrow \mathsf{SimOut}_{\mathsf{KeyGen}}(\mathscr{L}'_{\mathsf{KeyGen}})$ |
| $Q_s$ queries $\{$ | $\xrightarrow{m^{(i)}, \bar{\mathscr{I}}^{(i)}_{\mathsf{Sign}}}$ | $\mathsf{sk} \leftarrow \mathsf{KeyUpdate}(\mathsf{sk})$ |
| | | $\bar{\mathscr{I}}^{(i)\,\prime}_{\mathsf{Sign}} \leftarrow \mathsf{SimIn}_{\mathsf{Sign}}(\bar{\mathscr{I}}^{(i)}_{\mathsf{Sign}})$ |
| | | $(\mathsf{sig}^{(i)}, \perp, \mathscr{L}^{(i)\,\prime}_{\mathsf{Sign}}) \leftarrow \mathsf{ExecObs}(\bar{\mathscr{I}}^{(i)\,\prime}_{\mathsf{Sign}}, \mathsf{Sign}, \mathsf{sk}, m^{(i)})$ |
| | $\xleftarrow{\mathsf{sig}^{(i)}, \mathscr{L}^{(i)}_{\mathsf{Sign}}}$ | $\mathscr{L}^{(i)}_{\mathsf{Sign}} \leftarrow \mathsf{SimOut}_{\mathsf{Sign}}(\mathscr{L}^{(i)\,\prime}_{\mathsf{Sign}})$ |
| forgery $\{$ | $\xrightarrow{m^{*}, \mathsf{sig}^{*}}$ | |
| | | $b := \mathsf{Verify}(\mathsf{vk}, m^*, \mathsf{sig}^*) \wedge (m^* \notin \{m^{(1)}, \ldots, m^{(Q_s)}\}) \wedge$ |
| | | $\wedge \lvert \bar{\mathscr{I}}_{\mathsf{KeyGen}} \rvert \leq d-1 \wedge \forall i \in \{1, \ldots, Q_s\},\ \lvert \bar{\mathscr{I}}^{(i)}_{\mathsf{Sign}} \rvert \leq d-1$ |

**Fig. 8.** $\mathsf{Hybrid}_2$: The NIU properties proven in Lemma 6.2 ensure the existence of two PPT simulators ($\mathsf{SimIn}_{\mathsf{KeyGen}}, \mathsf{SimOut}_{\mathsf{KeyGen}}$) and ($\mathsf{SimIn}_{\mathsf{Sign}}, \mathsf{SimOut}_{\mathsf{Sign}}$). This ensures all probes can be moved to the randomness in the AddRepNoise gadgets in KeyGen and Sign. Differences with the EUF-CMA security game in the $(d-1)$-probing model (Fig. 3) are highlighted .

**Algorithm 15.** $\mathsf{KeyGen}_{\mathscr{L}}(1^{\kappa}, \bar{\mathscr{I}}) \to (\mathsf{vk}, \mathsf{sk}, \mathscr{L})$

**Input:** Probe set $\mathscr{I} = (\mathscr{I}_{\mathrm{s}}, \mathscr{I}_{\mathrm{e}})$, $\bar{\mathscr{I}}_{\mathrm{s}} \subset \left\{\bar{\rho}_{\mathrm{s},i,i_{\mathsf{rep}},j}; (i, i_{\mathsf{rep}}, j) \in [\ell] \times [\mathsf{rep}] \times [d]\right\}$ ,

$\bar{\mathscr{I}}_{\mathrm{e}} \subset \left\{\bar{\rho}_{\mathrm{e},i,i_{\mathsf{rep}},j}; (i, i_{\mathsf{rep}}, j) \in [k] \times [\mathsf{rep}] \times [d]\right\}$

