Raccoon

A Side-Channel Secure Signature Scheme

Rafael del Pino
PQShield, FR

Thomas Espitau
PQShield, FR

Shuichi Katsumata
PQShield, UK
AIST, JP

Mary Maller
PQShield, UK
Ethereum Foundation, UK

Fabrice Mouhartem
PQShield, FR

Thomas Prest
PQShield, FR

Mélissa Rossi
ANSSI, FR

Markku-Juhani Saarinen
PQShield, UK
Tampere University, FI

https://raccoonfamily.org

authors@raccoonfamily.org

Contents

1 Introduction to Raccoon

1.1 Motivation and Context

In the past decade, post-quantum cryptography has reached a turning point; institutional bodies and stakeholders initiated standardization and deployment efforts, and a variety of designs reached a high enough level of maturity to be deployed.

This is epitomized by NIST's recent standardization in 2020 of the hash-based signatures XMSS and LMS [CAD+20], 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+ [AAC+22].

Whilst the efficiency profiles and black-box security of these schemes are well-understood, resistance against side-channel attacks remains a weak spot for all of them.

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

Several side-channel attacks have been proposed against schemes considered by NIST for standardization, such as Dilithium [KAA21, FDK20, MUTS22], Falcon [KA21, GMRR22, ZLYW23], or SPHINCS and XMSS [KGB+18]. The list above is by no means exhaustive, and in general cryptographic algorithms require implementation countermeasures in order to achieve any meaningful security in the context of side-channel attacks.

Masking. The main countermeasure against side-channel attacks is masking [ISW03]. It consists of splitting sensitive information in d shares (concretely: $x = x_0 + \cdots + x_{d-1}$), and performing secure computation using MPC-based techniques. Masking incurs an overhead on the running time of the protected algorithm: this overhead is linear, quadratic, or worse than quadratic depending on the operation. On the other hand, the cost of a side-channel attack is expected to grow exponentially in the number of shares d [DFS19, MRS22, IUH22]. In other words, masking provides a *trade-off* between side-channel resistance and computational efficiency.

Unfortunately, lattice-based signatures contain subroutines that are extremely expensive to mask [MGTF19, ABC+22]. Hash-based signatures use hash functions as building blocks, and these are similarly expensive to mask even with state-of-the-art techniques [ZSS+21]. This state of affairs limits the applicability of masking to these schemes and, by extension, their ability to be deployed in highly adversarial environments.

A masking-friendly scheme. The main motivation of Raccoon is to cover use cases where side-channel resistance is important. All subroutines in Raccoon either can be masked with a quasilinear overhead or do not need to be masked at all. As a result, we can mask Raccoon at orders that are out of reach for all other existing signature schemes. For example, very high-order $d = 32$ signatures can have a latency that doesn't affect authentication user experience (tens of milliseconds – See Table 5). This efficiency gain is not limited to running time; we also propose techniques that minimize the *memory* overhead when masking Raccoon at high orders.

One of the most distinctive feature of raccoons is the "mask" around their eyes. In addition, a group of raccoons is sometimes called a mask of raccoons. Thus we decided to call our scheme Raccoon, due to its masking-friendly nature.

1.2 Design Rationale and Technical Overview

Fiat-Shamir with aborts. Raccoon is a lattice-based signature scheme based on the Fiat-Shamir paradigm. Examples of such schemes include BLISS [DDLL13], qTESLA [BAA+17], and of course, Dilithium [LDK+22], which was selected in 2022 by NIST as its primary standard for signatures. All these schemes follow the "Fiat-Shamir with aborts" framework proposed by Lyubashesvky in 2009 [Lyu09] and refined in subsequent works [Lyu12, GLP12, DDLL13, BG14, DKL+18a].

A prototypical instantiation of this framework is provided in Figure 1a. A key subroutine for achieving security is the *rejection sampling* step (Line 10). For simplicity, all optimizations are ignored in order to focus on the key ideas.

$\mathsf{Dilithium\text{-}Sign}(\mathsf{sk}, \mathsf{msg}) \longrightarrow \mathsf{sig}$

Input: A signing key sk, a message msg.
Output: A signature sig of msg under sk.

1: $(\mathsf{vk}, \mathbf{s}) := \mathsf{sk}, (\mathbf{A}, \mathbf{t}) := \mathsf{vk}$
2: $\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg})$
3: $\mathbf{r} \leftarrow \mathcal{U}(\{-\eta, \ldots, \eta\})^\ell$ ▷ **Slow**
4: $\mathbf{e}' \leftarrow \mathcal{U}(\{-\eta, \ldots, \eta\})^k$ ▷ **Slow**
5: $\mathbf{w} := \mathbf{A} \cdot \mathbf{r} + \mathbf{e}'$ ▷ Fast
6: $c := \mathsf{G}(\mathbf{w}, \mu)$
7: $\mathbf{z} := c \cdot \mathbf{s} + \mathbf{r}$ ▷ Fast
8: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t}$ ▷ Fast
9: $\mathbf{h} := \mathbf{w} - \mathbf{y}$ ▷ Fast
10: **if** $\mathsf{Rejection}(\mathbf{z}, \mathbf{h})$ is True ▷ **Slow**
11: **goto** Line 3
12: $\mathsf{sig} := (c, \mathbf{h}, \mathbf{z})$
13: **return** sig

(a) Blueprint for Dilithium

$\mathsf{Raccoon\text{-}Sign}(\mathsf{sk}, \mathsf{msg}) \longrightarrow \mathsf{sig}$

Input: A signing key sk, a message msg.
Output: A signature sig of msg under sk.

1: $(\mathsf{vk}, \mathbf{s}) := \mathsf{sk}, (\mathbf{A}, \mathbf{t}) := \mathsf{vk}$
2: $\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg})$
3: $(\mathbf{r}, \mathbf{e}') := (\mathbf{0}^k, \mathbf{0}^\ell)$ ▷ Fast
4: **for** $i \in [T]$ **do** ▷ Fast
5: $\mathbf{r} \leftarrow \mathbf{r} + \mathcal{U}(\{-\eta, \ldots, \eta - 1\})^\ell$ ▷ Fast
6: $\mathbf{e}' \leftarrow \mathbf{e}' + \mathcal{U}(\{-\eta, \ldots, \eta - 1\})^k$ ▷ Fast
7: $\mathbf{w} := \mathbf{A} \cdot \mathbf{r} + \mathbf{e}'$ ▷ Fast
8: $c := \mathsf{G}(\mathbf{w}, \mu)$
9: $\mathbf{z} := c \cdot \mathbf{s} + \mathbf{r}$ ▷ Fast
10: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t}$
11: $\mathbf{h} := \mathbf{w} - \mathbf{y}$
12: $\mathsf{sig} := (c, \mathbf{h}, \mathbf{z})$
13: **return** sig

(b) Blueprint for Raccoon

Figure 1: High-level blueprints for Dilithium and Raccoon. Differences between the blueprints are highlighted. Operations that need to be masked in the context of side-channels are indicated with comments: Fast when the overhead is $O(d \log d)$ or **Slow** when the overhead is $\Omega(d^2)$. We write $\mathcal{U}(S)$ to denote a set of polynomials in $\mathcal{R}_q[x]$ with coefficients in the set S. As an example, we note that with masked Dilithium, for coefficients r, one needs to come up with a sum $r = r_0 + \cdots + r_{d-1} \pmod q$ that is uniform in the range $r \in [-\eta, \eta]$ but such that each proper subset of r_i reveals nothing about r. This is a complex operation to implement securely. In Raccoon, the final r has a sum-of-uniform distribution, and the contributing uniform distributions are added to individual shares. This is much faster but still requires additional randomization and careful analysis. Furthermore, it is tempting not to implement complex masking for only potentially vulnerable steps in Dilithium, such as Line 10. Raccoon does not have such ambiguous steps.

Limitations in the context of side-channel attacks and masking. While the black-box security of Fiat-Shamir with aborts schemes is by now well-understood, its resistance against side-channel attacks is still in an exploratory state. Several side-channel attacks against unprotected implementations of schemes in this family have been documented. See [PBY17, EFGT17, BDE+18, BBE+19] and [KAA21, FDK20, MUTS22] for side-channel attacks against unprotected implementations of BLISS and Dilithium, respectively. Two particularly vulnerable points are the generation of ephemeral secrets (Lines 3 and 4) and the rejection sampling step (Line 10).

As discussed in Section 1.1, the main countermeasure to address this class of attacks is masking. When applying masking to Figure 1a, several difficulties arise:

1. **Randomness sampling.** Sampling random errors (Lines 3 and 4) is challenging in an arithmetic masked form. The most efficient known approach is to sample $\mathbf{r}$ in *Boolean* masked form, then convert the result in arithmetic masked form. This requires so-called *mask conversions* [CGV14, HT19, CGTV15]. Despite efficiency improvements since their introduction in [Gou01], known secure mask conversion algorithms run in time at least $O(d^2)$. See [BBE+18, Alg. 15], [MGTF19, Alg. 13] and [GR19, §3.2] for concrete instantiations of randomness generation with mask conversions.

2. **Challenge computation.** In Line 6, a challenge c is computed. In classical Fiat-Shamir schemes, since c is a function of public data, it is clear that the challenge computation can be securely performed unmasked. However, in Fiat-Shamir with aborts, not all signatures are output, so whether it remains safe to perform the challenge computation unmasked is an open question. Existing works conjecture that it is still true and perform the challenge computation unmasked, see for example [BBE+18, Definition 2].

3. **Rejection sampling.** The rejection sampling step is critical for the security of Fiat-Shamir with aborts and needs to be masked. In practice, most schemes verify that $(\mathbf{z}, \mathbf{h})$ belongs to a certain set. Once again, the most efficient known techniques require expensive mask conversions – it has been performed this way in existing masked designs. See [BBE+18, Alg. 16], [BBE+19, §4], [MGTF19, §5.3.3], and [GR19, Alg. 8] for concrete instantiations of masked rejection sampling.

Due to these three points, secure masking of schemes such as Dilithium is a challenging task.

A masking-friendly design. Raccoon is based on a design that makes it amenable to masking. Our main inspiration is the eponymous scheme from [dPPRS23], and Raccoon also shares similarities with a scheme from [ASY22]. Our main design rationale is to rely solely on masking-friendly operations. Our high-level design is presented in Figure 1b. Our main design decisions are the following:

1. **No rejection sampling.** Since rejection sampling is challenging to mask, we decide to remove it altogether. Similarly to [dPPRS23, ASY22], an analysis based on the Rényi divergence guarantees the security of Raccoon in certain regimes of parameters. The downside of this change is that parameters need to be adjusted in order to guarantee security. Compared to Dilithium, the signature size is increased by a factors 5, approximately. On the other hand, the verification key size remains similar.

2. **Sums of uniforms.** The security analysis of [ASY22] is valid for Gaussians. However, these are not amenable to masking, so we replace them with sums of uniform distributions. The security analysis becomes more delicate, but the implementation is made considerably simpler.

Masking Raccoon. We now briefly explain how to mask the blueprint of Figure 1b.

1. **Unmasked operations.** Computing the challenge c (Line 8) and the values $\mathbf{y}$ and $\mathbf{h}$ (Lines 10 and 11) can be performed unmasked. Indeed, these values can always be computed from public data, since there is no more rejection sampling.

2. **Linear operations.** These include the computation of $\mathbf{w}$ and $\mathbf{z}$ (Lines 7 and 9). Due to their linearity, these operations can be masked in linear overhead $O(d)$.

3. **Randomness sampling.** The most subtle part is related to masking Lines 3 to 6. We proceed as follows. Each share a_i (for $i \in [d]$) of each integer coefficient a of $(\mathbf{r}, \mathbf{e}')$ will be the sum of rep uniform random samples in an interval S. As a result, a is the sum of $\mathsf{rep} \cdot d$ uniform samples in S. By interleaving the addition of samples of S with refresh gadgets, we can ensure that even an adversary with the ability to probe t values can learn no more than t of the individual samples of S.

Note that the distribution of signatures is correlated to the number of shares d. In other words, one can determine d by observing the distribution of signatures. We do not expect this to be an issue in practice. In addition, we ensure that verifier-side parameters are independent of d.

1.3 Advantages and Limitations

1.3.1 Advantages

Masking-friendliness. The main design principle of Raccoon is amenability to masking. In effect, Raccoon can be masked at order $d - 1$ with an overhead $O(d \log d)$. This allows masking Raccoon at high orders with a small impact on efficiency.

At high masking orders, memory consumption becomes the new efficiency bottleneck due to the need to store polynomials masked at high orders. We resolve this by using techniques that allow significantly reduce the memory cost of masked values. This allows us to implement Raccoon with $d = 32$ shares, in as little as 128kB of SRAM.[1]

Standard lattice assumptions. Raccoon relies on (variants) of lattice assumptions that are well-understood. Indeed, we rely on variants of Module-LWE and Module-SIS (see Section 4.1 for formals statements), similarly to the (selected) primary standard Dilithium. Note that we rely on self-target Module-SIS for the Euclidean norm, as opposed to the slightly less usual infinity norm used in Dilithium [LDK+22, Remark 1].

Simple and portable implementation. Two ideas that permeate the design of Raccoon are the simplicity and portability of implementation. For example, our error distributions are based on uniform distributions over $\{0, \ldots, 2^u - 1\}$; this makes implementation straightforward across a wide range of platforms. Similarly, our 49-bit modulus can be split in two 24-bit and 25-bit moduli; this facilitates implementation on 32-bit architectures.

Unlike many other schemes, for side-channel security Raccoon does not require masked implementations of symmetric cryptographic components such as SHA-3/SHAKE. The number of distinct masking gadgets is relatively small, which results in simpler and easier-to-verify firmware and hardware.

In addition to the scalability of security and theoretical soundness, an essential advantage of masking countermeasures is that they are less dependent on the physical details of the implementation when compared to logic-level techniques such as dual-rail countermeasures [ABD+14]. Hence the implementations are – to a degree – portable.

Potential relevance to the NIST Threshold Cryptography project. NIST has recently issued a call for Multi-Party Threshold Cryptography (MPTC) [NIS23b]. This document indicates a high interest in threshold-friendly schemes, and this interest is also reflected in NIST's call for

[1] This functional specification does not go into the details of these implementation techniques.

additional PQC signatures [NIS22]. With its simple structure and lack of rejection sampling, we expect Raccoon to be easier to threshold than schemes such as Falcon or Dilithium.

Potential relevance to NIST's Masked Circuits project. NIST has indicated its interest to masking cryptographic schemes via the Masked Circuits project[2]. While this project is in an earlier stage than NIST's PQC and MPTC projects, we expect that Raccoon may be of interest for the scope of this project, due to its masking-friendly nature.

