




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

Rafaël del Pino¹, Shuichi Katsumata^{1,2}
, Thomas Prest¹, and Mélissa Rossi³

¹ PQShield (`firstname.lastname@pqshield.com`)

² AIST

³ ANSSI (`firstname.lastname@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 dependency 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.

1 Introduction

In the past decade, post-quantum cryptography has rapidly evolved from a mostly theoretical field to a mature one suitable for large-scale deployment. 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 Section 1.3 for references of concrete SCAs against Dilithium.

Masking. The main countermeasure against side-channel attacks is masking [ISW03]. It consists of splitting sensitive information in d shares (concretely: $x = x_0 + \dots + x_{d-1}$), 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 [DFS19,IUH22].

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. The best known ways to perform these operations is to rely on mask conversions [HT19,CGTV15], which convert between arithmetic and boolean masking. This typically incurs an overhead $O(d^2 \log q)$ [CGV14] or $O(2^{d/2})$ [Cor17], and quickly becomes the efficiency bottleneck. As an illustration, the only publicly available *masked* implementation of Dilithium [CGTZ23] 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 Section 1.3 for a more detailed discussion. Achieving a formally proven, masking-friendly signature has remained 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 [ISW03]. While Raccoon is inspired from the similarly named scheme from [dPPRS23], 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 [dEK⁺23].

Blueprint. Raccoon is based on the “Lyubashevsky signature without aborts” blueprint, also found in works on threshold signatures [ASY22], and which we recall below. Assume the public key 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 $\mathbf{w} = [\mathbf{A} \ \mathbf{I}] \cdot \mathbf{r}$;

(S2) Compute a challenge $c = H(\mathbf{w}, \text{vk}, \text{msg})$;

(S3) Compute a response $\mathbf{z} = \mathbf{s} \cdot c + \mathbf{r}$.

The verification procedure checks that $H(\mathbf{A} \cdot \mathbf{z} - \mathbf{t} \cdot c, \text{vk}, \text{msg}) = c$ 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 [ASY22] 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 [NAB⁺17] or Falcon [PFH⁺22]. 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 $\mathcal{O}(d^2 \log q)$. Similarly, all other existing sampling methods (see e.g. “Related works” in [HPRR20]) 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 generalization 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 $\llbracket \mathbf{r} \rrbracket$ as the sum of rep small uniform variates, so that \mathbf{r} is the sum of $d \cdot \text{rep}$ 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 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 $d \cdot \text{rep} - d + 1$ 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 Section 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 $\{\mathbf{s}_0, \dots, \mathbf{s}_{d-2}\}$, he can compute $\mathbf{vk}' = \mathbf{vk} - [\mathbf{A} \ \mathbf{I}] \sum_{i=0}^{d-2} \mathbf{s}_i = [\mathbf{A} \ \mathbf{I}] \mathbf{s}_{d-1}$. 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.

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 provided in [BBE⁺18]. Such a notion has been defined but it was not formally used in a game-based proof before.

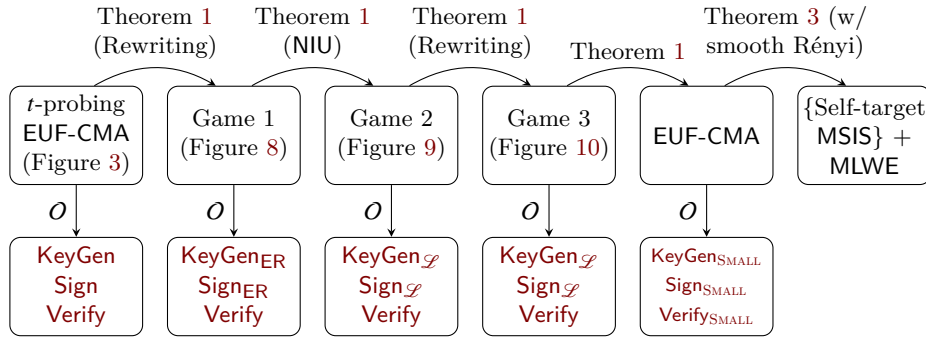


Fig. 1: Proof overview. Jump 1 consists in moving randomness to inputs as per Definition 7. Jump 2 uses Lemma 5 to move all probes to inputs. Jump 3 is a simple rewriting step. Jump 4 is a black-box reduction to a simpler unmasked signature Small Raccoon. Jump 5 is the security proof of Small Raccoon. \mathcal{O} denote access to an oracle to the corresponding algorithm.

The main contribution of this paper is a 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 [ISW03] and other masking proof techniques [BBD⁺16]. 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 [EEN⁺24] (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 Section 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) 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 to 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 is proven in Section 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 Section 7.3.

In Section 8, we instantiate the parameters to validate 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 [FDK20], [KAA21], [MUTS22], [BVC⁺23], [SLKG23], [BAE⁺23], [WNGD23].

Masking lattice schemes. The formal study of masking lattice-based signatures has been initiated by Barthe et al. [BBE⁺18], which studied the GLP signature. Since then, BLISS [BBE⁺19] and qTESLA [GR19] have also been studied from a masking perspective. Masked implementations of Dilithium have been proposed in [MGTF19], [ABC⁺23], [CGTZ23].

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

- *Mitaka.* Espitau et al. [EFG⁺22] 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 [Pre23].
- *IEEE SP Raccoon.* At IEEE S&P 2023, del Pino et al. [dPPRS23] presented a lattice-based masking-friendly signature, also called Raccoon. Our scheme is a conceptual descendent of the scheme from [dPPRS23], with significant improvements. While both versions of Raccoon are Fiat-Shamir lattice-based signatures, the security proof of [dPPRS23] 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 [dEK⁺23], Esgin et al. [EEN⁺24] 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, [EEN⁺24] 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 $\{0, 1, \dots, n-1\}$. Let $f : X \rightarrow Y$ 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 assumption (SelfTargetMSIS). This variant of MSIS, is defined with respect to a 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 $\mathbf{b} = G(\mathbf{x}_1, \dots, \mathbf{x}_k)$ the variables $\mathbf{b}, \mathbf{x}_1, \dots, \mathbf{x}_k$ 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 $\llbracket \mathbf{x} \rrbracket = (\mathbf{x}_i)_{i \in [d]}$ 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 $\llbracket \mathbf{x} \rrbracket$.

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 $(\bar{a}_1, \dots, \bar{a}_N)$ that calls the gadgets G_1, \dots, G_k , (with $N, k \in$

\mathbb{N}), we will consider the set of variables $\bar{\mathcal{V}}_P = \{\bar{a}_1, \dots, \bar{a}_N\} \uplus \bar{\mathcal{V}}_{G_1} \uplus \dots \uplus \bar{\mathcal{V}}_{G_k}$ (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 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 1 (Probes). For a well-formed program P with variables $\bar{\mathcal{V}}_P$ and input variables $\bar{a}_1, \dots, \bar{a}_N$, a set of probes is a set $\bar{\mathcal{F}} \subset \bar{\mathcal{V}}_P$. For any set $\bar{\mathcal{F}} \subset \bar{\mathcal{V}}_P$ and any scalars $\mathcal{X} = (a_1, \dots, a_N)$ we will denote as $\text{ExecObs}(P, \bar{\mathcal{F}}, \mathcal{X})$ the joint distribution of the (masked and unmasked) outputs of $P(a_1, \dots, a_N)$ and of all the values taken by the variables in $\bar{\mathcal{F}}$. In particular for

$$(out_{\text{masked}}, out_{\text{unmasked}}, \mathcal{L}) \leftarrow \text{ExecObs}(P, \bar{\mathcal{F}}, \mathcal{X}),$$

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

Probing model. We recall standard non-interference results from [BBD⁺16].

Definition 2 (Perfect simulatability, reformulation of [BBD⁺16]). Let $\bar{\mathcal{F}}$ be a set of probes of a gadget \mathbf{G} with input shares \bar{X} . We say that the PPT simulator $(\text{SimIn}, \text{SimOut})$ perfectly simulates the probes $\bar{\mathcal{F}}$ if and only if for any input values \mathcal{X} , $\text{SimIn}(\mathbf{G}, \bar{\mathcal{F}})$ outputs a subset $\bar{X}' \subset \bar{X}$ of the input variables of \mathbf{G} , and $\text{SimOut}(\mathbf{G}, \bar{X}')$ (where \bar{X}' is the values taken by \mathcal{X} at indices \bar{X}') outputs a tuple of values such that the marginal distribution of \mathcal{L} , for $(out_{\text{masked}}, out_{\text{unmasked}}, \mathcal{L}) \leftarrow \text{ExecObs}(P, \bar{\mathcal{F}}, \mathcal{X})$, and $\text{SimOut}(\mathbf{G}, \bar{X}')$ are identical.

Definition 3 (Non Interference [BBD⁺16]). A gadget is said $(d-1)$ -non-interfering (written $(d-1)$ -NI for short) iff any set of probes $\bar{\mathcal{F}}$ such that $|\bar{\mathcal{F}}| \leq d-1$ can be perfectly simulated (See Definition 2) by a simulator $(\text{SimIn}, \text{SimOut})$ such that $\text{SimIn}(\mathbf{G}, \bar{\mathcal{F}})$ outputs a set \bar{X}' of at most $d-1$ shares of each input.

Definition 4 (Strong Non Interference [BBD⁺16]). A gadget is said $(d-1)$ -strongly-non-interfering (written $(d-1)$ -sNI for short) iff any set $\bar{\mathcal{F}}$ of at most $d-1 = d_{\text{int}} + d_{\text{out}}$ probes, where d_{int} are made on internal data and d_{out} are made on the outputs, can be perfectly simulated by a simulator $(\text{SimIn}, \text{SimOut})$ such that $\text{SimIn}(\mathbf{G}, \bar{\mathcal{F}})$ outputs a set \bar{X}' of at most d_{int} shares of each input.

Lemma 1 (Composability of NI and sNI gadgets [BBE⁺18]). A well-formed algorithm is NI if all of its gadgets are NI or sNI and each sharing is used at most once as input of a non-sNI gadget. Moreover, a well-formed algorithm is sNI if it is NI and its output sharings are issued from a 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 NI/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:

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

The acronym SU stands for “sum of uniforms”. This class of distributions is

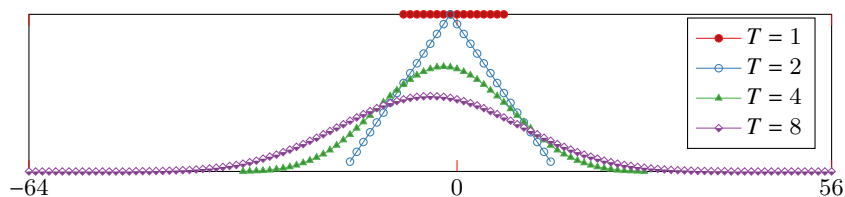


Fig. 2: The distribution $\text{SU}(4, T)$, for $T \in \{1, 2, 4, 8\}$

illustrated in Figure 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 $\text{RSU}(u, 1)$ the distribution over \mathcal{R} obtained by sampling each integer coefficient of $a \in \mathcal{R}$ according to $\text{SU}(u, 1)$, 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.

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

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

3.1 Key Generation

Masked key generation process is described by Algorithm 1. At a high-level, **KeyGen** generates d -sharings $(\llbracket \mathbf{s} \rrbracket, \llbracket \mathbf{e} \rrbracket)$ 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 $\llbracket \mathbf{s} \rrbracket, \llbracket \mathbf{e} \rrbracket$ are generated in Lines 4 and 6 using our novel algorithm **AddRepNoise** (Algorithm 5).

Algorithm 1 **KeyGen**(\emptyset) \rightarrow (vk, sk)

Output: Keypair vk, sk

- 1: $\text{seed} \leftarrow \{0, 1\}^\kappa$ $\triangleright \kappa$ -bit random seed for \mathbf{A} .
 - 2: $\mathbf{A} := \text{ExpandA}(\text{seed})$ \triangleright Similar to **ExpandA** in Dilithium. $\mathbf{A} \in \mathcal{R}_q^{k \times \ell}$.
 - 3: $\llbracket \mathbf{s} \rrbracket \leftarrow \ell \times \text{ZeroEncoding}(d)$ \triangleright Masked zero vector $\llbracket \mathbf{s} \rrbracket \in (\mathcal{R}_q^\ell)^d$. Algorithm 8.
 - 4: $\llbracket \mathbf{s} \rrbracket \leftarrow \text{AddRepNoise}(\llbracket \mathbf{s} \rrbracket, u_t, \text{rep})$ \triangleright Generate the secret distribution.
Algorithm 5.
 - 5: $\llbracket \mathbf{t} \rrbracket := \mathbf{A} \cdot \llbracket \mathbf{s} \rrbracket$ \triangleright Compute masked product $\llbracket \mathbf{t} \rrbracket \in (\mathcal{R}_q^k)^d$.
 - 6: $\llbracket \mathbf{t} \rrbracket \leftarrow \text{AddRepNoise}(\llbracket \mathbf{t} \rrbracket, u_t, \text{rep})$ \triangleright Add masked noise to $\llbracket \mathbf{t} \rrbracket$. Algorithm 5.
 - 7: $\mathbf{t} := \text{Decode}(\llbracket \mathbf{t} \rrbracket)$ \triangleright Collapse $\mathbf{t} \in \mathcal{R}_q^k$. Algorithm 6.
 - 8: $\mathbf{t} := \lfloor \mathbf{t} \rfloor_{\nu_t}$ \triangleright Rounding and right-shift to modulus $q_t = \lfloor q/2^{\nu_t} \rfloor$.
 - 9: **return** (vk := (seed, t), sk := (vk, $\llbracket \mathbf{s} \rrbracket$)) \triangleright Return serialized key pair.
-

3.2 Signing Procedure

The masked signing process is described by Algorithm 2. This signing procedure is similar to the “Lyubashevskys Signature Without Aborts” in [ASY22, Fig. 2]. Again, the use of **AddRepNoise** is crucial in this procedure. The challenge computation is divided in two parts, first a 2κ 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.

Algorithm 2 $\text{Sign}(\llbracket \text{sk} \rrbracket, \text{msg}) \rightarrow \text{sig}$

Input: Secret signing key $\text{sk} = (\text{vk}, \llbracket \text{s} \rrbracket)$, message to be signed $\text{msg} \in \{0, 1\}^*$.

Output: Signature $\text{sig} = (c_{\text{hash}}, \mathbf{h}, \mathbf{z})$ of msg under sk .

- 1: $\mu := \text{H}(\text{H}(\text{vk}) \parallel \text{msg})$ \triangleright Bind vk with msg to form $\mu \in \{0, 1\}^{2\kappa}$.
 - 2: $\mathbf{A} := \text{ExpandA}(\text{seed})$ \triangleright Similar to ExpandA in Dilithium. $\mathbf{A} \in \mathcal{R}_q^{k \times \ell}$.
 - 3: $\llbracket \mathbf{r} \rrbracket \leftarrow \ell \times \text{ZeroEncoding}(d)$ \triangleright Masked zero vector $\llbracket \mathbf{r} \rrbracket \in (\mathcal{R}_q^\ell)^d$. Algorithm 8.
 - 4: $\llbracket \mathbf{r} \rrbracket \leftarrow \text{AddRepNoise}(\llbracket \mathbf{r} \rrbracket, u_{\mathbf{w}}, \text{rep})$ \triangleright Add masked noise to $\llbracket \mathbf{r} \rrbracket$. Algorithm 5.
 - 5: $\llbracket \mathbf{w} \rrbracket := \mathbf{A} \cdot \llbracket \mathbf{r} \rrbracket$ \triangleright Compute masked product $\llbracket \mathbf{w} \rrbracket \in (\mathcal{R}_q^\ell)^d$.
 - 6: $\llbracket \mathbf{w} \rrbracket \leftarrow \text{AddRepNoise}(\llbracket \mathbf{w} \rrbracket, u_{\mathbf{w}}, \text{rep})$ \triangleright Add masked noise to $\llbracket \mathbf{w} \rrbracket$. Algorithm 5.
 - 7: $\mathbf{w} := \text{Decode}(\llbracket \mathbf{w} \rrbracket)$ \triangleright Collapse LWE commitment \mathbf{w} . Algorithm 6.
 - 8: $\mathbf{w} := \lfloor \mathbf{w} \rfloor_{v_{\mathbf{w}}}$ \triangleright Rounding and right-shift to modulus $q_{\mathbf{w}} = \lfloor q/2^{v_{\mathbf{w}}} \rfloor$.
 - 9: $c_{\text{hash}} := \text{ChalHash}(\mathbf{w}, \mu)$ \triangleright Map \mathbf{w} and μ to $c_{\text{hash}} \in \{0, 1\}^{2\kappa}$.
 - 10: $c_{\text{poly}} := \text{ChalPoly}(c_{\text{hash}})$ \triangleright Map c_{hash} to $c_{\text{poly}} \in \mathcal{C}$.
 - 11: $\llbracket \mathbf{s} \rrbracket \leftarrow \text{Refresh}(\llbracket \mathbf{s} \rrbracket)$ \triangleright Refresh $\llbracket \mathbf{s} \rrbracket$ before re-use. Algorithm 7.
 - 12: $\llbracket \mathbf{r} \rrbracket \leftarrow \text{Refresh}(\llbracket \mathbf{r} \rrbracket)$ \triangleright Refresh $\llbracket \mathbf{r} \rrbracket$ before re-use. Algorithm 7.
 - 13: $\llbracket \mathbf{z} \rrbracket := c_{\text{poly}} \cdot \llbracket \mathbf{s} \rrbracket + \llbracket \mathbf{r} \rrbracket$ \triangleright Masked response $\llbracket \mathbf{z} \rrbracket \in (\mathcal{R}_q^\ell)^d$.
 - 14: $\llbracket \mathbf{z} \rrbracket \leftarrow \text{Refresh}(\llbracket \mathbf{z} \rrbracket)$ \triangleright Refresh $\llbracket \mathbf{z} \rrbracket$ before collapsing it. Algorithm 7.
 - 15: $\mathbf{z} := \text{Decode}(\llbracket \mathbf{z} \rrbracket)$ \triangleright Collapse into response $\mathbf{z} \in \mathcal{R}_q^\ell$. Algorithm 6.
 - 16: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - 2^{v_{\mathbf{t}}} \cdot c_{\text{poly}} \cdot \mathbf{t}$ \triangleright “Noisy” LWE commitment.
 - 17: $\mathbf{h} := \mathbf{w} - \lfloor \mathbf{y} \rfloor_{v_{\mathbf{w}}}$ \triangleright Compute hint $\mathbf{h} \in \mathcal{R}_{q_{\mathbf{w}}}^k$. Subtraction mod $q_{\mathbf{w}}$.
 - 18: $\text{sig} := (c_{\text{hash}}, \mathbf{h}, \mathbf{z})$
 - 19: **if** $\{\text{CheckBounds}(\text{sig}) = \text{FAIL}\}$ **goto** Line 3 \triangleright Sanity check on the signature. Algorithm 4.
 - 20: **return** sig \triangleright Return encoded signature triplet.
-

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.

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

Input: Signature $\text{sig} = (c_{\text{hash}}, \mathbf{h}, \mathbf{z})$, message $\text{msg} \in \{0, 1\}^*$, public key $\text{vk} = (\text{seed}, \mathbf{t})$.

Output: Signature validity: OK (accept) or FAIL (reject).

- 1: **if** $\text{CheckBounds}(\text{sig}) = \text{FAIL}$ **return** FAIL \triangleright Norms check. Algorithm 4.
 - 2: $\mu := \text{H}(\text{H}(\text{vk}) \parallel \text{msg})$; $\mathbf{A} := \text{ExpandA}(\text{seed})$
 - 3: $c_{\text{poly}} := \text{ChalPoly}(c_{\text{hash}})$ \triangleright Map c_{hash} to $c_{\text{poly}} \in \mathcal{C}$.
 - 4: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - 2^{v_{\mathbf{t}}} \cdot c_{\text{poly}} \cdot \mathbf{t}$ \triangleright Scale \mathbf{t} from $\mathbb{Z}_{q_{\mathbf{t}}}$ to \mathbb{Z}_q and recompute the commitment.
 - 5: $\mathbf{w}' := \lfloor \mathbf{y} \rfloor_{v_{\mathbf{w}}} + \mathbf{h}$ \triangleright Adjust the LWE commitment with hint (mod $q_{\mathbf{w}}$).
 - 6: $c'_{\text{hash}} := \text{ChalHash}(\mathbf{w}', \mu)$ \triangleright Recompute $c'_{\text{hash}} \in \{0, 1\}^{2\kappa}$.
 - 7: **if** $c_{\text{hash}} \neq c'_{\text{hash}}$ **return** FAIL \triangleright Check commitment.
 - 8: **return** OK \triangleright Signature is accepted.
-

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_∞ and B_2 such that honest signatures verify with overwhelming probability (this is necessary for the reduction of Section 7.2 to go through).

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) \rightarrow {OK or FAIL}

Input: Signature sig = ($c_{\text{hash}}, \mathbf{h}, \mathbf{z}$).

Output: Format validity check OK or FAIL.

1: if $(\|\mathbf{z}, 2^{y_w} \cdot \mathbf{h}\|_\infty > B_\infty)$ or $(\|\mathbf{z}, 2^{y_w} \cdot \mathbf{h}\|_2 > B_2)$ return FAIL else return OK

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

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_{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

Algorithm 5 $\text{AddRepNoise}(\llbracket \mathbf{v} \rrbracket, u, \text{rep}) \rightarrow \llbracket \mathbf{v} \rrbracket$

Input: Masked vector $\llbracket \mathbf{v} \rrbracket = (\mathbf{v}_j)_{j \in [d]} = (v_{i,j})_{i \in [\text{len}(\mathbf{v})], j \in [d]}$.

Input: Bit size (distribution parameter) u .

Input: Global repetition count parameter rep .

Output: Updated $\llbracket \mathbf{v} \rrbracket$ with $\text{SU}(u, d \cdot \text{rep})$ distribution added to each coefficient of \mathbf{v} .

```

1: for  $i \in [\text{len}(\mathbf{v})]$  do ▷ Vector index.
2:   for  $i_{\text{rep}} \in [\text{rep}]$  do ▷ Repetition index.
3:     for  $j \in [d]$  do ▷ Share index.
4:        $\rho \leftarrow \text{RSU}(u, 1)$  ▷ uniform sample of  $u$  bits
5:        $v_{i,j} \leftarrow v_{i,j} + \rho$  ▷ Add small uniform to the polynomial.
6:        $\llbracket \mathbf{v}_i \rrbracket \leftarrow \text{Refresh}(\llbracket \mathbf{v}_i \rrbracket)$  ▷ Refresh polynomial on each repeat.
7: return  $\llbracket \mathbf{v} \rrbracket$ 

```

of entropy. While we present **ZeroEncoding** as a recursive algorithm, it is easy to see that it can be computed in-place and its memory requirement is $O(d)$. Remark 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 $\llbracket \mathbf{x} \rrbracket = (\mathbf{x}_i)_{i \in [t+1]}$ as input, refreshes it with Algorithm 7, then computes the sum $\mathbf{x}_0 + \dots + \mathbf{x}_{d-1} \bmod q$. In practice, when the decoding gadget is already preceded by a refresh gadget, one of them is omitted. **Decode** is similar to the algorithm “FullAdd” from [BBE⁺18, Alg. 16]. Note that since the underlying refresh is not a linear combination of $d+1$ refresh algorithms as in [BBE⁺18, Alg. 16], the security argument is different. One need to argue that **Decode** is free-SNI [CS21, Def. 10]. This has been proved in [BCRT23, Cor. 5]. Next, one need to show that it implies Nlo security using [CGMZ23, Cor. 1].⁴

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 We warmly thank Katharina Boudgoust, Loïc Demange, Laurent Imbert, Loïc Masure, Camille Mutschler and Thomas Roche for finding the issue and suggesting the correct arguments.