**Output:** Keypair vk, sk and Leakage $\mathscr{L}$

```
1: seed ← $\{0,1\}^{\kappa}$; $\mathbf{A} := \mathsf{ExpandA}(\mathsf{seed})$
2: $[\![\mathbf{s}]\!] = (\mathbf{s}_1, \ldots, \mathbf{s}_d) := (0, \ldots, 0) \in (\mathcal{R}_q^{\ell})^d$
3: for $(i, i_{\mathsf{rep}}, j) \in [\ell] \times [\mathsf{rep}] \times [d]$ do
4:     $\rho_{\mathrm{s},i,i_{\mathsf{rep}},j} \leftarrow \mathrm{RSU}(u, 1)$
5:     $\mathbf{s}_{j,i} \leftarrow \mathbf{s}_{j,i} + \rho_{\mathrm{s},i,i_{\mathsf{rep}},j}$
6: $[\![\mathbf{t}]\!] := \mathbf{A} \cdot [\![\mathsf{sk}]\!] \in (\mathcal{R}_q^{k})^d$
7: for $(i, i_{\mathsf{rep}}, j) \in [k] \times [\mathsf{rep}] \times [d]$ do
8:     $\rho_{\mathrm{e},i,i_{\mathsf{rep}},j} \leftarrow \mathrm{RSU}(u, 1)$
9:     $\mathbf{t}_{j,i} \leftarrow \mathbf{t}_{j,i} + \rho_{\mathrm{s},i,i_{\mathsf{rep}},j}$
10: $\mathbf{t} := \mathsf{Decode}([\![\mathbf{t}]\!])$
11: $\mathbf{t} := \lfloor \mathbf{t} \rceil_{\nu_{\mathbf{t}}}$
12: $\mathscr{L} := \left\{(\rho_{\mathrm{s},i,i_{\mathsf{rep}},j}, \rho_{\mathrm{e},i',i'_{\mathsf{rep}},j'})\right\}_{(\bar{\rho}_{\mathrm{s},i,i_{\mathsf{rep}},j}, \bar{\rho}_{\mathrm{e},i',i'_{\mathsf{rep}},j'}) \in \bar{\mathscr{I}}}$
13: return $(\mathsf{vk} := (\mathsf{seed}, \mathbf{t}), \mathsf{sk} := (\mathsf{vk}, [\![\mathbf{s}]\!]), \mathscr{L})$
```

| **Adversary** | | **Challenger** |
|---|---|---|
| | $\xleftarrow{(\mathsf{KeyGen},\mathsf{Sign},\mathsf{Verify},\mathsf{KeyUpdate})}$ | |
| | $\xrightarrow{\bar{\mathscr{I}}_{\mathsf{KeyGen}}}$ | |
| | | $\bar{\mathscr{I}}'_{\mathsf{KeyGen}} \leftarrow \mathsf{SimIn}_{\mathsf{KeyGen}}(\bar{\mathscr{I}}_{\mathsf{KeyGen}})$ |
| | | $((\mathsf{sk},\mathsf{vk}),\mathscr{L}'_{\mathsf{KeyGen}}) \leftarrow \mathsf{KeyGen}_{\mathscr{L}}(1^{\kappa},\bar{\mathscr{I}}'_{\mathsf{KeyGen}})$ |
| | $\xleftarrow{\mathsf{vk},\mathscr{L}_{\mathsf{KeyGen}}}$ | $\mathscr{L}_{\mathsf{KeyGen}} \leftarrow \mathsf{SimOut}_{\mathsf{KeyGen}}(\mathscr{L}'_{\mathsf{KeyGen}})$ |
| $Q_s$ queries $\{$ | $\xrightarrow{m^{(i)},\bar{\mathscr{I}}^{(i)}_{\mathsf{Sign}}}$ | $\mathsf{sk} \leftarrow \mathsf{KeyUpdate}(\mathsf{sk})$ |
| | | $\bar{\mathscr{I}}^{(i)\,\prime}_{\mathsf{Sign}} \leftarrow \mathsf{SimIn}_{\mathsf{Sign}}(\bar{\mathscr{I}}^{(i)}_{\mathsf{Sign}})$ |
| | | $(\mathsf{sig}^{(i)},\mathscr{L}^{(i)\,\prime}_{\mathsf{Sign}}) \leftarrow \mathsf{Sign}_{\mathscr{L}}(\bar{\mathscr{I}}^{(i)\,\prime}_{\mathsf{Sign}}, sk, m^{(i)})$ |
| | $\xleftarrow{\mathsf{sig}^{(i)},\mathscr{L}^{(i)}_{\mathsf{Sign}}}$ | $\mathscr{L}^{(i)}_{\mathsf{Sign}} \leftarrow \mathsf{SimOut}_{\mathsf{Sign}}(\mathscr{L}^{(i)\,\prime}_{\mathsf{Sign}})$ |
| forgery $\{$ | $\xrightarrow{m^{*},\mathsf{sig}^{*}}$ | |
| | | $b := \mathsf{Verify}(\mathsf{vk}, m^{*}, \mathsf{sig}^{*}) \wedge (m^{*} \notin \{m^{(1)},\dots,m^{(Q_s)}\})\wedge$ $\wedge\lvert\bar{\mathscr{I}}_{\mathsf{KeyGen}}\rvert \leq d-1 \wedge \forall i \in \{1,\dots,Q_s\},\ \lvert\bar{\mathscr{I}}^{(i)}_{\mathsf{Sign}}\rvert \leq d-1$ |

**Fig. 9.** $\mathsf{Hybrid}_3$: We replace the $\mathsf{ExecObs}$ calls with the functionally identical algorithms $\mathsf{KeyGen}_{\mathscr{L}}$ (cf. Algorithm 15) and $\mathsf{Sign}_{\mathscr{L}}$ (cf. full version).