1.3.2 Limitations

Larger sizes than Falcon and Dilithium. Due to the removal of rejection sampling, the signature size of Raccoon is quite larger than for Dilithium, despite being based on a similar blueprint and similar assumptions. The verification key sizes are similar to Dilithium's.

This increase in size is due to the fact that our signature sizes scale logarithmically with the number of queries. At the moment, our parameter sets and associated security proofs for NIST level I, III and V cover a maximal number of queries Q_s equal to $2^{53}, 2^{51}$ and 2^{55}, respectively. Our design can readily support higher number of queries if necessary; the signature size would increase accordingly.

Same assumptions as Dilithium. The assumptions underlying Raccoon and Dilithium are very similar. While this allows Raccoon to benefit from the same body of work which underpins the security of Dilithium, it also means that Raccoon does not diversify the security assumptions compared to already selected standards (Dilithium, Falcon, SPHINCS$^+$, LMS, XMSS).

No resistance against fault attacks. While the design of Raccoon makes it more resilient to side-channel attacks, fault attacks are also a meaningful threat in real-life adversarial environments. Our design does not necessarily make a system any more secure against fault attacks.

1.4 Use Cases

We view the RSA and ECC signature use cases requiring physical side-channel security as the primary use cases for Raccoon. This is a common industry requirement for Smart Cards, Secure Elements, Authentication Tokens, Hardware Security Modules, IoT Platform Security (secure boot, Firmware Updates, attestation), Crypto Wallets, and Mobile Phones.

Matching and surpassing the side-channel security of classical signature schemes. For a signature scheme, we opine that side-channel countermeasures (in FIPS 140-3 terms, "non-invasive attack mitigations" [NIS19, ISO22]) should be at least as powerful as the countermeasures available for classical RSA and Elliptic Curve based signatures defined in FIPS 186-5 [NIS23a].

Creating countermeasures for these older algorithms was relatively straightforward due to their simple algebraic structure. For example, Coron in CHES 1999 [Cor99] proposed three "standard" countermeasures for Elliptic Curve Cryptography implementations: Randomization of the Private Exponent, Blinding the Point P, and Randomized Projective Coordinates. All of these randomization techniques are based on Elliptic Curves' homogeneous "one big arithmetic operation" implementation structure. Similar algebraic randomization, masking, and blinding techniques are commonly applied to RSA. However, most PQC algorithms have a larger number of algebraically dissimilar algorithmic steps, so analogous techniques are not available. There is ample evidence that unsecured PQC implementations are highly vulnerable to attacks.

[2] https://csrc.nist.gov/Projects/masked-circuits

Upper-bounded signature latency for real-time applications. We wanted the Raccoon signature rejection rate to be low to facilitate applications that must complete signature generation within a specific timing bound, which is common in "real-world" authentication systems.

Such real-time bounds can be problematic for schemes based on rejection sampling; each loop iteration can be seen as an independent Bernoulli trial; there can be any number of iterations.

For Dilithium, the signature generation success rate is approximately $p \in \{0.23, 0.19, 0.26\}$ at security levels $\{2, 3, 5\}$, respectfully. On average $1/p$ iterations are run. After n iterations, the algorithm has a probability $(1-p)^n$ of not having succeeded. Budgeting $4\times$ the average Dilithium signature execution time (23 to 32 iterations) still leaves a 1% probability of missing the deadline. A single Raccoon iteration has a $p > 0.999$ success rate, and latency bounds can be met with a much smaller margin. This facilitates real-time and safety-critical applications.

2 Technical Specification

This section contains a technical specification for (Masked) Raccoon instantiated to provide the functionality required in the NIST Call for Additional Signature Schemes [NIS22] using the NIST PQC Testing API. We include low-level details such as padding, serialization, hashes, and other components used by the reference implementations and which are required for interoperability. This description does not address broader Raccoon applications such as threshold signatures or include an exhaustive description of implementation techniques that a real-life (non-reference) masked Raccoon implementation may require to achieve resistance against side-channel attacks.

2.1 Parameter Sets

Parameters are provided at security levels matching or exceeding best quantum or classical attacks against AES-κ where $\kappa \in \{128, 192, 256\}$. These correspond to NIST Post-Quantum Security Categories 1, 3, and 5 [NIS22, Section 4.B.3].

Furthermore, we offer internal parameters for masked key generation and signature computation with d masking shares, where $d \in \{1, 2, 4, 8, 16, 32\}$. The variant with $d = 1$ has masking disabled ("Vanilla Raccoon"), while others offer t-probing security at masking order $t = d - 1$. The masking order does not affect public keys or signature verification; the same verification function can verify signatures generated with any d.

A naming convention for these parameter sets is adopted in this document and with the supplied reference implementations:

$$\text{Raccoon-}\kappa\text{-}d$$

When the masking order does not matter (for example, for signature verification), the d parameter can be omitted, and Raccoon-κ suffices. Table 1 contains a brief guide to the parameters. Tables 2 to 4 contain the parameters for Raccoon-128, Raccoon-192, and Raccoon-256 respectively; parameters which are independent of masking order are marked with an equivalence symbol (=).

Table 1: Raccoon Parameter Legend

Parameter	References	Description
κ	Sections 2.1 and 4.3.1.	AES-equivalent security level.
Q_s	Section 4.3.6.	Maximal recommended number of queries.
q	Section 2.7.	Integer modulus used in $\mathcal{R}_q$ ring arithmetic.
n	Section 4.1.	Degree of ring $\mathcal{R}_q$ reduction polynomial $x^n + 1$.
k	Section 4.1.	Number of rows in $\mathbf{A}$, length of vector $\mathbf{t}$.
ℓ	Section 4.1.	Number of columns in $\mathbf{A}$, length of vector $\mathbf{s}$.
ν_{t}	Sections 2.3.1 and 2.3.3.	Public key low-bit truncation (bits.)
ν_{w}	Sections 2.3.2 and 2.3.3.	Commitment low-bit truncation (bits.)
ω	Section 2.4.6.	Number of nonzero coefficients in c_{poly}.
$2^{-64}B_2^2$	Sections 2.4.4 and 2.6.	Scaled squared Euclidean norm bound.
B_∞	Sections 2.4.4 and 2.6.	Infinity norm (absolute value) bound.
$\lvert\mathsf{sig}\rvert$	Sections 2.5.1 and 2.6.	Length of (detached) signature in bytes.
$\lvert\mathsf{vk}\rvert$	Section 2.5.2.	Length of public verification key in bytes.
d	Sections 2.2 and 2.4.5.	Number of masking shares (order $t + 1$).
rep	Section 2.4.5	The number of iterations for SU distribution.
u_{t}	Sections 2.3.1 and 2.4.5.	Distribution parameter for $\mathbf{s}$ and $\mathbf{t}$ (uniform bits).
u_{w}	Sections 2.3.2 and 2.4.5.	Distribution parameter for $\mathbf{r}$ and $\mathbf{w}$ (uniform bits).
$\lvert\mathsf{sk}\rvert$	Section 2.5.3.	Length of (reference code) secret key in bytes.

Table 2: Parameters for Raccoon-128, NIST Post-Quantum security strength category 1.

Parameter	Raccoon-128	128-2	128-4	128-8	128-16	128-32
κ	128	=	=	=	=	=
Q_s	2^{53}	=	=	=	=	=
q	549824583172097	=	=	=	=	=
n	512	=	=	=	=	=
k	5	=	=	=	=	=
ℓ	4	=	=	=	=	=
ν_{t}	42	=	=	=	=	=
ν_{w}	44	=	=	=	=	=
ω	19	=	=	=	=	=
$2^{-64}B_2^2$	14656575897	=	=	=	=	=
B_∞	41954689765971	=	=	=	=	=
|sig| (bytes)	11524	=	=	=	=	=
|vk| (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
|sk| (bytes)	14800	14816	14848	14912	15040	15296

Table 3: Parameters for Raccoon-192, NIST Post-Quantum security strength category 3.

Parameter	Raccoon-192	192-2	192-4	192-8	192-16	192-32
κ	192	=	=	=	=	=
Q_s	2^{51}	=	=	=	=	=
q	549824583172097	=	=	=	=	=
n	512	=	=	=	=	=
k	7	=	=	=	=	=
ℓ	5	=	=	=	=	=
ν_{t}	42	=	=	=	=	=
ν_{w}	44	=	=	=	=	=
ω	31	=	=	=	=	=
$2^{-64}B_2^2$	24964497408	=	=	=	=	=
B_∞	47419426657048	=	=	=	=	=
|sig| (bytes)	14544	=	=	=	=	=
|vk| (bytes)	3160	=	=	=	=	=
d	1	2	4	8	16	32
rep	8	4	2	4	2	4
u_{t}	7	7	7	6	6	5
u_{w}	41	41	41	40	40	39
|sk| (bytes)	18840	18864	18912	19008	19200	19584

Table 4: Parameters for Raccoon-256, NIST Post-Quantum security strength category 5.

Parameter	Raccoon-256	256-2	256-4	256-8	256-16	256-32
κ	256	=	=	=	=	=
Q_s	2^{55}	=	=	=	=	=
q	549824583172097	=	=	=	=	=
n	512	=	=	=	=	=
k	9	=	=	=	=	=
ℓ	7	=	=	=	=	=
$\nu_{\mathbf{t}}$	42	=	=	=	=	=
$\nu_{\mathbf{w}}$	44	=	=	=	=	=
ω	44	=	=	=	=	=
$2^{-64}B_2^2$	38439957299	=	=	=	=	=
B_∞	50958538642039	=	=	=	=	=
$\lvert\mathsf{sig}\rvert$ (bytes)	20330	=	=	=	=	=
$\lvert\mathsf{vk}\rvert$ (bytes)	4064	=	=	=	=	=
d	1	2	4	8	16	32
rep	8	4	2	4	2	4
$u_{\mathbf{t}}$	6	6	6	5	5	4
$u_{\mathbf{w}}$	41	41	41	40	40	39
$\lvert\mathsf{sk}\rvert$ (bytes)	26016	26048	26112	26240	26496	27008

2.2 Notation

Sets, functions and distributions. 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 $f : X \to Y$ be a function, and $x \in X$. When f is deterministic, we use the notation $y := f(x)$ to indicate that we assign the output of $f(x)$ to y. When f is randomized, we instead use the notation $y \leftarrow f(x)$. From a programming viewpoint, both of these notations indicate an assignment of the result to the variable on the left. Given a probability distribution $\mathcal{D}$ over Y, we note $y \leftarrow \mathcal{D}$ to express that $y \in Y$ is sampled from $\mathcal{D}$.

Lastly, we use $\omega_{\mathsf{asymp}}(g(\kappa))$ to denote the class of functions that grows asymptotically faster than $g(\kappa)$.

Integer representatives. Modular congruence classes $x \in \mathbb{Z}_q$ have a canonical non-negative integer representative $x \in [q]$, and a canonical signed integer representative $-\lfloor q/2 \rfloor \le x < \lceil q/2 \rceil$. The signed representative is used for quantities representing norms and distances and for functions such as $\mathrm{abs}(x)$. For details of serialization and deserialization of integers for transmission or storage, see Section 2.4.1.

Modulus rounding. Let $\nu \in \mathbb{N}\backslash\{0\}$. Any integer $x \in \mathbb{Z}$ can be decomposed as:

$$x = 2^\nu \cdot x_{\mathsf{top}} + x_{\mathsf{bot}}, \quad (x_{\mathsf{top}}, x_{\mathsf{bot}}) \in \mathbb{Z} \times [-2^{\nu-1}, 2^{\nu-1} - 1]. \tag{1}$$

The decomposition in Eq. (1) is unique. We define the function

$$\lfloor \cdot \rceil_\nu : \mathbb{Z} \mapsto \mathbb{Z} \quad \text{s.t.} \quad \lfloor x \rceil_\nu = \lfloor x/2^\nu \rceil = x_{\mathsf{top}}, \tag{2}$$

where $\lfloor \cdot \rceil : \mathbb{R} \mapsto \mathbb{Z}$ denotes the rounding operator. More precisely the "rounding half-up" method $\lfloor x \rceil = \lfloor x + 1/2 \rfloor$ where half-way values are rounded up: $\lfloor 2.5 \rceil = 3$ and $\lfloor -2.5 \rceil = -2$.

With a slight overload of notation, for any $q \in \mathbb{N}\backslash\{0\}$ with $q > 2^\nu$, we allow $\lfloor\cdot\rceil_\nu$ to take inputs in $\mathbb{Z}_q$, in which case, we assume the output is an element in $\mathbb{Z}_{q_\nu}$ where $q_\nu = \lfloor q/2^\nu \rfloor$. I.e. we define:

$$\lfloor\cdot\rceil_\nu : \mathbb{Z}_q \mapsto \mathbb{Z}_{q_\nu} = \mathbb{Z}_{\lfloor q/2^\nu \rfloor} \quad \text{s.t.} \quad \lfloor x \rceil_\nu = \lfloor x/2^\nu \rceil \mod q_\nu = x_{\mathsf{top}} \mod q_\nu \tag{3}$$

where we use the non-negative representative for $\mathbb{Z}_q$ and $\mathbb{Z}_{q_\nu}$. Concretely, in the Raccoon signature scheme, we define additional smaller moduli $q_{\mathrm{t}} = \lfloor q/2^{\nu_{\mathrm{t}}} \rfloor$ (used for the public key $\mathbf{t}$ and related quantities), and $q_{\mathrm{w}} = \lfloor q/2^{\nu_{\mathrm{w}}} \rfloor$ (for an MLWE commitment $\mathbf{w}$ and hint computation).

Programming note: The range of x_{bot} matches that of a standard ν-bit (two's complement) signed integer. To obtain x_{bot}, an implementation can mask low ν bits and sign-extend bit $\nu - 1$. To obtain rounded $\lfloor x \rceil_\nu = x_{\mathsf{top}}$, add a rounding bit and right shift by ν bits: $x_{\mathsf{top}} = \lfloor (x + 2^{\nu-1})/2^\nu \rfloor$.

Polynomials, vectors, and matrices. Let $q \in \mathbb{N}$ and n a power-of-two. We note $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ the quotient ring of integers modulo q. We also note $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$ the quotient ring obtained by taking the quotient of $\mathbb{Z}_q[x]$ by the ideal generated by $(x^n + 1)$. The canonical representative of $f \in \mathcal{R}_q$ is the unique polynomial in $\mathbb{Z}_q[x]$ of degree $< n$ in the equivalence class f. Details of these rings are discussed in Section 2.7.