<hr/> Algorithm 6 $\text{Decode}(\llbracket x \rrbracket) \rightarrow x$ <hr/> Input: d -sharing $\llbracket x \rrbracket = (x_i)_i$ of $x \in \mathcal{R}_q$ Output: The clear value $x \in \mathcal{R}_q$ 1: $\llbracket x \rrbracket \leftarrow \text{Refresh}(\llbracket x \rrbracket)$ 2: return $x := \sum_{i \in [d]} x_i$ <hr/>	<hr/> Algorithm 7 $\text{Refresh}(\llbracket x \rrbracket) \rightarrow \llbracket x \rrbracket'$ <hr/> Input: A d -sharing $\llbracket x \rrbracket$ of $x \in \mathcal{R}_q$ Output: A fresh d -sharing $\llbracket x \rrbracket'$ of x 1: $\llbracket z \rrbracket \leftarrow \text{ZeroEncoding}(d)$ 2: return $\llbracket x \rrbracket' := \llbracket x \rrbracket + \llbracket z \rrbracket$ <hr/>
<hr/> Algorithm 8 $\text{ZeroEncoding}(d) \rightarrow \llbracket z \rrbracket_d$ <hr/> Input: A power-of-two integer d , a ring \mathcal{R}_q Output: A uniform d -sharing $\llbracket z \rrbracket \in \mathcal{R}_q^d$ of $0 \in \mathcal{R}_q$ 1: if $d = 1$ then 2: return $\llbracket z \rrbracket_1 := (0)$ \triangleright There is only one way to encode zero into 1 share. 3: $\llbracket z_1 \rrbracket_{d/2} \leftarrow \text{ZeroEncoding}(d/2)$ \triangleright Recursively obtain left side. 4: $\llbracket z_2 \rrbracket_{d/2} \leftarrow \text{ZeroEncoding}(d/2)$ \triangleright Recursively obtain right side. 5: $\llbracket r \rrbracket_{d/2} \xleftarrow{M} \mathcal{R}_q^{d/2}$ \triangleright Sampled using a Mask Random Generator (MRG). 6: $\llbracket z_1 \rrbracket_{d/2} := \llbracket z_1 \rrbracket_{d/2} + \llbracket r \rrbracket_{d/2}$ \triangleright Add to the left side. 7: $\llbracket z_2 \rrbracket_{d/2} := \llbracket z_2 \rrbracket_{d/2} - \llbracket r \rrbracket_{d/2}$ \triangleright Subtract from the right side. 8: return $\llbracket z \rrbracket_d := (\llbracket z_1 \rrbracket_{d/2} \parallel \llbracket z_2 \rrbracket_{d/2})$ \triangleright Concatenate the two. <hr/>	

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 $P = \text{SU}(u, T)$ and $Q = P + a$, for any $a \neq 0$. The *smooth* Rényi divergence (Definition 5) 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 5 (Smooth Rényi divergence). *Let $\epsilon \geq 0$ and $1 < \alpha < \infty$. Let P, Q be two distributions of countable supports $\text{Supp}(P) \subseteq \text{Supp}(Q) = X$. The smooth Rényi divergence of parameters (α, ϵ) between P and Q is defined as:*

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

where Δ_{SD} and R_α denote the statistical distance and the Rényi divergence, respectively:

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

While [DFPS22] has also provided a definition of smooth Rényi divergence, we argue that our definition is more natural. Indeed, it satisfies variations of prop-

erties that are expected from classical Rényi divergences. These are listed in Lemma 2.

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

Lemma 2. *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) $\text{Supp}(P) \cup \text{Supp}(Q)$.*

$$R_\alpha^\epsilon(g(P); g(Q)) \leq R_\alpha^\epsilon(P; Q). \quad (2)$$

2. **Probability preservation.** *For any event $E \subseteq \text{Supp}(Q)$:*

$$P(E) \leq (Q(E) + \epsilon)^{(\alpha-1)/\alpha} \cdot R_\alpha^\epsilon(P; Q) + \epsilon. \quad (3)$$

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 $\epsilon = \sum_{i \in I} \epsilon_i$.*

$$R_\alpha^\epsilon \left(\prod_{i \in I} P_i; \prod_{i \in I} Q_i \right) \leq \prod_{i \in I} R_\alpha^{\epsilon_i}(P_i; Q_i). \quad (4)$$

Proof. We recall that Δ_{SD} and $(R_\alpha^\alpha - 1)$ can be cast as f -divergences, following Csiszár's terminology [Csi63]. 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. \square

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 3. *Let $T, u, N \in \mathbb{N}$ and $c \in \mathbb{Z}$ such that $T \geq 4$ and $N = 2^u$. Let $P = \text{SU}(u, T)$ 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:

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

Gap with practice. In practice, Lemma 3 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 3 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, [ASY22, Lemma 2.28] tells that if P is instead a Gaussian of parameter σ , then $\log R_\alpha(P; Q) \leq \frac{\alpha c^2}{2\sigma^2}$. Thus there is a gap $O(T^3)$ between Lemma 3 and [ASY22, 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 1 in the full version.

Conjecture 1. Under the conditions of Lemma 3, we have

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

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 $\alpha = \omega_{\text{asympt}}(1)$ and $T = o(\alpha|c_i|)$ for all the i -th ($i \in [M]$) entry of \mathbf{c} , we have:

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

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

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 Section 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 [EEN⁺24] (cf. Section 1.3).

5.1 EUF-CMA Security in the Probing Model

We use the definition of [BBE⁺18] 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 1).

Definition 6. Let $d \geq 1$ an integer, Q_s be a fixed maximum amount of signature queries. A signature scheme $(\text{KeyGen}, \text{Sign}, \text{Verify})$ 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 Figure 3.

As in [BBE⁺18], 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 [BBE⁺18, Remark 3] for more details.

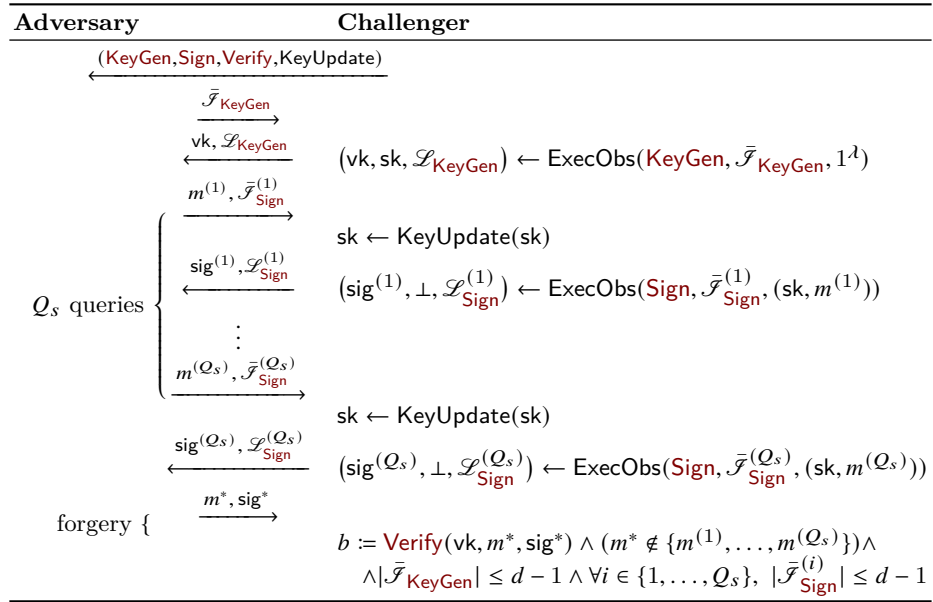


Fig. 3: EUF-CMA security game in the $d - 1$ -probing model. See Definition 1 for the definition of ExecObs .

Remark 2 (Standard EUF-CMA security). We note that Definition 6 incorporates the standard notion of standard EUF-CMA (i.e., 0-probing). For this, we define KeyUpdate to be the identity 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 knowledge of some shares of $\llbracket \mathbf{s} \rrbracket$ combined with the knowledge of the public key \mathbf{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 Section 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 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 where the definition requires $\text{SimOut}(\mathbf{G}, \mathcal{X}')$ and \mathcal{L} to be identically distributed, but not $(\text{out}_{\text{unmasked}}, \text{SimOut}(\mathbf{G}, \mathcal{X}'))$ and $(\text{out}_{\text{unmasked}}, \mathcal{L})$.

non-NIU($\llbracket \mathbf{v} \rrbracket$)	NIU($\llbracket \mathbf{v} \rrbracket, (\rho_i)_{i \in [d]}$)
1: for $j \in [d]$ do 2: $\rho_j \leftarrow \text{RSU}(u, 1)$ 3: $v'_j \leftarrow v_j + \rho_j$	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 Figure 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 . If we however consider the gadgets of Figure 4, append a **Decode**(\mathbf{v}') gadget to them, and output \mathbf{v}' , then the NI notion is insufficient to say anything about the distribution of \mathbf{v}' conditioned on the values of the probes: a simulator taking as input shares of $\llbracket \mathbf{v} \rrbracket$ cannot output probes that are correlated to the decoded output \mathbf{v}' . For example, in gadget non-NIU, consider the set of probes $\bar{\mathcal{F}} = \{\bar{v}'_1\}$ which corresponds to the sum of v_1 and ρ_1 . A simulator ($\text{SimIn}, \text{SimOut}$) can perfectly simulate $\bar{\mathcal{F}}$ by setting $\text{SimIn}(\text{non-NIU}, \bar{\mathcal{F}}) := \bar{v}'_1$, and $\text{SimOut}(\text{non-NIU}, v_1) := v_1 + \text{RSU}(u, 1)$. Then the variable $\mathcal{L} = v'_1$ being probed has the same distribution as $\text{SimOut}(\text{non-NIU}, v_1)$. However the distribution of $(\mathcal{L}, \text{out}_{\text{unmasked}}) = (v_1 + \rho_1, v + \rho_1 + \dots + \rho_d)$ is clearly not the same as that of $(\text{SimOut}(\text{non-NIU}, v_1), \text{out}_{\text{unmasked}}) = (v_1 + \text{RSU}(u, 1), v + \rho_1 + \dots + \rho_d)$.

5.3 NI/sNI with Unshared Inputs

To be able to handle cases where the values being probed are correlated with the 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 7 and 8 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), we note that once an algorithm \mathcal{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 Figure 4 where we parse the random samples ρ_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 $\mathcal{F} = \bar{v}'_1$ as before, this time the simulator needs to use two input values to simulate the probe: $\text{SimIn}(\text{NIU}, \mathcal{F}) := \{\bar{v}_1, \bar{\rho}_1\}$, however since each input variable is in a different shared input this simulator fits the definition of 2-NI in Definition 3, and we can set $\text{SimOut}(\text{NIU}, \{v_1, \rho_1\}) := v_1 + \rho_1$. It is obvious that in this case $(\mathcal{L}, \text{out}_{\text{unmasked}}) = (v_1 + \rho_1, v + \rho_1 + \dots + \rho_d) = (\text{SimOut}(\text{NIU}, \{v_1, \rho_1\}), \text{out}_{\text{unmasked}})$.

We will now first formalize the $(d - 1)$ -NIU notion, introduced in [EEN⁺24], in Definitions 7 and 8. Using the formalism of Section 2.2 we can then state and prove composition properties in Lemma 4, which are straightforward though never made explicit in [EEN⁺24]. Finally we can prove the core simulatability property of Lemma 5 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 [EEN⁺24], it was actually never proven.

Definition 7 (Non Interference with Unshared input [EEN⁺24]). *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 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 \mathcal{F} such that $|\mathcal{F}| \leq d - 1$ can be perfectly simulated (See Definition 2) by a simulator $(\text{SimIn}, \text{SimOut})$ such that $\text{SimIn}(G, \mathcal{F})$ outputs a set $\bar{\mathcal{X}}' \cup \bar{\mathcal{U}}$ of at most $d - 1$ shares of each shared input ($\bar{\mathcal{X}}'$) and each unshared input ($\bar{\mathcal{U}}$).

Definition 8 (Strong Non Interference with Unshared input [EEN⁺24]).

A gadget is said $(d - 1)$ -strongly-non-interfering with unshared inputs (written $(d - 1)$ -sNIU for short) iff any set \mathcal{F} of at most $d - 1 = d_{\text{int}} + d_{\text{out}}$ probes where d_{int} are made on internal data and d_{out} are made on the outputs can be simulated as in Definition 7 with d_{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.

Lemma 4 (Composability of NIU and sNIU gadgets). *A well-formed algorithm is sNIU if it is NIU and its output sharings are issued from a sNIU gadget.*⁵

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. *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 X and any probes \mathcal{F} (with $|\mathcal{F}| \leq d - 1$), the distribution of $(\text{out}_{\text{unmasked}}, \text{SimOut}(P, (X', \mathcal{U}')))$ and $(\text{out}_{\text{unmasked}}, \mathcal{L})$ over the randomness \mathcal{U} and the random coins of P and SimOut are identical, where $(\text{out}_{\text{masked}}, \text{out}_{\text{unmasked}}, \mathcal{L}) \leftarrow \text{ExecObs}(P, \mathcal{F}, X)$ and $(X', \mathcal{U}') \leftarrow \text{SimIn}(P, \mathcal{F})$.*

Proof. We will fix the input X and not \mathcal{D} the distribution from which \mathcal{U} is sampled. \mathcal{L} and $\text{out}_{\text{unmasked}}$ are random variables over the choice of \mathcal{U} and the random coins of P which we will note rc_P , and $\text{SimOut}(P, (X', \mathcal{U}'))$ 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 \mathcal{L} and $\text{SimOut}(P, (X', \mathcal{U}'))$ are identical, i.e. for any possible leakage Λ we have:

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

⁵ We thank Katharina Boudgoust, Loïc Demange, Laurent Imbert, Loïc Masure, Camille Mutschler and Thomas Roche for noting an incorrect statement in a previous version of this paper.

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

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

which is the desired result. \square

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 $\mathcal{L}_{\text{KeyGen}}$ and $\mathcal{L}_{\text{Sign}}^{(i)}$ in the EUF-CMA security game in the probing model (cf. Figure 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.

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 **h**).

Name	Property	Proof reference
$\times \mathbf{A}$ and Line 13 of Algorithm 2	NI	\mathbb{Z}_q -linear
Refresh (Algorithm 7)	sNI	[BCPZ16, Mat21, GPRV21]
ZeroEncoding (Algorithm 8)	sNI	[Mat21]
Decode (Algorithm 6)	NI	[BCRT23, Cor. 5] and [CGMZ23, Cor. 1]
AddRepNoise (Algorithm 5)	sNIU	Proved in Section 6.2, Lemma 6

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 $a \leftarrow a + \text{SU}(u, T)$ for each coefficient a , for $T = d \cdot \text{rep}$. The internal use of **Refresh** operations does not affect this behavior but is meant to offer some resilience to probing adversaries.

Without **Refresh**, a viable strategy would be to probe individual shares of $\llbracket a \rrbracket$ at the start and at the end of **AddRepNoise**, allowing to learn the sum b of $\text{rep} \cdot (d - 1)/2$ small uniform errors. The conditional distribution of the additive noise (conditioned on the $d - 1$ probed values) is now $b + \text{SU}(u, T - (d - 1) \cdot \text{rep}/2)$. 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 $b + \text{SU}(u, T - (d - 1))$, where the adversary learns b but knows nothing about the realization of $\text{SU}(u, T - (d - 1))$.

While **AddRepNoise** performs operations share by share, the underlying distributions are not uniform. Short noise values are added together and as stated in the introduction the knowledge of any intermediate short value 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 **AddRepNoise_{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 that **AddRepNoise_{ER}** is sNIU. A similar result was proven in [EEN⁺24] 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. *The **AddRepNoise_{ER}** gadget is $(d-1)$ -sNIU.*

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

Step 1: MiniAddRepNoise. We first show that any probe inside **MiniAddRepNoise** can be perfectly simulated (see Definition 2) with $\rho_{i, i_{\text{rep}}, j}$ and the input \mathbf{v}_j , where (i, i_{rep}, j) corresponds to the targeted loop. Indeed, let p be a probe inside **MiniAddRepNoise**. The description of this probe necessarily includes (i, i_{rep}, j) to specify the involved loop. The intermediate value targeted by p can be

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

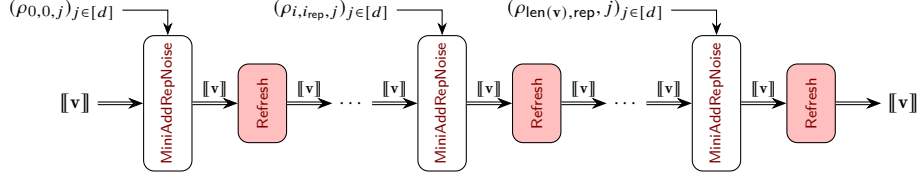


Fig. 5: Structure of $\text{AddRepNoise}_{\text{ER}}$ (using Algorithm 10). A gadget proven sNI is noted $\boxed{\text{gadget}}$. The gadgets with no proven property are noted $\boxed{\text{gadget}}$. Single arrows (\rightarrow) and double arrows (\Rightarrow) represent plain and masked values, respectively.

Algorithm 9 $\text{AddRepNoise}_{\text{ER}}(\llbracket \mathbf{v} \rrbracket, (\rho_{i,i_{\text{rep}},j})) \rightarrow \llbracket \mathbf{v}' \rrbracket$, w/ partial explicit randomness

Input: Masked vector $\llbracket \mathbf{v} \rrbracket = (\mathbf{v}_j)_{j \in [d]} = (v_{i,j})_{i \in [\text{len}(\mathbf{v})], j \in [d]}$.

Input: Randomness $(\rho_{i,i_{\text{rep}},j})_{i \in [\text{len}(\mathbf{v})], i_{\text{rep}} \in [\text{rep}], j \in [d]}$

Output: Updated $\llbracket \mathbf{v} \rrbracket$ with $\text{SU}(u, d \cdot \text{rep})$ distribution added to each coefficient of \mathbf{v} .

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

Algorithm 10 $\text{MiniAddRepNoise}(\llbracket \mathbf{v} \rrbracket, i_{\text{rep}}, (\rho_{i,i_{\text{rep}},j})) \rightarrow \llbracket \mathbf{v}' \rrbracket$

Input: Masked vector $\llbracket \mathbf{v} \rrbracket$, index $i_{\text{rep}} \in [\text{rep}]$, randomness $(\rho_{i,i_{\text{rep}},j})_{i \in [\text{len}(\mathbf{v})], j \in [d]}$

Output: Updated $\llbracket \mathbf{v} \rrbracket$.

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

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

Step 2: AddRepNoise_{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 $\rho_{i,i_{\text{rep}},j}$ and \mathbf{v}_j . Let $\tilde{\mathcal{F}}$ be the given set of at most $d - 1$ probes in AddRepNoise . We decompose $\tilde{\mathcal{F}}$ as follows.

- Let $\delta_{\text{MiniAddRepNoise}}^{i,i_{\text{rep}}}$ be the number intermediate variables that are probed inside the MiniAddRepNoise gadget of the loop with indexes i, i_{rep} .
- Let $\delta_{\text{Refresh}}^{i,i_{\text{rep}}}$ be the number intermediate variables that are probed inside the Refresh gadget of the loop with indexes i, i_{rep} .

By definition,

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

Going from right to left in Figure 5, we first consider the last **Refresh** of the last loop (where $i = \text{len}(\mathbf{v})$ and $i_{\text{rep}} = \text{rep}$). Thanks to the sNI property of the last **Refresh** algorithm, all the $\delta_{\text{Refresh}}^{\text{len}(\mathbf{v}), \text{rep}}$ probes can be perfectly simulated from $\delta_{\text{Refresh}}^{\text{len}(\mathbf{v}), \text{rep}}$ 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:

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

The set of $\bar{\mathcal{F}}'_{\text{len}(\mathbf{v}), \text{rep}}$ 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 $\delta_{\text{Refresh}}^{\text{len}(\mathbf{v})-1, \text{rep}-1}$ intermediate variables probed inside the penultimate **Refresh** algorithm. In conclusion, the $\bar{\mathcal{F}}'_{\text{len}(\mathbf{v}), \text{rep}}$ probes can be simulated from uniform random.

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

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

We define $\bar{\mathcal{U}} = \bar{\mathcal{F}}_{0,0} \cup \dots \cup \bar{\mathcal{F}}_{\text{len}(\mathbf{v}), \text{rep}}$ and $\bar{\mathcal{X}}' = \bar{\mathcal{F}}'_{0,0}$. Thanks to Eq. (9) and Lemma 1, we have shown that **AddRepNoise_{ER}** is (d-1)-sNIU. \square

6.3 Security Property of **KeyGen** and **Sign**

Now that **AddRepNoise_{ER}** is proved, one needs to derive the security of the key generation and signature algorithms with a composition proof. Let us first introduce **KeyGen_{ER}** and **Sign_{ER}**, simple modifications of **KeyGen** and **Sign** algorithms where the small uniform randomness is provided as input. **KeyGen_{ER}** is formally described in Algorithm 11. The formal description of **Sign_{ER}** is deferred to the appendix (Algorithm 16).

Lemma 7. *The algorithm **KeyGen_{ER}** is (d - 1)-NIU.*

Lemma 8. *The algorithm **Sign_{ER}** is (d - 1)-NIU.*

We now prove Lemma 7. The proof of Lemma 8 proceeds in a similar fashion and is deferred to Appendix D.

Algorithm 11 $\text{KeyGen}_{\text{ER}}((\rho_{i,i_{\text{rep}},j}^{(0)}), (\rho_{i,i_{\text{rep}},j}^{(1)})) \rightarrow (\text{vk}, \text{sk})$

▷ KeyGen with explicit randomness for AddRepNoise

Input: Randomness $(\rho_{i,i_{\text{rep}},j}^{(0)})_{i \in [\text{len}(\mathbf{v})], i_{\text{rep}} \in [\text{rep}], j \in [d]}$, $(\rho_{i,i_{\text{rep}},j}^{(1)})_{i \in [\text{len}(\mathbf{v})], i_{\text{rep}} \in [\text{rep}], j \in [d]}$

Output: Keypair vk, sk

1: $\text{seed} \leftarrow \{0, 1\}^k$; $\mathbf{A} := \text{ExpandA}(\text{seed})$

2: $\llbracket \mathbf{s} \rrbracket \leftarrow \ell \times \text{ZeroEncoding}(d)$

3: $\llbracket \mathbf{s} \rrbracket \leftarrow \text{AddRepNoise}_{\text{ER}}(\llbracket \mathbf{s} \rrbracket, u_{\mathbf{t}}, \text{rep}, (\rho_{i,i_{\text{rep}},j}^{(0)}))$ ▷ Partially derandomized
 AddRepNoise.

4: $\llbracket \mathbf{t} \rrbracket := \mathbf{A} \cdot \llbracket \mathbf{s} \rrbracket$

5: $\llbracket \mathbf{t} \rrbracket \leftarrow \text{AddRepNoise}_{\text{ER}}(\llbracket \mathbf{t} \rrbracket, u_{\mathbf{t}}, \text{rep}, (\rho_{i,i_{\text{rep}},j}^{(1)}))$ ▷ Partially derandomized
 AddRepNoise.

6: $\mathbf{t} := \text{Decode}(\llbracket \mathbf{t} \rrbracket)$

7: $\mathbf{t} := \lfloor \mathbf{t} \rfloor_{v_{\mathbf{t}}}$

8: **return** $(\text{vk} := (\text{seed}, \mathbf{t}), \text{sk} := (\text{vk}, \llbracket \mathbf{s} \rrbracket))$

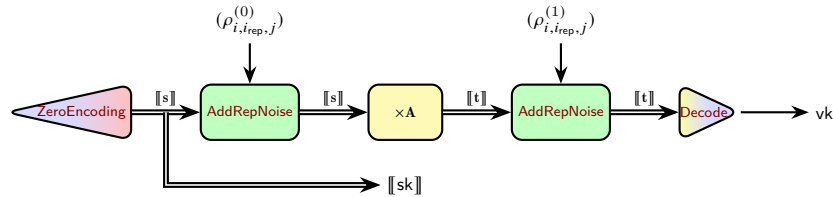


Fig. 6: Structure of KeyGen (Algorithm 11). Gadgets proven NI (resp. sNIU) is noted gadget (resp. gadget). Triangular gadgets either start from a masked input and output an unmasked value, or the other way around.

Proof (Lemma 7). Let us decompose the key generation as a succession of gadgets. The gadgets may be represented as in Figure 6. We assume the respective NI/sNI/sNIU properties of each gadget as presented in Table 2.