$\bar{\mathscr{I}}'_{\mathrm{e}'}$ for some $(i, i_{\mathsf{rep}}, j) \in [k, \mathsf{rep}, d]$, and $\bar{\mathbf{s}}_i \in \bar{\mathscr{I}}'_{\mathsf{sk}}$ for some $i \in [d]$, and $\max(\lvert\bar{\mathscr{I}}'_{\mathrm{r}}\rvert, \lvert\bar{\mathscr{I}}'_{\mathrm{e}'}\rvert, \lvert\bar{\mathscr{I}}'_{\mathsf{sk}}\rvert) \leq t$.
It also holds that $(\mathsf{sig}, \mathsf{SimOut}(\mathsf{ExecObs}(\bar{\mathscr{I}}', \mathsf{Sign}, 1^{\lambda})))$ follows the same distribution as $\mathsf{ExecObs}(\bar{\mathscr{I}}_{\mathsf{Sign}}, \mathsf{Sign}, 1^{\lambda})$. From Lemma 5.2, $\mathsf{SimOut}(\mathsf{Sign}_{\mathsf{ER}}, \bar{\mathscr{I}}')$ follows the same distribution as $(\mathsf{sig}, \mathscr{L})$, where $(\mathsf{sig}, \mathscr{L}) \leftarrow \mathsf{ExecObs}(\bar{\mathscr{I}}_{\mathsf{Sign}}, \mathsf{Sign}_{\mathsf{ER}}, \mathsf{msg})$. Thus the two hybrids are identical.

$\mathsf{Hybrid}_3$**:** This hybrid corresponds to Fig. 9, in which the algorithms $\mathsf{ExecObs}(\bar{\mathscr{I}}, \mathsf{KeyGen}, 1^{\kappa})$ and $\mathsf{ExecObs}(\bar{\mathscr{I}}, \mathsf{Sign}, \mathsf{sk}, \mathsf{msg})$ are replaced by $\mathsf{KeyGen}_{\mathscr{L}}(1^{\kappa}, \bar{\mathscr{I}})$ and $\mathsf{Sign}_{\mathscr{L}}(\mathsf{sk}, \mathsf{msg}, \bar{\mathscr{I}})$, respectively. The former is presented in Algorithm 15. The latter is defined analogously and deferred to the full version due to page limitations. Observe that since $\mathsf{ExecObs}(\bar{\mathscr{I}}, \mathsf{KeyGen}, 1^{\kappa})$ outputs the same output as $\mathsf{KeyGen}(1^{\kappa})$ as well as the value of the variables at indices $\bar{\mathscr{I}}$, any algorithm that outputs the same distribution is semantically identical. Since the variables in $\bar{\mathscr{I}}$ are now restricted to the randomness used in $\mathsf{AddRepNoise}$ it is clear that the algorithm $\mathsf{KeyGen}_{\mathscr{L}}$ outputs the same distribution . The same argument goes for $\mathsf{ExecObs}(\bar{\mathscr{I}}, \mathsf{Sign}, \mathsf{sk}, \mathsf{msg})$. Hence, the two hybrids are identical.

$\mathsf{Hybrid}_4$**:** This hybrid corresponds to Fig. 10, in which the challenger artificially extends the set of probes queried to the key generation and signing algorithm. More specifically, we define $\mathsf{Extend}$ so that for any $\rho_{\mathrm{s},i,i_{\mathsf{rep}},j} \in \bar{\mathscr{I}}_{\mathrm{s}}$, all variables $\rho_{\mathrm{s},i',i_{\mathsf{rep}},j}$ for $i' \in [\ell]$ are in $\mathsf{Extend}(\bar{\mathscr{I}}_{\mathrm{s}})$ (same for $\mathsf{Extend}(\bar{\mathscr{I}}_{\mathrm{e}})$, $\mathsf{Extend}(\bar{\mathscr{I}}_{\mathrm{r}})$, $\mathsf{Extend}(\bar{\mathscr{I}}_{\mathrm{e}'})$). Conversely $\mathsf{Collapse}(\mathscr{L}'_{\mathrm{s}})$ discards the values of any variables that are in $\bar{\mathscr{I}}_{\mathrm{r}}$ but not $\bar{\mathscr{I}}'_{\mathrm{r}}$. Clearly, this does not modify the view of the