Scalars are noted in italic lowercase (e.g., n); this includes elements of $\mathbb{Z}, \mathbb{Z}_q$, and $\mathcal{R}$. Vectors are noted in bold lowercase (e.g., $\mathbf{v}$), and matrices are noted in bold uppercase (e.g., $\mathbf{M}$). Vectors are column vectors by default; given an $m \times n$ matrix $\mathbf{M}$ and an n-element column vector $\mathbf{v}$, their product $\mathbf{M} \cdot \mathbf{v}$ is an m-element column vector.

For $p \in [1, \infty]$ and a vector $\mathbf{v} = (v_i)_{i \in [n]} \in \mathbb{R}^n$, we note $\|\mathbf{v}\|_p$ the L_p norm of $\mathbf{v}$, that is $\|\mathbf{v}\|_p = (\sum |v_i|^p)^{1/p}$ when $p < \infty$ and $\|\mathbf{v}\|_\infty = \max_i |v_i|$. If $p = 2$, we may drop the subscript p: $\|\mathbf{v}\| := \|\mathbf{v}\|_2$. We recall that the L_p norm is a non-increasing function of p; for $r \leq q$, $\|\mathbf{v}\|_q \leq \|\mathbf{v}\|_r$.

Throughout the document, we may define functions with inputs in $\mathbb{Z}$ (resp. $\mathbb{Z}_q$) and extend them to inputs in $\mathbb{Z}_q$ (resp. $\mathcal{R}_q$). This is simply done by identifying these inputs with their canonical representatives. We may also extend them freely to inputs that are vectors or matrices with entries in $\mathcal{R}_q$; this is simply done by the entry-wise application.

Random sampling. Given a finite set S, the notation $x \leftarrow S$ indicates that x is sampled uniformly at random in S. Following NIST terminology and requirements, sampling uses Random Bit Generators (RBGs) [BK15, TBK+18, BKM+22]. There are also secondary random quantities that are used only for masking: These are sampled with Masking Random Generators (MRGs), denoted $x \xleftarrow{M} S$. For further details, see Section 2.8.

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

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

The acronym SU stands for "sum of uniforms". This class of distributions is 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, see Appendix A. 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. Given a random variable $X \sim \mathrm{SU}(u, T)$,

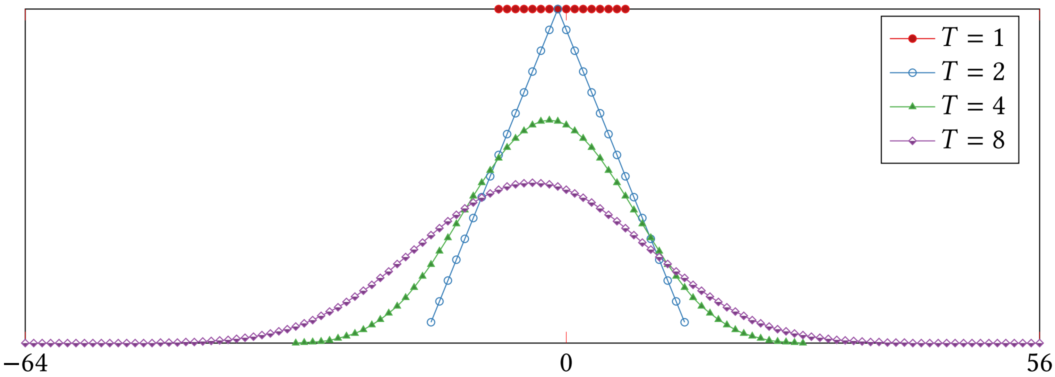


Figure 2: The distribution $\mathrm{SU}(4, T)$, for $T \in \{1, 2, 4, 8\}$

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:

$$\mathbb{P}[|X + T/2| > \mu] \leq \exp\left(-\frac{\mu^2}{2\sigma^2}\right) = \exp\left(-\frac{3\mu^2}{TN^2}\right) \tag{4}$$

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

$$\Pr\left[\sum_1^m X_i^2 > r \cdot m \cdot \sigma^2\right] \leq \exp\left(-\frac{r \cdot m}{8}\right) \tag{5}$$

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

$$\mathbb{E}[X] = -\frac{T}{2}; \qquad \mathbb{E}[X^2] = \frac{T(N^2-1)}{12} + \frac{T^2}{4}; \qquad \mathbb{V}[X] = \frac{T(N^2-1)}{12}; \tag{6}$$

Note that $\mathrm{SU}(u, T)$ 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.

Masking. Masking consists of randomizing any secret-dependent intermediate variable. Each of these secret-dependent intermediate variables, say $\mathbf{x}$, is split into $d = t + 1$ variables $(\mathbf{x}_i)_{i \in [d]}$ called “shares”. The integer t is referred to as the masking order. We define $d = t+1$ to differentiate the masking order and the number of shares.

The two most deployed types of masking are *arithmetic masking* and *Boolean masking*. The masking of Raccoon has the advantage of being of only one type: *arithmetic masking*. Concretely, in Raccoon, a sensitive variable $\mathbf{x}$ is shared in $(\mathbf{x}_i)_{i \in [d]}$ such that

$$\mathbf{x} = \sum_{i \in [d]} \mathbf{x}_i \bmod q.$$

A d-shared variable $(\mathbf{x}_i)_{i \in [d]}$ will be denoted $[\![\mathbf{x}]\!]$ for readability. The variable d is usually clear from context, but we may use the notation $[\![\mathbf{x}]\!]_d$ when we need to make d explicit.

2.3 Main Functions

This section describes our main functions: key generation (Section 2.3.1), signing (Section 2.3.2) and verification (Section 2.3.3). 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.

2.3.1 Key Generation

Masked key generation process is described by Algorithm 1, with pointers to auxiliary functions in Section 2.4. For details about the encoding of public and private keys, see Section 2.5.

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 the specialized Algorithm 8. This ensures that even a t-probing adversary only learns limited information about $(\mathbf{s}, \mathbf{e})$.

Algorithm 1 KeyGen$() \to (\mathsf{vk}, \mathsf{sk})$

Input: $\emptyset$
Output: Public (signature verification) key vk.
Output: Private (signing) key sk

1: $\mathsf{seed} \leftarrow \{0,1\}^\kappa$ ▷ κ-bit random seed for $\mathbf{A}$.
2: $\mathbf{A} := \mathsf{ExpandA}(\mathsf{seed})$ ▷ Uniform matrix $\mathbf{A} \in \mathcal{R}_q^{k \times \ell}$. Algorithm 6.
3: $\llbracket \mathbf{s} \rrbracket \leftarrow \ell \times \mathsf{ZeroEncoding}(d)$ ▷ Masked zero vector $\llbracket \mathbf{s} \rrbracket \in (\mathcal{R}_q^\ell)^d$. Algorithm 12.
4: $\llbracket \mathbf{s} \rrbracket \leftarrow \mathsf{AddRepNoise}(\llbracket \mathbf{s} \rrbracket, u_\mathrm{t}, \mathsf{rep})$ ▷ Generate the secret distribution. Algorithm 8.
5: $\llbracket \mathbf{t} \rrbracket := \mathbf{A} \cdot \llbracket \mathbf{s} \rrbracket$ ▷ Compute masked product $\llbracket \mathbf{t} \rrbracket \in (\mathcal{R}_q^k)^d$.
6: $\llbracket \mathbf{t} \rrbracket \leftarrow \mathsf{AddRepNoise}(\llbracket \mathbf{t} \rrbracket, u_\mathrm{t}, \mathsf{rep})$ ▷ Add masked noise to $\llbracket \mathbf{t} \rrbracket$. Algorithm 8.
7: $\mathbf{t} := \mathsf{Decode}(\llbracket \mathbf{t} \rrbracket)$ ▷ Collapse $\mathbf{t} \in \mathcal{R}_q^k$. Algorithm 13.
8: $\mathbf{t} := \lfloor \mathbf{t} \rceil_{\nu_t}$ ▷ Rounding and right-shift to modulus $q_t = \lfloor q/2^{\nu_t} \rfloor$.
9: **return** $(\mathsf{vk} := (\mathsf{seed}, \mathbf{t}), \mathsf{sk} := (\mathsf{vk}, \llbracket \mathbf{s} \rrbracket))$ ▷ Return serialized key pair.

2.3.2 Signing Procedure

Masked signing process is described by Algorithm 2. We recall that Raccoon has a Fiat-Shamir structure. Concretely, the signing procedure follows a similar flow to, e.g., Schnorr or ECDSA signatures:

1. **Commit.** Ephemeral random noise is generated in masked form, and an LWE commitment $\mathbf{w}$ is computed in masked form, then unmasked (Lines 4 to 9); in KeyGen, ephemeral random noise generation is done through AddRepNoise, in order to limit the information learned by a probing adversary;

2. **Challenge.** A challenge is computed as a function of the message msg, the verification key vk and the LWE commitment $\mathbf{w}$ (Lines 10 and 11); as in Dilithium, this computation is split in two subroutines (ChalHash and ChalPoly), as it is more convenient for implementation;

3. **Response.** A response $(\mathbf{h}, \mathbf{z})$ is computed from the ephemeral random noise, private key and challenge (Lines 14 to 18). The first part of this computation is performed masked, and the second part is unmasked since it only involves public data. Note that $\mathbf{y}$ is computed over $\mathcal{R}_q$ by naturally lifting $\mathbf{t} \in \mathcal{R}_{q_t}^k$ to $\mathcal{R}_q^k$.

The signature is a serialization of the challenge and the response (Line 19). The signer checks some bounds relative to the signature before outputting it (Line 20). Note that Line 20 is *not* a rejection sampling step; there is no need to mask it.

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

Input: Secret signing key $\mathsf{sk} = (\mathsf{vk}, [\![\mathbf{s}]\!])$
Input: Message to be signed $\mathsf{msg} \in \{0,1\}^*$.
Output: Signature $\mathsf{sig} = (c_{\mathsf{hash}}, \mathbf{h}, \mathbf{z})$ of msg under sk.

1: $(\mathsf{vk}, [\![\mathbf{s}]\!]) := \mathsf{sk}, (\mathsf{seed}, \mathbf{t}) := \mathsf{vk}$ ▷ Deserialize variables from $[\![\mathsf{sk}]\!]$.
2: $\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk}) \| \mathsf{msg})$ ▷ Bind vk with msg to form $\mu \in \{0,1\}^{2\kappa}$.
3: $\mathbf{A} := \mathsf{ExpandA}(\mathsf{seed})$ ▷ Uniform matrix $\mathbf{A} \in \mathcal{R}_q^{k \times \ell}$. Algorithm 6.
4: $[\![\mathbf{r}]\!] \leftarrow \ell \times \mathsf{ZeroEncoding}(d)$ ▷ Masked zero vector $[\![\mathbf{r}]\!] \in (\mathcal{R}_q^{\ell})^d$. Algorithm 12.
5: $[\![\mathbf{r}]\!] \leftarrow \mathsf{AddRepNoise}([\![\mathbf{r}]\!], u_{\mathrm{w}}, \mathsf{rep})$ ▷ Add masked noise to $[\![\mathbf{r}]\!]$. Algorithm 8.
6: $[\![\mathbf{w}]\!] := \mathbf{A} \cdot [\![\mathbf{r}]\!]$ ▷ Compute masked product $[\![\mathbf{w}]\!] \in (\mathcal{R}_q^{k})^d$.
7: $[\![\mathbf{w}]\!] \leftarrow \mathsf{AddRepNoise}([\![\mathbf{w}]\!], u_{\mathrm{w}}, \mathsf{rep})$ ▷ Add masked noise to $[\![\mathbf{w}]\!]$. Algorithm 8.
8: $\mathbf{w} := \mathsf{Decode}([\![\mathbf{w}]\!])$ ▷ Collapse LWE commitment $\mathbf{w}$. Algorithm 13.
9: $\mathbf{w} := \lfloor \mathbf{w} \rceil_{\nu_{\mathrm{w}}}$ ▷ Rounding and right-shift to modulus $q_{\mathrm{w}} = \lfloor q/2^{\nu_{\mathrm{w}}} \rfloor$.
10: $c_{\mathsf{hash}} := \mathsf{ChalHash}(\mathbf{w}, \mu)$ ▷ Map $\mathbf{w}$ and μ to $c_{\mathsf{hash}} \in \{0,1\}^{2\kappa}$. Algorithm 9.
11: $c_{\mathsf{poly}} := \mathsf{ChalPoly}(c_{\mathsf{hash}})$ ▷ Map c_{hash} to $c_{\mathsf{poly}} \in \mathcal{R}_q$. Algorithm 10.
12: $[\![\mathbf{s}]\!] \leftarrow \mathsf{Refresh}([\![\mathbf{s}]\!])$ ▷ Refresh $[\![\mathbf{s}]\!]$ before re-use. Algorithm 11.
13: $[\![\mathbf{r}]\!] \leftarrow \mathsf{Refresh}([\![\mathbf{r}]\!])$ ▷ Refresh $[\![\mathbf{r}]\!]$ before re-use. Algorithm 11.
14: $[\![\mathbf{z}]\!] := c_{\mathsf{poly}} \cdot [\![\mathbf{s}]\!] + [\![\mathbf{r}]\!]$ ▷ Masked response $[\![\mathbf{z}]\!] \in (\mathcal{R}_q^{\ell})^d$.
15: $[\![\mathbf{z}]\!] \leftarrow \mathsf{Refresh}([\![\mathbf{z}]\!])$ ▷ Refresh $[\![\mathbf{z}]\!]$ before collapsing it. Algorithm 11.
16: $\mathbf{z} := \mathsf{Decode}([\![\mathbf{z}]\!])$ ▷ Collapse into response $\mathbf{z} \in \mathcal{R}_q^{\ell}$. Algorithm 13.
17: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - 2^{\nu_t} \cdot c_{\mathsf{poly}} \cdot \mathbf{t}$ ▷ "Noisy" LWE commitment.
18: $\mathbf{h} := \mathbf{w} - \lfloor \mathbf{y} \rceil_{\nu_{\mathrm{w}}}$ ▷ Compute hint $\mathbf{h} \in \mathcal{R}_{q_{\mathrm{w}}}^{k}$. Subtraction mod q_{w}.
19: $\mathsf{sig} := (c_{\mathsf{hash}}, \mathbf{h}, \mathbf{z})$ ▷ Serialize signature. Section 2.5.
20: **if** $\{\mathsf{CheckBounds}(\mathsf{sig}) = \mathsf{FAIL}\}$ **goto** Line 4 ▷ Sanity check on the signature. Algorithm 4.
21: **return** sig ▷ Return encoded signature triplet.