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

Let us decompose the set $\bar{\mathcal{F}}$ of at most $d - 1$ probes in $\text{KeyGen}_{\text{ER}}$ 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:

- δ_0 the number of intermediate variables probed in Line 6 (final **Decode** gadget);
- δ_1 the number of intermediate variables probed in Line 5 (second **AddRepNoise_{ER}**);
- δ_2 the number of intermediate variables probed in Line 4 (multiplication with **A**);
- δ_3 the number of intermediate variables probed in Line 3 (first **AddRepNoise_{ER}**);
- δ_4 the number of intermediate variables probed in Line 2 (**ZeroEncoding**);

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

The proof is similar to a standard composition proof. Thanks to the NI property of the **Decode** gadget, all the δ_0 intermediate variables can be perfectly simulated (see Definition 2) with at most δ_0 shares of $\llbracket \mathbf{t} \rrbracket$. Since the second **AddRepNoise_{ER}** is $d - 1$ -sNIU, the $\delta_1 + \delta_0$ intermediate variables observed during **Decode** and the last **AddRepNoise_{ER}** may be perfectly simulated with δ_1 shares of $\llbracket t \rrbracket$ (the output of the $\times \mathbf{A}$ operation) and δ_1 shares of $(\rho_{i, i_{\text{rep}}}^{(1)})$. We note $\bar{\mathcal{F}}_1$ this set. Note that δ_0 has been 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 **AddRepNoise_{ER}** can be perfectly simulated with at most δ_3 shares of $\llbracket s \rrbracket$ (which are also the output of **ZeroEncoding**) and at most δ_3 shares of $(\rho_{i, i_{\text{rep}}}^{(0)})$. We note $\bar{\mathcal{F}}_0$ this sets. In addition, the δ_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 $\bar{\mathcal{F}}$ may be perfectly simulated from :

- the set $\bar{\mathcal{F}}_0$ containing at most δ_3 shares of $(\rho_{i, i_{\text{rep}}}^{(0)})$
- the sets $\bar{\mathcal{F}}_1$ containing at most δ_1 shares of $(\rho_{i, i_{\text{rep}}}^{(1)})$

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

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 Figure 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. Section 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. Section 4), and consists the computation step.

7.1 Description of a Non-Masked Small Raccoon

We first formally define a *non-masked* and simplified variant of Raccoon, which we call *small Raccoon*. This is depicted in Figure 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}_∞ and \bar{B}_2 , whose formal definition appears in Appendix E.

7.2 EUF-CMA Security of Small Raccoon \Rightarrow 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 1. *For any PPT adversary \mathcal{A} against the EUF-CMA security on Raccoon in the $(d-1)$ -probing model with time T and advantage ε , there exists a PPT adversary \mathcal{B} against the EUF-CMA security on small Raccoon (cf. Figure 7) with time $O(T)$ and advantage:*

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

Above, Q_H and Q_S denote the number of random oracle queries and signing queries performed by \mathcal{A} .

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

Hybrid₀: This hybrid corresponds to real the EUF-CMA security game in the $(d-1)$ -probing model (cf. Figure 3).

Hybrid₁: In this hybrid we replace **KeyGen** with **KeyGen_{ER}** and **Sign** with **Sign_{ER}**, in which all randomnesses are sampled prior to running the algorithm. Since the algorithms are functionnaly identical the advantage is unchanged.

Hybrid₂: This hybrid corresponds to Figure 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 Lemmas 7 and 8.

More precisely there is a first PPT simulator ($\text{SimIn}_{\text{KeyGen}}, \text{SimOut}_{\text{KeyGen}}$) such that for any probe set $|\bar{\mathcal{F}}_{\text{KeyGen}}| \leq t$ in $\text{KeyGen}(1^\kappa)$, all probes in $\bar{\mathcal{F}}' := (\bar{\mathcal{F}}'_s, \bar{\mathcal{F}}'_e) := \text{SimIn}_{\text{KeyGen}}(\bar{\mathcal{F}}_{\text{KeyGen}})$ are of the form $\bar{\rho}_{s,i,i_{\text{rep}},j} \in \bar{\mathcal{F}}'_s$ for some $(i, i_{\text{rep}}, j) \in [\ell, \text{rep}, d]$, and $\bar{\rho}_{e,i,i_{\text{rep}},j} \in \bar{\mathcal{F}}'_e$ for some $(i, i_{\text{rep}}, j) \in [k, \text{rep}, d]$ (note that the variable names $\bar{\rho}$ are also indexed by the **AddRepNoise** gadget to which they belong to ensure unique namings), and $\max(|\bar{\mathcal{F}}'_s|, |\bar{\mathcal{F}}'_e|) \leq d-1$. Using Lemma 5 we have that $(\text{vk}, \text{SimOut}(\text{KeyGen}_{\text{ER}}, \mathcal{F}'))$ follows the same distribution as (vk, \mathcal{L}) , where $(\text{sk}, \text{vk}, \mathcal{L}) \leftarrow \text{ExecObs}(\bar{\mathcal{F}}_{\text{KeyGen}}, \text{KeyGen}_{\text{ER}}, 1^\lambda)$. Similarly there is a second PPT simulator ($\text{SimIn}_{\text{Sign}}, \text{SimOut}_{\text{Sign}}$) such that for any message msg , masked secret key $\llbracket \text{sk} \rrbracket$, and probe set $|\bar{\mathcal{F}}_{\text{Sign}}| \leq t$

<p>Algorithm 12 $\text{KeyGen}_{\text{SMALL}}(\emptyset) \rightarrow (\text{vk}, \text{sk})$</p> <hr/> <p>Output: Keypair vk, sk</p> <p>1: $\text{seed} \leftarrow \{0, 1\}^k$</p> <p>2: $\mathbf{A} := \text{ExpandA}(\text{seed})$</p> <p>3: $(\mathbf{s}, \mathbf{e}) \leftarrow \text{RSU}(u_t, d(\text{rep} - 1) + 1)^\ell \times \text{RSU}(u_t, d(\text{rep} - 1) + 1)^k$</p> <p>4: $\mathbf{t} := \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ \triangleright No rounding of $\mathbf{t} \in \mathcal{R}_q^k$</p> <p>5: return $(\text{vk} := (\text{seed}, \mathbf{t}), \text{sk} = (\text{vk}, \mathbf{s}))$</p> <hr/> <p>Algorithm 13 $\text{Sign}_{\text{SMALL}}(\text{sk}, \text{msg}) \rightarrow \text{sig}$</p> <hr/> <p>Input: Secret signing key $\text{sk} = (\text{vk}, \mathbf{s})$, message $\text{msg} \in \{0, 1\}^*$.</p> <p>Output: Signature $\text{sig} = (c_{\text{hash}}, \mathbf{h}, \mathbf{z})$ of msg under sk.</p> <p>1: $\mu := \text{H}(\text{H}(\text{vk}) \parallel \text{msg})$</p> <p>2: $(\mathbf{r}, \mathbf{e}') \leftarrow \text{RSU}(u_w, d(\text{rep} - 1) + 1)^\ell \times \text{RSU}(u_w, d(\text{rep} - 1) + 1)^k$</p> <p>3: $\mathbf{w} = \mathbf{A} \cdot \mathbf{r} + \mathbf{e}'$ \triangleright No rounding of $\mathbf{w} \in \mathcal{R}_q^k$</p> <p>4: $c_{\text{hash}} := \text{ChalHash}(\mathbf{w}, \mu)$ \triangleright ChalHash redefined to take $\mathbf{w} \in \mathcal{R}_q^k$</p> <p>5: $c_{\text{poly}} := \text{ChalPoly}(c_{\text{hash}})$</p> <p>6: $\mathbf{z} := c_{\text{poly}} \cdot \mathbf{s} + \mathbf{r}$</p> <p>7: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - c_{\text{poly}} \cdot \mathbf{t}$ \triangleright No need to lift \mathbf{t} anymore</p> <p>8: $\mathbf{h} := \mathbf{w} - \mathbf{y}$ \triangleright Hint \mathbf{h} now defined over \mathcal{R}_q^k</p> <p>9: $\text{sig} := (c_{\text{hash}}, \mathbf{z}, \mathbf{h})$</p> <p>10: if $(\ (\mathbf{z}, \mathbf{h})\ _\infty > \bar{B}_\infty)$ or $(\ (\mathbf{z}, \mathbf{h})\ _2 > \bar{B}_2)$ goto Line 2 \triangleright Check smaller bound</p> <p>11: return sig</p> <hr/> <p>Algorithm 14 $\text{Verify}_{\text{SMALL}}(\text{sig}, \text{msg}, \text{vk}) \rightarrow \{\text{OK or FAIL}\}$</p> <hr/> <p>Input: Signature $\text{sig} = (c_{\text{hash}}, \mathbf{h}, \mathbf{z}) := \text{sig}$.</p> <p>Output: Signature validity: OK (accept) or FAIL (reject).</p> <p>1: if $(\ (\mathbf{z}, \mathbf{h})\ _\infty > \bar{B}_\infty)$ or $(\ (\mathbf{z}, \mathbf{h})\ _2 > \bar{B}_2)$ return FAIL else return OK</p> <p>2: $\mu := \text{H}(\text{H}(\text{vk}) \parallel \text{msg}); \mathbf{A} := \text{ExpandA}(\text{seed})$</p> <p>3: $c_{\text{poly}} := \text{ChalPoly}(c_{\text{hash}})$</p> <p>4: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - c_{\text{poly}} \cdot \mathbf{t}$</p> <p>5: $\mathbf{w} := \mathbf{y} + \mathbf{h}$</p> <p>6: $c'_{\text{hash}} := \text{ChalHash}(\mathbf{w}', \mu)$</p> <p>7: if $c_{\text{hash}} \neq c'_{\text{hash}}$ return FAIL</p> <p>8: return OK</p>
--

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 $c_{\text{poly}} \cdot \mathbf{e} + \mathbf{e}'$ without rounding errors.

in $\text{Sign}(\llbracket \text{sk} \rrbracket, \text{msg})$, all probes in $\bar{\mathcal{F}}' := (\bar{\mathcal{F}}'_r, \bar{\mathcal{F}}'_{e'}, \bar{\mathcal{F}}'_{\text{sk}}) := \text{SimIn}_{\text{Sign}}(\bar{\mathcal{F}}_{\text{Sign}})$ are of the form $\bar{\rho}_{r,i,i_{\text{rep}},j} \in \bar{\mathcal{F}}'_r$ for some $(i, i_{\text{rep}}, j) \in [\ell, \text{rep}, d]$, $\bar{\rho}_{e',i,i_{\text{rep}},j} \in \bar{\mathcal{F}}'_{e'}$ for some $(i, i_{\text{rep}}, j) \in [k, \text{rep}, d]$, and $\bar{s}_i \in \bar{\mathcal{F}}'_{\text{sk}}$ for some $i \in [d]$, and $\max(|\bar{\mathcal{F}}'_r|, |\bar{\mathcal{F}}'_{e'}|, |\bar{\mathcal{F}}'_{\text{sk}}|) \leq t$. We also have that $(\text{sig}, \text{SimOut}(\text{ExecObs}(\bar{\mathcal{F}}', \text{Sign}, 1^\lambda)))$ follows the same distribution as $\text{ExecObs}(\bar{\mathcal{F}}_{\text{Sign}}, \text{Sign}, 1^\lambda)$. Using Lemma 5 we have that $\text{SimOut}(\text{Sign}_{\text{ER}}, \bar{\mathcal{F}}')$ follows the same distribution as $(\text{sig}, \mathcal{L})$,

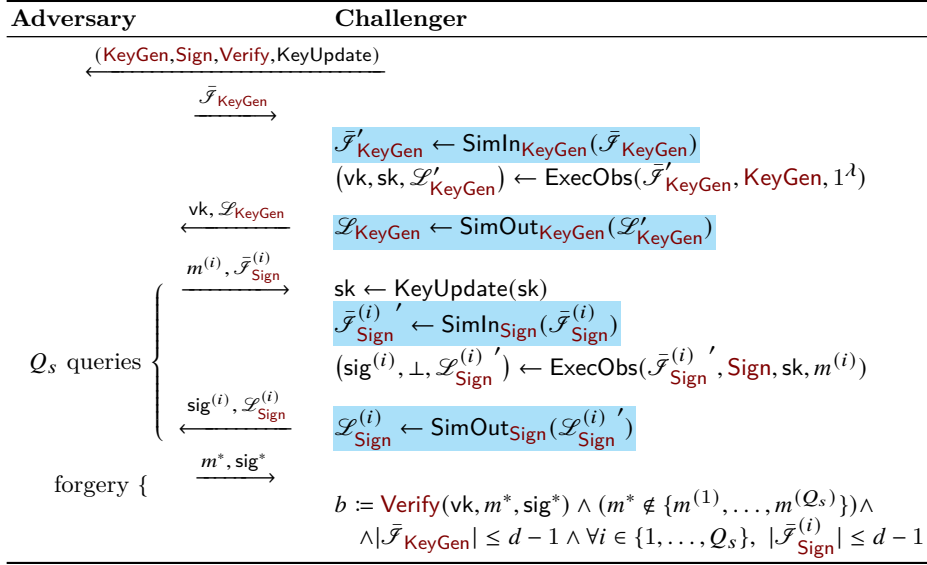


Fig. 8: Hybrid₂: The NIU properties proven in Lemma 7 ensure the existence of two PPT simulators ($\text{SimIn}_{\text{KeyGen}}, \text{SimOut}_{\text{KeyGen}}$) and ($\text{SimIn}_{\text{Sign}}, \text{SimOut}_{\text{Sign}}$). This ensures all probes can be moved to the randomness in the **AddRepNoise** gadgets in **KeyGen** and **Sign**. Differences with the EUF-CMA security game in the $(d-1)$ -probing model (Figure 3) are highlighted.

where $(\text{sig}, \mathcal{L}) \leftarrow \text{ExecObs}(\bar{\mathcal{F}}_{\text{Sign}}, \text{Sign}_{\text{ER}}, \text{msg})$. Thus the two hybrids are identical.

Hybrid₃: This hybrid corresponds to Figure 9, in which the algorithms $\text{ExecObs}(\bar{\mathcal{F}}, \text{KeyGen}, 1^\kappa)$ and $\text{ExecObs}(\bar{\mathcal{F}}, \text{Sign}, \text{sk}, \text{msg})$ are replaced by $\text{KeyGen}_{\mathcal{F}}(1^\kappa, \bar{\mathcal{F}})$ and $\text{Sign}_{\mathcal{F}}(\text{sk}, \text{msg}, \bar{\mathcal{F}})$, respectively. The former is presented in Algorithm 15. The latter is defined analogously and deferred to Appendix E.1, Algorithm 17 due to page limitations. Observe that since $\text{ExecObs}(\bar{\mathcal{F}}, \text{KeyGen}, 1^\kappa)$ outputs the same output as $\text{KeyGen}(1^\kappa)$ as well as the value of the variables at indices $\bar{\mathcal{F}}$, any algorithm that outputs the same distribution is semantically identical. Since the variables in $\bar{\mathcal{F}}$ are now restricted to the randomness used in **AddRepNoise** it is clear that the algorithm $\text{KeyGen}_{\mathcal{F}}$ outputs the same distribution. The same argument goes for $\text{ExecObs}(\bar{\mathcal{F}}, \text{Sign}, \text{sk}, \text{msg})$. Hence, the two hybrids are identical.

Hybrid₄: This hybrid corresponds to Figure 10, in which the challenger artificially extends the set of probes queried to the key generation and signing algorithm. More specifically, we define Extend so that for any $\rho_{s,i,i_{\text{rep}},j} \in \bar{\mathcal{F}}_s$, all variables $\rho_{s,i',i_{\text{rep}},j}$ for $i' \in [\ell]$ are in $\text{Extend}(\bar{\mathcal{F}}_s)$ (same for $\text{Extend}(\bar{\mathcal{F}}_e)$, $\text{Extend}(\bar{\mathcal{F}}_r)$, $\text{Extend}(\bar{\mathcal{F}}_{e'})$). Conversely $\text{Collapse}(\mathcal{L}'_s)$ discards the values of any variables that are in $\bar{\mathcal{F}}_r$ but not $\bar{\mathcal{F}}'_r$. Clearly, this does not modify the

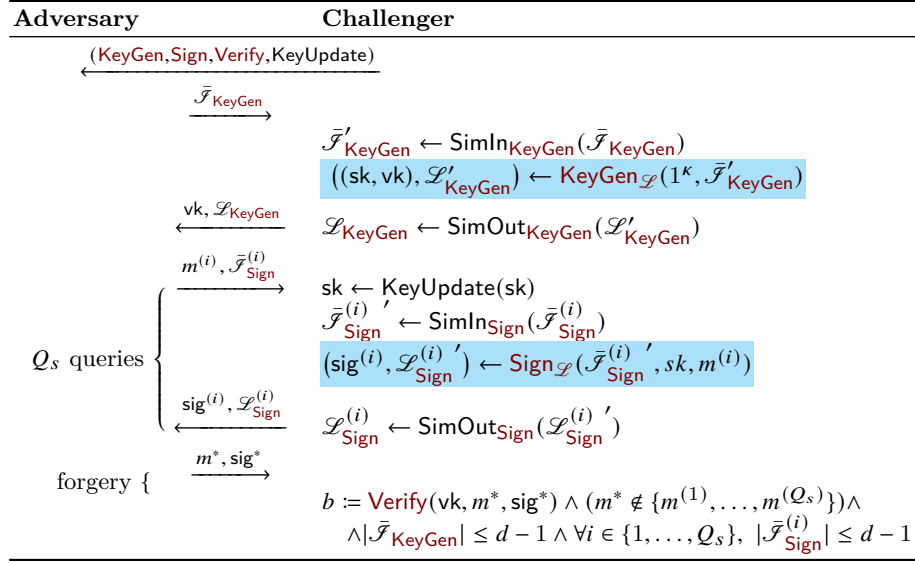


Fig. 9: Hybrid₃: We replace the ExecObs calls with the functionally identical algorithms $\text{KeyGen}_{\mathcal{L}}$ (cf. Algorithm 15) and $\text{Sign}_{\mathcal{L}}$ (cf. full version).

Algorithm 15 $\text{KeyGen}_{\mathcal{L}}(1^\kappa, \bar{\mathcal{F}}) \rightarrow (\text{vk}, \text{sk}, \mathcal{L})$

Input: Probe set $\mathcal{F} = (\mathcal{F}_s, \mathcal{F}_e), \bar{\mathcal{F}}_s \subset \{\bar{\rho}_{s,i,i_{\text{rep}},j}; (i, i_{\text{rep}}, j) \in [\ell] \times [\text{rep}] \times [d]\},$

$\bar{\mathcal{F}}_e \subset \{\bar{\rho}_{e,i,i_{\text{rep}},j}; (i, i_{\text{rep}}, j) \in [k] \times [\text{rep}] \times [d]\}$

Output: Keypair vk, sk and Leakage \mathcal{L}

- 1: $\text{seed} \leftarrow \{0, 1\}^\kappa; \mathbf{A} := \text{ExpandA}(\text{seed})$
 - 2: $\llbracket \mathbf{s} \rrbracket = (s_1, \dots, s_d) := (0, \dots, 0) \in (\mathcal{R}_d^\ell)^d$
 - 3: **for** $(i, i_{\text{rep}}, j) \in [\ell] \times [\text{rep}] \times [d]$ **do**
 - 4: $\rho_{s,i,i_{\text{rep}},j} \leftarrow \text{RSU}(u, 1)$
 - 5: $s_{j,i} \leftarrow s_{j,i} + \rho_{s,i,i_{\text{rep}},j}$
 - 6: $\llbracket \mathbf{t} \rrbracket := \mathbf{A} \cdot \llbracket \mathbf{s} \rrbracket \in (\mathcal{R}_d^k)^d$
 - 7: **for** $(i, i_{\text{rep}}, j) \in [k] \times [\text{rep}] \times [d]$ **do**
 - 8: $\rho_{e,i,i_{\text{rep}},j} \leftarrow \text{RSU}(u, 1)$
 - 9: $t_{j,i} \leftarrow t_{j,i} + \rho_{e,i,i_{\text{rep}},j}$
 - 10: $\mathbf{t} := \text{Decode}(\llbracket \mathbf{t} \rrbracket)$
 - 11: $\mathbf{t} := \lfloor \mathbf{t} \rfloor_{\mathcal{V}_t}$
 - 12: $\mathcal{L} := \left\{ (\rho_{s,i,i_{\text{rep}},j}, \rho_{e,i',i'_{\text{rep}},j'}) \right\}_{(\bar{\rho}_{s,i,i_{\text{rep}},j}, \bar{\rho}_{e,i',i'_{\text{rep}},j'}) \in \bar{\mathcal{F}}}$
 - 13: **return** $(\text{vk} := (\text{seed}, \mathbf{t}), \text{sk} := (\text{vk}, \llbracket \mathbf{s} \rrbracket), \mathcal{L})$
-

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

Adversary	Challenger
	$(\text{KeyGen}, \text{Sign}, \text{Verify}, \text{KeyUpdate})$
	$\xrightarrow{\bar{\mathcal{F}}_{\text{KeyGen}}}$
	$(\bar{\mathcal{F}}'_s, \bar{\mathcal{F}}'_e) := \bar{\mathcal{F}}'_{\text{KeyGen}} \leftarrow \text{SimIn}_{\text{KeyGen}}(\bar{\mathcal{F}}_{\text{KeyGen}})$ $\bar{\mathcal{F}}'_s = \text{Extend}(\bar{\mathcal{F}}'_s)$ $\bar{\mathcal{F}}'_e = \text{Extend}(\bar{\mathcal{F}}'_e)$ $((\text{sk}, \text{vk}), (\mathcal{L}'_s, \mathcal{L}'_e)) \leftarrow \text{KeyGen}_{\mathcal{G}}((\bar{\mathcal{F}}'_s, \bar{\mathcal{F}}'_e), 1^\lambda)$
	$\xleftarrow{\text{vk}, \mathcal{L}_{\text{KeyGen}}}$
	$\mathcal{L}_{\text{KeyGen}} \leftarrow \text{SimOut}_{\text{KeyGen}}(\text{Collapse}(\mathcal{L}'_s), \text{Collapse}(\mathcal{L}'_e))$
Q_s queries	$\xrightarrow{m^{(i)}, \bar{\mathcal{F}}_{\text{Sign}}^{(i)}}$ $\text{sk} \leftarrow \text{KeyUpdate}(\text{sk})$ $(\bar{\mathcal{F}}'_r, \bar{\mathcal{F}}'_{e'}, \bar{\mathcal{F}}'_{\text{sk}}) := \bar{\mathcal{F}}'^{(i)}_{\text{Sign}} \leftarrow \text{SimIn}_{\text{Sign}}(\bar{\mathcal{F}}_{\text{Sign}}^{(i)})$ $\bar{\mathcal{F}}'_r = \text{Extend}(\bar{\mathcal{F}}'_r)$ $\bar{\mathcal{F}}'_{e'} = \text{Extend}(\bar{\mathcal{F}}'_{e'})$ $(\text{sig}^{(i)}, (\mathcal{L}'_r, \mathcal{L}'_{e'}, \mathcal{L}'_{\text{sk}})) \leftarrow \text{Sign}_{\mathcal{G}}((\bar{\mathcal{F}}'_r, \bar{\mathcal{F}}'_{e'}, \bar{\mathcal{F}}'_{\text{sk}}), \text{sk}, m^{(i)})$
	$\xleftarrow{\text{sig}^{(i)}, \mathcal{L}_{\text{Sign}}^{(i)}}$
	$\mathcal{L}_{\text{Sign}}^{(i)} \leftarrow \text{SimOut}_{\text{Sign}}(\text{Collapse}(\mathcal{L}'_r), \text{Collapse}(\mathcal{L}'_{e'}), \mathcal{L}'_{\text{sk}})$
Forgery{	$\xrightarrow{m^*, \text{sig}^*}$ $b := \text{Verify}(\text{vk}, m^*, \text{sig}^*) \wedge (m^* \notin \{m^{(1)}, \dots, m^{(Q_s)}\}) \wedge$ $\wedge \bar{\mathcal{F}}_{\text{KeyGen}} \leq d-1 \wedge \forall i \in \{1, \dots, Q_s\}, \bar{\mathcal{F}}_{\text{Sign}}^{(i)} \leq d-1$

Fig. 10: Hybrid_4 : In this game, for any variable name $\bar{\rho}_{s,i,i_{\text{rep}},j}$ the challenger artificially leaks all variables $\rho_{s,i,i_{\text{rep}},j'}$ 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.