| **Adversary** | | **Challenger** |
|---|---|---|
| | $\xleftarrow{(\mathsf{KeyGen},\mathsf{Sign},\mathsf{Verify},\mathsf{KeyUpdate})}$ | |
| | $\xrightarrow{\bar{\mathscr{I}}_{\mathsf{KeyGen}}}$ | |
| | | $(\bar{\mathscr{I}}'_{\mathrm{s}},\bar{\mathscr{I}}'_{\mathrm{e}}) := \bar{\mathscr{I}}'_{\mathsf{KeyGen}} \leftarrow \mathsf{SimIn}_{\mathsf{KeyGen}}(\bar{\mathscr{I}}_{\mathsf{KeyGen}})$ |
| | | $\bar{\mathscr{I}}'_{\mathrm{s}} = \mathsf{Extend}(\bar{\mathscr{I}}'_{\mathrm{s}})$ |
| | | $\bar{\mathscr{I}}'_{\mathrm{e}} = \mathsf{Extend}(\bar{\mathscr{I}}'_{\mathrm{e}})$ |
| | | $((\mathsf{sk},\mathsf{vk}),(\mathscr{L}'_{\mathrm{s}},\mathscr{L}'_{\mathrm{e}})) \leftarrow \mathsf{KeyGen}_{\mathscr{L}}((\bar{\mathscr{I}}'_{\mathrm{s}},\bar{\mathscr{I}}'_{\mathrm{e}}),1^{\lambda})$ |
| | $\xleftarrow{\mathsf{vk},\,\mathscr{L}_{\mathsf{KeyGen}}}$ | $\mathscr{L}_{\mathsf{KeyGen}} \leftarrow \mathsf{SimOut}_{\mathsf{KeyGen}}(\mathsf{Collapse}(\mathscr{L}'_{\mathrm{s}}),\mathsf{Collapse}(\mathscr{L}'_{\mathrm{e}}))$ |
| $Q_s$ queries $\{$ | $\xrightarrow{m^{(i)},\,\bar{\mathscr{I}}^{(i)}_{\mathsf{Sign}}}$ | $\mathsf{sk} \leftarrow \mathsf{KeyUpdate}(\mathsf{sk})$ |
| | | $(\bar{\mathscr{I}}'_{\mathrm{r}},\bar{\mathscr{I}}'_{\mathrm{e}'},\bar{\mathscr{I}}'_{\mathsf{sk}}) := \bar{\mathscr{I}}^{(i)\,\prime}_{\mathsf{Sign}} \leftarrow \mathsf{SimIn}_{\mathsf{Sign}}(\bar{\mathscr{I}}^{(i)}_{\mathsf{Sign}})$ |
| | | $\bar{\mathscr{I}}'_{\mathrm{r}} = \mathsf{Extend}(\bar{\mathscr{I}}'_{\mathrm{r}})$ |
| | | $\bar{\mathscr{I}}'_{\mathrm{e}'} = \mathsf{Extend}(\bar{\mathscr{I}}'_{\mathrm{e}'})$ |
| | | $(\mathsf{sig}^{(i)},(\mathscr{L}'_{\mathrm{r}},\mathscr{L}'_{\mathrm{e}'},\mathscr{L}'_{\mathsf{sk}})) \leftarrow \mathsf{Sign}_{\mathscr{L}}((\bar{\mathscr{I}}'_{\mathrm{r}},\bar{\mathscr{I}}'_{\mathrm{e}'},\bar{\mathscr{I}}'_{\mathsf{sk}}),\mathsf{sk},m^{(i)})$ |
| | $\xleftarrow{\mathsf{sig}^{(i)},\,\mathscr{L}^{(i)}_{\mathsf{Sign}}}$ | $\mathscr{L}^{(i)}_{\mathsf{Sign}} \leftarrow \mathsf{SimOut}_{\mathsf{Sign}}(\mathsf{Collapse}(\mathscr{L}'_{\mathrm{r}}),\mathsf{Collapse}(\mathscr{L}'_{\mathrm{e}'}),\mathscr{L}'_{\mathsf{sk}})$ |
| Forgery$\{$ | $\xrightarrow{m^*,\,\mathsf{sig}^*}$ | |
| | | $b := \mathsf{Verify}(\mathsf{vk},m^*,\mathsf{sig}^*) \wedge (m^* \notin \{m^{(1)},\dots,m^{(Q_s)}\}) \wedge$ |
| | | $\wedge \lvert\bar{\mathscr{I}}_{\mathsf{KeyGen}}\rvert \leq d-1 \wedge \forall i \in \{1,\dots,Q_s\},\ \lvert\bar{\mathscr{I}}^{(i)}_{\mathsf{Sign}}\rvert \leq d-1$ |

**Fig. 10.** $\mathsf{Hybrid}_4$: In this game, for any variable name $\bar{\rho}_{\mathrm{s},i,i_{\mathsf{rep}},j}$ the challenger artificially leaks all variables $\rho_{\mathrm{s},i,i_{\mathsf{rep}},j'}$ for $j' \in [\ell]$ (and similarly when $\mathbf{s}$ is replaced by $\mathbf{e},\mathbf{r},\mathbf{e}'$). He then discards the extra leakage before sending it to the adversary. The view of the adversary is unchanged.

adversary. This conceptual change will be necessary to reduce to a simpler signing algorithm in the following section.