2.3.3 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. Verification operates in a similar way to most lattice-based Fiat-Shamir signatures:

1. A bound check is performed (Line 2);

2. An equality check $c_{\mathsf{hash}} \stackrel{?}{=} \mathsf{ChalHash}\left(\lfloor \mathbf{A} \cdot \mathbf{z} - 2^{\nu_t} \cdot c_{\mathsf{poly}} \cdot \mathbf{t} \rceil_{\nu_{\mathrm{w}}} + \mathbf{h}, \mu\right)$ is performed.

2.4 Auxiliary Functions

2.4.1 Encoding of Variables

Raccoon uses the little-endian convention for byte serialization: the least significant byte of an integer comes first. Bits and polynomial coefficients are also numbered from the least significant bit/coefficient (bit/coefficient zero) toward the more significant ones.

Let $b_0, b_1, \ldots b_{n-1}$ be a sequence of n bits $b_i \in \{0,1\}$. We write $b := \mathsf{Ser}(x)$ to be the serialization of object x into bits, and $x := \mathsf{Deser}(b)$ its inverse.

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

Input: Detached signature $\mathsf{sig} = (c_{\mathsf{hash}}, \mathbf{h}, \mathbf{z})$.
Input: Message whose signature is verified: $\mathsf{msg} \in \{0,1\}^*$.
Input: Public verification key $\mathsf{vk} = (\mathsf{seed}, \mathbf{t})$.
Output: Signature validity: OK (accept) or FAIL (reject).

$(c_{\mathsf{hash}}, \mathbf{h}, \mathbf{z}) := \mathsf{sig}, (\mathsf{seed}, \mathbf{t}) := \mathsf{vk}$ ▷ Deserialize sig and vk. Section 2.5.
if $\mathsf{CheckBounds}(\mathsf{sig}) = \mathsf{FAIL}$ **return** FAIL ▷ Norms check. Algorithm 4.
$\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk}) \| \mathsf{msg})$ ▷ Bind public key with message to form $\mu \in \{0,1\}^{2\kappa}$.
$\mathbf{A} := \mathsf{ExpandA}(\mathsf{seed})$ ▷ Uniform matrix $\mathbf{A} \in \mathcal{R}_q^{k \times \ell}$. Algorithm 6.
$c_{\mathsf{poly}} := \mathsf{ChalPoly}(c_{\mathsf{hash}})$ ▷ Map c_{hash} to $c_{\mathsf{poly}} \in \mathcal{R}_q$. Algorithm 10.
$\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - 2^{\nu_t} \cdot c_{\mathsf{poly}} \cdot \mathbf{t}$ ▷ Scale $\mathbf{t}$ from $\mathbb{Z}_{q_t}$ to $\mathbb{Z}_q$ and recompute the commitment.
$\mathbf{w}' := \lfloor \mathbf{y} \rceil_{\nu_w} + \mathbf{h}$ ▷ Adjust the LWE commitment with hint $(\mathrm{mod}\ q_w)$.
$c'_{\mathsf{hash}} := \mathsf{ChalHash}(\mathbf{w}', \mu)$ ▷ Recompute $c'_{\mathsf{hash}} \in \{0,1\}^{2\kappa}$. Algorithm 9.
if $c_{\mathsf{hash}} \neq c'_{\mathsf{hash}}$ **return** FAIL ▷ Check commitment.
return OK ▷ Signature is accepted.

- **Bit strings and concatenation.** With $c = a \,\|\, b$ we mean that c equals the concatenation of bit strings a and b. Single vertical denotes the lengths: $n_a = |a|$, $n_b = |b|$, $n_c = n_a + n_b = |c|$. Concatenated bit strings satisfy $c_i = a_i$ for $0 \le i < n_a$ and $c_i = b_{i-n_a}$ for $n_a \le i < n_c$.

- **Unsigned integers.** For non-negative integers, $x = \mathsf{Deser}_{2^n}(b) = \sum_{i=0}^{n-1} 2^i \cdot b_i$ with resulting range $0 \le x < 2^n$. Conversely, serialization $b = \mathsf{Ser}_{2^n}(x)$ yields bits $b_i = \lfloor 2^{-i} \cdot x \rfloor \bmod 2$. For unsigned serialization $\mathsf{Ser}_q(x)$ of $x \in \mathbb{Z}_q$ (modular congruence classes) into $n = \lceil \log_2 q \rceil$ bits we normalize x to range $0 \le x < q$ and use $b = \mathsf{Ser}_{2^n}(x)$. Similarly deserialization of $\mathbb{Z}_q$ elements computes $x = \mathsf{Deser}_{2^n}(b)$ but checks that $0 \le x < q$. The encoding is invalid if this condition is not satisfied.

- **Signed integers.** In signed deserialization, we interpret the highest bit b_{n-1} as a sign bit: $x = \mathsf{Deser}^{\pm}_{2^n}(b) = (\sum_{i=0}^{n-2} 2^i \cdot b_i) - (2^{n-1} \cdot b_{n-1})$. This is equivalent to the common "two's complement" representation. The numerical range representable by n bits is therefore $-2^{n-1} \le x < 2^{n-1}$. We normalize $x \in \mathbb{Z}_r$ to range $-\lfloor r/2 \rfloor \le x < \lceil r/2 \rceil$ before signed serialization $\mathsf{Ser}^{\pm}_r(x)$ into $\lceil \log_2 r \rceil$ bits. When deserializing a signed integer to the result of $x = \mathsf{Deser}^{\pm}_r(b)$ must be in this range or the encoding is invalid.

- **Bits as bytes.** Bit strings are commonly manipulated as arrays of bytes ($\mathbb{Z}_{2^8}^m$). Serialization of m bytes $b = \mathsf{Ser}_{2^8}(v_0, v_1, \ldots, v_{m-1})$ produces $|b| = 8m$ bits. Each byte v_j has a non-negative numerical value satisfying $x = v_j = \sum_{i=0}^{7} 2^i \cdot b_{8j+i}$. Conversely, bit $b_i \in \{0,1\}$ can be extracted from byte $x = v_j$, $j = \lfloor i/8 \rfloor$ with $b_i = \lfloor 2^{(8j-i)} \cdot x \rfloor \bmod 2$.

- **Polynomials and Vectors of Polynomials.** Polynomials (such as $\mathcal{R}_q$ ring elements) $F(x) = \sum_{i=0}^{n-1} f_i \cdot x^i$ are serialized as a concatenation of n coefficient integers $f_i \in \mathbb{Z}_q$: $\mathsf{Ser}_q(F) = \mathsf{Ser}_q(f_0) \mid \cdots \mid \mathsf{Ser}_q(f_{n-1})$, requiring $n\lceil \log_2 q \rceil$ bits. Signed polynomial serialization $b = \mathsf{Ser}^{\pm}_r(F)$ and deserialization $F = \mathsf{Deser}^{\pm}_r(b)$ works the same way but uses signed integer representation for all coefficients. Vectors of polynomials $\mathsf{Ser}_q(\mathbf{x})$ are concatenated in increasing index order.

2.4.2 Symmetric Cryptography: SHAKE256

Raccoon uses the SHAKE256 Extendable-Output Function (XOF) as its sole symmetric cryptographic building block. It is defined in the SHA-3 standard FIPS 202 [NIS15]. A permutation-based XOF such as SHAKE256 can be abstracted into initialize, absorb (write), and squeeze (read) phases:

- **Initialize**: $\mathsf{XOF.init}(m_0)$ clears the state of XOF and loads the first (possibly zero-length) message chunk m_0 into it. Note that parameter m_0 is optional: $\mathsf{XOF.init}(m_0)$ is equivalent to $\mathsf{XOF.init}()$ followed by $\mathsf{XOF.input}(m_0)$
- **Absorb**: $\mathsf{XOF.input}(m_i)$ mixes an arbitrarily-length message m_i into the internal state of the XOF. This step can be repeated any number of times. Note that chunk lengths $|m_i|$ are not authenticated: repeated updates with data items $m_1, m_2, m_3 \cdots$ result in the same state as a single update with their concatenation $(m_1 \| m_2 \| m_3 \| \cdots)$.
- **Squeeze**: $h_i \coloneqq \mathsf{XOF.output}(n_i)$ extracts n_i bits of hash output from the state: we have $h_i \in \{0,1\}^{n_i}$ or $|h_i| = n_i$. The First $\mathsf{XOF.output}()$ call performs padding on the state before producing the first h_0, and no further $\mathsf{XOF.input}()$ calls are possible before the state is re-initialized with $\mathsf{XOF.init}()$. For ease of implementation, n_i is always a multiple of 8 in Raccoon: h_i is $n_i/8$ full bytes.

2.4.3 XOF Inputs: The Domain Separation Prefix

For its internal hashes Raccoon uses a domain-separating input prefix hdr. This is a 64-bit (8-byte) identifier that defines the structure of the rest of the XOF input and also the purpose of the XOF call. The first byte of the prefix is an ASCII letter related to the variable name, while subsequent bytes define further information. This allows many random quantities to be derived from a single seed and also prevents some potential attacks.

For vectors and matrices, the prefix also contains indices that allow computations of large quantities such as $\mathbf{A}$ to be parallelized in "counter mode"; some SIMD architectures (such as AVX2) can compute several Keccak permutations in parallel faster than executing them sequentially.

Note that not all domain separation is technically necessary to thwart perceived attacks. The public key and message hashes (in the μ variable computation) do not use prefix encoding. This is to simplify interfacing as in some use cases the μ message hash is computed externally and passed to a Raccoon hardware module.

2.4.4 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). For information about how the bounds were selected, see Section 2.6.

The function also checks that the compressed signature fits in the allocated fixed space. In signing, the length check is done by the encoding process (after norms checks), while in the verification, it is performed by the decoding process (before the norms checks). The order of checks in this function is not important from a security viewpoint, and an "early FAIL" in CheckBounds will not cause (timing attack) security issues.

The decoded hint vectors $\mathbf{h}$ contain relatively small signed integers, so the computation of L_2 and L_∞ norms is straightforward; these correspond to the sum of squares and the largest absolute number. For the $\mathbf{z}$ coefficients $\mathbf{z}_i$, we use an approximate L_2 bound $2^{-64}B_2^2$ to avoid "big integer" arithmetic: We first compute the absolute value ($\mathrm{abs}(x) = q - x$ if $(x \bmod q) > q/2$ and x otherwise.) The absolute value is shifted right by 32 bits before squaring so that the sum of squares will fit into a 64-bit integer. Comparison is approximate as it is performed with a L_2 bound scaled by $(2^{-32})^2 = 2^{-64}$. Note that forgoing the $\mathrm{abs}(x)$ step before division may give slightly different results if the rounding is not toward zero.

Algorithm 4 $\mathsf{CheckBounds}(\mathsf{sig}) \rightarrow \{\mathsf{OK} \text{ or } \mathsf{FAIL}\}$

Input: Serialized signature $\mathsf{sig} = (c_{\mathsf{hash}}, \mathbf{h}, \mathbf{z})$.
Output: Format validity check OK or FAIL.

if $|\mathsf{sig}| \neq |\mathsf{sig}|_{\mathsf{default}}$ **return** FAIL ▷ Fail if signature length or encoding are anomalous.
$(c_{\mathsf{hash}}, \mathbf{h}, \mathbf{z}) := \mathsf{sig}$ ▷ Deserialize signature.
if $\|\mathbf{h}\|_\infty > \lfloor B_\infty / 2^{\nu_{\mathbf{w}}} \rceil$ **return** FAIL ▷ Scale and round the bound for hints.
if $\|\mathbf{z}\|_\infty > B_\infty$ **return** FAIL ▷ Absolute value check on $\mathbf{z}$.
$h_2 := 2^{(2\nu_{\mathbf{w}} - 64)} \cdot \|\mathbf{h}\|_2^2$ ▷ Scaled sum of squares of $\mathbf{h}$ coefficients.
$z_2 := \sum_i \lfloor \mathsf{abs}(\mathbf{z}_i)/2^{32} \rceil^2$ ▷ Sum of squares of $\mathbf{z}$ coefficient shifted right by 32 bits.
if $(h_2 + z_2) > 2^{-64} B_2^2$ **return** FAIL ▷ Scaled / Approximate Squared Euclidean Norm.
return OK ▷ Passed norms checks.

Seed expansion. SampleQ (Algorithm 5) is used to implement the ExpandA seed expansion function (Algorithm 6). It maps its inputs (a header hdr and a seed σ) to a pseudo-random uniform polynomial $f \in \mathcal{R}_q$. It is also used by the reference implementation in the secret key encoding and decoding process (Section 2.5.3).

Algorithm 5 $\mathsf{SampleQ}(\mathsf{hdr}, \sigma) \rightarrow \mathcal{R}_q$

Input: Domain separation header $\mathsf{hdr} \in \{0,1\}^{64}$.
Input: Secret key or public seed $\sigma \in \{0,1\}^{\kappa}$.
Output: Uniform polynomial $f \in \mathcal{R}_q$.

$\mathsf{XOF.init}(\mathsf{hdr})$ ▷ 64-bit domain separation header. XOF defined in Section 2.4.2.
$\mathsf{XOF.input}(\sigma)$ ▷ Absorb the public seed or secret key material.
for $i \in \{0, 1, \cdots, n-1\}$ **do** ▷ Generate coefficients $f_0, f_1, \cdots, f_{n-1}$ in this order.
  repeat ▷ Rejection sampler loop.
    $b := \mathsf{XOF.output}(56)$ ▷ Squeeze $\lceil 49/8 \rceil = 7$ bytes (56 bits) from XOF.
    $f_i := \mathsf{Deser}_{2^{49}}(b_0 \| b_1 \| \cdots \| b_{48})$ ▷ Take low 49 bits, unsigned little-endian.
  until $0 \leq f_i < q$ ▷ Discard if not in range $0 \leq f_i < q$.
return $f(x) := \sum_{i \in [n]} f_i \cdot x^i$ ▷ Coefficients of (at most) $n-1$ degree polynomial.

Generating A. One can generate the entries of the matrix $\mathbf{A}$ in any order (or entirely in parallel) with SampleQ (Algorithm 5) as shown in ExpandA (Algorithm 6). Based on memory and performance considerations, implementors may choose to generate elements $\mathbf{A}_{i,j}$ on the fly (when required), or to generate several elements in parallel.