Lastly, we prove that for any PPT adversary \mathcal{A} against the game described in Hybrid_4 (cf. Figure 10), we can construct an adversary \mathcal{B} against the standard EUF-CMA security of small Raccoon in Figure 7. We defer the formal proof to Appendix E.2. At a high level a challenger can simulate queries from $\text{KeyGen}_{\mathcal{G}}$ by querying the public key $\bar{\mathbf{t}}$ from the oracle for $\text{KeyGen}_{\text{Small}}$ and artificially sampling additional noises $(\bar{\mathbf{s}}, \bar{\mathbf{e}})$ as the sum of $d-1$ small uniforms and outputting the public key $\mathbf{t} := \lfloor \bar{\mathbf{t}} + \mathbf{A}\bar{\mathbf{s}} + \bar{\mathbf{e}} \rfloor_{v_t}$ which will be distributed exactly as a public key for $\text{KeyGen}_{\mathcal{G}}$. Similarly the a signature from $\text{Sign}_{\text{Small}}$ can be mapped to a signature for $\text{Sign}_{\mathcal{G}}$ by sampling the appropriate sums of uniform $(\bar{\mathbf{r}}, \bar{\mathbf{e}}')$ and setting $\mathbf{w} = \lfloor \bar{\mathbf{w}} + \mathbf{A}\bar{\mathbf{r}} + \bar{\mathbf{e}}' \rfloor_{v_w}$. Finally we show that a forgery for $\text{Sign}_{\mathcal{G}}$ can be mapped to a forgery for $\text{Sign}_{\text{Small}}$.

This completes the proof of the main theorem.

7.3 MLWE + SelfTargetMSIS \Rightarrow EUF-CMA Security of Small Raccoon

It remains to prove that small Raccoon in Figure 7 is (standard) EUF-CMA secure. This is established in the following theorem. A more formal statement

with a concrete security loss is provided in Appendix F. The detailed security proof along with a candidate asymptotic parameter selection is deferred to Appendix F. The main technical contribution of the proof consists of relying on the *smooth* Rényi divergence introduced in Section 4.

Theorem 2 (Informal). *The small Raccoon in Figure 7 is EUF-CMA secure under the $\text{MLWE}_{q,\ell,k,\text{SU}(u_t,d,\text{rep})}$ and $\text{SelfTargetMSIS}_{q,\ell+1,k,C,\nu_w,\beta}$ assumptions.*

8 Concrete Instantiation

Looking at Theorem 3, it is clear that the security bottlenecks are the hardness of MLWE, of SelfTargetMSIS, and the smooth Rényi divergence (ϵ_{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 $\nu\mathbf{k}$ and sig).

- Our analysis of MLWE and SelfTargetMSIS is fairly standard. We rely on the lattice estimator [APS15] for the concrete analysis of MLWE. Following the Dilithium methodology [LDK⁺22, §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 [CPS⁺20, §4.2].
- For the smooth Rényi divergence, one could use Lemma 3 for a provable bound. However, it is not tight so we opt instead to use Conjecture 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: Parameters for Raccoon-128, NIST Post-Quantum security strength category 1. For all Raccoon-128 masking orders, we fix: $\kappa = 128$, $Q_s = 2^{53}$, $q = (2^{24} - 2^{18} + 1) \cdot (2^{25} - 2^{18} + 1)$, $n = 512$, $k = 5$, $\ell = 4$, $\nu_t = 42$, $\nu_w = 44$, $\omega = 19$, $2^{-64}B_2^2 = 14656575897$, $B_\infty = 41954689765971$.

Parameter	Raccoon-128	128-2	128-4	128-8	128-16	128-32
$ \text{sig} $ (bytes)	11524	=	=	=	=	=
$ \nu\mathbf{k} $ (bytes)	2256	=	=	=	=	=
d	1	2	4	8	16	32
rep	8	4	2	4	2	4
u_t	6	6	6	5	5	4
u_w	41	41	41	40	40	39
$ \text{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. [KLSS23] seems perfectly suited to study Raccoon, as illustrated by a thresholdized variant of Raccoon [DKM⁺24]. For Raccoon itself, an obstacle to a direct application is that [KLSS23] 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

- ABC⁺23. Melissa Azouaoui, Olivier Bronchain, Gaëtan Cassiers, Clément Hoffmann, Yulia Kuzovkova, Joost Renes, Tobias Schneider, Markus Schönauer, François-Xavier Standaert, and Christine van Vredendaal. Protecting Dilithium against leakage revisited sensitivity analysis and improved implementations. *IACR TCHES*, 2023(4):58–79, 2023.
- APS15. Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of Learning with Errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.
- ASY22. Shweta Agrawal, Damien Stehlé, and Anshu Yadav. Round-optimal lattice-based threshold signatures, revisited. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *ICALP 2022*, volume 229 of *LIPICs*, pages 8:1–8:20. Schloss Dagstuhl, July 2022.
- BAE⁺23. Olivier Bronchain, Melissa Azouaoui, Mohamed ElGhamrawy, Joost Renes, and Tobias Schneider. 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>.
- BBD⁺16. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 116–129. ACM Press, October 2016.
- BBE⁺18. Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. Masking the GLP lattice-based signature scheme at any order. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 354–384. Springer, Cham, April / May 2018.

- BBE⁺19. Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Mélissa Rossi, and Mehdi Tibouchi. GALACTICS: Gaussian sampling for lattice-based constant-time implementation of cryptographic signatures, revisited. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2147–2164. ACM Press, November 2019.
- BCPZ16. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 23–39. Springer, Berlin, Heidelberg, August 2016.
- BCRT23. Sonia Belaïd, Gaëtan Cassiers, Matthieu Rivain, and Abdul Rahman Taleb. Unifying freedom and separation for tight probing-secure composition. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part III*, volume 14083 of *LNCS*, pages 440–472. Springer, Cham, August 2023.
- BN06. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399. ACM Press, October / November 2006.
- BVC⁺23. Alexandre Berzati, Andersson Calle Viera, Maya Chartouny, Steven Madec, Damien Vergnaud, and David Vigilant. Exploiting intermediate value leakage in Dilithium: A template-based approach. *IACR TCHES*, 2023(4):188–210, 2023.
- CGMZ23. Jean-Sébastien Coron, François Gérard, Simon Montoya, and Rina Zeitoun. High-order polynomial comparison and masking lattice-based encryption. *IACR TCHES*, 2023(1):153–192, 2023.
- CGTV15. Jean-Sébastien Coron, Johann Großschädl, Mehdi Tibouchi, and Praveen Kumar Vadnala. Conversion from arithmetic to Boolean masking with logarithmic complexity. In Gregor Leander, editor, *FSE 2015*, volume 9054 of *LNCS*, pages 130–149. Springer, Berlin, Heidelberg, March 2015.
- CGTZ23. Jean-Sébastien Coron, François Gérard, Matthias Trannoy, and Rina Zeitoun. High-order masking of NTRU. *IACR TCHES*, 2023(2):180–211, 2023.
- CGV14. Jean-Sébastien Coron, Johann Großschädl, and Praveen Kumar Vadnala. Secure conversion between Boolean and arithmetic masking of any order. In Lejla Batina and Matthew Robshaw, editors, *CHES 2014*, volume 8731 of *LNCS*, pages 188–205. Springer, Berlin, Heidelberg, September 2014.
- Cor17. Jean-Sébastien Coron. High-order conversion from Boolean to arithmetic masking. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 93–114. Springer, Cham, September 2017.
- CPS⁺20. Chitchanok Chuengsatiansup, Thomas Prest, Damien Stehlé, Alexandre Wallet, and Keita Xagawa. ModFalcon: Compact signatures based on module-NTRU lattices. In Hung-Min Sun, Shih-Pyng Shieh, Guofei Gu, and Giuseppe Ateniese, editors, *ASIACCS 20*, pages 853–866. ACM Press, October 2020.
- CS21. Jean-Sébastien Coron and Lorenzo Spignoli. Secure wire shuffling in the probing model. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 215–244, Virtual Event, August 2021. Springer, Cham.

- Csi63. Imre Csiszár. Eine informationstheoretische Ungleichung und ihre Anwendung auf den Beweis der Ergodizität von Markoffschen Ketten. *Magyar. Tud. Akad. Mat. Kutató Int. Közl*, 8:85–108, 1963.
- dEK⁺23. Rafael del Pino, Thomas Espitau, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, Mélissa Rossi, and Markku-Juhani Saarienen. Raccoon. Technical report, National Institute of Standards and Technology, 2023. available at <https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures>.
- DFPS22. Julien Devevey, Omar Fawzi, Alain Passelègue, and Damien Stehlé. On rejection sampling in Lyubashevsky’s signature scheme. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part IV*, volume 13794 of *LNCS*, pages 34–64. Springer, Cham, December 2022.
- DFS19. Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete (or how to evaluate the security of any leaking device), extended version. *Journal of Cryptology*, 32(4):1263–1297, October 2019.
- DKM⁺24. Rafaël Del Pino, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, and Markku-Juhani O. Saarienen. Threshold raccoon: Practical threshold signatures from standard lattice assumptions. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 219–248. Springer, Cham, May 2024.
- dPPRS23. Rafaël del Pino, Thomas Prest, Mélissa Rossi, and Markku-Juhani O. Saarienen. High-order masking of lattice signatures in quasilinear time. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1168–1185, 2023.
- EEN⁺24. Muhammed F. Esgin, Thomas Espitau, Guilhem Niot, Thomas Prest, Amin Sakzad, and Ron Steinfeld. Plover: Masking-friendly hash-and-sign lattice signatures. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VII*, volume 14657 of *LNCS*, pages 316–345. Springer, Cham, May 2024.
- EFG⁺22. Thomas Espitau, Pierre-Alain Fouque, François Gérard, Mélissa Rossi, Akira Takahashi, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu. Mitaka: A simpler, parallelizable, maskable variant of falcon. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 222–253. Springer, Cham, May / June 2022.
- FDK20. Apostolos P. Fournaris, Charis Dimopoulos, and Odysseas G. Koufopavlou. Profiling Dilithium Digital Signature Traces for Correlation Differential Side Channel Attacks. In Alex Orailoglu, Matthias Jung, and Marc Reichenbach, editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation - 20th International Conference, SAMOS 2020, Samos, Greece, July 5-9, 2020, Proceedings*, volume 12471 of *Lecture Notes in Computer Science*, pages 281–294. Springer, 2020.
- GPRV21. Dahmun Goudarzi, Thomas Prest, Matthieu Rivain, and Damien Vergnaud. Probing security through input-output separation and revisited quasilinear masking. *IACR TCHES*, 2021(3):599–640, 2021.
- GR19. François Gérard and Mélissa Rossi. An efficient and provable masked implementation of qtesla. In Sonia Belaïd and Tim Güneysu, editors, *Smart Card Research and Advanced Applications - 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11-13, 2019, Revised Selected Papers*, volume 11833 of *Lecture Notes in Computer Science*, pages 74–91. Springer, 2019.

- HPRR20. James Howe, Thomas Prest, Thomas Ricosset, and Mélissa Rossi. Isochronous gaussian sampling: From inception to implementation. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 53–71. Springer, Cham, 2020.
- HT19. Michael Hutter and Michael Tunstall. Constant-time higher-order Boolean-to-arithmetic masking. *Journal of Cryptographic Engineering*, 9(2):173–184, June 2019.
- ISW03. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Berlin, Heidelberg, August 2003.
- IUH22. Akira Ito, Rei Ueno, and Naofumi Homma. On the success rate of side-channel attacks on masked implementations: Information-theoretical bounds and their practical usage. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1521–1535. ACM Press, November 2022.
- KAA21. Emre Karabulut, Erdem Alkim, and Aydin Aysu. Single-Trace Side-Channel Attacks on ω -Small Polynomial Sampling: With Applications to NTRU, NTRU Prime, and CRYSTALS-DILITHIUM. In *IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2021, Tysons Corner, VA, USA, December 12-15, 2021*, pages 35–45. IEEE, 2021.
- KLSS23. Duhyeong Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Toward practical lattice-based proof of knowledge from hint-MLWE. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 549–580. Springer, Cham, August 2023.
- LDK+22. Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022. available at <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>.
- LPR13. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, Berlin, Heidelberg, May 2013.
- LS15. Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.
- Mat21. Axel Mathieu-Mahias. *Securisation of implementations of cryptographic algorithms in the context of embedded systems. (Sécurisation des implémentations d’algorithmes cryptographiques pour les systèmes embarqués)*. PhD thesis, University of Paris-Saclay, France, 2021.
- MGTF19. Vincent Migliore, Benoît Gérard, Mehdi Tibouchi, and Pierre-Alain Fouque. Masking Dilithium - efficient implementation and side-channel evaluation. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19International Conference on Applied Cryptography and Network Security*, volume 11464 of *LNCS*, pages 344–362. Springer, Cham, June 2019.
- Muk15. Samrat Mukhopadhyay. Decreasing ratio of two Partial Sums. Mathematics Stack Exchange, 2015. Author profile: <https://math.stackexchange.com/users/83973/samrat-mukhopadhyay>. Version: 2015-02-09.

- MUTS22. Soundes Marzougui, Vincent Ulitzsch, Mehdi Tibouchi, and Jean-Pierre Seifert. Profiling side-channel attacks on Dilithium: A small bit-fiddling leak breaks it all. *Cryptology ePrint Archive*, Report 2022/106, 2022.
- NAB⁺17. Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. FrodoKEM. Technical report, National Institute of Standards and Technology, 2017. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions>.
- PFH⁺22. Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022. available at <https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022>.
- Pre17. Thomas Prest. Sharper bounds in lattice-based cryptography using the Rényi divergence. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 347–374. Springer, Cham, December 2017.
- Pre23. Thomas Prest. A key-recovery attack against mitaka in the t -probing model. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 205–220. Springer, Cham, May 2023.
- SLKG23. Hauke Malte Steffen, Georg Land, Lucie Johanna Kogelheide, and Tim Güneysu. Breaking and protecting the crystal: Side-channel analysis of dilithium in hardware. In *PQCrypto*, volume 14154 of *Lecture Notes in Computer Science*, pages 688–711. Springer, 2023.
- WNGD23. Ruize Wang, Kalle Ngo, Joel Gärtner, and Elena Dubrova. Single-trace side-channel attacks on crystals-dilithium: Myth or reality? *Cryptology ePrint Archive*, Paper 2023/1931, 2023. <https://eprint.iacr.org/2023/1931>.

A Parameters

Table 4: Impact of parameters on security/performance metrics. ↗↗ (resp. ↗, =, ↘, ↘↘) indicates that increasing this parameter has a very positive (resp. positive, negative, neutral, very negative) impact on the considered metric.

	q	u_t	u_w	$d \cdot \text{rep}$	v_t	v_w	n	ℓ	k	ω
MLWE	↘	↗↗	=	↗	=	=	↗↗	↗↗	=	=
SelfTargetMSIS	↗↗	=	↘↘	↘	↘	↘	↗↗	↗	↗↗	↘
ϵ_{TAIL}	=	↘↘	↗↗	↗↗	=	=	↘	↘	↘	↘
$R_{\alpha}^{\epsilon_{\text{TAIL}}}$	=	↘↘	↗↗	=	=	=	↘	↘	↘	↘
Size of vk	↘	↗	=	=	↗↗	=	↘↘	=	↘↘	=
Size of sig	↘	↘	↘	↘	=	↗↗	↘↘	↘↘	↘	=

B Deferred Preliminaries

We review some useful properties on samples from the sum of uniform distribution.

B.1 Hardness assumptions

Let us recall the hardness assumptions used to establish the security of Raccoon.

Definition 9 (MLWE). Let ℓ, k, q be integers, and \mathcal{D} be a probability distribution over \mathcal{R}_q . The advantage of an adversary \mathcal{A} against the Module Learning with Errors $\text{MLWE}_{q,\ell,k,\mathcal{D}}$ problem is defined as follows:

$$\text{Adv}_{\mathcal{A}}^{\text{MLWE}}(\kappa) = |\Pr [1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e})] - \Pr [1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{b})]|,$$

where $(\mathbf{A}, \mathbf{b}, \mathbf{s}, \mathbf{e}) \leftarrow \mathcal{R}_q^{k \times \ell} \times \mathcal{R}_q^k \times \mathcal{D}^\ell \times \mathcal{D}^k$. The $\text{MLWE}_{q,\ell,k,\mathcal{D}}$ assumption states that any efficient adversary \mathcal{A} against this problem has negligible advantage.

Definition 10 (MSIS). Let ℓ, k, q be integers and $\beta > 0$ a real number. The advantage of an adversary \mathcal{A} against the Module Short Integer Solution $\text{MSIS}_{q,\ell,k,\beta}$ problem is defined as follows:

$$\text{Adv}_{\mathcal{A}}^{\text{MSIS}}(\kappa) = \Pr [\mathbf{A} \leftarrow \mathcal{R}_q^{k \times \ell}, \mathbf{s} \leftarrow \mathcal{A}(\mathbf{A}) : 0 < \|\mathbf{s}\|_2 \leq \beta \wedge [\mathbf{A} \mid \mathbf{I}] \cdot \mathbf{s} = \mathbf{0} \bmod q].$$

The $\text{MSIS}_{q,\ell,k,\beta}$ assumption states that any efficient adversary \mathcal{A} has negligible advantage.

Definition 11 (SelfTargetMSIS). Let ℓ, k, q be integers, and $\beta > 0$ a real number. Let C be a subset of \mathcal{R}_q and let $G : \mathcal{R}_q^k \times \{0, 1\}^{2k} \rightarrow C$ be a cryptographic hash function modeled as a random oracle. The advantage of an adversary \mathcal{A} against the Self Target MSIS problem, noted $\text{SelfTargetMSIS}_{q,\ell,k,C,\beta}$, is defined as:

$$\text{Adv}_{\mathcal{A}}^{\text{SelfTargetMSIS}}(\kappa) = \Pr \left[\mathbf{A} \leftarrow \mathcal{R}_q^{k \times \ell}, (\text{msg}, \mathbf{s}) \leftarrow \mathcal{A}^G(\mathbf{A}), (\text{msg}, \mathbf{s}) \in \{0, 1\}^{2k} \times \mathcal{R}_q^{\ell+k} : \left(\mathbf{s} = \begin{bmatrix} c \\ \mathbf{s}' \end{bmatrix} \right) \wedge (0 < \|\mathbf{s}\|_2 \leq \beta) \wedge G([\mathbf{A} \mid \mathbf{I}] \cdot \mathbf{s}, \text{msg}) = c \right].$$

The $\text{SelfTargetMSIS}_{q,\ell,k,C,\beta}$ assumption states that any efficient adversary \mathcal{A} has no more than negligible advantage.

The following are important worst-case to average-case reductions, establishing the hardness of the MLWE and MSIS problems.

Lemma 9 (Hardness of MLWE ([LS15])). Let $k(\kappa), \ell(\kappa), q(\kappa), n(\kappa), \sigma(\kappa)$ such that $q \leq \text{poly}(\ell \cdot n)$, $k \leq \text{poly}(\ell)$, and $\sigma \geq \sqrt{\ell} \cdot \omega_{\text{asympt}}(\sqrt{\log n})$. If \mathcal{D} is a discrete Gaussian distribution with standard deviation σ , then the $\text{MLWE}_{q,\ell,k,\mathcal{D}}$ problem is as hard as the worst-case lattice Generalized-Independent-Vector-Problem (GIVP) in dimension $N = \ell n$ with approximation factor $\sqrt{8} \cdot N \ell \cdot \omega_{\text{asympt}}(\sqrt{\log \ell}) \cdot q / \sigma$.

Lemma 10 (Hardness of MSIS ([LS15])). For any $k(\kappa), \ell(\kappa), q(\kappa), n(\kappa), \beta(\kappa)$ such that $q > \beta \sqrt{\ell n} \cdot \omega_{\text{asympt}}(\log(\ell n))$, $k \leq \text{poly}(\ell)$, and $\log q \leq \text{poly}(\ell n)$. The $\text{MSIS}_{q,\ell,k,\beta}$ problem is as hard as the worst-case lattice Generalized-Independent-Vector-Problem (GIVP) in dimension $N = \ell n$ with approximation factor $\beta \sqrt{N} \cdot \omega_{\text{asympt}}(\sqrt{\log N})$.

Finally, the following lemma, an immediate implication of the forking lemma [BN06], reduces MSIS to SelfTargetMSIS.

Lemma 11 (Hardness of SelfTargetMSIS). Let ℓ, k, q be integers and $\beta > 0$ be a real number. Let C be a subset of \mathcal{R}_q and let $G : \mathcal{R}_q^k \times \{0, 1\}^{2k} \rightarrow C$ be a cryptographic hash function modeled as a random oracle. Then, for any adversary \mathcal{A} against the $\text{SelfTargetMSIS}_{q,\ell,k,C,\beta}$ problem making at most Q queries to H , there exists an adversary \mathcal{B} against the $\text{MSIS}_{q,\ell,k,2\beta}$ problem such that

$$\text{Adv}_{\mathcal{A}}^{\text{SelfTargetMSIS}}(\kappa) \leq \sqrt{Q \cdot \text{Adv}_{\mathcal{B}}^{\text{MSIS}}(\kappa)} + \frac{Q}{|C|}.$$

B.2 Min-entropy of MLWE for the sum of uniform distribution.

In the security proofs, we use the $\text{MLWE}_{q,\ell,k,\mathcal{D}}$ distribution with bit dropping: BD-MLWE, formally defined as follows.

Definition 12. Let BD-MLWE be the $\text{MLWE}_{q,\ell,k,\mathcal{D}}$ distribution with v_w dropped bits. Namely, given $\mathbf{A} \in \mathcal{R}_q^{k \times \ell}$, BD-MLWE is defined as the ensemble $\{[\mathbf{A} \cdot \mathbf{s} + \mathbf{e}]_{v_w} \mid (\mathbf{s}, \mathbf{e}) \leftarrow \mathcal{D}^{\ell+k}\}$.

Conjecture 2. For the parameters of our scheme in Figure 7, i.e., $\text{MLWE}_{q,\ell,k,\text{SU}(u,d\cdot\text{rep})}$, we have

$$H_\infty(\text{BD-MLWE}) \geq 2 \cdot \kappa.$$

While we do not have a proof that the above min-entropy is large enough, there are strong heuristic arguments toward this. First, if the distribution of (\mathbf{s}, \mathbf{e}) was Gaussian (even with a much smaller standard deviation) we would be able to use the regularity theorem of [LPR13] to argue $n > 2\kappa$ bits of security. Second, even without Gaussians if we assume that $\mathbf{A}\mathbf{s} + \mathbf{e}$ is “well distributed”, i.e. if we assume that the distributions $[\mathbf{A}\mathbf{s} + \mathbf{e}]_{v_w}$ and $\mathbf{A}\mathbf{s} + \mathbf{e} \bmod 2^{v_w}$ are independent, then we have:

$$\begin{aligned} H_\infty(\mathbf{A}\mathbf{s} + \mathbf{e}) &= H_\infty([\mathbf{A}\mathbf{s} + \mathbf{e}]_{v_w}, \mathbf{A}\mathbf{s} + \mathbf{e} \bmod 2^{v_w}) \\ &\leq H_\infty(\text{BD-MLWE}) + knv_w \end{aligned}$$

If we also assume that the function $(\mathbf{s}, \mathbf{e}) \mapsto \mathbf{A}\mathbf{s} + \mathbf{e}$ is injective, we get

$$H_\infty(\text{BD-MLWE}) \geq H_\infty(\mathcal{D}^{\ell+k}) - knv_w$$

Since \mathcal{D} is the sum of T uniform distribution with support $N = 2^{u_w}$ it has min-entropy larger than u_w , hence:

$$H_\infty(\text{BD-MLWE}) \geq (\ell + k)nu_w - knv_w$$

For all parameters we consider we will have $H_\infty(\text{BD-MLWE}) > 100\kappa$, meaning that even if the two assumptions we have made are not very accurate we have high confidence in the amount of entropy used for the input of the hash function.