Lastly, we prove that for any PPT adversary $\mathcal{A}$ against the game described in $\mathsf{Hybrid}_4$ (cf. Fig. 10), we can construct an adversary $\mathbf{B}$ against the standard $\mathsf{EUF\text{-}CMA}$ security of small Raccoon in Fig. 7. At a high level a challenger can simulate queries from $\mathsf{KeyGen}_{\mathscr{L}}$ by querying the public key $\bar{\mathbf{t}}$ from the oracle for $\mathsf{KeyGen}_{Small}$ and artificially sampling additional noises $(\tilde{\mathbf{s}},\tilde{\mathbf{e}})$ as the sum of $d-1$ small uniforms and outputting the public key $\mathbf{t} := \lfloor \bar{\mathbf{t}} + \mathbf{A}\tilde{\mathbf{s}} + \tilde{\mathbf{e}} \rceil_{\nu_{\mathrm{t}}}$ which will be distributed exactly as a public key for $KeyGen_{\mathscr{L}}$. Similarly, a signature from $\mathsf{Sign}_{\mathrm{SMALL}}$ can be mapped to a signature for $\mathsf{Sign}_{\mathscr{L}}$ by sampling the appropriate sums of uniform $(\tilde{\mathbf{r}},\tilde{\mathbf{e}}')$ and setting $\mathbf{w} = \lfloor \bar{\mathbf{w}} + \mathbf{A}\tilde{\mathbf{r}} + \tilde{\mathbf{e}}' \rceil_{\nu_{\mathrm{w}}}$. Finally we show a forgery for $\mathsf{Sign}_{\mathscr{L}}$ can be mapped to a forgery for $\mathsf{Sign}_{\mathrm{SMALL}}$ . The formal proof is given in the full version. This completes the proof.

## 7.3 MLWE + SelfTargetMSIS ⇒ EUF-CMA Security of Small Raccoon

*Notations for Smooth Rényi Divergence.* We further define some useful notations to aid the readability. For any $c \in C, \mathbf{s} \in \mathcal{R}_q^{\ell}$, and $\mathbf{e} \in \mathcal{R}_q^{k}$, we note $\mathsf{center} :=$

$c \cdot \begin{bmatrix} \mathbf{s} \\ \mathbf{e} \end{bmatrix} \in \mathcal{R}_q^{\ell+k}$ and recall $T = d \cdot (\mathsf{rep} - 1) + 1$. We define two distributions: $\mathcal{P} := \mathrm{SU}(u_{\mathrm{w}}, T)^{n(\ell+k)}$ and $\mathcal{Q}(\mathsf{center}) := \mathsf{center} + \mathcal{P}$.

We bound the smooth Rényi divergence of $\mathcal{P}$ and $\mathcal{Q}$. For any $\alpha = \omega_{\mathsf{asymp}}(1)$ and $\epsilon_{\mathrm{TAIL}}(\mathsf{center}) = \frac{1}{\sqrt{2\pi T}} \left( \frac{\alpha\, e\, \|\mathsf{center}\|_2}{2^{u_{\mathrm{w}}} \cdot T} \right)^T$ (see Conjecture 4.1 or Lemma 4.2), we define $\epsilon_{\mathrm{TAIL}}$ and $R_\alpha^{\epsilon_{\mathrm{TAIL}}}(\mathcal{P}; \mathcal{Q})$ to be any two values satisfying

$$\Pr\left[\epsilon_{\mathrm{TAIL}} \geq \max_{c \in C} \epsilon_{\mathrm{TAIL}}(\mathsf{center})\right] \geq 1 - \mathsf{negl}(\kappa). \tag{10}$$

$$\Pr\left[R_\alpha^{\epsilon_{\mathrm{TAIL}}}(\mathcal{P}; \mathcal{Q}) \geq \max_{c \in C} R_\alpha^{\epsilon_{\mathrm{TAIL}}(\mathsf{center})}(\mathcal{P}; \mathcal{Q}(\mathsf{center}))\right] \geq 1 - \mathsf{negl}(\kappa). \tag{11}$$

where both probabilities are taken under the randomness of $(\mathbf{s}, \mathbf{e}) \leftarrow \mathrm{RSU}(u_{\mathrm{t}}, T)^\ell \times \mathrm{RSU}(u_{\mathrm{t}}, T)^k$. For efficiency and better parameters, we set $\epsilon_{\mathrm{TAIL}}$ and $R_\alpha^{\epsilon_{\mathrm{TAIL}}}(\mathcal{P}; \mathcal{Q})$ to be the smallest values satisfying the above inequality. The above parameters we provide is one set of candidate asymptotic parameters.

It remains to prove that small Raccoon in Fig. 7 is (standard) EUF-CMA secure. This is established in the following theorem.