Algorithm 6 $\mathsf{ExpandA}(\mathsf{seed}) \rightarrow \mathbf{A}$

Input: Public $\mathsf{seed} \in \{0,1\}^{\kappa}$.
Output: $k \times \ell$ generator matrix $\mathbf{A}$.

for $i \in [k]$ **do** ▷ Rows: Calculation order is arbitrary.
  for $j \in [\ell]$ **do** ▷ Columns: Calculation order is arbitrary.
    $\mathsf{hdr}_A := \mathsf{Ser}_{2^8}(65, i, j, 0, 0, 0, 0, 0)$ ▷ 64 bits: 'A', row, column, 5 zero bytes.
    $\mathbf{A}_{i,j} := \mathsf{SampleQ}(\mathsf{hdr}_A, \mathsf{seed})$ ▷ Uniform polynomial.
return $\mathbf{A}$ ▷ Public generator matrix.

2.4.5 Error Distributions

AddRepNoise (Algorithm 8) implements the Sum of Uniforms (SU) distribution $\mathrm{SU}(u, d \cdot \mathsf{rep})$ (Section 2.2) in a masked implementation. AddRepNoise interleaves noise additions and refresh operations; more precisely, for each (masked) coefficient $[\![a]\!]$ of $[\![\mathbf{v}]\!]$, small uniform noise is added to each share of $[\![a]\!]$, then $[\![a]\!]$ is refreshed, and this operation is repeated rep times.

Internally, the function uses SampleU (Algorithm 7) to expand κ-bit seeds σ into pseudorandom polynomials with coefficients in the set $\{-2^{u-1}, \ldots, 2^{u-1} - 1\}$.

Algorithm 7 $\mathsf{SampleU}(\mathsf{hdr}, \sigma, u) \to (f_i)_{i \in [n]}$

Input: Domain separation header $\mathsf{hdr} \in \{0,1\}^{64}$.
Input: Distribution parameter u ("bits").
Input: Secret key material $\sigma \in \{0,1\}^{\kappa}$.
Output: $\mathcal{R}_q$ coefficients satisfying $-2^{u-1} \le f_i < 2^{u-1}$.

```
XOF.init(hdr)                      ▷ Domain separation header. XOF defined in Section 2.4.2.
XOF.input(σ)                       ▷ Absorb the secret.
for i ∈ [n] do                     ▷ Generate f_0, f_1, ··· , f_{n−1}.
    b := XOF.output(8⌈u/8⌉)        ▷ Squeeze full bytes from XOF.
    f_i := Deser±_{2^u}(b_0‖b_1‖···‖b_{u−1})   ▷ Deserialize low u bits, signed little-endian.
return (f_0, f_1, ··· , f_{n−1})   ▷ Polynomial Σ_{i∈[n]} f_i x^i coefficients.
```

Algorithm 8 $\mathsf{AddRepNoise}([\![\mathbf{v}]\!], u, \mathsf{rep}) \to [\![\mathbf{v}]\!]$

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

```
for i ∈ [len(v)] do                        ▷ Vector index.
    for i_rep ∈ [rep] do                   ▷ Repetition index.
        for j ∈ [d] do                     ▷ Share index.
            σ ← {0,1}^κ                    ▷ Secret key material (use RBG).
            hdr_u := Ser_{2^8}(117, i_rep, i, j, 0, 0, 0, 0)   ▷ 64 bits: 'u', rep, idx, share.
            v_{i,j} ← v_{i,j} + SampleU(hdr_u, σ, u)   ▷ Add small uniform to the polynomial.
        ⟦v_i⟧ ← Refresh(⟦v_i⟧)             ▷ Refresh polynomial on each repeat.
return ⟦v⟧
```

2.4.6 Challenge Computation

As in Dilithium, the challenge computation is split in two subroutines: ChalHash (Algorithm 9) and ChalPoly (Algorithm 9). This makes implementation simpler, as the signing procedure calls ChalHash followed by ChalPoly, whereas the verification procedure calls ChalPoly followed by ChalHash. These functions do not need masking or timing attack protection.

Challenge hash computation. The function ChalHash (Algorithm 9) is used to compute a 2κ-bit digest of the commitment $\mathbf{w}$ and message hash μ (bound to public key vk). This is a straightforward hash computation.

Algorithm 9 $\mathsf{ChalHash}(\mathbf{w}, \mu) \to c_{\mathsf{hash}}$

Input: Commitment $\mathbf{w} = (w_i)_{i\in[k]}$.
Input: Message hash $\mu = \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg}) \in \{0,1\}^{2\kappa}$.
Output: A challenge digest $c_{\mathsf{hash}} \in \{0,1\}^{2\kappa}$.

1: $\mathsf{hdr}_h := \mathsf{Ser}_{2^8}(104, k, 0, 0, 0, 0, 0, 0)$ ▷ 64-bits: `'h'`, authenticate k, 6 zero bytes.
2: $\mathsf{XOF.init}(\mathsf{hdr}_h)$ ▷ Add header. XOF defined in Section 2.4.2.
3: $\mathsf{XOF.input}(\mathsf{Ser}_{2^8}(\mathbf{w}))$ ▷ The $\mathbf{w}$ vector is serialized as bytes.
4: $\mathsf{XOF.input}(\mu)$ ▷ Add the message hash.
5: **return** $c_{\mathsf{hash}} := \mathsf{XOF.output}(2\kappa)$ ▷ Collision resistance.

Challenge polynomial computation. ChalPoly (Algorithm 10) expands a 2κ-bit challenge hash into a "ternary" polynomial with "weight" ω elements: Exactly ω coefficients of the resulting c_{poly} polynomial are either $+1$ or -1 and the rest are zeros. The set C of such polynomials is commonly referred to as the "challenge space".

Internally, ChalPoly starts from an all-zero vector $\mathbf{c}$ of dimension n, selects pseudo-random coefficients of $\mathbf{c}$, and sets them to ± 1 until the Hamming weight is ω. If a selected coefficient is already set, then ChalPoly moves on to the next coefficient. The pseudo-randomness is obtained by passing c_{hash} into a XOF. By a coupon collector argument, the expected number of iterations of the **while** loop in ChalPoly is $\sum_{i<\omega} \frac{n}{n-i} \leq \frac{\omega}{1-\omega/n}$. Thus its average bit consumption is $\frac{\omega}{1-\omega/n} \log n$, which is equivalent to $\omega \log n$ when $\omega = o(n)$.

ChalPoly serves the same purpose as Dilithium's "SampleInBall" algorithm: mapping a hash digest to a polynomial of fixed weight. Both algorithms employ different strategies, but their (pseudo-)randomness consumptions are similar.

Algorithm 10 $\mathsf{ChalPoly}(c_{\mathsf{hash}}) \to c_{\mathsf{poly}}$

Input: A hash digest $c_{\mathsf{hash}} \in \{0,1\}^{2\kappa}$
Output: A polynomial c_{poly} in the challenge space C

1: $\mathsf{hdr}_c := \mathsf{Ser}_{2^8}(99, \omega, 0, 0, 0, 0, 0, 0)$ ▷ 64 bits: `'c'`, authenticate ω.
2: $\mathsf{XOF.init}(\mathsf{hdr}_c)$ ▷ Add header. XOF defined in Section 2.4.2.
3: $\mathsf{XOF.input}(c_{\mathsf{hash}})$ ▷ Challenge hash.
4: $\mathbf{c} = (c_i)_{i\in[n]} := 0^n$ ▷ Initialize as a zero polynomial.
5: **while** $\|\mathbf{c}\|_1 \leq \omega$ **do** ▷ Less than ω non-zero coefficients.
6: $\quad b := \mathsf{XOF.output}(16)$ ▷ Squeeze two bytes from the XOF.
7: $\quad i = \mathsf{Deser}_n(b_1\|\cdots\|b_9)$ ▷ Shift 1 bit right, mask to get $0 \leq i < n$.
8: $\quad$ **if** $(c_i = 0)$ **then** ▷ Is this a zero coefficient?
9: $\qquad c_i := (-1)^{b_0}$ ▷ The least significant bit determines sign.
10: **return** $c_{\mathsf{poly}} := \sum_{i\in[n]} c_i \cdot x^i$ ▷ Coefficients of $n-1$ degree polynomial.

2.4.7 Refresh and Decoding Gadgets

Algorithms 11 and 12 describe the refresh gadgets that can be used to achieve $O(d \log d)$ complexity for the overall key generation and signing processes.

Refresh. Refresh (Algorithm 11) is used to generate 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 12) as a subroutine. Both algorithms perform $O(d \log d)$ basic operations over $\mathcal{R}_q$ and require $O(d\ \log(d)\ \log(q))$ bits of entropy. While we present

ZeroEncoding as a recursive algorithm, it is easy to see that it can be computed in-place and its memory requirement is $O(d)$.

Algorithm 11 Refresh$(\llbracket x \rrbracket) \to \llbracket x \rrbracket'$

Input: A d-sharing $\llbracket x \rrbracket$ of $x \in \mathcal{R}_q$
Output: A fresh d-sharing $\llbracket x \rrbracket$ of x

$\llbracket z \rrbracket \leftarrow$ ZeroEncoding(d)
return $\llbracket x \rrbracket' := \llbracket x \rrbracket + \llbracket z \rrbracket$

Algorithm 12 ZeroEncoding$(d) \to \llbracket z \rrbracket_d$

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 $0 \in \mathcal{R}_q$

if $d = 1$ **then**
    return $\llbracket z \rrbracket_1 := (0)$ ▷ There is only one way to encode zero into 1 share.
$\llbracket z_1 \rrbracket_{d/2} \leftarrow$ ZeroEncoding$(d/2)$ ▷ Recursively obtain left side.
$\llbracket z_2 \rrbracket_{d/2} \leftarrow$ ZeroEncoding$(d/2)$ ▷ Recursively obtain right side.
$\llbracket r \rrbracket_{d/2} \xleftarrow{M} \mathcal{R}_q^{d/2}$ ▷ Sampled using a Mask Random Generator (MRG).
$\llbracket z_1 \rrbracket_{d/2} := \llbracket z_1 \rrbracket_{d/2} + \llbracket r \rrbracket_{d/2}$ ▷ Add to the left side.
$\llbracket z_2 \rrbracket_{d/2} := \llbracket z_2 \rrbracket_{d/2} - \llbracket r \rrbracket_{d/2}$ ▷ Subtract from the right side.
return $\llbracket z \rrbracket_d := \left(\llbracket z_1 \rrbracket_{d/2} \parallel \llbracket z_2 \rrbracket_{d/2}\right)$ ▷ Concatenate the two.

Decoding. Decode (Algorithm 13) is a decoding gadget. We expect that the shares are refreshed before Decode is called. In KeyGen the Decode gadget (Line 7 of Algorithm 1) immediately follows a Refresh contained as the last step of AddRepNoise (Line 7 of Algorithm 8.) Similarly, the first instance of Decode in Sign (Line 8 of Algorithm 2) follows a AddRepNoise. The second instance of Decode in Sign (Line 16) follows an explicit Refresh on Line 15.

Algorithm 13 Decode$(\llbracket x \rrbracket) \to x$

Input: A d-sharing $\llbracket x \rrbracket = (x_i)_{i \in [d]}$ of $x \in \mathcal{R}_q$
Output: The clear value $x \in \mathcal{R}_q$

return $x := \sum_{i \in [d]} x_i$

2.5 Serialization and Deserialization

Raccoon specifies encoding formats for serializing signatures and public keys as fixed-length sequences of bytes ("blobs"). We recommend that these byte strings are embedded into application formats (e.g., as OCTET STRING or BIT STRING types in ASN.1 encoded X.509 certificates) and not dissected and re-serialized into custom formats.

Signatures (sig) and public keys (vk) at a given κ security level are interoperable: their encoding does *not* depend on the number of shares d used in the signing process, and the verifier does not need to know d. Hence a single format and |sig| and |vk| byte size is given for each of Raccoon-128 (Table 2), Raccoon-192 (Table 3), and Raccoon-256 (Table 4).

The encoding and size of the private key sk does depend on the number of shares d, and generally, a private key generated with a given d parameter set should also be used with a signing process with the same parameters.

We view the storage and encoding of masked private keys to be an application-specific issue; secret key interoperability is secondary to side-channel security considerations, which impact secret key formatting and management techniques. The format given here was selected mainly for the benefit of KAT testability and is not necessarily ideal for all use cases.

For strong unforgeability, we aim at having a unique representation for all quantities available to attackers; if an "invalid/alternative encoding" is discovered during deserialization, implementations must reject the entire signature or key.

2.5.1 Signature Format (sig)

The Raccoon signature as used in Signing and Verification functions consists of three components $\mathsf{sig} := (c_{\mathsf{hash}}, \mathbf{h}, \mathbf{z})$. It is encoded as a concatenation of these three elements as bit strings. While c_{hash} is always 2κ bits, a simple Huffman/unary-type entropy encoding is used to condense $\mathbf{h}$ and $\mathbf{z}$ components as they have non-uniform, roughly Gaussian distributions.

Zero padding and length rejection. The encoding introduces variation to the length of $\mathbf{h}$ and $\mathbf{z}$; however, the Raccoon signature blob is constant length $|\mathsf{sig}|$ as the remainder is padded with zero bits. Over-long signatures (greater than $|\mathsf{sig}|$ bytes) are rejected by CheckBounds (Line 20 in Algorithm 2), the signing process is restarted, until a signature fits the length. The rejection probability is designed to be low, with a restart rate $p < 10^{-4}$ due to signature length overflow (See Section 2.6).

Encoding of hint h. Given the zero-dominant distribution of the hint vector (which is signed), zeros $x = 0$ are encoded directly as a single zero bit. Encoding of nonzero values starts with the unary encoding of the absolute value: $a = \mathrm{abs}(x)$ as $a \times 1$-bits, followed by a single 0 stop bit. In the case of $x \neq 0$, the last (highest) bit is the sign bit; 0 for $x > 0$ or 1 for $x < 0$.

Code	Binary	Hint
`(0)`	`0b0`	0
`(1,0,0)`	`0b001`	+1
`(1,0,1)`	`0b101`	−1
`(1,1,0,0)`	`0b0011`	+2
`(1,1,0,1)`	`0b1011`	−2
`(1,1,1,0,0)`	`0b00111`	+3
`(1,1,1,0,1)`	`0b10111`	−3
`...`		
`(1,1,1,1,1,1,0,0)`	`0b00111111`	+6
`(1,1,1,1,1,1,0,1)`	`0b10111111`	−6

We note that due to the little-endian interpretation of bits in serialization, a common binary representation such as `0b1011 = 0xB` is read from "right to the left" and interpreted as $(1, 1, 0, 1) = -2$. The sign bit (for nonzero values) is the most significant bit and also the last bit when proceeding in little-endian order. As an example, a stand-alone zero byte (at byte boundary) would represent a segment of eight zero coefficients, while `0x3F` would represent a single coefficient +6, and `0xBF` byte decodes as −6.