B.3 Deferred preliminaries on sum of uniforms

Given a random variable $X \sim \text{SU}(u, T)$, its moment-generating function is easily computed, here with $N = 2^u$:

$$\mathbb{E}[e^{kX}] = \left(\frac{e^{Nk/2} - e^{-Nk/2}}{N(e^k - 1)} \right)^T$$

One can check that $X + T/2$ is sub-Gaussian for $\sigma^2 = \frac{N^2 T}{6}$. Hence the sub-Gaussian tail bounds:

$$\mathbb{P}[|X + T/2| > \mu] \leq \exp\left(-\frac{\mu^2}{2\sigma^2}\right) = \exp\left(-\frac{3\mu^2}{TN^2}\right) \quad (10)$$

Using Lemma 2.2 from [LPR13] we get a bound on the norm of a vector $Y = (X_1, \dots, X_m)$ of m iid. variables from $SU(u, T)$. For any $r \geq 16$:

$$\Pr \left[\sum_1^m X_i^2 > r \cdot m \cdot \sigma^2 \right] \leq \exp \left(-\frac{r \cdot m}{8} \right) \quad (11)$$

Derivating the moment-generating function gives us the moments and variance of X :

$$\mathbb{E}[X] = -\frac{T}{2}; \quad \mathbb{E}[X^2] = \frac{T(N^2 - 1)}{12} + \frac{T^2}{4}; \quad \mathbb{V}[X] = \frac{T(N^2 - 1)}{12}; \quad (12)$$

Note that $SU(u, T)$ is not “symmetric”, in the sense that its expected value and skewness (third-order moment) are not equal to zero. This could be problematic in applications such as trapdoor sampling, but is unimportant in the case of Raccoon. Indeed, since the offset is public we can remove it and assume for the study of the MLWE hardness that the distribution is centered around zero. On the other hand, the offset is very small and has a negligible impact on the hardness of SelfTargetMSIS.

B.4 Tail-cut bounds

We provide some norm bounds regarding the sum of uniform distribution.

Lemma 12. *Let $\mathbf{v} \in \mathcal{R}^L$ and $c \in \mathcal{R}$ such that each integer coefficient of \mathbf{v} is sampled from $SU(u, T)$ and $\|c\|_\infty = 1$. Let $N = 2^u$ and $\nu^2 = \frac{TN^2}{3} \cdot (\kappa + \log(nL)) \cdot \log(2)$. Then, we have*

$$\Pr \left[\|\mathbf{v} \cdot c\|_2 \leq \|c\|_1 \cdot \sqrt{nL} \cdot \left(\frac{T}{2} + \nu \right) \right] \geq 1 - 2^{-\kappa}.$$

$$\Pr \left[\|\mathbf{v} \cdot c\|_\infty \leq \|c\|_1 \cdot \left(\frac{T}{2} + \nu \right) \right] \geq 1 - 2^{-\kappa}.$$

Proof. Minkowski’s inequality implies $\|\mathbf{v} \cdot c\|_2 \leq \|c\|_1 \cdot \|\mathbf{v}\|_2$. Moreover, since the absolute value of each coefficient of c is less than 1, we have $\|\mathbf{v} \cdot c\|_\infty \leq \|c\|_1 \cdot \|\mathbf{v}\|_\infty$. Since $\mathbf{v}' = \mathbf{v} + \frac{T}{2} \cdot \mathbf{1}$ is a sub-Gaussian of parameter $\sigma^2 = \frac{N^2 \cdot T}{6}$, we can combine Eq. (10) with the union bound:

$$\Pr [\|\mathbf{v}'\|_\infty > \nu] \leq 2^{-\kappa}.$$

We obtain the bound using $\|\mathbf{v}\|_2 \leq \sqrt{nL} \cdot \left(\frac{T}{2} + \|\mathbf{v}'\| \right)$. □

Lemma 13. *For $0 < p < 1$ and $k > 0$, let \mathcal{D} be the binomial distribution of parameter $\left(\left\lceil \frac{2k}{p} \right\rceil, p \right)$. Hoeffding’s Inequality for \mathcal{D} entails:*

$$\Pr [\mathcal{D} \leq k] \leq \exp(-pk)$$

C Deferred Details from Section 4

This section provides a collection of results related to sums of discrete uniform variables that were omitted from Section 4.

C.1 Symmetry for Symmetric Distributions

For a class of “symmetric” distributions which includes shifted copies of sums uniforms, the Rényi divergence is symmetric.

Lemma 14 (Symmetry for symmetric distributions). *Let P, Q be distributions of support included in \mathbb{Z} . Suppose that P, Q are “symmetric” in the sense that there exists $C \in \mathbb{Z}$ such that $P(x) = Q(C - x)$. Then for any $\alpha > 1, \epsilon > 0$, it holds that $R_\alpha^\epsilon(P; Q) = R_\alpha^\epsilon(Q; P)$.*

In particular, for $P_{\text{SU}} = \text{SU}(u, T)$ and Q_{SU} the distributions corresponding to shifting the support of P by c , we have $R_\alpha^\epsilon(P_{\text{SU}}; Q_{\text{SU}}) = R_\alpha^\epsilon(Q_{\text{SU}}; P_{\text{SU}})$.

Proof. The bijection $x \mapsto C - x$ maps the distributions (P, Q) to (Q, P) . Therefore (P, Q) and (Q, P) are identical up to reindexing the support. In particular, $R_\alpha^\epsilon(P; Q) = R_\alpha^\epsilon(Q; P)$. Lastly, by defining $C = T \cdot (2^u - 1) + c$, we have $P_{\text{SU}}(x) = Q_{\text{SU}}(C - x)$. \square

C.2 The Sum of Discrete Uniform Variables

Definition 13. *Let $N, T \geq 1$ be integers. We note $P_{N,T}$ the distribution corresponding to the sum of T independent and identically distributed (iid) random variables $(X_i)_{i \in [T]}$, each X_i being uniformly distributed in $[N]$.*

The support of $P_{N,T}$ is $[T(N - 1) + 1]$. Lemma 15 links the cumulative distribution function (CDF) of $P_{N,T}$ and the probability distribution function (PDF) of $P_{N,T+1}$.

Lemma 15. *For any $x \geq 0$, $P_{N,T+1}(x) = \frac{1}{N} P_{N,T}(\{\max(0, x - N + 1), \dots, x\})$.*

Proof. $(T + 1)$ random variables X_i sum to x if and only if the T first sum to x_1 , the last one is equal to x_2 , and $x_1 + x_2 = x$. If we note $y = \max(0, x - N + 1)$, this is formalized as follows:

$$\begin{aligned} P_{N,T+1}(x) &= \sum_{x_1+x_2=x} P_{N,T}(x_1) P_{N,1}(x) \\ &= \frac{1}{N} \sum_{x_1=y}^x P_{N,T}(x_1) \\ &= \frac{1}{N} P_{N,T}(\{y, \dots, x\}) \end{aligned}$$

\square

Our next lemma provides a neat closed formula for the weight of the tail of $P_{N,T}$.

Lemma 16. *For $x \in [N]$:*

$$P_{N,T}(\{0, \dots, x\}) = \binom{x+T}{x} \frac{1}{N^T} = \binom{x+T}{T} \frac{1}{N^T}$$

By Lemma 15, this implies:

$$P_{N,T}(x) = \binom{x+T-1}{T-1} \frac{1}{N^T} \quad (13)$$

Proof. We prove the result by induction on T . First, one can check that it is true for $T \leq 2$. Now, following the same reasoning as in the proof of Lemma 15, $(T+1)$ random variables X_i sum to a value $\leq x$ if and only if the T first sum to a value $\leq x_1$, the last one is equal to x_2 , and $x_1 + x_2 = x$. This can be formalized as:

$$\begin{aligned} P_{N,T+1}(\{0, \dots, x\}) &= \sum_{x_1+x_2=x} P_{N,T}(\{0, \dots, x_1\}) P_{N,1}(x) \\ &= \frac{1}{N^{T+1}} \sum_{x_1 \leq x} \binom{x_1+T}{x_1} \\ &= \frac{1}{N^{T+1}} \binom{x+T+1}{x} \end{aligned}$$

The final equality is due to the hockey-stick identity. \square

Monotony of $x \mapsto P_{N,T}(x+c)/P_{N,T}(x)$. The goal of this section is to prove that $P_{N,T}(x+c)/P_{N,T}(x)$ is non-increasing in x . This will later be useful for computing the smooth Rényi divergence between shifted copies of $P_{N,T}$.

Lemma 17. *Let $c \geq 0$ be an integer. The function*

$$x \in \{0, \dots, T(N-1) - c\} \mapsto \frac{P_{N,T}(x+c)}{P_{N,T}(x)}$$

is non-increasing.

Proof. It suffices to prove Lemma 17 in the special case $c = 1$. The general case follows from the telescopic product:

$$\frac{P_{N,T}(x+c)}{P_{N,T}(x)} = \frac{P_{N,T}(x+c)}{P_{N,T}(x+c-1)} \times \dots \times \frac{P_{N,T}(x+1)}{P_{N,T}(x)}.$$

For the rest of the proof, let $c = 1$. For $x < N$, the statement can be verified using Lemma 16. For $x \geq n$, we proceed by induction on T . The statement is

true for $T = 1$. For $x \geq n$, it holds that:

$$\begin{aligned} P_{N,T+1}(x) &= \sum_{x_1+x_2=x} P_{N,T}(x_1) P_{N,1}(x_2) \\ &= \frac{1}{N} \sum_{c=0}^{N-1} P_{N,T}(x-c) \end{aligned}$$

Therefore the ratio $P_{N,T+1}(x+1)/P_{N,T+1}(x)$ can be written as a ratio of partial sums:

$$\frac{P_{N,T+1}(x+1)}{P_{N,T+1}(x)} = \frac{\sum_{c=0}^{N-1} P_{N,T}(x+1-c)}{\sum_{c=0}^{N-1} P_{N,T}(x-c)}$$

Since the ratio $P_{N,T}(x+1)/P_{N,T}(x)$ is non-increasing, this is also the case for the ratio of their partial sums [Muk15]. \square

C.3 Smooth Rényi Divergence Between Shifted Copies of $P_{N,T}$

The goal of this section is to prove Lemma 3. We first partition $\text{Supp}(P) \cup \text{Supp}(Q)$ in five sections, as illustrated in Figure 11:

Tails. The tails are $T_\ell = \{0, \dots, \tau-1\}$ and $T_r = \{T(N-1)+c-\tau+1, \dots, T(N-1)+c\}$. By symmetry, $P(T_\ell) = Q(T_r)$. Moreover, τ is chosen such that $P(T_\ell) \leq \epsilon$.

Sides. The sides are $S_\ell = \{\tau, \dots, N-1\}$ and $S_r = \{(T-1)(N-1)+c+1, \dots, T(N-1)+c-\tau\}$. Over the sides, $P(x)$ and $Q(x)$ can be computed explicitly, which allows computing precise bounds on partial Rényi divergence sums.

Head. The head is $H = \{N, \dots, (T-1)(N-1)+c\}$. Over the head, the ratio P/Q is constrained in a very narrow interval, which allows bounding the partial Rényi divergence sum over H using generic results.

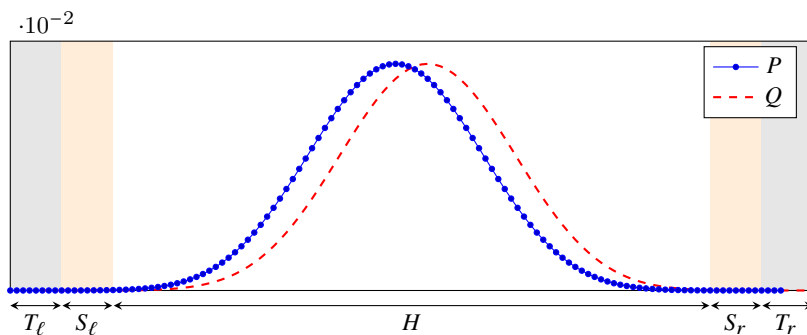


Fig. 11: Two copies of $P_{\{n=15, t=8\}}$, shifted by an offset $c = 5$. The areas T_ℓ, S_ℓ, H, S_r and T_r relate to a proof in Appendix C.3

Informally speaking, our proof strategy is to separately bound the statistical distance over the tails (Appendix C.3), and the partial Rényi divergence over the sides (Appendix C.3) and head (Appendix C.3). The smooth Rényi divergence between P and Q is obtained (Appendix C.3) as a simple consequence of these separate bounds.

Selecting P' and Q' . Let $\tau > 0$ such that $\frac{(\tau+T)^T}{T!N^T} \leq \epsilon$. By Lemma 16, we can bound the weight of P (resp. Q) over the left (resp. right) tail: $P(T_\ell) = Q(T_r) \leq \epsilon$.

Let $Q' = Q$, and P' be such that $P'(x) = Q(x)$ if $x \in T_\ell \sqcup T_r$, otherwise $P'(x) = P(x)$. This implies $\Delta_{\text{SD}}(P', P) \leq \epsilon$ and $\Delta_{\text{SD}}(Q', Q) = 0$.

Partial Sum Over the Sides. We now compute partial Rényi divergences sums over the sides. This is perhaps the most tedious part of our overall proof, as we computed this sum explicitly, as opposed to relying on more generic bounds.

Lemma 18. *Let $T \geq 2$, $\alpha \geq 4$, $\tau, c \geq 0$ be such that $\alpha c \leq \tau$. Let $a = (T-1)\alpha c$ and assume $a = o(N)$. Then:*

$$\sum_{x \in S_\ell \sqcup S_r} \left(\frac{P(x)}{Q(x)} \right)^\alpha Q(x) \leq \frac{1}{T!} \left(2 + \frac{T}{T-2} \left(\frac{a}{N} \right)^2 + O((a/N)^3) \right). \quad (14)$$

Proof. Without loss of generality, we assume $c > 0$, as the result is otherwise straightforward. We focus on the left side S_ℓ . By computing their derivatives, one can see that:

$$x \mapsto \left(\frac{x+c}{x} \right)^\alpha x \quad \text{is non-decreasing over } [(\alpha-1)c; +\infty) \quad (15)$$

$$f_{a,T} : x \mapsto \exp\left(\frac{a}{x}\right) x^T \quad \text{is non-decreasing over } [a/T; +\infty) \quad (16)$$

In particular, since $\max((\alpha-1)c, a/T) \leq \tau - c$, they are non-decreasing over $[\tau - c; +\infty)$. Since Lemma 12, Eq. (13) provides exact formulae for $P(x)$ and $Q(x)$ on the sides, we can upper bound each term $\left(\frac{P(x)}{Q(x)} \right)^\alpha Q(x)$ for $x \in S_\ell$:

$$\begin{aligned} \left(\frac{P(x)}{Q(x)} \right)^\alpha Q(x) &= \frac{1}{(T-1)!N^T} \prod_{u=x-c}^{x-c+T-1} \left(\frac{u+c}{u} \right)^\alpha u \\ &\leq \frac{1}{(T-1)!N^T} \left(\frac{x+T-1}{x-c+T-1} \right)^{(T-1)\alpha} (x-c+T-1)^{T-1} \end{aligned} \quad (17)$$

$$\leq \frac{1}{(T-1)!N^T} \exp\left(\frac{c(T-1)\alpha}{x-c+T-1}\right) (x-c+T-1)^{T-1} \quad (18)$$

(17) follows from the non-decreasingness of (15), while (18) follows from Bernoulli's inequality $1+x \leq \exp(x)$. We now bound the partial Rényi divergence sum over

the left side S_ℓ :

$$\begin{aligned} \sum_{x \in S_\ell} \left(\frac{P(x)}{Q(x)} \right)^\alpha Q(x) &\leq \frac{1}{(T-1)! N^T} \sum_{x=\tau-c+T-2}^{n-c+T-1} \exp\left(\frac{c(T-1)\alpha}{x}\right) x^{T-1} \\ &\leq \frac{1}{(T-1)! N^T} \sum_{x=\tau}^{N-1} f_{a,T-1}(x) \end{aligned} \quad (19)$$

$$\leq \frac{1}{(T-1)! N^T} \int_{\tau}^N f_{a,T-1}(u) du, \quad (20)$$

where (19) is implied by $f_{a,T-1}$ being non-decreasing, see (16), and (20) bounds a sum by an integral, by casting it as a Riemann sum and using its monotonicity. Let us note:

$$F_{a,T} = \int_{\tau}^N f_{a,T}(u) du.$$

Our next goal is to bound $F_{a,T}$. An iterated integration by parts gives us:

$$\begin{aligned} F_{a,T} &= \left[\frac{1}{T+1} f_{T+1} \right]_{\tau}^N + \frac{a}{T+1} F_{T-1} \\ &= \frac{1}{T+1} \left[f_{T+1} + \frac{a}{T} f_T + \frac{a^2}{T(T-1)} f_{T-1} + \cdots + \frac{a^{T-1}}{T!} f_2 \right]_{\tau}^N + \frac{a^T}{(T+1)!} F_0 \end{aligned}$$

Since $\tau \leq a$ and $a = o(N)$, we can approximate $F_{a,T}$ using Taylor series as follows:

$$\begin{aligned} F_{a,T} &= \frac{\exp(a/N)}{T+1} \left(N^{T+1} + \frac{a}{T} N^T + \frac{a^2}{T(T-1)} N^{T-1} + O(a^3 N^{T-2}) \right) \\ &= \frac{N^{T+1}}{T+1} \left(1 + \left(1 + \frac{1}{T} \right) \frac{a}{N} + \frac{T+1}{2(T-1)} \left(\frac{a}{N} \right)^2 + O((a/N)^3) \right) \end{aligned} \quad (21)$$

Combining (20) and (21) gives the partial sum on the left tail:

$$\sum_{x \in S_\ell} \left(\frac{P(x)}{Q(x)} \right)^\alpha Q(x) \leq \frac{1}{T!} \left(1 + \left(1 + \frac{1}{T-1} \right) \frac{a}{N} + \frac{T}{2(T-2)} \left(\frac{a}{N} \right)^2 + O((a/N)^3) \right) \quad (22)$$

Applying the same techniques provides a similar bound for the right tail T_r .

$$\sum_{x \in S_r} \left(\frac{P(x)}{Q(x)} \right)^\alpha Q(x) \leq \frac{1}{T!} \left(1 - \left(1 + \frac{1}{T-1} \right) \frac{a}{N} + \frac{T}{2(T-2)} \left(\frac{a}{N} \right)^2 + O((a/N)^3) \right) \quad (23)$$

Adding (22) and (23) gives the result.

$$\sum_{x \in S_\ell \sqcup S_r} \left(\frac{P(x)}{Q(x)} \right)^\alpha Q(x) \leq \frac{1}{T!} \left(2 + \frac{T}{T-2} \left(\frac{a}{N} \right)^2 + O((a/N)^3) \right).$$

□

Partial Sum Over the Head.

Lemma 19. *Let $cT = o(N)$. Then:*

$$\sum_{x \in H} \left(\frac{P(x)}{Q(x)} \right)^\alpha Q(x) \leq 1 + \frac{\alpha(\alpha-1)}{2} \left(\frac{(cT)^2}{N^2} + O\left(\frac{(cT)^3}{N^3}\right) \right) - \frac{2}{T!} \left(1 + 4(c/N)^2 + O((c/N)^4) \right) \quad (24)$$

Proof. Our goal is to apply [Pre17, Lemma 3]. This lemma requires us to bound the ratio $P(x)/Q(x)$ over $H = \{N, \dots, (T-1)(N-1) + c\}$. Thanks to the monotonicity of this ratio (Lemma 17), we know that it suffices to bound it at the extremities of H .

$$\frac{P(N)}{Q(N)} = \prod_{x=N}^{N+T-1} \frac{x}{x-c} \leq \left(\frac{N}{N-c} \right)^T \leq \exp\left(\frac{cT}{N-c}\right) \quad (25)$$

Similarly, by symmetry:

$$\frac{P((T-1)(N-1) + c)}{Q((T-1)(N-1) + c)} = \frac{Q(N)}{P(N)} \geq \exp\left(-\frac{cT}{N-c}\right)$$

A second issue is that [Pre17, Lemma 3] provides us the complete Rényi divergence sum over the full support of a distribution, while we only require a partial sum over H . We resolve this by assuming that all values $x \notin H$ are collapsed into a single value. Note that $P(\mathbb{Z} \setminus H) = Q(\mathbb{Z} \setminus H)$. If we note $\delta = \exp\left(\frac{cT}{N-c}\right) - 1 = \frac{cT}{N} + O\left(\frac{cT}{N}\right)^2$, then we obtain:

$$\sum_{x \in H} \left(\frac{P(x)}{Q(x)} \right)^\alpha Q(x) \leq 1 + \frac{\alpha(\alpha-1)\delta^2}{2(1-\delta)^{\alpha+1}} - Q(\mathbb{Z} \setminus H) \quad (26)$$

Finally, we can explicitly compute $Q(\mathbb{Z} \setminus H)$ via Lemma 12, and approximate it via Taylor series:

$$Q(\mathbb{Z} \setminus H) = \frac{2}{T!} \left(1 + 4(c/N)^2 + O((c/N)^4) \right)$$

□

Putting it together: Proof of Lemma 3.

Proof. It suffices to prove Lemma 3 for $c \geq 0$ since $R_\alpha^\epsilon(P; Q) = R_\alpha^\epsilon(Q; P)$. We apply Lemma 16 to compute ϵ , and Lemmas 18 and 19 to compute the Rényi divergence sum. □

C.4 Conjecture on the Smooth Rényi Divergence for Sums of Uniforms

In practice, Lemma 3 is a bit sub-optimal. For instance, [ASY22, Lemma 2.28] tells that if P is instead a Gaussian of parameter σ , then $\log R_\alpha(P; Q) \leq \frac{\alpha c^2}{2\sigma^2}$.

Thus there is a gap $O(T^3)$ between Lemma 3 and [ASY22, Lemma 2.28]. For this reason, we put forward Conjecture 1, which ignores this gap and which we use when setting our concrete parameters.

We note that Eq. (7) is obtained by invoking the tensorization property of the smooth Rényi divergence (see Lemma 2, Item 3) on Eq. (6).

D Deferred Details from Section 6

The `SignER` algorithm is described in Algorithm 16. The following is the proof of Lemma 8.

Algorithm 16 `SignER`($\llbracket \text{sk} \rrbracket, \text{msg}, (\rho_{i,i_{\text{rep}},j}^{(0)}), (\rho_{i,i_{\text{rep}},j}^{(1)})) \rightarrow \text{sig}$
 \triangleright `Sign` with explicit randomness for `AddRepNoise`

Input: Secret signing key $\text{sk} = (\text{vk}, \llbracket \text{s} \rrbracket)$
Input: Message to be signed $\text{msg} \in \{0, 1\}^*$.
Input: Randomness $(\rho_{i,i_{\text{rep}},j}^{(0)})_{i \in [\text{len}(\text{v})], i_{\text{rep}} \in [\text{rep}], j \in [d]}, (\rho_{i,i_{\text{rep}},j}^{(1)})_{i \in [\text{len}(\text{v})], i_{\text{rep}} \in [\text{rep}], j \in [d]}$
Output: Signature $\text{sig} = (c_{\text{hash}}, \mathbf{h}, \mathbf{z})$ of msg under sk .