**Theorem 7.2.** *The small Raccoon in Fig. 7 is* EUF-CMA *secure under the* $\mathsf{MLWE}_{q,\ell,k,\mathrm{SU}(u_{\mathrm{t}},T)}$ *and* $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,C,\beta}$ *assumptions.*

*Formally, for any adversary* $\mathcal{A}$ *against the* EUF-CMA *security game making at most* $Q_h$ *random oracle queries and* $Q_s$ *signing queries, and* $\epsilon_{\mathrm{TAIL}}$ *and* $R_\alpha^{\epsilon_{\mathrm{TAIL}}}(\mathcal{P}; \mathcal{Q})$ *satisfying Eqs. (10) and (11), there exists adversaries* $\mathcal{B}$ *and* $\mathcal{B}'$ *against the* $\mathsf{MLWE}_{q,\ell,k,\mathrm{SU}(u_{\mathrm{t}},T)}$ *and* $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,C,\beta}$ *problems such that*

$$\begin{aligned} \mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}} \leq\ & 2^{-\kappa} \cdot Q_h \cdot (1 + 2^{-\kappa+1} \cdot Q_s) + Q_s \cdot \epsilon_{\mathrm{TAIL}} \\ & + \left( \mathsf{Adv}_{\mathcal{B}}^{\mathsf{MLWE}} + \mathsf{Adv}_{\mathcal{B}'}^{\mathsf{SelfTargetMSIS}} + Q_s \cdot \epsilon_{\mathrm{TAIL}} \right)^{\frac{\alpha-1}{\alpha}} \cdot \left( R_\alpha^{\epsilon_{\mathrm{TAIL}}}(\mathcal{P}; \mathcal{Q}) \right)^{Q_s}, \end{aligned} \tag{12}$$

*where* $\mathsf{Time}(\mathcal{A}) \approx \mathsf{Time}(\mathcal{B}) \approx \mathsf{Time}(\mathcal{B}')$.

We now present an overview of the proof which, due to page constraints, is left to the full version. As a first step we replace the hash function by a random oracle which we will program by first sampling $c_{\mathsf{poly}} \leftarrow C$ and setting the hash function accordingly. Once this is done $\mathbf{w}$ can be defined as a function of $c_{\mathsf{poly}}$ rather than $(\mathbf{r}, \mathbf{e}')$, using the equation $\mathbf{w} := \mathbf{A} \cdot \mathbf{z} - c_{\mathsf{poly}} \cdot \mathbf{t} + \mathbf{z}'$ where $\mathbf{z}' := c_{\mathsf{poly}} \cdot \mathbf{e} + \mathbf{e}'$. We now observe that all variables can be computed as deterministic functions of $(\mathbf{z}, \mathbf{z}')$, we thus want to prove that $(\mathbf{z}, \mathbf{z}')$ are independent of $(\mathbf{s}, \mathbf{e})$. Using the Smooth-Renyie divergence property of Lemma 4.2 we can bound the divergence between $(\mathbf{z}, \mathbf{z}') = (c_{\mathsf{poly}} \cdot \mathbf{s} + \mathbf{r}, c_{\mathsf{poly}} \cdot \mathbf{e} + \mathbf{e}')$ and $(\mathbf{r}, \mathbf{e}')$ which are sums of uniforms independent of the secret. Finally we can replace the public key with a uniform vector using MLWE, and use the forgery output by the adversary to break MSIS.

## 8 Concrete Instantiation

Looking at Theorem 7.2, it is clear that the security bottlenecks in Theorem 7.2 are the hardness of MLWE, of SelfTargetMSIS, and the smooth Rényi divergence ($\epsilon_{\text{Tail}}$ and $R_\alpha^{\epsilon_{\text{Tail}}}$). Instantiating Raccoon boils down to an optimization problem where we need to balance the hardness assumptions (MLWE, SelfTargetMSIS), the smooth Rényi divergence and the performance metrics (size of vk and sig).

- Our analysis of MLWE and SelfTargetMSIS is fairly standard. We rely on the lattice estimator [2] for the concrete analysis of MLWE. Following the Dilithium methodology [31, §C.3], we assume that breaking SelfTargetMSIS requires to either (a) break the second-preimage resistance of the hash function, or (b) break an inhomogeneous MSIS instance, for which the best known attack is in [10, §4.2].
- For the smooth Rényi divergence, one could use Lemma 4.2 for a provable bound. However, it is not tight so we opt instead to use Conjecture 4.1.

We refer the reader to the full version of this paper where we provide the relationship between parameters the security/efficiency metrics is in. In addition, we provide example parameters for the NIST security level I (Table 3).

**Table 3.** Parameters for Raccoon-128, NIST Post-Quantum security strength category 1. For all Raccoon-128 masking orders, we fix: $\kappa = 128$, $Q_s = 2^{53}$, $q = (2^{24} - 2^{18} + 1) \cdot (2^{25} - 2^{18} + 1)$, $n = 512$, $k = 5$, $\ell = 4$, $\nu_{\text{t}} = 42$, $\nu_{\text{w}} = 44$, $\omega = 19$, $2^{-64}B_2^2 = 14656575897$, $B_\infty = 41954689765971$.