Encoding of response z. The distribution $\mathbf{z}$ is also approximately Gaussian but with a much higher standard deviation, around 2^{41}. We first encode the low 40 bits of absolute coefficient $a = \mathrm{abs}(x) \bmod 2^{40}$ directly as 40 bits. Then the high part $b = \lfloor \mathrm{abs}(x)/2^{40} \rfloor$ is encoded as $b \times 1$ bits, followed by stop bit 0, and a sign bit for nonzero values: 0 for $x > 0$ and 1 for $x < 0$.

There is no sign bit for $x = 0$: The encoding of coefficient $x = 0$ for $\mathbf{z}$ consists of 40 bits for $a = 0$ and a single 0 stop bit for unary encoding of $b = 0$ (41 zero bits total, no sign bit.)

Example: Encoding a coefficient $x = -\lfloor 2^{40}\pi \rfloor = -\texttt{0x3243F6A8885}$ for z yields low part $a = \mathrm{abs}(x) \bmod 2^{40} = \texttt{0x243F6A8885}$ and high part $b = 3$. If the bit encoding starts from a byte boundary, the 45-bit encoding would be `85 88 6A 3F 24 17` where the first 5 bytes correspond to a (bytes in little-endian order), the hexadecimal digit 7 in the last byte is an encoding of $b = (1, 1, 1, 0)$, and the final bit 44 (odd-numbered high digit in the last byte) indicates a negative sign. The high 3 bits of the last byte `0x17` are determined by the next coefficient.

2.5.2 Public Key Format (vk)

The Raccoon public key $\mathsf{vk} := (\mathsf{seed}, \mathbf{t})$ is a concatenation of a κ-bit seed (used to generate $\mathbf{A}$), and a vector $\mathbf{t}$ encoded in unsigned format $\mathsf{Ser}_{q_t}(\mathbf{t})$ with modulus $q_t = \lfloor q/2^{\nu_t} \rfloor$. Hence each coefficient encoded into $\lceil \log_2 q_t \rceil$ bits. The size of the public key is $(\kappa + kn(49 - \nu_t))$ bits.

2.5.3 Secret Keys (sk) in the Reference Implementation

Warning. A masked secret key can't be treated as a static key. For secure usage, one needs to refresh the masked encoding of the $[\![\mathbf{s}]\!]$ secret key component every time it is used, even though the (decoded) key itself remains the same. There are solutions based on masked symmetric cryptography such as "WrapQ" [Saa23] that allow fixed key re-use and which have a comparable encoding size to the secret key serialization used in the reference implementation.

Description and rationale. The main purpose of this format is to support the NIST API and Known Answer Test (KAT) testing functionality. During serialization these functions produce a unique masking serialization for a given $[\![\mathbf{s}]\!]$ depending on the state of the RBG(s) only; an implementation can use an arbitrary MRG (See Section 2.8) and still have matching KATs without having to store the secret key in a completely insecure decoded format.

Due to NIST API conventions, the public key vk is not separately available for the signing process but is duplicated in the secret key $\mathsf{sk} = (\mathsf{vk}, [\![\mathbf{s}]\!]_c)$. The public values $\mathsf{vk} = (\mathsf{seed}, \mathbf{t})$ are decoded from the beginning of the secret key blob as described in Section 2.5.2.

Note that the secret key vector is encoded in NTT transformed domain $[\![\hat{\mathbf{s}}]\!]$ (See Section 2.7.1). The reference implementation uses MaskCompress (Algorithm 14) to serialize (export) secret keys $[\![\hat{\mathbf{s}}]\!]$ into $[\![\hat{\mathbf{s}}]\!]_c$ and MaskDecompress (Algorithm 15) to deserialize (import) them back from $[\![\hat{\mathbf{s}}]\!]_c$ to $[\![\hat{\mathbf{s}}]\!]$. Different real use cases may wish to use different types of secret key encodings for additional protection of masking security.

The secret key consists of $(d-1)$ symmetric keys $z_i \in \{0,1\}^\kappa$, followed by a single share $\mathbf{x}$ of ℓ polynomials, encoded in bit-packed format $\mathsf{Ser}_q(\mathbf{x})$. The encoded $[\![\hat{\mathbf{s}}]\!]_c$ size is hence $(d-1)\kappa + \ell n \lceil \log_2 q \rceil$ bits.

Note on key management APIs in high-assurance cryptography. Refreshing of $[\![\hat{\mathbf{s}}]\!]$ between consecutive signature calls is impossible with the API used by the NIST Reference Implementation. APIs in high-assurance implementations avoid passing secret variables directly but operate on them via opaque "handles." These abstract references don't necessarily contain key material; they allow secure key management to be performed in an implementation-specific manner behind the scenes. Examples of such abstractions include the key identifier `psa_key_id_t` in ARM Platform Security Architecture (PSA) Crypto API [ARM22] and `TEE_ObjectHandle` in GlobalPlatform Trusted Execution Environment (TEE) Crypto API [Glo21].

Algorithm 14 MaskCompress($[\![\mathbf{s}]\!]$) $\rightarrow [\![\mathbf{s}]\!]_c$

Input: Shares $[\![\mathbf{s}]\!] \in (\mathcal{R}_q^\ell)^d = (\mathbf{s}_0, \mathbf{s}_1, \cdots, \mathbf{s}_{d-1})$ with $\mathbf{s}_i \in \mathcal{R}_q^\ell$.
Output: Serialized $[\![\mathbf{s}]\!]_c = (z_1, z_2, \cdots, z_{d-1}, \mathbf{x})$ with $z_i \in \{0,1\}^\kappa$ and $\mathbf{x} \in \mathcal{R}_q^\ell$.

$\mathbf{x} := \mathbf{s}_0$ ▷ First share of $[\![\mathbf{s}]\!]$.
for $i \in \{1, 2, \ldots, d-1\}$ **do**
  $z_i \leftarrow \{0,1\}^\kappa$ ▷ Random seed for share i.
  for $j \in [\ell]$ **do** ▷ Sample vector $\mathbf{r} \in \mathcal{R}_q^\ell$ using key z_i.
    $\mathsf{hdr}_K :=$ Ser$_{2^8}(75, i, j, 0, 0, 0, 0, 0)$ ▷ 64 bits: `'K'`, share, index, padding.
    $\mathbf{r}_j :=$ SampleQ(hdr_K, z_i) ▷ Sample uniform polynomial.
  $\mathbf{x} := \mathbf{x} - \mathbf{r}$ ▷ Update $\mathbf{x}$ with new share $\mathbf{r} = (\mathbf{r}_j)_{j \in [\ell]}$.
  $\mathbf{x} := \mathbf{x} + \mathbf{s}_i$ ▷ Update $\mathbf{x}$ with old share $\mathbf{s}_i$.
return $[\![\mathbf{s}]\!]_c := (z_1, z_2, \cdots, z_{d-1}, \mathbf{x})$

Algorithm 15 MaskDecompress($[\![\mathbf{s}]\!]_c$) $\rightarrow [\![\mathbf{s}]\!]$

Input: Serialized $[\![\mathbf{s}]\!]_c = (z_1, z_2, \cdots, z_{d-1}, \mathbf{x})$ with $z_i \in \{0,1\}^\kappa$ and $\mathbf{x} \in \mathcal{R}_q^\ell$.
Output: Shares $[\![\mathbf{s}]\!] \in (\mathcal{R}_q^\ell)^d = (\mathbf{s}_0, \mathbf{s}_1, \cdots, \mathbf{s}_{d-1})$ with $\mathbf{s}_i \in \mathcal{R}_q^\ell$.

$\mathbf{s}_0 := \mathbf{x}$ ▷ Used as-is.
for $i \in \{1, 2, \ldots, d-1\}$ **do** ▷ Expand other shares.
  for $j \in [\ell]$ **do** ▷ Sample vector $\mathbf{s}_i \in \mathcal{R}_q^\ell$ using key z_i.
    $\mathsf{hdr}_K :=$ Ser$_{2^8}(75, i, j, 0, 0, 0, 0, 0)$ ▷ 64 bits: `'K'`, share, index, padding.
    $\mathbf{s}_{i,j} :=$ SampleQ(hdr_K, z_i) ▷ Expand uniform share using z_i.
return $[\![\mathbf{s}]\!] := (\mathbf{s}_0, \mathbf{s}_1, \cdots, \mathbf{s}_{d-1})$

2.6 Provenance of Rejection Bounds

The Raccoon CheckBounds function (Section 2.4.4) verifies that the signature fits into a fixed space |sig| (in signing) or that it is not otherwise malformed or manipulated (in verification.)

Even though Raccoon has a rejection condition in signature generation (CheckBounds on Line 20 in Algorithm 2), the security of Raccoon does not directly depend on "rejection sampling" during the signing phase – which is the purpose of similar bounds with Dilithium.

Instead, the bounds B_2^2 and B_∞ can be seen as an attack countermeasure for the signature verification phase (Line 2 in Algorithm 3). These bounds are merely confirmed in signing so that invalid signatures are not generated.

Consequently, the input variables of CheckBounds can be considered as "already public", and implementing CheckBounds in the signature generation phase does not require special side-channel countermeasures.

2.6.1 Signature Field Size |sig|

We observe that the $\mathbf{h}$ and $\mathbf{z}$ coefficients have a roughly Gaussian distribution and are encoded with Huffman-type variable-length encoding (Section 2.5.1). As a sum of a relatively large number of independent random variables with static distributions, the actual encoded length of a Raccoon signature sig $:= (c_{\mathsf{hash}}, \mathbf{h}, \mathbf{z})$ will also have a roughly Gaussian distribution.

We wanted the scheme to have a fixed signature length but a low rejection rate due to "signature too long" (encoding not fitting the signature field.) The field size was determined experimentally: We approximated the length average $\mu_{|\mathsf{sig}|}$ and standard deviation $\sigma_{|\mathsf{sig}|}$ from 1000 signatures at each parameter set. The signature field size was set four standard deviations from average at $\mu_{|\mathsf{sig}|} + 4 \cdot \sigma_{|\mathsf{sig}|}$, rounded up to next even number (multiple of two bytes.)

- Raccoon-128: $|\mathsf{sig}| = 11524$ bytes (from $11490.3 + 4 \cdot 8.26$)
- Raccoon-192: $|\mathsf{sig}| = 14544$ bytes (from $14502.9 + 4 \cdot 10.19$)
- Raccoon-256: $|\mathsf{sig}| = 20330$ bytes (from $20280.8 + 4 \cdot 12.18$)

The one-sided tail of a normal distribution at $\sigma = 4$ is $\frac{1}{\sqrt{2\pi}} \int_{x=4}^{\infty} e^{-x^2/2} \approx 3.17 \cdot 10^{-5}$, so we can expect that rejections occur (due to insufficient signature space) with a rate $< 10^{-4}$.

2.6.2 Scaled Squared Norm $2^{-64}B_2^2$ and Infinity Norm B_∞

We now discuss how we compute the bounds B_2^2 and B_∞ for the (squared) L_2 (Euclidean) and L_∞ (infinity) norms. Remember that $(\mathbf{z}, \mathbf{h})$ is of this form:

$$(\mathbf{z}, \mathbf{h}) = \left(c \cdot \mathbf{s} + \mathbf{r}, \lfloor \mathbf{w} \rceil_{\nu_\mathrm{w}} - \lfloor \mathbf{u} - c \cdot \mathbf{e} - c \cdot \boldsymbol{\delta}_\mathrm{t} - \mathbf{e}' \rceil_{\nu_\mathrm{w}}\right),$$

Under mild heuristics, we approximate the expected value of the squared Euclidean norm $\|(\mathbf{z}, 2^{\nu_\mathrm{w}}\mathbf{h})\|^2$ as:

$$\beta_{\mathrm{NORM}} := n\Big[\underbrace{(k+\ell)\frac{d \cdot \mathsf{rep}}{12}\left(2^{2u_\mathrm{w}} + \omega\, 2^{2u_\mathrm{t}}\right)}_{\text{Additive errors}} + \underbrace{k\left(\frac{2^{2\nu_\mathrm{w}}}{6} + \omega\,\frac{2^{2\nu_\mathrm{t}}}{12}\right)}_{\text{Rounding}}\Big]. \tag{7}$$

While the approximation Eq. (7) is no longer true if $\nu_\mathrm{w}, \nu_\mathrm{t}$ are too large, it closely matches our experiments for the parameter sets considered in this document. From Eq. (7) we derive bounds for B_2^2 and B_∞, for each variant Raccoon-κ by Eqs. (8) and (9).

$$2^{-64}B_2^2 := \left\lfloor 1.2\,\frac{\beta_{\mathrm{NORM}}}{2^{64}} \right\rfloor \tag{8}$$

$$B_\infty := \left\lfloor 6 \cdot \sqrt{\frac{1}{n(k+\ell)}\,\beta_{\mathrm{NORM}}} \right\rfloor \tag{9}$$

These bound selections result in $p > 0.999$ overall acceptance rate in signing. The choice of the constants 1.2 and 6 in Eqs. (8) and (9) is motivated by heuristically treating $(\mathbf{z}, 2^{\nu_\mathrm{w}}\mathbf{h})$ as a Gaussian vector and using Gaussian tail bounds, e.g. [Lyu12, Lemma 4.4].

2.7 Number Theoretic Transforms

Like many lattice-based schemes, Raccoon implementations use Number Theoretic Transforms (NTT) to implement multiplication in the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$. The use of the specific NTT techniques discussed in this section is optional with Raccoon in the sense that the public keys and signatures are compatible even if ring multiplication is implemented in some other way (public keys or signatures do not contain NTT-domain quantities). However, producing test vectors deterministically that match the reference implementation requires a compatible NTT implementation, and the reference implementation uses the NTT domain for secret key encoding.

2.7.1 NTT Conventions

We use the "hat" (^) to denote transformed ring elements; $\hat{f} = (\hat{f}_0, \hat{f}_1, \cdots \hat{f}_{n-1})$ is a vector of n element $\hat{f}_i \in \mathbb{Z}_q$, just like the polynomial coefficients of $f(x) = \sum_{i=0}^{n-1} f_i x^i$. For forward and inverse transforms we have $\hat{f} = \mathsf{NTT}(f)$, $f = \mathsf{NTT}^{-1}(\hat{f})$. Polynomial multiplication fg mod (x^n+1) satisfies $fg = \mathsf{NTT}^{-1}(\hat{f} \odot \hat{g})$, where the NTT domain (pointwise) multiplication operation $\hat{h} = \hat{f} \odot \hat{g}$ in multiplies vector elements individually: $\hat{h}_i = \hat{f}_i \hat{f}_i$.