- 1: $(\text{vk}, \llbracket \text{s} \rrbracket) := \text{sk}, (\text{seed}, \mathbf{t}) := \text{vk}$
- 2: $\mu := \text{H}(\text{H}(\text{vk}) \parallel \text{msg})$
- 3: $\mathbf{A} := \text{ExpandA}(\text{seed})$
- 4: $\llbracket \mathbf{r} \rrbracket \leftarrow \ell \times \text{ZeroEncoding}(d)$
- 5: $\llbracket \mathbf{r} \rrbracket \leftarrow \text{AddRepNoise}_{\text{ER}}(\llbracket \mathbf{r} \rrbracket, u_{\mathbf{w}}, \text{rep}, \rho_{i,i_{\text{rep}},j}^{(0)})$ \triangleright Partially derandomized
`AddRepNoise.`
- 6: $\llbracket \mathbf{w} \rrbracket := \mathbf{A} \cdot \llbracket \mathbf{r} \rrbracket$
- 7: $\llbracket \mathbf{w} \rrbracket \leftarrow \text{AddRepNoise}_{\text{ER}}(\llbracket \mathbf{w} \rrbracket, u_{\mathbf{w}}, \text{rep}, \rho_{i,i_{\text{rep}},j}^{(1)})$ \triangleright Partially derandomized
`AddRepNoise.`
- 8: $\mathbf{w} := \text{Decode}(\llbracket \mathbf{w} \rrbracket)$
- 9: $\mathbf{w} := \lfloor \mathbf{w} \rfloor_{v_{\mathbf{w}}}$
- 10: $c_{\text{hash}} := \text{ChalHash}(\mathbf{w}, \mu)$
- 11: $c_{\text{poly}} := \text{ChalPoly}(c_{\text{hash}})$
- 12: $\llbracket \mathbf{s} \rrbracket \leftarrow \text{Refresh}(\llbracket \mathbf{s} \rrbracket)$
- 13: $\llbracket \mathbf{r} \rrbracket \leftarrow \text{Refresh}(\llbracket \mathbf{r} \rrbracket)$
- 14: $\llbracket \mathbf{z} \rrbracket := c_{\text{poly}} \cdot \llbracket \mathbf{s} \rrbracket + \llbracket \mathbf{r} \rrbracket$
- 15: $\llbracket \mathbf{z} \rrbracket \leftarrow \text{Refresh}(\llbracket \mathbf{z} \rrbracket)$
- 16: $\mathbf{z} := \text{Decode}(\llbracket \mathbf{z} \rrbracket)$
- 17: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - 2^{\nu_{\mathbf{t}}} \cdot c_{\text{poly}} \cdot \mathbf{t}$
- 18: $\mathbf{h} := \mathbf{w} - \lfloor \mathbf{y} \rfloor_{v_{\mathbf{w}}}$
- 19: $\text{sig} := (c_{\text{hash}}, \mathbf{h}, \mathbf{z})$
- 20: **if** $\{\text{CheckBounds}(\text{sig}) = \text{FAIL}\}$ **goto** Line 4
- 21: **return** sig

Proof (Lemma 8). We proceed similarly to the proof of Lemma 7. We decompose Algorithm 2 as a succession of gadgets. The gadgets may be represented

as Figure 12. We decompose the set $\bar{\mathcal{F}}$ of at most $d - 1$ probes in Sign_{ER} as

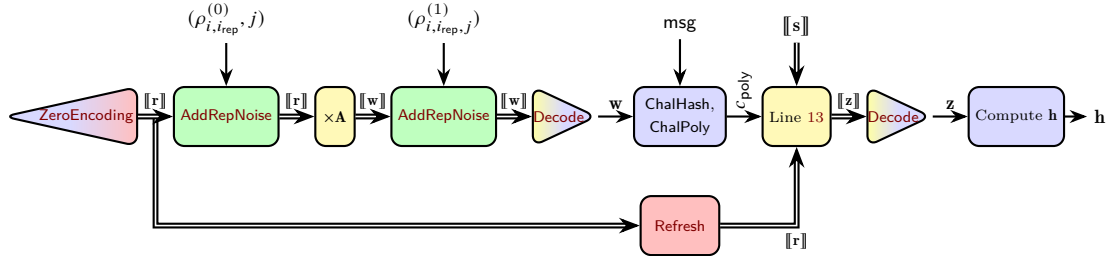


Fig. 12: Structure of Sign (Algorithm 16). Notations inherited from Figures 5 and 6. The gadgets containing only unmasked data are noted gadget.

follows (with the same convention excluding the probes on the outputs from the count).

- δ_0 be the number of intermediate variables probed in Line 16 ($\text{Decode}(\llbracket z \rrbracket)$).
- δ_1 be the number of intermediate variables probed in Line 14.
- δ_2 be the number of intermediate variables probed in Line 13 ($\text{Refresh}(\llbracket r \rrbracket)$).
- δ_3 be the number of intermediate variables probed in Line 8 ($\text{Decode}(\llbracket w \rrbracket)$).
- δ_4 be the number of intermediate variables probed in Line 6 (second $\text{AddRepNoise}_{\text{ER}}$).
- δ_5 be the number of intermediate variables probed in Line 5 (multiplication by \mathbf{A}).
- δ_6 be the number intermediate variables probed in Line 5 (first $\text{AddRepNoise}_{\text{ER}}$).
- δ_7 be the number of intermediate variables probed in Line 4 (ZeroEncoding).

Some probes may be performed during the unmasked computations like in the computation of ChalHash , ChalPoly or the computation of the hint, their number does not matter. By definition, $\sum_{i=0}^7 \delta_i \leq d - 1$.

Starting at the end of the algorithm, thanks to the NI property of the final Decode gadget, all the intermediate variables from the final Decode can be perfectly simulated with at most δ_0 shares of $\llbracket z \rrbracket$. Identically, since Line 14 is NI, all the intermediate variables from Line 14 to the end of the signature can be perfectly simulated with at most $\delta_0 + \delta_1$ shares of $\llbracket r \rrbracket$ and $\llbracket s \rrbracket$.

Continuing to the preceding masked gadget, thanks to the NI property of the first Decode gadget, all the intermediate variables after its execution until the computation of ChalHash and ChalPoly can be simulated with δ_3 shares of $\llbracket w \rrbracket$. Hence, all the intermediate variables after the second $\text{AddRepNoise}_{\text{ER}}$ can be perfectly simulated with δ_4 shares of $\llbracket w \rrbracket$ and $(\rho_{i,irep}^{(1)})$. Note that we exclude the δ_3 observations as they are on the outputs (this is due to the sNIU property of the second $\text{AddRepNoise}_{\text{ER}}$). Similarly, as the multiplication with \mathbf{A} is linear thus NI, the probes inside its execution can be perfectly simulated with at most

δ_6 shares of $\llbracket r \rrbracket$ (output of the first **AddRepNoise_{ER}**) and at most δ_6 shares of $(\rho_{i, i_{\text{rep}}}^{(0)})$.

Finally, as the **Refresh** gadget is sNI and **ZeroEncoding** is sNI (and requires no input), the probes inside the **ZeroEncoding** and **Refresh** can be perfectly simulated from uniform random and public parameters.

To recap, we have proved that the distribution of the intermediate variables in $\bar{\mathcal{F}}$ may be perfectly simulated from :

- δ_6 shares of $(\rho_{i, i_{\text{rep}}}^{(0)})$
- δ_4 shares of $(\rho_{i, i_{\text{rep}}}^{(1)})$
- $\delta_0 + \delta_1$ shares of $\llbracket s \rrbracket$.

Since $\delta_6 + \delta_4, \delta_0 + \delta_1 \leq \sum_{i=0}^7 \delta_i \leq d - 1$, this concludes the proof of Lemma 8. \square

E Deferred Details from Section 7.2

This section contains the deferred details from Section 7.2.

E.1 Formal Description of **Sign_S**

The following is the formal description of **Sign_S** used in Hybrid₃ of Theorem 1.

Algorithm 17 $\text{Sign}_{\mathcal{L}}(\text{sk}, \text{msg}, \mathcal{F}) \rightarrow \text{sig}$

Input: Signing key $\text{sk} = (\text{vk}, \llbracket \text{s} \rrbracket)$, message $\text{msg} \in \{0, 1\}^*$.

Input: Probe set $\mathcal{F} = (\mathcal{F}_{\mathbf{r}}, \mathcal{F}_{e'}, \mathcal{F}_{\text{sk}}), \mathcal{F}_{\mathbf{r}} \subset \{\bar{\rho}_{\mathbf{r}, i, i_{\text{rep}}, j}\}_{(i, i_{\text{rep}}, j) \in [\ell] \times [\text{rep}] \times [d]}$,

$\mathcal{F}_{e'} \subset \{\bar{\rho}_{e', i, i_{\text{rep}}, j}\}_{(i, i_{\text{rep}}, j) \in [k] \times [\text{rep}] \times [d]}, \mathcal{F}_{\text{sk}} \subset \{\bar{\text{sk}}_j\}_{j \in [d]}$

Output: Signature $\text{sig} = (c_{\text{hash}}, \mathbf{h}, \mathbf{z})$ of msg under sk , leakage \mathcal{L}

1: $\mu := \text{H}(\text{H}(\text{vk}) \parallel \text{msg}); \mathbf{A} := \text{ExpandA}(\text{seed})$

2: $\llbracket \mathbf{r} \rrbracket := (0, \dots, 0) \in (\mathcal{R}_q^\ell)^d$

3: **for** $(i, i_{\text{rep}}, j) \in [\ell] \times [\text{rep}] \times [d]$ **do**

4: $\rho_{\mathbf{r}, i, i_{\text{rep}}, j} \leftarrow \text{RSU}(u, 1)$

5: $\mathbf{r}_{j, i} \leftarrow \mathbf{r}_{j, i} + \rho_{\mathbf{r}, i, i_{\text{rep}}, j}$

6: $\llbracket \mathbf{w} \rrbracket := \mathbf{A} \cdot \llbracket \mathbf{r} \rrbracket \in (\mathcal{R}_q^k)^d$

7: **for** $(i, i_{\text{rep}}, j) \in [k] \times [\text{rep}] \times [d]$ **do**

8: $\rho_{e', i, i_{\text{rep}}, j} \leftarrow \text{RSU}(u, 1)$

9: $\mathbf{w}_{j, i} \leftarrow \mathbf{w}_{j, i} + \rho_{e', i, i_{\text{rep}}, j}$

10: $\mathbf{w} := \text{Decode}(\llbracket \mathbf{w} \rrbracket)$

11: $\mathbf{w} := \lfloor \mathbf{w} \rfloor_{\mathcal{Y}_{\mathbf{w}}}$

12: $c_{\text{hash}} := \text{ChalHash}(\mathbf{w}, \mu)$

13: $c_{\text{poly}} := \text{ChalPoly}(c_{\text{hash}})$

14: $\llbracket \text{s} \rrbracket \leftarrow \text{Refresh}(\llbracket \text{s} \rrbracket)$

15: $\text{s} := \text{Decode}(\llbracket \text{s} \rrbracket)$

16: $\mathbf{z} := c_{\text{poly}} \cdot \text{s} + \mathbf{r}$

17: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - 2^{v_t} \cdot c_{\text{poly}} \cdot \mathbf{t}$

18: $\mathbf{h} := \mathbf{w} - \lfloor \mathbf{y} \rfloor_{\mathcal{Y}_{\mathbf{w}}}$

19: $\text{sig} := (c_{\text{hash}}, \mathbf{h}, \mathbf{z})$

20: **if** $\{\text{CheckBounds}(\text{sig}) = \text{FAIL}\}$ **goto** Line 4

21: $\mathcal{L} := \left\{ (\rho_{\mathbf{r}, i, i_{\text{rep}}, j}, \rho_{e', i', i'_{\text{rep}}, j'}, \mathbf{s}_{j''}) \right\}_{(\bar{\rho}_{\mathbf{r}, i, i_{\text{rep}}, j}, \bar{\rho}_{e', i', i'_{\text{rep}}, j'}, \bar{\mathbf{s}}_{j''}) \in \mathcal{F}}$

22: **return** $(\text{sig}, \mathcal{L})$

E.2 Completing the Proof of Theorem 1

The following lemma is the final component required to prove Theorem 1, establishing that for any PPT adversary \mathcal{A} against the game described in Hybrid₄ (cf. Figure 10), we can construct an adversary \mathcal{B} against the standard EUF-CMA security of small Raccoon in Figure 7.

Lemma 20. *Let $\bar{B}_\infty \geq B_\infty + \omega \cdot (d-1) \cdot \left(\frac{1}{2} + \frac{2^{3u_t}}{3}\right) \cdot (\kappa + \log(n(k+\ell))) + 2^{v_w} + \omega \cdot 2^{v_t}$, let $\bar{B}_2 \geq B_2 + \omega \cdot \sqrt{n(k+\ell)} \cdot (d-1) \cdot \left(\frac{1}{2} + \frac{2^{3u_t}}{3}\right) \cdot (\kappa + \log(n(k+\ell))) + 2^{v_w} \cdot \sqrt{nk} + \omega \cdot 2^{v_t} \cdot \sqrt{nk}$. Let \mathcal{A} be a PPT adversary against the game defined in Hybrid₄ (cf. Figure 10), then there exists a PPT adversary \mathcal{B} against the EUF-CMA security game for small Raccoon in Figure 7 with advantage:*

$$\text{Adv}_{\mathcal{B}} \geq \text{Adv}_{\mathcal{A}} - 4Q_H Q_S \cdot 2^{-H_\infty} - 2^{-\kappa} - \frac{1}{|\mathcal{C}|} - 2^{-\kappa}$$

Proof. For clarity we will assume that all probes queried by \mathcal{A} are on variables $\bar{\rho}_{b,i,i_{\text{rep}},j}$ in some **AddRepNoise** gadget and that for any $\bar{\rho}_{b,i,i_{\text{rep}},j} \in \bar{\mathcal{F}}$ we have that $\bar{\rho}_{b,i',i_{\text{rep}},j} \in \bar{\mathcal{F}}$ for all $i' \neq i$, i.e. we abstract away the applications of **SimIn**, **Extend**, **SimOut**, and **Collapse**. When \mathcal{A} queries $\text{OKeyGen}(\bar{\mathcal{F}}_s, \bar{\mathcal{F}}_e)$, as per Figure 10, \mathcal{B} will run the following algorithm.

Algorithm 18 $\text{KeyGen}_{\mathcal{G}}((\bar{\mathcal{F}}_s, \bar{\mathcal{F}}_e))$

1: $(\mathbf{A}, \bar{\mathbf{v}}\mathbf{k}) \leftarrow \mathcal{O}^{\text{KeyGen}_{\text{SMALL}}}()$ 2: $\bar{\mathbf{s}} \leftarrow \text{SU}(d-1 - \bar{\mathcal{F}}_s /\ell, u_t)^\ell$ 3: $\bar{\mathbf{e}} \leftarrow \text{SU}(d-1 - \bar{\mathcal{F}}_e /k, u_t)^k$ 4: for $(i_{\text{rep}}, j) \in [\text{rep}] \times [d]$ such that $\bar{\rho}_{s,1,i_{\text{rep}},j} \in \bar{\mathcal{F}}_1$ do 5: $(\rho_{s,1,i_{\text{rep}},j}, \dots, \rho_{s,\ell,i_{\text{rep}},j}) \leftarrow \text{RSU}(u, 1)^\ell$ 6: $\bar{\mathbf{s}} := \bar{\mathbf{s}} + (\rho_{s,1,i_{\text{rep}},j}, \dots, \rho_{s,\ell,i_{\text{rep}},j})$ 7: for $(i_{\text{rep}}, j) \in [\text{rep}] \times [d]$ such that $\bar{\rho}_{e,1,i_{\text{rep}},j} \in \bar{\mathcal{F}}_e$ do 8: $(\rho_{e,1,i_{\text{rep}},j}, \dots, \rho_{e,\ell,i_{\text{rep}},j}) \leftarrow \text{RSU}(u, 1)^k$ 9: $\bar{\mathbf{e}} := \bar{\mathbf{e}} + (\rho_{e,1,i_{\text{rep}},j}, \dots, \rho_{e,\ell,i_{\text{rep}},j})$ 10: $\mathbf{t} := \lfloor \bar{\mathbf{t}} + \bar{\mathbf{t}} \rfloor_{\gamma_t}$ 11: return $(\mathbf{t}, \mathcal{L} := (\rho_{b,i,i_{\text{rep}},j})_{\bar{\rho}_{b,i,i_{\text{rep}},j} \in \bar{\mathcal{F}}}, \bar{\mathbf{s}}, \bar{\mathbf{e}})$	<div style="display: flex; align-items: center;"> <div style="font-size: 3em; margin-right: 10px;">}</div> <div> <p style="margin: 0;">▷ Call to $\text{KeyGen}_{\text{SMALL}}$ (Algorithm 12)</p> <p style="margin: 0;">We simulate the distribution of $\text{KeyGen}_{\mathcal{G}}$.</p> </div> </div>
--	---

\mathcal{B} then returns $(\mathbf{t}, \mathcal{L})$ to \mathcal{A} and keeps $(\bar{\mathbf{s}}, \bar{\mathbf{e}})$ which will be necessary when signing. Observe that if $\bar{\mathbf{t}}$ comes from $\text{KeyGen}_{\text{SMALL}}$ then $\mathbf{t} = \lfloor \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \rfloor_{\gamma_w}$ where \mathbf{s} and \mathbf{e} are the sums of $N = \text{rep} \cdot d$ small uniforms, exactly as in $\text{KeyGen}_{\mathcal{G}}$, the leakage \mathcal{L} is also identical to the one in $\text{KeyGen}_{\mathcal{G}}$, note as well that since $|\bar{\mathcal{F}}_s| + |\bar{\mathcal{F}}_e| \leq d-1$, lines 2& 3 of the previous algorithm always make sense. When \mathcal{A} queries $\text{OSgn}(\text{msg}, \bar{\mathcal{F}})$, \mathcal{B} will run the following algorithm.

Algorithm 19 $\text{Sign}_{\mathcal{G}}(\text{msg}, \bar{\mathcal{F}})$

```

1:  $(\bar{\mathcal{F}}_{\mathbf{r}}, \bar{\mathcal{F}}_{e'}, \bar{\mathcal{F}}_{sk}) = \bar{\mathcal{F}}$ 
2:  $(\bar{c}_{\text{hash}}, \bar{\mathbf{z}}, \bar{\mathbf{h}}) \leftarrow \text{OSgn}_{\text{SMALL}}(\text{msg})$  ▷ Call to  $\text{Sign}_{\text{SMALL}}$  (Algorithm 13)
3:  $\bar{c}_{\text{poly}} := \text{ChalPoly}(\bar{c}_{\text{hash}})$ 
4:  $\mu := H(H(\text{vk}) || \text{msg})$ 
5:  $\bar{\mathbf{r}} \leftarrow \text{SU}(d-1-|\bar{\mathcal{F}}_{\mathbf{r}}|, u_w)^\ell$ 
6:  $\bar{\mathbf{e}}' \leftarrow \text{SU}(d-1-|\bar{\mathcal{F}}_{e'}|, u_w)^k$ 
7: for  $(i_{\text{rep}}, j) \in [\text{rep}] \times [d]$  such that  $\bar{\rho}_{\mathbf{r},1,i_{\text{rep}},j} \in \bar{\mathcal{F}}_{\mathbf{r}}$  do
8:    $(\rho_{\mathbf{r},1,i_{\text{rep}},j}, \dots, \rho_{\mathbf{r},\ell,i_{\text{rep}},j}) \leftarrow \text{RSU}(u, 1)^\ell$ 
9:    $\bar{\mathbf{r}} := \bar{\mathbf{r}} + (\rho_{\mathbf{r},1,i_{\text{rep}},j}, \dots, \rho_{\mathbf{r},\ell,i_{\text{rep}},j})$ 
10: for  $(i_{\text{rep}}, j) \in [\text{rep}] \times [d]$  such that  $\bar{\rho}_{e',1,i_{\text{rep}},j} \in \bar{\mathcal{F}}_{e'}$  do
11:    $(\rho_{e',1,i_{\text{rep}},j}, \dots, \rho_{e',\ell,i_{\text{rep}},j}) \leftarrow \text{RSU}(u, 1)^k$ 
12:    $\bar{\mathbf{e}}' := \bar{\mathbf{e}}' + (\rho_{e',1,i_{\text{rep}},j}, \dots, \rho_{e',\ell,i_{\text{rep}},j})$ 
13:  $\bar{\mathbf{w}} := \mathbf{A} \cdot \bar{\mathbf{z}} - \bar{c}_{\text{poly}} - \bar{\mathbf{v}}\mathbf{k} + \bar{\mathbf{h}}$ 
14:  $\tilde{\mathbf{w}} := \mathbf{A} \cdot \bar{\mathbf{r}} + \bar{\mathbf{e}}'$ 
15:  $\tilde{\mathbf{z}} := \bar{\mathbf{s}} \cdot \bar{c}_{\text{poly}} + \bar{\mathbf{r}}$ 
16:  $\mathbf{w} := \lfloor \tilde{\mathbf{w}} + \tilde{\mathbf{w}} \rfloor_{v_w}$ 
17:  $\text{ChalHash}(\mathbf{w}, \mu) := \bar{c}_{\text{hash}}$ 
18:  $\mathbf{z} := \bar{\mathbf{z}} + \tilde{\mathbf{z}}$ 
19:  $\mathbf{h} := \mathbf{w} - \lfloor \mathbf{A} \cdot \mathbf{z} - \bar{c}_{\text{poly}} \cdot 2^{\nu_t} \cdot \mathbf{t} \rfloor_{v_w}$ 
20:  $\text{sig} := (\bar{c}_{\text{poly}}, \mathbf{z}, \mathbf{h})$ 
21: if  $\{\text{CheckBounds}(\text{sig}) = \text{FAIL}\}$  goto Line 4
22: return  $(\text{sig}, \mathcal{L} = ((\rho_{b,i,i_{\text{rep}},j})_{\bar{\rho}_{b,i,i_{\text{rep}},j} \in \bar{\mathcal{F}}}, \$))$ 

```

We simulate the distribution of $\text{Sign}_{\mathcal{G}}$.

First note that since $(\bar{c}_{\text{hash}}, \bar{\mathbf{z}}, \bar{\mathbf{h}})$ comes from $\text{Sign}_{\text{SMALL}}$ then $\mathbf{z} = (\bar{\mathbf{s}} + \tilde{\mathbf{s}}) \cdot \bar{c}_{\text{poly}} + \bar{\mathbf{r}} + \tilde{\mathbf{r}}$, and $\mathbf{w} = \lfloor \mathbf{A} \cdot (\bar{\mathbf{r}} + \tilde{\mathbf{r}}) + \bar{\mathbf{e}}' + \tilde{\mathbf{e}}' \rfloor_{v_w}$ are distributed exactly as in $\text{Sign}_{\mathcal{G}}$, thus so is \mathbf{h} . When \mathcal{A} directly queries $\text{ChalHash}(\mathbf{x}, y)$, \mathcal{B} checks that $\text{ChalHash}(\mathbf{x}, y)$ is not set and queries $\bar{c}_{\text{hash}} := \text{ChalHash}(2^{\nu_w} \cdot \mathbf{x}, y)$ and sets $\text{ChalHash}(\mathbf{x}, y) := \bar{c}_{\text{hash}}$. When \mathcal{A} outputs a forgery $(c_{\text{hash}}^*, \mathbf{z}^*, \mathbf{h}^*)$ for message msg^* , let $c_{\text{poly}} := \text{ChalPoly}(c_{\text{hash}})$, $\bar{\mathbf{z}} := \mathbf{z}^* - c_{\text{poly}} \cdot \bar{\mathbf{s}}$, and $\bar{\mathbf{h}} := 2^{\nu_w} \mathbf{h}^* - c_{\text{poly}} \cdot \bar{\mathbf{e}} + \delta_1 + c_{\text{poly}} \cdot \delta_2$, where $\delta_1 := 2^{\nu_w} \cdot \lfloor \mathbf{A} \cdot \mathbf{z}^* - c_{\text{poly}}^* \cdot 2^{\nu_t} \cdot \mathbf{t} \rfloor_{v_w} - (\mathbf{A} \cdot \mathbf{z}^* - c_{\text{poly}}^* \cdot 2^{\nu_t} \cdot \mathbf{t})$, and $\delta_2 := 2^{\nu_t} \lfloor \tilde{\mathbf{t}} + \mathbf{A} \cdot \tilde{\mathbf{s}} + \tilde{\mathbf{e}} \rfloor_{v_t} - (\tilde{\mathbf{t}} + \mathbf{A} \cdot \tilde{\mathbf{s}} + \tilde{\mathbf{e}})$. \mathcal{B} then checks that $(c_{\text{hash}}^*, \mathbf{z}^*, \mathbf{h}^*)$ is a valid forgery for $\text{Sign}_{\mathcal{G}}$ and outputs his forgery $(c_{\text{hash}}, \bar{\mathbf{z}}, \bar{\mathbf{h}})$. We first prove that \mathcal{A} cannot distinguish between \mathcal{B} and the challenger of Figure 10. We have shown that the keys and signatures output by \mathcal{B} come from the appropriate distribution unless \mathcal{B} aborts because it is required to program ChalHash on a value that was already queried by the adversary. If we set p the probability that CheckBounds succeeds (which is constant and identical in $\text{Sign}_{\mathcal{G}}$ and $\text{Sign}_{\mathcal{G}}$), using Lemma 13 (for our parameters we always have $p > 1/2$) we have that the total number of queries done to ChalHash during signature is less than $\frac{2Q_s}{p}$ with probability $1 - 2^{-pQ_s} > 1 - 2^{-\kappa}$, since each call to ChalHash is done on a vector \mathbf{w} from $[\text{LWE}]$ and at most Q_H values are set at any time, the probability that \mathcal{B} aborts

because the input to `ChalHash` was already set is at most:

$$\begin{aligned}
\Pr[\text{BAD}] &= \Pr[\text{BAD} | Q_{S,\text{total}} \leq 2Q_S/p] \cdot \Pr[Q_{S,\text{total}} \leq 2Q_S/p] \\
&\quad + \Pr[\text{BAD} | Q_{S,\text{total}} > 2Q_S/p] \cdot \Pr[Q_{S,\text{total}} > 2Q_S/p] \\
&\leq (1 - 2^{-\kappa}) \cdot (1 - (1 - Q_H \cdot 2^{-H_\infty})^{2Q_S/p}) + 2^{-\kappa} \\
&\leq \frac{Q_H \cdot Q_S}{p} \cdot 2^{-H_\infty+1} + 2^{-\kappa}
\end{aligned}$$

We now show that when the event `BAD` does not occur the forgery output by \mathcal{B} is valid with overwhelming probability. We first note that since $(c_{\text{hash}}^*, \mathbf{z}^*, \mathbf{h}^*)$ is a valid forgery for `msg*`, `OSgn` was never queried on `msg*`. Let $\mathbf{w}^* = \lfloor \mathbf{A} \cdot \mathbf{z}^* - c^* \cdot 2^{\nu_t} \cdot \mathbf{t} \rfloor_{\nu_w} + \mathbf{h}^*$, if \mathcal{A} never queried `ChalHash(w*, msg)` and it is consequently not set, then the probability of forging successfully is at most $1/|C|$. If \mathcal{A} queried `ChalHash(w*, msg)`, then

$$\text{ChalHash}(\mathbf{w}^*, \text{msg}) = \text{ChalHash}(2^{\nu_t} \cdot \mathbf{w}^*, \text{msg}) = \text{ChalHash}(\mathbf{A} \cdot \bar{\mathbf{z}} - c \cdot \bar{\mathbf{t}} + \bar{\mathbf{h}}, \text{msg})$$

where $(\bar{\mathbf{z}}, \bar{\mathbf{h}}) = (\mathbf{z}^* - c_{\text{poly}} \cdot \tilde{\mathbf{s}}, 2^{\nu_w} \mathbf{h}^* - c_{\text{poly}} \cdot \tilde{\mathbf{e}} + \delta_1 + c_{\text{poly}} \cdot \delta_2)$ are as defined previously with $\|\delta_1\|_\infty \leq 2^{\nu_w}$ and $\|\delta_2\|_\infty \leq 2^{\nu_t}$. Using Lemma 12, we have:

$$\begin{aligned}
\|(\bar{\mathbf{z}}, \bar{\mathbf{h}})\| &\leq \|(\mathbf{z}^*, 2^{\nu_w} \mathbf{h}^*)\| + \|(c_{\text{poly}} \cdot \tilde{\mathbf{s}}, c_{\text{poly}} \cdot \tilde{\mathbf{e}} + \delta_1 + c_{\text{poly}} \cdot \delta_2)\| \\
&\leq B_2 + \omega \cdot \sqrt{n(k+\ell)} \cdot (d-1) \cdot \left(\frac{1}{2} + \frac{2^{3u_t}}{3}\right) \cdot (\kappa + \log(n(k+\ell))) + 2^{\nu_w} \cdot \sqrt{nk} + \omega \cdot 2^{\nu_t} \cdot \sqrt{nk} \\
&= \bar{B}_2
\end{aligned}$$

Similarly we have

$$\begin{aligned}
\|(\bar{\mathbf{z}}, \bar{\mathbf{h}})\|_\infty &\leq B_\infty + \omega \cdot (d-1) \cdot \left(\frac{1}{2} + \frac{2^{3u_t}}{3}\right) \cdot (\kappa + \log(n(k+\ell))) + 2^{\nu_w} + \omega \cdot 2^{\nu_t} \\
&= \bar{B}_\infty
\end{aligned}$$

Hence $(c, \bar{\mathbf{z}}, \bar{\mathbf{h}})$ is a valid forgery. \square

F Deferred Details in Section 7.3

This section establishes the standard EUF-CMA security of small Raccoon in Figure 7.