| Parameter | Raccoon-128 | 128-2 | 128-4 | 128-8 | 128-16 | 128-32 |
|---|---|---|---|---|---|---|
| \|sig\| (bytes) | 11524 | = | = | = | = | = |
| \|vk\| (bytes) | 2256 | = | = | = | = | = |
| $d$ | 1 | 2 | 4 | 8 | 16 | 32 |
| rep | 8 | 4 | 2 | 4 | 2 | 4 |
| $u_{\text{t}}$ | 6 | 6 | 6 | 5 | 5 | 4 |
| $u_{\text{w}}$ | 41 | 41 | 41 | 40 | 40 | 39 |
| \|sk\| (bytes) | 14800 | 14816 | 14848 | 14912 | 15040 | 15296 |

## 9 Conclusion and Next Steps

We have presented Raccoon, a masking-friendly signature scheme with a formal security proof in the $t$-probing model based on standard lattice assumptions. We present a few natural extensions of our work:

- **Tighter proof.** The recent Hint-MLWE assumption by Kim et al. [30] seems perfectly suited to study Raccoon, as illustrated by a thresholded variant of Raccoon [36]. For Raccoon itself, an obstacle to a direct application is that [30] provided security reductions for Gaussian distributions, whereas Raccoon uses sums of uniform distributions.

– **More realistic models.** While the $t$-probing model is a simple and convenient abstraction of real-world leakage, there exist more realistic models such as the random probing and noisy leakage models. We expect a security analysis in these models to be informative and to raise its own challenges.
– **Real-world assessment.** Since side-channel analysis are grounded in real-world deployment, this work needs to be completed with a study of the concrete leakage of Raccoon when implemented on real-world devices.

## References

1. Agrawal, S., Stehlé, D., Yadav, A.: Round-optimal lattice-based threshold signatures, revisited. In: Bojanczyk, M., Merelli, E., Woodruff, D.P. (eds.) ICALP 2022. LIPIcs, vol. 229, pp. 8:1–8:20. Schloss Dagstuhl, July 2022. https://doi.org/10.4230/LIPIcs.ICALP.2022.8
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. **9**(3), 169–203 (2015). https://doi.org/10.1515/jmc-2015-0016
3. Azouaoui, M., et al.: Protecting Dilithium against leakage revisited sensitivity analysis and improved implementations. IACR TCHES **2023**(4), 58–79 (2023). https://doi.org/10.46586/tches.v2023.i4.58-79
4. Barthe, G., et al.: 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, October 2016. https://doi.org/10.1145/2976749.2978427
5. Barthe, G., et al.: Masking the GLP lattice-based signature scheme at any order. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. LNCS, vol. 10821, pp. 354–384. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78375-8_12
6. Barthe, G., Belaïd, S., Espitau, T., Fouque, P.A., Rossi, M., Tibouchi, M.: GALACTICS: Gaussian sampling for lattice-based constant-time implementation of cryptographic signatures, revisited. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019, pp. 2147–2164. ACM Press, November 2019. https://doi.org/10.1145/3319535.3363223
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 (2016). https://doi.org/10.1007/978-3-662-53140-2_2
8. Berzati, A., Viera, A.C., Chartouny, M., Madec, S., Vergnaud, D., Vigilant, D.: Exploiting intermediate value leakage in Dilithium: a template-based approach. IACR TCHES **2023**(4), 188–210 (2023). https://doi.org/10.46586/tches.v2023.i4.188-210
9. Bronchain, O., Azouaoui, M., ElGhamrawy, M., Renes, J., Schneider, T.: Exploiting small-norm polynomial multiplication with physical attacks: application to crystals-Dilithium. Cryptology ePrint Archive, Paper 2023/1545 (2023). https://eprint.iacr.org/2023/1545
10. 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, October 2020. https://doi.org/10.1145/3320269.3384758