Mathematically Raccoon's forward NTT transform evaluates a polynomial $f(x)$ at n roots of unity; specially selected points $z_i \in \mathbb{Z}_q$ that satisfy $z^{2n} \equiv 1 \pmod{q}$. Each evaluation point $\hat{f}_i = f(z_i)$ is an odd power of a subgroup generator $z_0 = g$. To facilitate in-place implementation techniques, the order of these points is "bit-reversed": Let $\mathsf{rev}(b) = \sum_{i=0}^{8} 2^{8-i}(\lfloor 2^{-i} b \rfloor \bmod 2)$. We have $z_i = g^{2\mathsf{rev}(i)+1} (\bmod\ q)$ for $0 \le i < n$. Conversely, the inverse operation NTT^{-1} determines a unique ("interpolation") polynomial f that satisfies $\hat{f}_i = \sum_{j=0}^{n-1} f_j z_i^j$ for all $\hat{f}_i, 0 \le i < n$.

Thanks to the special properties of z_i roots of unity, both $\mathsf{NTT}$and NTT^{-1}can be computed in $O(n \log n)$ arithmetic operations, while the $\hat{f} \odot \hat{g}$ pointwise multiplication is $O(n)$. Hence the overall complexity of NTT-based multiplication $fg = \mathsf{NTT}^{-1}(\mathsf{NTT}(f) \odot \mathsf{NTT}(g))$ is also $O(n \log n)$, compared to quadratic complexity $O(n^2)$ of more straightforward methods.

2.7.2 Provenance of Modulus q and Tweak Constants

The 49-bit Raccoon modulus $q = 549824583172097$ is a composite number $q = q_1 q_2$ consisting of two primes: 24-bit $q_1 = 16515073 = 2^{24} - 2^{18} + 1$ and 25-bit $q_2 = 33292289 = 2^{25} - 2^{18} + 1$. Since both multiplicative orders $q_1 - 1$ and $q_2 - 1$ are divisible by a large power-of-two ($2^{18} \ge 2n$), sufficient roots of unity are available for NTT. NTT can be computed either with composite modulus $(\bmod\ q)$, which is suitable for common 64-bit architectures, or separately $(\bmod\ q_1)$ and $(\bmod\ q_2)$ with 32-bit multipliers. The latter "CRT" (Chinese Remainder Theorem) NTT option may be preferable with some 32-bit microcontroller targets but also with vector/SIMD architectures that only have parallel 32-bit integer multipliers.

The "subgroup generator" $g = 358453792785495$ was selected the following way: We first find the smallest number $x > 1$ that is simultaneously a generator of full multiplicative groups $\mathbb{Z}_{q_1}^*$ and $\mathbb{Z}_{q_2}^*$: this is $x = 15$. We then determine its multiplicative order in composite $\mathbb{Z}_q^*$; smallest m with $x^m \equiv 1 \pmod{q}$ is $m = \mathrm{lcm}(q_1 - 1, q_2 - 1) = 2097414144$. The $2n$:th root of unity g is derived as $g = x^{m/(2n)} \pmod{q} = 358453792785495$.

Note that all odd powers g^{2i+1} also have order $2n$ modulo both q_1 and q_2; this is a required property for "negacyclic" convolutions in ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$. (For "cyclic" convolutions required for $\mathbb{Z}_q[x]/(x^n - 1)$ multiplication, one would choose z_i from n:th roots of unity.)

2.7.3 Sampling into the NTT Domain

The result of ExpandA (Algorithm 6) also requires a forward NTT transform before pointwise multiplication, even though it is uniform in $\mathcal{R}_q$ (the NTT transform does not alter its distribution). This is so that signing and verification functions can be interoperable regardless of the details of the polynomial multiplication implementation (In the case of NTT: The root of unity, ordering of coefficients in the transformed domain, Montgomery reduction, etc.) The relative performance penalty of this technically unnecessary transformation is relatively small.

Functions ChalPoly (Algorithm 10) and SampleU (Algorithm 7) create specific distribution in the "normal" $\mathcal{R}_q$ polynomial representation domain. If NTT-based ring multiplication is implemented, these require a forward NTT transformation before pointwise multiplication.

However, the secret key $[\![\hat{\mathbf{s}}]\!]$ serialization and deserialization processes (Algorithms 14 and 15) used in the reference implementation operate in the NTT domain. If serialization of secret keys is performed in the NTT domain, then they can only be loaded and used by an implementation with a compatible NTT representation.

2.8 RBGs for Secret Key Bits and MRBGs for Masking Bits

We write $x \leftarrow \{0,1\}^\kappa$ to denote a call to a secure sampling of κ random bits. In NIST cryptography, all secret key material is generated with Random Bit Generators (RBGs). FIPS-compliant implementations are expected to use approved methods described in the SP 800-90 series of publications [BK15, TBK+18, BKM+22] for generating secret key bits.

Raccoon uses random bits relatively sparingly, expanding short seeds with an XOF. Only uniform distributions are used in Raccoon, so straightforward rejection samplers suffice to translate random bits to these target distributions.

2.8.1 Random Bit Generators (RBGs)

The NIST standards support Deterministic Random Bit Generators (DRBGs [BK15]) for secret key generation in most applications. DRBGs are more problematic than physical "True" random number generators such as RBG3 [BKM+22] from the viewpoint of masking theory and practice. For a theoretical treatment of deterministic or semi-deterministic generators in the probing model, it may be helpful to consider that there exist d independent random bit generators RBG_i, at least one for each share. Implementors will need to consider various approaches to facilitate both testability and side-channel security.

The RBG1 construction of SP 800-90C 3pd [BKM+22] is intended for devices that don't have an internal randomness source; it is difficult to see how such a device can maintain side-channel security in the long term. RBG2 (which combines a physical entropy source with deterministic generation) may succeed if the entropy source and the symmetric mixing components are implemented carefully. Instantiation of Raccoon with the RBG3 "full entropy" construction is recommended: It has the advantage of using a large amount of true entropy to produce each seed output, making them uncorrelated even under a very powerful side-channel adversary.

Limitations of the Reference Implementation and the NIST API. For cryptographic randomness, the reference implementation makes calls to the NIST-defined API `randombytes()` function, which represents an abstract RBG. This approach is only appropriate to facilitate Known Answer Test (KAT) generation and verification.

2.8.2 Masking Random Generators (MRGs)

The masking countermeasures also require randomness. Its purpose is to make it more difficult for an attacker to determine the algorithm's secret variables from side-channel observations. Masking randomness is distinct from secret key material as its security requirements are determined by the physical attack model rather than purely cryptanalytic factors. We term these generators as Masking Random Generators (MRG) and write $x \xleftarrow{M} \mathcal{D}$ to denote that an MRG suffices to sample x from $\mathcal{D}$ (or some other distribution – different MRGs may be appropriate for different distributions.)

RBGs and MRGs are orthogonal in the sense that the scheme remains secure against purely cryptanalytic (non-side channel) attacks even if an attacker compromises all MRG-generated randomness but somehow none of the RBG-generated material. In Raccoon, the masking random number generators have no effect on the actual secret key or signature generated; the secure RBG entirely determines those. Hence the implementation details of masking randomness do not affect high-level test vectors (albeit they do affect intermediate values).

The reference implementations use placeholder generators as MRGs to facilitate testing; these should not be viewed as a part of Raccoon itself. Implementors should use more appropriate generators in "real life" than the placeholder MRGs contained in the reference implementations.

2.9 Known Answer Tests (KATs)

The electronic submission package contains 18 Known Answer Test (KAT) sets, each with 100 test vectors. These are in the `.rsp` format generated by the NIST-provided KAT generator called `PQCgenKAT_sign.c`. Note that generated files `PQCsignKAT_X.rsp` have secret key size as their index `X`; see Tables 2 to 4 for values of |sk|.

SHA-256 hashes of the generated 100-KAT files `PQCsignKAT_|sk|.rsp`:

```
Raccoon-128-1:  039383b9d9b29c5a9cda63cb93666771c7c09791afaadc941341e0df670229e0
Raccoon-128-2:  71586c2fd1ae47f17cb5c44c2b5351ab48531344041a76357ffc695098d2506c
Raccoon-128-4:  ae6e775feaf9d26eac5d10bec3c742fb7ab8f6716ee96a2ce3cf2c3aa23b8ef0
Raccoon-128-8:  ffbd4df642d15da96624e2b8489b5303a97a7f6a5d60416c72108880746394ea
Raccoon-128-16: 579fbaafde26049c4f4993b28568abfb657da76e5cd0c7a83239e37d4cc43325
Raccoon-128-32: dff454bf03e9c027d70d4443bb394cae3c5af23ed81179889a62bf98a8a916d8
Raccoon-192-1:  bb577467a15ff20d6ac88c3eb7ba3fd6b3a3e7bf8e5bc627890bb027bba8bda5
Raccoon-192-2:  1543992c77e4a3ee08cd93daf1044e2d7816efbb6c572f167e500ee5b6e68d02
Raccoon-192-4:  82f2b834889bacdbcbb48d51f99c15639a235a764714ba858b415fdf546c9dbc
Raccoon-192-8:  b21ecba12cafa88a8337a813e9dac131a50f043f860241f7cd36f8b502233971
Raccoon-192-16: 57e3c6d014c7283806f4cd3d9c83737c6d381202a1649042c499c5c354f7606b
Raccoon-192-32: 49a552559d6a68175996de373232e0863496834c16b4d2772781f0e01469b621
Raccoon-256-1:  031d4976f4c09b90ecec5c535b5ab3bcb020b9cb4f95e17dfdcedb10de1425fc
Raccoon-256-2:  8936afaf3fd6cf5b43716e006977e1c14a2624913bfd23adb850aa141ef2ae91
Raccoon-256-4:  2e3ae8a29435ce8621a98390874fa2193756c87741f02934018650163c57e369
Raccoon-256-8:  893bf614327740610c29781db7973bbfa7069010039bfa9b2ba02a9a675a78ab
Raccoon-256-16: 663ce05beb35184b0012e638ed8c918f945b379a9bd35a97e37141798c320acf
Raccoon-256-32: 594169ee1ddc6238fbbfae0178d0ed8fab9eb0205066fe382f6ff788c775bd58
```

3 Performance Analysis

Quasilinear masking. While demonstrating good performance at each cryptanalytic security level $\kappa \in \{128, 192, 256\}$, a key feature of Raccoon is the $O(d \log d)$ complexity of its masking; performance at each side-channel security level $d \in \{1, 2, 4, 8, 16, 32\}$. Quasilinear masking creates a significant advantage in favor of side-channel defense since side-channel attacks (with realistic "noisy measurements") have been shown to have exponential asymptotics in relation to the number of shares d and the masking order $d - 1$.

In pioneering work, Chari et al. [CJRR99] showed that in the presence of Gaussian noise, the number of side-channel observations required to determine a masked secret value grows exponentially with the masking order $d - 1$. The understanding of this exponential relationship has since been made more precise both theoretically and in practice [DFS19, MRS22, IUH22].

Hence we suggest that Raccoon is compared against signature algorithms and implementations that support masking (or other sufficiently robust side-channel attack countermeasures.) Finding such comparison data for other PQC Signature schemes is difficult as most have not been designed to directly support side-channel countermeasures.

3.1 General Implementation Characteristics

Since the building blocks of Raccoon greatly resemble those of Dilithium, very similar implementation and optimization strategies can be used. Especially when the 32-bit "CRT" arithmetic option (Section 2.7.2) is used, NTT code for Raccoon is essentially equivalent to that of Dilithium on both microcontroller and high-end SIMD targets. The Keccak computations can be parallelized similarly on SIMD targets (Section 2.4.3).

3.2 Performance on the NIST x64 Reference Platform

We emphasize that even though Raccoon is designed for various masking levels, a portable, completely deterministic reference implementation can't have a realistic expectation of high side-channel security. In addition to limitations related to random numbers (Section 2.8.1), the reference implementation is severely limited in its key management (Section 2.5.3), and overall leakage characteristics (due to high-level programming language used, and other factors.)

However, indicative performance characteristics in relation to other schemes can be evaluated this way, as well as the impact of d and various other implementation options. Table 5 and Figure 3 summarize the results. Performance is given in milliseconds and millions of cycles, while memory usage is indicated with the static stack usage of the core function (in bytes.) These apply only to the reference implementation; different speed/memory tradeoffs and much lower memory usage has been attained in size-optimized implementations (Section 3.3.2.)

3.2.1 Description of the Reference Implementation

The functional reference implementation[3] is written in ANSI C for the x64 NIST reference platform. This implementation is self-contained apart from the NIST KAT generation code, which uses an AES implementation from OpenSSL to implement a deterministic random bit generator. The total size of the implementation is roughly 5000 LOC. This includes the Keccak permutation (for SHAKE256), also in ANSI C language, and some other optional components.

For $(\mathrm{mod}\ q)$ modular arithmetic, the implementation uses Montgomery reduction (the NTT code avoids some of these reduction steps with the "lazy reduction" technique.) The code also

[3] Current version of the reference code is available from: `https://github.com/masksign/raccoon`

Table 5: Performance of the Raccoon reference implementation on an Intel PC (Section 3.2.2). Units: ms = milliseconds, Mclk = millions of clock cycles, stack = stack usage in bytes.