F.1 Asymptotic Parameter Selection and Preparation

To formally rely on the smooth Rényi divergence and the hardness assumptions, we must first provide a set of candidate asymptotic parameters for which the EUF-CMA security proof holds.

Random oracle model. The small Raccoon signature relies on two hash functions $H : \{0, 1\}^* \rightarrow \{0, 1\}^{2\kappa}$ and $G : \mathcal{R}_{q_{vt}}^k \times \{0, 1\}^{2\kappa} \rightarrow C$, where C is the challenge space defined as

$$C = \{c \in \mathcal{R}_q \mid \|c\|_\infty = 1 \wedge \|c\|_1 = \omega\}. \quad (27)$$

It further relies on the `ExpandA` function serving as a pseudorandom number generator. These hash functions and `ExpandA` are modeled as random oracles throughout the security proof.

Constraints on parameters. We then give the intermediate variables that will be used during the proof and their value when applicable. Below, denote $\omega_{\text{asympt}}(f)$ the class of functions that grows asymptotically faster than f . Also, denote $T = d(\text{rep} - 1) + 1$ below.

- $B_{u_t, \infty}^{\text{sRD}}, B_{u_t, 2}^{\text{sRD}}$ bounds on the L_∞, L_2 -norm, respectively, of $c \cdot (\mathbf{s}, \mathbf{e})$ for any $c \in C$ and $(\mathbf{s}, \mathbf{e}) \leftarrow \text{RSU}(u_t, T)^\ell \times \text{RSU}(u_t, T)^k$,
- $B_{u_w, \infty}^{\text{sRD}}, B_{u_w, 2}^{\text{sRD}}$ bounds on the L_∞, L_2 -norm, respectively, of $(\mathbf{r}, \mathbf{e}') \leftarrow \text{RSU}(u_w, T)^\ell \times \text{RSU}(u_w, T)^k$,
- $\beta = \sqrt{\omega} + \bar{B}_2$,
- α the order used in the smooth Rényi divergence,
- ϵ_{TAIL} the statistical component that will be used in the smooth Rényi divergence argument,
- $\epsilon_{\text{Adv}} = \text{Adv}_{\mathcal{B}}^{\text{MLWE}} + \text{Adv}_{\mathcal{B}'}^{\text{SelfTargetMSIS}} + \epsilon_{\text{negl}}$, for Lemma 22 where \mathcal{B} and \mathcal{B}' are constructed from the EUF-CMA adversary \mathcal{A} with similar advantages as \mathcal{A} , and a fixed negligible function ϵ_{negl} .

We now list the constraints which will appear in the proof:

- $\text{Adv}_{\mathcal{B}}^{\text{MLWE}} = \text{negl}(\kappa)$. I.e. $\frac{T(2^{2u_t} - 1)}{12} \geq \sqrt{\ell} \cdot \omega_{\text{asympt}}(\sqrt{\log n})$, using Eq. (12) and Lemma 9.
 - $\text{Adv}_{\mathcal{B}'}^{\text{SelfTargetMSIS}} = \text{negl}(\kappa)$. I.e. $\beta' \sqrt{n(\ell + 1)} \cdot \omega_{\text{asympt}}(\sqrt{\log(n\ell)}) \leq q$ where $\beta' = 2\beta$, using Lemmas 10 and 11.
 - $\alpha B_{u_t, \infty}^{\text{sRD}} = o\left(\frac{2^{u_w}}{T-1}\right)$, $\alpha = \omega_{\text{asympt}}(1)$ and $\epsilon_{\text{TAIL}} = \frac{(\alpha B_{u_t, \infty}^{\text{sRD}} + T)^T}{2^{u_w \cdot T} T!} = \text{negl}(\kappa)$. So that we can use the smooth Rényi divergence as per Lemma 3 and Conjecture 1.
 - $\alpha = \frac{2^{u_w}}{B_{u_t, 2}^{\text{sRD}}} \sqrt{\frac{-\log(\epsilon_{\text{Adv}})T}{C_{\text{RÉNYI}} Q_s}}$, $\frac{B_{u_t, 2}^{\text{sRD}}}{2^{u_w}} \cdot \sqrt{\frac{-C_{\text{RÉNYI}} \cdot \log(\epsilon_{\text{Adv}}) \cdot Q_s}{T}} = O(\log \kappa)$, and $Q_s \cdot \epsilon_{\text{TAIL}} \leq \epsilon_{\text{negl}}$. So we can use Lemma 22.
 - $B_{u_t, \infty}^{\text{sRD}} = \omega \cdot \left(\frac{T}{2} + \delta_{u_t}\right)$, $B_{u_t, 2}^{\text{sRD}} = \sqrt{n(\ell + k)} \cdot B_{u_t, \infty}^{\text{sRD}}$, and $\delta_{u_t} = 2^{u_t} \sqrt{\frac{T}{3} \cdot \kappa + \log(Q_s \cdot n(\ell + k)) \cdot \log(2)}$.
- For Lemma 12.
- $B_{u_w, \infty}^{\text{sRD}} = 2^{u_w} \cdot T$, and $B_{u_w, 2}^{\text{sRD}} = 2^{u_w} \sqrt{3T \cdot n(\ell + k)}$ for Eq. (11).
 - $B_2 = B_{u_w, \infty}^{\text{sRD}} + B_{u_w, 2}^{\text{sRD}} + 2^{v_w} \cdot \sqrt{nk}$ for overwhelming correctness.

Candidate asymptotic parameters. Finally, we give a set of asymptotic parameters which fit the above constraints. It is worth noting that the only parameters that may depend on the EUF-CMA adversary \mathcal{A} are α , ϵ_{TAIL} , and ϵ_{Adv} used in the security proof. All other parameters are scheme specific and defined independently of \mathcal{A} .

- $n, \ell, k = \text{poly}(\kappa)$ such that $n \geq \kappa$,
- $Q_h = \text{poly}(\kappa)$: the maximum number of hash queries is any unbounded polynomial,
- $Q_s = \text{poly}(\kappa)$: the maximum number of signing queries is polynomially bounded, i.e., the parameter of the scheme depends on Q_s . Without loss of generality, we assume $Q_s \geq \kappa$,
- $T = d \cdot (\text{rep} - 1) + 1 = \omega_{\text{asyp}}(1)$, e.g., $T = \sqrt{\log \kappa}$,
- $\omega = \omega_{\text{asyp}}(1)$, e.g., $\omega = \log \kappa$,
- $\epsilon_{\text{negl}} = 2^{-\kappa}$,
- $\nu_t, \nu_w = O(\log \kappa)$. From which, we get $\beta, \beta' = \text{poly}(\kappa)$, and set polynomially sized modulus q such that $\beta' \sqrt{n(\ell + 1)} \cdot \omega_{\text{asyp}}(\log(n\ell)) \leq q$,
- $2^{u_t} = \sqrt[4]{\ell \cdot \log \kappa}$,
- $2^{u_w} = B_{u_t, 2}^{\text{sRD}} \sqrt{\frac{C_{\text{RÉNYI}} Q_s}{T}} \cdot \sqrt{\frac{\kappa}{\log \kappa}} \cdot n(\ell + k)$. From which we get the condition on Lemma 22, as well as $\epsilon_{\text{TAIL}} = \text{negl}(\kappa)$ since

$$\epsilon_{\text{TAIL}} \leq \left(\frac{\alpha B_{u_t, \infty}^{\text{sRD}}}{2^{u_w}} + \frac{1}{\sqrt{\kappa}} \right)^T \leq \left(\sqrt{\frac{-\log(\epsilon_{\text{Adv}}) \cdot T}{Q_s \cdot n(\ell + k)}} + \frac{1}{\sqrt{\kappa}} \right)^T \leq \left(\frac{2}{\sqrt[4]{\kappa}} \right)^{\sqrt{\log \kappa}} = \text{negl}(\kappa).$$

Where the first inequality comes from $T \cdot \kappa^{1/2} \leq 2^{u_w}$ and the second inequality comes from $B_{u_t, 2}^{\text{sRD}} = \sqrt{n(\ell + k)} \cdot B_{u_t, \infty}^{\text{sRD}}$. The last fact comes from $Q_s \geq \kappa$, $T = \sqrt{\log \kappa}$, and the fact that we can assume $\epsilon_{\text{Adv}} \geq 2^{-\sqrt{\frac{\kappa}{\log \kappa}} \cdot n(\ell + k)}$ as any adversary against SelfTargetMSIS with $\beta = \text{poly}(\kappa)$ can achieve better advantage than ϵ_{Adv} by random guessing.⁶ Following a similar computation, $\frac{B_{u_t, 2}^{\text{sRD}}}{2^{u_w}} \cdot \sqrt{\frac{-C_{\text{RÉNYI}} \cdot \log(\epsilon_{\text{Adv}}) \cdot Q_s}{T}} \leq 1$. Lastly, we have $Q_s \cdot \epsilon_{\text{TAIL}} \leq \epsilon_{\text{negl}}$,

- Using how we set 2^{u_w} , $\alpha = \frac{2^{u_w}}{B_{u_t, 2}^{\text{sRD}}} \sqrt{\frac{-\log(\epsilon_{\text{Adv}}) T}{C_{\text{RÉNYI}} Q_s}} = \sqrt{-\log(\epsilon_{\text{Adv}})} \leq \sqrt[4]{\kappa} \cdot \sqrt{n(\ell + k)}$.

From this we get $\alpha B_{u_t, \infty}^{\text{sRD}} = o\left(\frac{2^{u_w}}{T-1}\right)$. Moreover, assuming the hardness of MLWE and SelfTargetMSIS, we can bound $\epsilon_{\text{Adv}} \leq \kappa^{-1}$, which establishes $\alpha \geq \log(\kappa) = \omega_{\text{asyp}}(1)$.

Theorem 3. *The small Raccoon in Figure 7 is EUF-CMA secure under the MLWE $_{q, \ell, k, \text{SU}(u_t, T)}$ and 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 ϵ_{TAIL} , there exists adversaries \mathcal{B} and \mathcal{B}' against the MLWE $_{q, \ell, k, \text{SU}(u_t, T)}$ and SelfTargetMSIS $_{q, \ell+1, k, C, \beta}$ problems such that

$$\begin{aligned} \text{Adv}_{\mathcal{A}}^{\text{EUF-CMA}} &\leq 2^{-\kappa} \cdot Q_h \cdot (1 + 2^{-\kappa+1} \cdot Q_s) + Q_s \cdot \epsilon_{\text{TAIL}} \\ &\quad + \left(\text{Adv}_{\mathcal{B}}^{\text{MLWE}} + \text{Adv}_{\mathcal{B}'}^{\text{SelfTargetMSIS}} + Q_s \cdot \epsilon_{\text{TAIL}} \right)^{\frac{\alpha-1}{\alpha}} \cdot (R_{\alpha}^{\epsilon_{\text{TAIL}}}(\mathcal{P}; \mathcal{Q}))^{Q_s}, \end{aligned} \tag{28}$$

⁶ Note that when setting concrete parameters, we can use a lower bound derived from the best known attack against the ExtMLWE and SelfTargetMSIS problems.

where $\text{Time}(\mathcal{A}) \approx \text{Time}(\mathcal{B}) \approx \text{Time}(\mathcal{B}')$. Concretely, plugging in our candidate asymptotic parameters in Appendix F.1, we conclude $\text{Adv}_{\mathcal{A}}^{\text{EUF-CMA}}$ is bounded by $\text{negl}(\kappa)$.

Hybrid ₀	OSgn(msg)
1: $\text{seed} \leftarrow \{0, 1\}^\kappa$	1: $\mu := \text{H}(\text{H}(\text{vk}) \parallel \text{msg})$
2: $\mathbf{A} := \text{ExpandA}(\text{seed})$	2: $\mathbf{r} \leftarrow \text{RSU}(u_{\mathbf{w}}, N - d + 1)^\ell$
3: $\mathbf{s} \leftarrow \text{RSU}(u_{\mathbf{t}}, N - d + 1)^\ell$	3: $\mathbf{e}' \leftarrow \text{RSU}(u_{\mathbf{w}}, N - d + 1)^k$
4: $\mathbf{e} \leftarrow \text{RSU}(u_{\mathbf{t}}, N - d + 1)^k$	4: $\mathbf{w} := \mathbf{A} \cdot \mathbf{r} + \mathbf{e}'$
5: $\mathbf{t} := \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$	5: $c_{\text{poly}} := \text{G}(\mathbf{w}, \mu)$
6: $\text{vk} := (\mathbf{A}, \mathbf{t})$	6: $\mathbf{z} := c_{\text{poly}} \cdot \mathbf{s} + \mathbf{r}$
7: $Q_{\text{Sign}} := \emptyset$	7: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - c_{\text{poly}} \cdot \mathbf{t}$
8: $(\text{msg}^*, \text{sig}^*) \leftarrow \mathcal{A}^{\text{OSgn}()}(\text{vk})$	8: $\mathbf{h} := \mathbf{w} - \mathbf{y}$
9: if $\exists \text{sig}'$ s.t. $(\text{msg}^*, \text{sig}') \in Q_{\text{Sign}}$ re- turn FAIL	9: $\text{sig} := (c_{\text{poly}}, \mathbf{h}, \mathbf{z})$
10: return $\text{Verify}(\text{sig}^*, \text{msg}^*, \text{vk})$	10: if $\text{CheckBounds}(\text{sig}) = \text{FAIL}$ goto Line 2
	11: $Q_{\text{Sign}} := Q_{\text{Sign}} \cup \{(\text{msg}, \text{sig})\}$
	12: return sig

Fig. 13: First hybrid Hybrid₀.

Proof. Below, we consider a sequence of hybrids, where the first hybrid is the original game and the last is a game that can be reduced from the SelfTargetMSIS problem. We relate the advantage of \mathcal{A} for each adjacent hybrids.

Hybrid₀: This is the original EUF-CMA security game (cf. Remark 2). See Figure 13. For ease of reading, we use the hash function G that corresponds to $\text{ChalPoly} \circ \text{ChalHash}$ to sample $c_{\text{poly}} := \text{G}(\mathbf{w}, \mu)$. Without loss of generality, throughout the security proof, we assume c_{poly} is included in the signature sig , as opposed to c_{hash} .

$$\text{Adv}_{\mathcal{A}}^{\text{Hybrid}_0} = \text{Adv}_{\mathcal{A}}^{\text{EUF-CMA}}.$$

Hybrid₁: In this hybrid, the challenger samples \mathbf{A} uniformly at random from its target set $\mathcal{R}_q^{k \times \ell}$ and programs $\text{ExpandA}(\text{seed}) := \mathbf{A}$. This is depicted in Figure 14. As ExpandA is modeled as a random oracle and there are at most Q_h random oracle queries, the probability that the programming of the random oracle fail is bounded by $Q_h \cdot 2^{-\kappa}$. Thus, we have

$$\left| \text{Adv}_{\mathcal{A}}^{\text{Hybrid}_1} - \text{Adv}_{\mathcal{A}}^{\text{Hybrid}_0} \right| \leq Q_h \cdot 2^{-\kappa}.$$

Hybrid₂: In this hybrid, the challenger adds a winning condition. This is depicted in Figure 14. Namely, when the adversary produces a forgery on a

<hr/> <p>Hybrid₁</p> <ol style="list-style-type: none"> 1: seed $\leftarrow \{0, 1\}^\kappa$ 2: $\mathbf{A} \leftarrow \mathcal{R}_q^{k \times \ell}$ 3: $\text{ExpandA}(\text{seed}) := \mathbf{A}$ 4: $\mathbf{s} \leftarrow \text{RSU}(u_t, N - d + 1)^\ell$ 5: $\mathbf{e} \leftarrow \text{RSU}(u_t, N - d + 1)^k$ 6: $\mathbf{t} := \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ 7: $\text{vk} := (\text{seed}, \mathbf{t})$ 8: $Q_{\text{Sign}} := \emptyset$ 9: $(\text{msg}^*, \text{sig}^*) \leftarrow \mathcal{A}^{\text{OSgn}(\cdot)}(\text{vk})$ 10: if $\exists \text{sig}'$ s.t. $(\text{msg}^*, \text{sig}') \in Q_{\text{Sign}}$ return FAIL 11: return Verify(sig[*], msg[*], vk) <hr/>	<hr/> <p>Hybrid₂</p> <ol style="list-style-type: none"> 1: seed $\leftarrow \{0, 1\}^\kappa$ 2: $\mathbf{A} \leftarrow \mathcal{R}_q^{k \times \ell}$ 3: $\text{ExpandA}(\text{seed}) := \mathbf{A}$ 4: $\mathbf{s} \leftarrow \text{RSU}(u_t, N - d + 1)^\ell$ 5: $\mathbf{e} \leftarrow \text{RSU}(u_t, N - d + 1)^k$ 6: $\mathbf{t} := \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ 7: $\text{vk} := (\text{seed}, \mathbf{t})$ 8: $Q_{\text{Sign}} := \emptyset$ 9: $(\text{msg}^*, \text{sig}^*) \leftarrow \mathcal{A}^{\text{OSgn}(\cdot)}(\text{vk})$ 10: if $\exists (\text{msg}, \cdot) \in Q_{\text{Sign}} : \text{H}(\text{H}(\text{vk}) \parallel \text{msg}^*) = \text{H}(\text{H}(\text{vk}) \parallel \text{msg})$ return FAIL 11: if $\exists \text{sig}'$ s.t. $(\text{msg}^*, \text{sig}') \in Q_{\text{Sign}}$ return FAIL 12: return Verify(sig[*], msg[*], vk) <hr/>
<hr/> <p>Hybrid₃</p> <hr/>	<hr/> <p>Hybrid₄</p> <hr/>
<hr/> <p>OSgn(msg)</p> <ol style="list-style-type: none"> 1: $\mu := \text{H}(\text{H}(\text{vk}) \parallel \text{msg})$ 2: $\mathbf{r} \leftarrow \text{RSU}(u_w, N - d + 1)^\ell$ 3: $\mathbf{e}' \leftarrow \text{RSU}(u_w, N - d + 1)^k$ 4: $\mathbf{w} := \mathbf{A} \cdot \mathbf{r} + \mathbf{e}'$ 5: $c_{\text{poly}} \leftarrow \mathcal{C}$ 6: $\mathbf{z} := c_{\text{poly}} \cdot \mathbf{s} + \mathbf{r}$ 7: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - c_{\text{poly}} \cdot \mathbf{t}$ 8: $\mathbf{h} := \mathbf{w} - \mathbf{y}$ 9: $\mathbf{G}(\mathbf{w}, \mu) := c_{\text{poly}} \triangleright$ Abort if already programmed 10: $\text{sig} := (c_{\text{poly}}, \mathbf{h}, \mathbf{z})$ 11: if CheckBounds(sig) = FAIL goto Line 2 12: $Q_{\text{Sign}} := Q_{\text{Sign}} \cup \{(\text{msg}, \text{sig})\}$ 13: return sig <hr/>	<hr/> <p>OSgn(msg)</p> <ol style="list-style-type: none"> 1: $\mu := \text{H}(\text{H}(\text{vk}) \parallel \text{msg})$ 2: $\mathbf{r} \leftarrow \text{RSU}(u_w, N - d + 1)^\ell$ 3: $\mathbf{e}' \leftarrow \text{RSU}(u_w, N - d + 1)^k$ 4: $c_{\text{poly}} \leftarrow \mathcal{C}$ 5: $\mathbf{z} := c_{\text{poly}} \cdot \mathbf{s} + \mathbf{r}$ 6: $\mathbf{z}' := c_{\text{poly}} \cdot \mathbf{e} + \mathbf{e}'$ 7: $\mathbf{w} := \mathbf{A} \cdot \mathbf{z} - c_{\text{poly}} \cdot \mathbf{t} + \mathbf{z}'$ 8: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - c_{\text{poly}} \cdot \mathbf{t}$ 9: $\mathbf{h} := \mathbf{w} - \mathbf{y}$ 10: $\mathbf{G}(\mathbf{w}, \mu) := c_{\text{poly}} \triangleright$ Abort if already programmed 11: $\text{sig} := (c_{\text{poly}}, \mathbf{h}, \mathbf{z})$ 12: if CheckBounds(sig) = FAIL goto Line 2 13: $Q_{\text{Sign}} := Q_{\text{Sign}} \cup \{(\text{msg}, \text{sig})\}$ 14: return sig <hr/>

Fig. 14: Hybrid₁ to Hybrid₄ used in the proof of Theorem 3. Differences from Hybrid_{*i*-1} to Hybrid_{*i*} are highlighted.

message msg that provokes a collision in $H(H(\text{vk})\|\text{msg}^*)$ for a message msg^* previously queried to the signing oracle, the challenger does not view this as a valid forgery. Since $H : \{0, 1\}^* \rightarrow \{0, 1\}^{2\kappa}$ is modeled as a random oracle, this event happens with probability at most $Q_s \cdot 2^{-2\kappa}$:

$$\left| \text{Adv}_{\mathcal{A}}^{\text{Hybrid}_2} - \text{Adv}_{\mathcal{A}}^{\text{Hybrid}_1} \right| \leq Q_s \cdot 2^{-2\kappa}.$$

Hybrid₃: In this hybrid, the challenger replaces non-programmed random oracle outputs in the signing oracle with programmed outputs. Namely, it first samples an element c_{poly} uniformly at random from the challenge space \mathcal{C} . Then it programs the hash function to consistently return this value c_{poly} on input (μ, \mathbf{w}) during further interactions with the adversary. This is depicted in Figure 14.