11. Coron, J.-S.: High-order conversion from Boolean to arithmetic masking. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 93–114. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_5
12. Coron, J.S., Gérard, F., Trannoy, M., Zeitoun, R.: Improved gadgets for the high-order masking of Dilithium. IACR TCHES **2023**(4), 110–145 (2023). https://doi.org/10.46586/tches.v2023.i4.110-145
13. Coron, J.-S., Großschädl, J., Tibouchi, M., Vadnala, P.K.: Conversion from arithmetic to Boolean masking with logarithmic complexity. In: Leander, G. (ed.) FSE 2015. LNCS, vol. 9054, pp. 130–149. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48116-5_7
14. Coron, J.-S., Großschädl, J., Vadnala, P.K.: Secure conversion between Boolean and arithmetic masking of any order. In: Batina, L., Robshaw, M. (eds.) CHES 2014. LNCS, vol. 8731, pp. 188–205. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44709-3_11
15. Csiszár, I.: Eine informationstheoretische Ungleichung und ihre Anwendung auf den Beweis der Ergodizitat von Markoffschen Ketten. Magyar. Tud. Akad. Mat. Kutató Int. Közl **8**, 85–108 (1963)
16. del Pino, R., et al.: Raccoon. Technical report, National Institute of Standards and Technology (2023). https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures
17. del Pino, R., Prest, T., Rossi, M., Saarinen, M.J.O.: High-order masking of lattice signatures in quasilinear time. In: 2023 IEEE Symposium on Security and Privacy, pp. 1168–1185. IEEE Computer Society Press, May 2023. https://doi.org/10.1109/SP46215.2023.10179342
18. Devevey, J., Fawzi, O., Passelègue, A., Stehlé, D.: On rejection sampling in Lyubashevsky's signature scheme. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part IV. LNCS, vol. 13794, pp. 34–64. Springer, Heidelberg (2022). https://doi.org/10.1007/978-3-031-22972-5_2
19. Duc, A., Faust, S., Standaert, F.X.: Making masking security proofs concrete (or how to evaluate the security of any leaking device), extended version. J. Cryptol. **32**(4), 1263–1297 (2019). https://doi.org/10.1007/s00145-018-9277-0
20. Esgin, M., Espitau, T., Niot, G., Prest, T., Sakzad, A., Steinfeld, R.: Plover: masking-friendly hash-and-sign lattice signatures. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024. LNCS, vol. 14657. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-58754-2_12. https://tprest.github.io/pdf/pub/plover.pdf
21. Espitau, T., et al.: 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 (2022). https://doi.org/10.1007/978-3-031-07082-2_9
22. Fournaris, A.P., Dimopoulos, C., Koufopavlou, O.: Profiling Dilithium digital signature traces for correlation differential side channel attacks. In: Orailoglu, A., Jung, M., Reichenbach, M. (eds.) SAMOS 2020. LNCS, vol. 12471, pp. 281–294. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-60939-9_19
23. Gérard, F., Rossi, M.: An efficient and provable masked implementation of `qTESLA`. In: Belaïd, S., Güneysu, T. (eds.) CARDIS 2019. LNCS, vol. 11833, pp. 74–91. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-42068-0_5
24. Goudarzi, D., Prest, T., Rivain, M., Vergnaud, D.: Probing security through input-output separation and revisited quasilinear masking. IACR TCHES **2021**(3), 599–640 (2021). https://doi.org/10.46586/tches.v2021.i3.599-640. https://tches.iacr.org/index.php/TCHES/article/view/8987

25. Paquin, C., Stebila, D., Tamvada, G.: Benchmarking post-quantum cryptography in TLS. In: Ding, J., Tillich, J.-P. (eds.) PQCrypto 2020. LNCS, vol. 12100, pp. 72–91. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44223-1_5
26. Hutter, M., Tunstall, M.: Constant-time higher-order Boolean-to-arithmetic masking. J. Cryptographic Eng. **9**(2), 173–184 (2019). https://doi.org/10.1007/s13389-018-0191-z
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 (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, November 2022. https://doi.org/10.1145/3548606.3560579
29. Karabulut, E., Alkim, E., Aysu, A.: Single-trace side-channel attacks on $\omega$-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, 12–15 December 2021, pp. 35–45. IEEE (2021). https://doi.org/10.1109/HOST49136.2021.9702284
30. 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 (2023). https://doi.org/10.1007/978-3-031-38554-4_18
31. Lyubashevsky, V., et al.: CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology (2022). https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
32. 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
33. Mathieu-Mahias, A.: Securisation of implementations of cryptographic algorithms in the context of embedded systems. (Sécurisation des implémentations d'algorithmes cryptographiques pour les systèmes embarqués). Ph.D. thesis, University of Paris-Saclay, France (2021). https://tel.archives-ouvertes.fr/tel-03537322
34. Migliore, V., Gérard, B., Tibouchi, M., Fouque, P.-A.: Masking Dilithium. In: Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M. (eds.) ACNS 2019. LNCS, vol. 11464, pp. 344–362. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21568-2_17
35. Naehrig, M., et al.: FrodoKEM. Technical report, National Institute of Standards and Technology (2017). https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions
36. Pino, R.D., Katsumata, S., Maller, M., Mouhartem, F., Prest, T., Saarinen, M.O.: Threshold raccoon: practical threshold signatures from standard lattice assumptions. In: EUROCRYPT 2024. LNCS, vol. 14652, pp. 219–248. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-58723-8_8
37. 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 (2023). https://doi.org/10.1007/978-3-031-31368-4_8
38. Prest, T., et al.: FALCON. Technical report, National Institute of Standards and Technology (2022). https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022

39. Steffen, H.M., Land, G., Kogelheide, L.J., Güneysu, T.: Breaking and protecting the crystal: side-channel analysis of Dilithium in hardware. In: PQCrypto 2023. LNCS, vol. 14154, pp. 688–711. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-40003-2_25
40. Wang, R., Ngo, K., Gärtner, J., Dubrova, E.: Single-trace side-channel attacks on crystals-Dilithium: myth or reality? Cryptology ePrint Archive, Paper 2023/1931 (2023). https://eprint.iacr.org/2023/1931