Note that the signing responses in **Hybrid₂** are identically distributed to **Hybrid₁** unless $\text{OSgn}(\cdot)$ is required to program a value that has already been queried by the adversary. As \mathbf{w} is sampled randomly following the BD-MLWE distribution without any rounding as in Definition 12, this happens with probability at most $Q_h \cdot 2^{-H_\infty(\text{BD-MLWE})}$ in each signing query. Thus it follows that

$$\begin{aligned} \left| \text{Adv}_{\mathcal{A}}^{\text{Hybrid}_3} - \text{Adv}_{\mathcal{A}}^{\text{Hybrid}_2} \right| &\leq 1 - \left(1 - Q_h \cdot 2^{-H_\infty(\text{BD-MLWE})} \right)^{Q_s} \\ &\leq Q_s \cdot Q_h \cdot 2^{-H_\infty(\text{BD-MLWE})}, \end{aligned}$$

where we have used Bernoulli's inequality and $Q_h < 2^{H_\infty(\text{BD-MLWE})}$ from Conjecture 2.

Hybrid₄: In this hybrid, the challenger computes the commitment \mathbf{w} using the public key as opposed to an ephemeral LWE sample $\mathbf{A} \cdot \mathbf{r} + \mathbf{e}'$. This is depicted in Figure 14. As the challenger computes $\mathbf{w} = \mathbf{A} \cdot \mathbf{r} + \mathbf{e}' = \mathbf{A} \cdot \mathbf{z} - c_{\text{poly}} \cdot \mathbf{A} \cdot \mathbf{s} + \mathbf{e}'$ in the previous game, one can verify that

$$\mathbf{A} \cdot \mathbf{z} - c_{\text{poly}} \cdot \mathbf{A} \cdot \mathbf{s} + \mathbf{e}' = \mathbf{A} \cdot \mathbf{z} - c_{\text{poly}} \cdot \mathbf{t} + \underbrace{c_{\text{poly}} \cdot \mathbf{e} + \mathbf{e}'}_{=: \mathbf{z}'},$$

which yields the equation in **Hybrid₃**.

As it is simply a rewriting of \mathbf{w} , it remains indistinguishable from **Hybrid₃**:

$$\text{Adv}_{\mathcal{A}}^{\text{Hybrid}_4} = \text{Adv}_{\mathcal{A}}^{\text{Hybrid}_3}.$$

Hybrid₅: In this hybrid, the challenger computes the response $(\mathbf{z}, \mathbf{z}')$ without using the secret key \mathbf{s} or the noise \mathbf{e} . This is depicted in Figure 15. However, to do this the challenger has removed an explicit dependence on \mathbf{s}, \mathbf{e} in \mathbf{z} and \mathbf{z}' so the distribution of the signing responses are not statistically identical. We argue that the two distributions are indistinguishable for an adversary that can make no more than Q_s queries.

We recall the \mathcal{P} and $\mathcal{Q}(\text{center})$ defined in Appendix F.1. Let \mathcal{P} be the distribution $\text{SU}(\mathbf{u}_{\mathbf{w}}, T)^{n(\ell+k)}$ and $\mathcal{Q}(\text{center}_{\mathbf{q}_s})$ be the distribution $\text{center}_{\mathbf{q}_s} + \mathcal{P}$, where

$c_{q_s} \leftarrow C$ is the q_s -th ($q_s \in [Q_s]$) challenge used to respond to the signing oracle OSgn and $\text{center}_{q_s} := c_{q_s} \cdot \begin{bmatrix} \mathbf{s} \\ \mathbf{e} \end{bmatrix} \in \mathcal{R}_q^{\ell+k}$. Define $\mathcal{P}^* := \mathcal{P}^{Q_s}$ and let $\mathcal{Q}_{\text{centers}}^*$ be the tensorized distribution $\otimes_{q_s \in [Q_s]} \mathcal{Q}(\text{center}_{q_s})$. Then, \mathcal{P}^* and $\mathcal{Q}_{\text{centers}}^*$ correspond to the distributions of $(\mathbf{z}, \mathbf{z}')$ in Hybrid_5 and Hybrid_4 , respectively. Lastly, let $\epsilon_{\text{TAIL}, \text{centers}} := \sum_{q_s \in [Q_s]} \epsilon_{\text{TAIL}}(\text{center}_{q_s})$ where recall Appendix F.1 for the definition of $\epsilon_{\text{TAIL}}(\text{center}_{q_s})$.

We can now relate the advantage of this hybrid from the previous hybrid. Using the probability preservation property and the tensorization of the smooth Rényi divergence in Lemma 2, we have the following with overwhelming probability:

$$\begin{aligned}
\text{Adv}_{\mathcal{A}}^{\text{Hybrid}_4} &\leq (\text{Adv}_{\mathcal{A}}^{\text{Hybrid}_5} + \epsilon_{\text{TAIL}, \text{centers}})^{\frac{\alpha-1}{\alpha}} \cdot (R_{\alpha}^{\epsilon_{\text{TAIL}, \text{centers}}}(\mathcal{Q}_{\text{centers}}^*; \mathcal{P}^*)) + \epsilon_{\text{TAIL}, \text{centers}} \\
&\leq (\text{Adv}_{\mathcal{A}}^{\text{Hybrid}_5} + \epsilon_{\text{TAIL}, \text{centers}})^{\frac{\alpha-1}{\alpha}} \cdot \prod_{q_s \in [Q_s]} \left(R_{\alpha}^{\epsilon_{\text{TAIL}}(\text{center}_{q_s})}(\mathcal{Q}(\text{center}_{q_s}); \mathcal{P}) \right) \\
&\quad + \epsilon_{\text{TAIL}, \text{centers}} \\
&\leq (\text{Adv}_{\mathcal{A}}^{\text{Hybrid}_5} + \epsilon_{\text{TAIL}, \text{centers}})^{\frac{\alpha-1}{\alpha}} \cdot \prod_{q_s \in [Q_s]} \left(R_{\alpha}^{\epsilon_{\text{TAIL}}(\text{center}_{q_s})}(\mathcal{P}; \mathcal{Q}(\text{center}_{q_s})) \right) \\
&\quad + \epsilon_{\text{TAIL}, \text{centers}} \\
&\leq (\text{Adv}_{\mathcal{A}}^{\text{Hybrid}_5} + Q_s \cdot \epsilon_{\text{TAIL}})^{\frac{\alpha-1}{\alpha}} \cdot (R_{\alpha}^{\epsilon_{\text{TAIL}}}(\mathcal{P}; \mathcal{Q}))^{Q_s} + Q_s \cdot \epsilon_{\text{TAIL}},
\end{aligned}$$

where the third bound follows from Lemma 14 and the final bound follows from the definitions of ϵ_{TAIL} and $R_{\alpha}^{\epsilon_{\text{TAIL}}}(\mathcal{P}; \mathcal{Q})$. So as not to interrupt the proof, we postpone the proof showing that the two advantages are polynomially related.

Hybrid₆: In this hybrid, the verification key $\text{vk} = (\text{seed}, \mathbf{A} \cdot \mathbf{s} + \mathbf{e})$ is replaced with (\mathbf{A}, \mathbf{t}) where \mathbf{t} is sampled uniformly at random from \mathcal{R}_q^k . Since the secret key \mathbf{s} is not used anywhere in Hybrid_5 , the only change in the view of the adversary is the distribution of the verification key vk . Meaning that an adversary capable of distinguishing between Hybrid_5 and Hybrid_6 can be used to construct an adversary \mathcal{B} solving the $\text{MLWE}_{q, \ell, k, \text{SU}(u, T)}$ problem:

$$\left| \text{Adv}_{\mathcal{A}}^{\text{Hybrid}_6} - \text{Adv}_{\mathcal{A}}^{\text{Hybrid}_5} \right| \leq \text{Adv}_{\mathcal{B}}^{\text{MLWE}}.$$

Moreover we have $\text{Time}(\mathcal{B}) \approx \text{Time}(\mathcal{A})$.

Hybrid₇: Lastly, in this hybrid, the challenger prepares an empty list L_{SimT} and a fresh random oracle G' , and modifies the description of the random oracle G provided to the adversary. Notably, the adversary is not provided access to G' . The list L_{SimT} stores all the input for which G was queried in the previous hybrid. The challenger checks the same abort condition using L_{SimT} , corresponding to the fact that G was already programmed in the previous hybrid. Finally, $(\mathbf{w}, \mu, \perp) \in L_{\text{SimT}}$ denotes the point of G that the adversary queried, and not something programmed by the challenger. Since the view

of the adversary remains identical in both hybrids, we have

$$\text{Adv}_{\mathcal{A}}^{\text{Hybrid}_7} = \text{Adv}_{\mathcal{A}}^{\text{Hybrid}_6}.$$

We show in Lemma 21 that there exists an adversary \mathcal{B}' solving the $\text{SelfTargetMSIS}_{q,\ell+1,k,C,\beta}$ problem such that

$$\text{Adv}_{\mathcal{A}}^{\text{Hybrid}_7} \leq \text{Adv}_{\mathcal{B}'}^{\text{SelfTargetMSIS}}.$$

Before providing the proof of Lemma 21, we finish the proof of Theorem 3.

Collecting the bounds, we obtain

$$\begin{aligned} \text{Adv}_{\mathcal{A}}^{\text{Hybrid}_0} &\leq 2^{-\kappa} \cdot Q_h \cdot (1 + 2^{-\kappa+1} \cdot Q_s) + Q_s \cdot \epsilon_{\text{TAIL}} \\ &\quad + \left(\text{Adv}_{\mathcal{B}}^{\text{MLWE}} + \text{Adv}_{\mathcal{B}'}^{\text{SelfTargetMSIS}} + Q_s \cdot \epsilon_{\text{TAIL}} \right)^{\frac{\alpha-1}{\alpha}} \cdot (R_{\alpha}^{\epsilon_{\text{TAIL}}}(\mathcal{P}; \mathcal{Q}))^{Q_s}, \\ &\leq \left(\text{Adv}_{\mathcal{B}}^{\text{MLWE}} + \text{Adv}_{\mathcal{B}'}^{\text{SelfTargetMSIS}} + \epsilon_{\text{negl}} \right)^{\frac{\alpha-1}{\alpha}} \cdot (R_{\alpha}^{\epsilon_{\text{TAIL}}}(\mathcal{P}; \mathcal{Q}))^{Q_s} + \text{negl}(\kappa), \end{aligned}$$

where $Q_s \cdot \epsilon_{\text{TAIL}} \leq \epsilon_{\text{negl}} = \text{negl}(\kappa)$ due to our parameter selection in Appendix F.1.

Relying on Conjecture 1, we can bound $(R_{\alpha}^{\epsilon_{\text{TAIL}}}(\mathcal{P}; \mathcal{Q}))^{Q_s} \leq \exp\left(\frac{C_{\text{RENYI}} \cdot Q_s \cdot \alpha \cdot (B_{u_t,2}^{\text{sRD}})^2}{T \cdot 2^{2 \cdot u_w}}\right)$.

Hence, plugging in our choice of α , i.e., $\alpha = \frac{2^{u_w}}{B_{u_t,2}^{\text{sRD}}} \cdot \sqrt{\frac{\log(\epsilon_{\text{Adv}}) \cdot T}{C_{\text{RENYI}} \cdot Q_s}}$ with $\epsilon_{\text{Adv}} = \text{Adv}_{\mathcal{B}}^{\text{MLWE}} + \text{Adv}_{\mathcal{B}'}^{\text{SelfTargetMSIS}} + \epsilon_{\text{negl}}$, we obtain

$$\text{Adv}_{\mathcal{A}}^{\text{Hybrid}_0} \leq \underbrace{\epsilon_{\text{Adv}} \cdot \exp\left(\frac{2 \cdot B_{u_t,2}^{\text{sRD}}}{2^{u_w}} \sqrt{\frac{-C_{\text{RENYI}} \cdot \log(\epsilon_{\text{Adv}}) \cdot Q_s}{T}}\right)}_{=: \Lambda} + \text{negl}(\kappa).$$

We finally show in Lemma 22 that $\Lambda = \text{negl}(\kappa)$, assuming the hardness of the MLWE and SelfTargetMSIS assumptions. This completes the proof of Theorem 3. \square

It remains to prove the following two Lemmas 21 and 22.

Lemma 21. *There exists an adversary \mathcal{B}' solving the $\text{SelfTargetMSIS}_{q,\ell+1,k,C,\beta}$ problem with*

$$\text{Adv}_{\mathcal{A}}^{\text{Hybrid}_7} \leq \text{Adv}_{\mathcal{B}'}^{\text{SelfTargetMSIS}}.$$

Moreover we have $\text{Time}(\mathcal{B}') \approx \text{Time}(\mathcal{A})$.

Proof. Let \mathcal{A} be an adversary against the EUF-CMA security game in Hybrid₇. We construct an adversary \mathcal{B}' solving the SelfTargetMSIS problem having the same advantage as \mathcal{A} . Assume \mathcal{B}' is given $\mathbf{M} \in \mathcal{R}_q^{k \times (\ell+1)}$ as the SelfTargetMSIS problem. We denote by \mathcal{G}' the oracle \mathcal{B}' is given access to as part of the SelfTargetMSIS problem. The description of \mathcal{B}' follows.

First, \mathcal{B}' lazily simulates the random oracles H and $\mathsf{ExpandA}$. It also simulates G by relying on G' in the case $(\mathbf{w}, \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg}))$ was not used to answer the signing query (see Figure 15). Furthermore, \mathcal{B}' sets $-\mathbf{t} \in \mathcal{R}_q^k$ to be the first column of \mathbf{M} and $\mathbf{A} \in \mathcal{R}_q^{k \times \ell}$ to be the last ℓ columns and prepares the verification key vk . Note that \mathcal{B}' perfectly simulates the challenger in Hybrid_7 as the matrix \mathbf{A} and the vector \mathbf{t} are distributed uniformly in their respective sets. At the end of the game, the adversary \mathcal{A} outputs a forgery $(c_{\text{poly}}^*, \mathbf{h}^*, \mathbf{z}^*)$ for a message msg^* .

\mathcal{B}' sets $\mu^* = \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg}^*)$ and $\mathbf{s} = \begin{bmatrix} c_{\text{poly}}^* \\ \mathbf{z}^* \\ \mathbf{h}^* \end{bmatrix} \in \mathcal{R}_q^{\ell+k+1}$. It then outputs (μ^*, \mathbf{s}) as the solution to the $\mathsf{SelfTargetMSIS}$ problem.

Let us analyze the success probability of \mathcal{B}' . Conditioning on \mathcal{A}' breaking EUF-CMA security, no $(\mathbf{w}', c'_{\text{poly}})$ such that $c'_{\text{poly}} \neq \perp$ and $(\mathbf{w}', \mu^*, c'_{\text{poly}}) \in L_{\text{SimT}}$ exists due to the modification we made in Hybrid_2 . Since the forgery is valid, this implies $c_{\text{poly}}^* = \mathsf{G}'(\mathbf{A} \cdot \mathbf{z}^* - c_{\text{poly}}^* \cdot \mathbf{t} + \mathbf{h}^*, \mu^*)$ and $\|(\mathbf{z}^*, \mathbf{h}^*)\|_2 \leq \bar{B}_2$. Now, notice that

$$[\mathbf{M} \mid \mathbf{I}] \cdot \mathbf{s} = [\mathbf{M} \mid \mathbf{I}] \cdot \begin{bmatrix} c_{\text{poly}}^* \\ \mathbf{z}^* \\ \mathbf{h}^* \end{bmatrix} = -c_{\text{poly}}^* \cdot \mathbf{t} + \mathbf{A} \cdot \mathbf{z}^* + \mathbf{h}^*$$

In particular, $c_{\text{poly}}^* = \mathsf{G}'([\mathbf{M} \mid \mathbf{I}] \cdot \mathbf{s}, \mu^*)$ as desired. Finally, we have

$$\begin{aligned} \|\mathbf{s}\|_2 &= \|(c_{\text{poly}}^*, \mathbf{z}^*, \mathbf{h}^*)\|_2 \\ &\leq \|c_{\text{poly}}^*\|_2 + \|(\mathbf{z}^*, \mathbf{h}^*)\|_2 \\ &\leq \sqrt{\omega} + \bar{B}_2 \\ &= \beta. \end{aligned}$$

Since $\mathbf{s} \neq \mathbf{0}$ as c_{poly}^* has ω non-zero coefficients, we conclude that (μ^*, \mathbf{s}) is a valid solution for the $\mathsf{SelfTargetMSIS}_{q, \ell+1, k, C, \beta}$ problem. It is clear that $\mathsf{Time}(\mathcal{B}') \approx \mathsf{Time}(\mathcal{A}')$. This completes the proof. \square

Lemma 22. *Under the assumption that $\mathsf{MLWE}_{q, \ell, k, \text{SU}(u_i, T)}$ and $\mathsf{SelfTargetMSIS}_{q, \ell+1, k, C, \beta}$ are hard, we have the following according to our parameter selection in Appendix F.1:*

$$\Lambda = \varepsilon_{\text{Adv}} \cdot \exp\left(\frac{2 \cdot B_{u_i, 2}^{\text{SRD}}}{2^{u_w}} \sqrt{\frac{-C_{\text{RÉNYI}} \cdot \log(\varepsilon_{\text{Adv}}) \cdot Q_s}{T}}\right) = \text{negl}(\kappa).$$

Proof. Due to our assumption, we can assume $\varepsilon_{\text{Adv}} = \text{negl}(\kappa)$. Plugging our value for 2^{u_w} , we get $\Lambda = O(\varepsilon_{\text{Adv}} \cdot \exp(\log \kappa)) = \text{negl}(\kappa)$ as desired. \square

<div style="border-bottom: 1px solid black; padding-bottom: 5px;"> Hybrid₅ </div> <hr/> <div style="border-bottom: 1px solid black; padding-bottom: 5px;"> OSgn(msg) </div> <ol style="list-style-type: none"> 1: $\mu := H(H(\text{vk})\ \text{msg})$ 2: $\mathbf{z} \leftarrow \text{RSU}(u_{\mathbf{w}}, N - d + 1)^\ell$ 3: $\mathbf{z}' \leftarrow \text{RSU}(u_{\mathbf{w}}, N - d + 1)^k$ 4: $c_{\text{poly}} \leftarrow C$ 5: $\mathbf{w} := \mathbf{A} \cdot \mathbf{z} - c_{\text{poly}} \cdot \mathbf{t} + \mathbf{z}'$ 6: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - c_{\text{poly}} \cdot \mathbf{t}$ 7: $\mathbf{h} := \mathbf{w} - \mathbf{y}$ 8: $G(\mathbf{w}, \mu) := c_{\text{poly}} \triangleright$ Abort if already programmed 9: $\text{sig} := (c_{\text{poly}}, \mathbf{h}, \mathbf{z})$ 10: if CheckBounds(sig) = FAIL goto Line 2 11: $Q_{\text{Sign}} := Q_{\text{Sign}} \cup \{(\text{msg}, \text{sig})\}$ 12: return sig <hr/>	<div style="border-bottom: 1px solid black; padding-bottom: 5px;"> Hybrid₆ </div> <ol style="list-style-type: none"> 1: $\text{seed} \leftarrow \{0, 1\}^\kappa$ 2: $\mathbf{A} \leftarrow \mathcal{R}_q^{k \times \ell}$ 3: $\text{ExpandA}(\text{seed}) := \mathbf{A}$ 4: $\mathbf{t} \leftarrow \mathcal{R}_q^k$ 5: $\text{vk} := (\text{seed}, \mathbf{t})$ 6: $Q_{\text{Sign}} := \emptyset$ 7: $(\text{msg}^*, \text{sig}^*) \leftarrow \mathcal{A}^{\text{OSgn}(\cdot)}(\text{vk})$ 8: if $\exists (\text{msg}, \cdot) \in Q_{\text{Sign}} : H(H(\text{vk})\ \text{msg}^*) = H(H(\text{vk})\ \text{msg})$ return FAIL 9: if $\exists \text{sig}' \text{ s.t. } (\text{msg}^*, \text{sig}') \in Q_{\text{Sign}}$ return FAIL 10: return Verify(sig[*], msg[*], vk) <hr/>
<div style="border-bottom: 1px solid black; padding-bottom: 5px;"> Hybrid₇ </div> <ol style="list-style-type: none"> 1: $\text{seed} \leftarrow \{0, 1\}^\kappa$ 2: $\mathbf{A} \leftarrow \mathcal{R}_q^{k \times \ell}$ 3: $\text{ExpandA}(\text{seed}) := \mathbf{A}$ 4: $\mathbf{t} \leftarrow \mathcal{R}_q^k$ 5: $\text{vk} := (\text{seed}, \mathbf{t})$ 6: $Q_{\text{Sign}} := \emptyset$ 7: $L_{\text{SimT}} := \emptyset$ 8: $(\text{msg}^*, \text{sig}^*) \leftarrow \mathcal{A}^{\text{OSgn}(\cdot)}(\text{vk})$ 9: if $\exists (\text{msg}, \cdot) \in Q_{\text{Sign}} : H(H(\text{vk})\ \text{msg}^*) = H(H(\text{vk})\ \text{msg})$ return FAIL 10: if $\exists \text{sig}' \text{ s.t. } (\text{msg}^*, \text{sig}') \in Q_{\text{Sign}}$ return FAIL 11: return Verify(sig[*], msg[*], vk) <hr/>	<div style="border-bottom: 1px solid black; padding-bottom: 5px;"> OSgn(msg) </div> <ol style="list-style-type: none"> 1: $\mu := H(H(\text{vk})\ \text{msg})$ 2: $\mathbf{z} \leftarrow \text{RSU}(u_{\mathbf{w}}, N - d + 1)^\ell$ 3: $\mathbf{z}' \leftarrow \text{RSU}(u_{\mathbf{w}}, N - d + 1)^k$ 4: $c_{\text{poly}} \leftarrow C$ 5: $\mathbf{w} := \mathbf{A} \cdot \mathbf{z} - c_{\text{poly}} \cdot \bar{\mathbf{t}} + \mathbf{z}'$ 6: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - c_{\text{poly}} \cdot \mathbf{t}$ 7: $\mathbf{h} := \mathbf{w} - \mathbf{y}$ 8: $L_{\text{SimT}} \leftarrow L_{\text{SimT}} \cup \{(\mathbf{w}, \mu, c_{\text{poly}})\}$ \triangleright Abort if $\exists c'_{\text{poly}} \in C \cup \{\perp\} \text{ s.t. } (\mathbf{w}, \mu, c'_{\text{poly}}) \in L_{\text{SimT}}$ 9: $\text{sig} := (c_{\text{poly}}, \mathbf{h}, \mathbf{z})$ 10: if CheckBounds(sig) = FAIL goto Line 2 11: $Q_{\text{Sign}} := Q_{\text{Sign}} \cup \{(\text{msg}, \text{sig})\}$ 12: return sig <hr/>
	<div style="border-bottom: 1px solid black; padding-bottom: 5px;"> G(w, msg) </div> <ol style="list-style-type: none"> 1: $\mu := H(H(\text{vk})\ \text{msg})$ 2: if $\exists c_{\text{poly}} \text{ s.t. } (\mathbf{w}, \mu, c_{\text{poly}}) \in L_{\text{SimT}}$ return c_{poly} 3: $L_{\text{SimT}} \leftarrow L_{\text{SimT}} \cup \{(\mathbf{w}, \mu, \perp)\}$ 4: return $G'(\mathbf{w}, \text{msg})$ <hr/>

Fig. 15: Last three Hybrid games for the proof of Theorem 3. The differences between Hybrid_{i-1} and Hybrid_i are highlighted. Note that in Hybrid_6 , the signing oracle $\text{OSgn}(\text{msg})$ remains the same as in Hybrid_5 . Moreover, in Hybrid_7 , the game uses another random oracle G' (non-accessible from \mathcal{A}) and modifies the description of the random oracle G . We assume \mathcal{A} is given access to the random oracles $(H, G, \text{ExpandA})$.