Variant	KeyGen			Sign			Verify		
κ-d	ms	Mclk	stack	ms	Mclk	stack	ms	Mclk	stack
128-1	1.000	2.112	24784	2.281	4.817	155952	0.832	1.757	53488
128-2	1.242	2.624	24800	2.563	5.412	180560	=	=	=
128-4	1.646	3.477	33040	3.061	6.465	229776	=	=	=
128-8	4.457	9.413	49472	8.361	17.658	328144	=	=	=
128-16	6.156	13.001	152048	10.695	22.588	529008	=	=	=
128-32	19.829	41.879	283360	35.104	74.140	922480	=	=	=
192-1	1.540	3.252	28896	3.248	6.860	233808	1.309	2.764	65776
192-2	1.872	3.953	28928	3.644	7.697	262512	=	=	=
192-4	2.415	5.101	37152	4.292	9.064	319904	=	=	=
192-8	6.282	13.268	53600	11.410	24.099	434672	=	=	=
192-16	8.542	18.041	156160	14.476	30.574	668288	=	=	=
192-32	26.451	55.866	287472	46.867	98.984	1127296	=	=	=
256-1	2.462	5.199	37088	4.764	10.062	373072	2.156	4.554	82176
256-2	2.926	6.180	37120	5.266	11.123	409968	=	=	=
256-4	3.699	7.811	45344	6.238	13.174	483744	=	=	=
256-8	8.870	18.734	61792	15.830	33.433	631280	=	=	=
256-16	12.149	25.659	164352	20.233	42.732	930432	=	=	=
256-32	36.587	77.272	295664	63.972	135.111	1520512	=	=	=

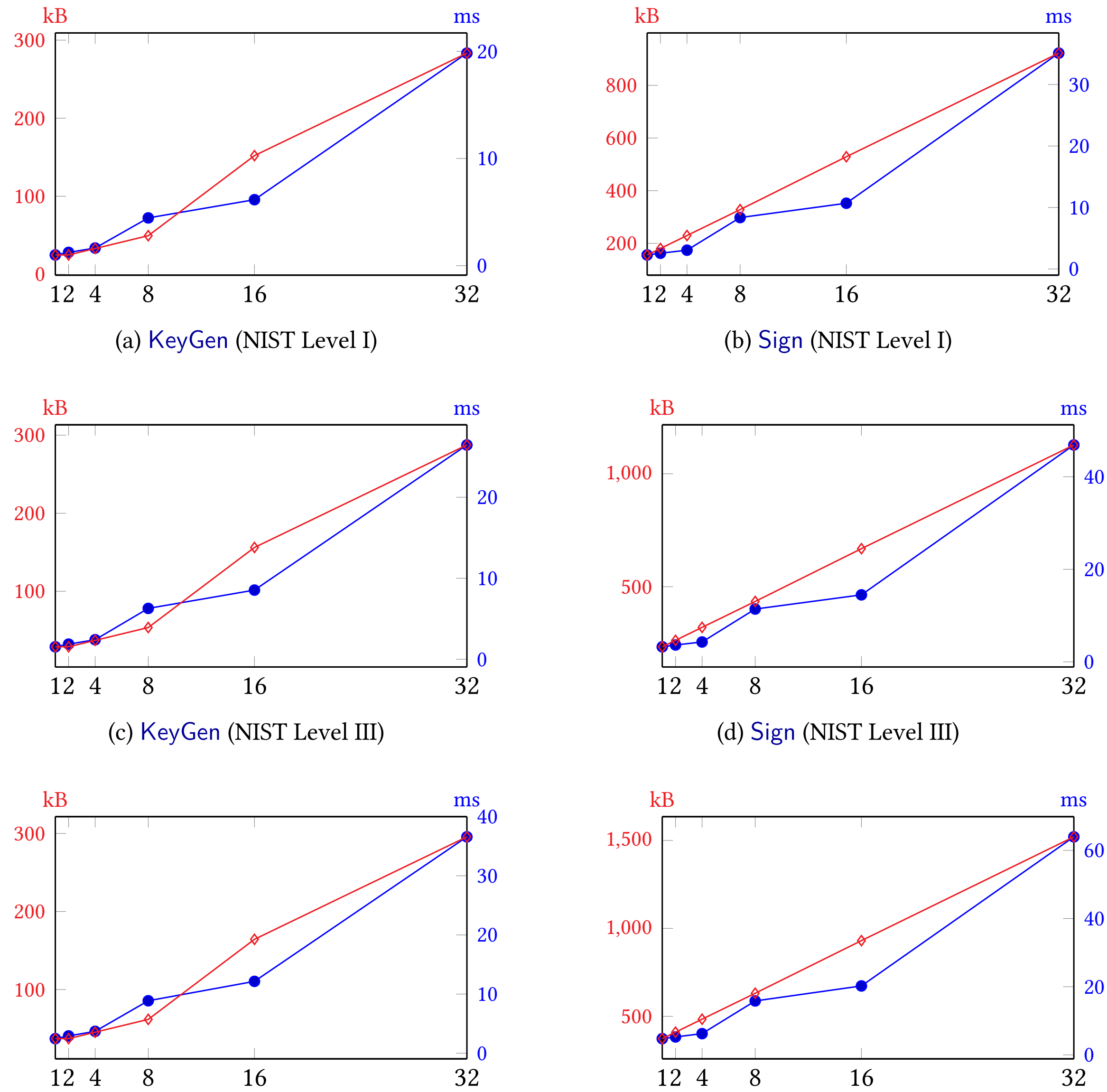
includes a compile-time macro option for 32-bit "Chinese Remainder Theorem" arithmetic (Section 2.7.2), mainly to demonstrate how this technique can be used.

For evaluation purposes, two different alternative MRG implementations are provided; a fast MRG based on a 127-bit LFSR and a second one based on the Ascon [DEMS21] permutation. These can be used to study how the MRG performance impacts the overall performance of Raccoon. Even though the MRGs run in an entirely predictable, deterministic fashion and hence are not actually secure, the interfaces allow multiple independent instances of these generators to be used (an additional measure for probing security.)

3.2.2 Benchmarking Details

The measurements in Table 5 were made using the ANSI C Reference Implementation on Dell OptiPlex XE4, a mid-range 2022 desktop system with an Intel Core i7-12700 CPU running at 2.1gHz. The test programs were executed on a single CPU thread with frequency scaling disabled. The system had 64GB of physical RAM and was running Ubuntu 22.04.2 LTS Linux operating system. The C test code was compiled with gcc 11.3.0 packaged in that operating system. No SIMD/Vector (AVX2 or similar) intrinsics or assembly-level optimizations were used. Compilation and optimization flags were `-Wall -Wextra -Ofast -march=native`.

The test program uses the NIST API calls `crypto_sign_keypair()`, `crypto_sign()`, and `crypto_sign_open()`. Processor time was measured with `clock()` POSIX call (for milliseconds) and with `rdtsc` inline instruction (for cycles.) 64-bit integer arithmetic and the LFSR127-based Masking Random Generator was used. Stack frame size (indicating memory usage) for core functions was obtained with the `-fstack-usage` compile-time option.



(a) KeyGen (NIST Level I)

(b) Sign (NIST Level I)

(c) KeyGen (NIST Level III)

(d) Sign (NIST Level III)

(e) KeyGen (NIST Level V)

(f) Sign (NIST Level V)

Figure 3: Reference implementation running time in milliseconds (—●—, scale on the right side) and stack usage in kilobytes (—◇—, scale on the left side) as functions of the number of shares $d \in \{1, 2, 4, 8, 16, 32\}$, for KeyGen (Algorithm 1) and Sign (Algorithm 2), for NIST levels {I, III, V}.

3.3 Hardware Architectures

Several versions of Raccoon were implemented on FPGA hardware during its development process; one is reported in [dPPRS23]. The current version is being implemented for ASIC.

These implementations contain a RISC-V controller, a Keccak accelerator, and a lattice unit with direct memory access via a 64-bit interface. The lattice unit has hard-coded support for Raccoon's mod q arithmetic. The architecture implements modular multipliers with a fixed-modulus reduction circuit. All variants of Raccoon utilize the same modulus q, allowing "hard-coded" reduction circuitry to be used to implement them all. The unit can perform arbitrary-length vector arithmetic operations such as polynomial addition, coefficient multiplication, and NTT butterfly operations. It also supports Boolean operations and shifts on arrays of words.

Table 6: Raccoon (IEEE SP23 Version [dPPRS23]) hardware implementation cycle counts at various side-channel security levels. The device also supported two-share Dilithium; first-order masking was the highest attainable with the design, but we note that Dilithium2 with 2 shares is already slower than Raccoon-128 with 8 shares.

Algorithm	Shares	KeyGen()	Sign()	Verify()
Raccoon-128	$d = 2$	1,366,000	2,402,000	1,438,000
Raccoon-128	$d = 4$	2,945,000	3,714,230	1,433,034
Raccoon-128	$d = 8$	6,100,000	6,345,000	1,389,000
Raccoon-128	$d = 16$	12,413,000	11,605,000	1,389,000
Raccoon-128	$d = 32$	25,073,000	22,160,000	1,393,000
Dilithium2	$d = 1$	572,000	3,102,000	510,000
Dilithium3	$d = 1$	886,000	5,010,000	725,000
Dilithium5	$d = 1$	1,399,000	5,889,000	1,174,000
Dilithium2	$d = 2$	1,633,000	7,866,000	510,000
Dilithium3	$d = 2$	2,538,000	12,326,000	725,000
Dilithium5	$d = 2$	3,389,000	13,489,000	1,174,000

Since the implementation is designed for side-channel security and masking, the circuitry also has a fast "random fill" MRG function that generates non-deterministic masking randomness rapidly. In a production implementation, this function requires special attention to guarantee that the randomness used in each share is genuinely independent.

3.3.1 XOF Samplers in Hardware

Crucially, the hardware implementation can directly perform streaming SHAKE output and mod q sampling to implement SampleU and SampleQ. Since a full Keccak round is implemented in hardware, it produces output at a very high rate, theoretically a full block (136 bytes for SHAKE-256) every 24 cycles, directly filling arrays in memory.

The hardware XOF sampler eliminates perhaps the most significant performance bottleneck in microcontroller lattice-based PQC implementations: It was initially intended to generate the $k \times \ell$ polynomial matrix $\mathbf{A}$ on the fly with ExpandA (Algorithm 6) in key generation and verification functions. However, in the present implementation, it is also used for AddRepNoise (Algorithm 8), and is also used with MaskCompress and MaskDecompress.

3.3.2 Mask Compression Techniques to Reduce Memory

Hardware implementations of Raccoon may use different types of masking gadgets and implementation techniques to achieve the desired performance, size, and security trade-offs. Our prototype implementations utilize mask compression gadgets (analogous to Algorithms 14 and 15) to reduce the amount of working memory required at higher masking orders. There is a trade-off between memory saving and performance, but even at $d = 32$, the Raccoon-128 implementation operated well with 128 kB of SRAM, while at least 2000 kB would have been required without compression. The (externally stored) secret key $[\![\mathbf{s}]\!]$ size also shrunk from 392 kB to 13 kB, which is essential as non-volatile storage can be more scarce than working memory.

3.3.3 Size and Performance

On an XC7A100T (Xilinx Artix 7) FPGA target, this size-optimized design (including a control Core, Keccak unit, lattice coprocessor, masking random number generator, and communication peripherals) was 10,638 Slice LUTs (16.78%), 4,140 Slice registers / Flip Flops, (3.26%) and only 3 DSPs (as logic was used for multipliers – the design is ASIC-oriented). The design was rated for 78.3 MHz. Table 6 summarizes its performance at various masking levels. We note that some design changes have been made to Raccoon since the publication of [dPPRS23], so these results are merely indicative.

3.4 Leakage Assessments and Vulnerability Analysis

We performed leakage assessments on the FPGA implementations (Section 3.3), following the general test procedure of ISO 17825:2022 [ISO23] and the tools of ISO 20085 [ISO19, ISO20]. This "TVLA" type random-vs-fixed test was adapted to detect leakage from $[\![\mathbf{s}]\!]$ in the signature generation function. The ISO 17825 testing procedure is generally limited to first-order leakage; hence, a $d = 2$ version of Raccoon was used. At $N = 200,000$ traces, the maximum t-value was 4.89 [dPPRS23] – no leakage was detected.

We note that ISO 17825 is referenced as an *approved non-invasive attack mitigation test metric* in Annex F of the current ISO/IEC 19790:2022 Working Draft [ISO22], and hence the most likely testing method to be adopted for FIPS 140-3.

When evaluated under the Common Criteria methodology, high-assurance cryptographic implementations are often required to undergo AVA_VAN.5, Advanced methodical vulnerability analysis [Cri22]. Security against higher-order DPA – and other more advanced attacks where higher-order masking is an effective countermeasure – are assessed and required in these evaluations [SOG22].

4 Security Analysis

This section performs a security analysis of Raccoon. We first provide a black-box security reduction in Section 4.1. Then we argue in Section 4.2 that the impact of a t-probing adversary on the security of Raccoon is small. Based on these two sections, Section 4.3 estimates the concrete security of Raccoon based on state-of-the-art techniques. Finally, Section 4.4 discusses additional security notions.

4.1 Black-box Security Reduction

4.1.1 Hardness Assumptions

We first recall two well-known hardness assumptions over lattices: the MLWE and MSIS assumptions.

Definition 1 (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 $\mathsf{MLWE}_{q,\ell,k,\mathcal{D}}$ *problem is defined as follows:*

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

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 $\mathsf{MLWE}_{q,\ell,k,\mathcal{D}}$ assumption states that any efficient adversary $\mathcal{A}$ against this problem has negligible advantage.

Definition 2 (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 $\mathsf{MSIS}_{q,\ell,k,\beta}$ *problem is defined as follows:*

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

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

In this work, we further rely on the *self-target* MSIS (SelfTargetMSIS) problem [DKL+18a, KLS18]. This is a variant of the standard MSIS problem, where the problem is defined relative to some hash function modeled as a random oracle. Following a standard proof using the forking lemma [FS87, BN06], when the range of the hash function is exponentially large, SelfTargetMSIS is shown to be as hard as MSIS in the random oracle model.[4] In our work, we directly work with SelfTargetMSIS instead since it allows for a simpler proof compared to using MSIS, while putting a focus on concrete security, ignoring the reduction loss incurred by the forking lemma. Indeed, SelfTargetMSIS also underlies the hardness of the signature scheme Dilithium [DKL+18a], recently selected by NIST for standardization, and widely understood to be as concretely secure as MSIS. Formally, SelfTargetMSIS is defined as follows. The concrete hardness of SelfTargetMSIS is analyzed in Section 4.3.5. For completeness, we include more details on the asymptotic hardness of SelfTargetMSIS in Appendix C.1 showing how bit dropping (i.e., $\lfloor\cdot\rceil_v$) interplays with the norm bound β below.

Definition 3 (SelfTargetMSIS)**.** *Let ℓ, k, q, v be integers such that $k < \ell$, and $\beta > 0$ a real number. Let C be a subset of $\mathcal{R}_q$ and let $\mathsf{G} : \mathcal{R}_{q_v}^k \times \{0,1\}^{2\kappa} \to C$ be a cryptographic hash function modeled as a random oracle, where $q_v := \lfloor q/2^v \rfloor$. The advantage of an adversary $\mathcal{A}$ against the* Self Target MSIS *problem, noted $\mathsf{SelfTargetMSIS}_{q,\ell,k,C,v,\beta}$, is defined as:*

[4] As with any invocation of the forking lemma, the reduction comes with a reduction loss dependent on the number of random oracle queries the adversary performs. We note the reduction loss can be tuned using alternative forking strategies [MR02, OO98, PS00].

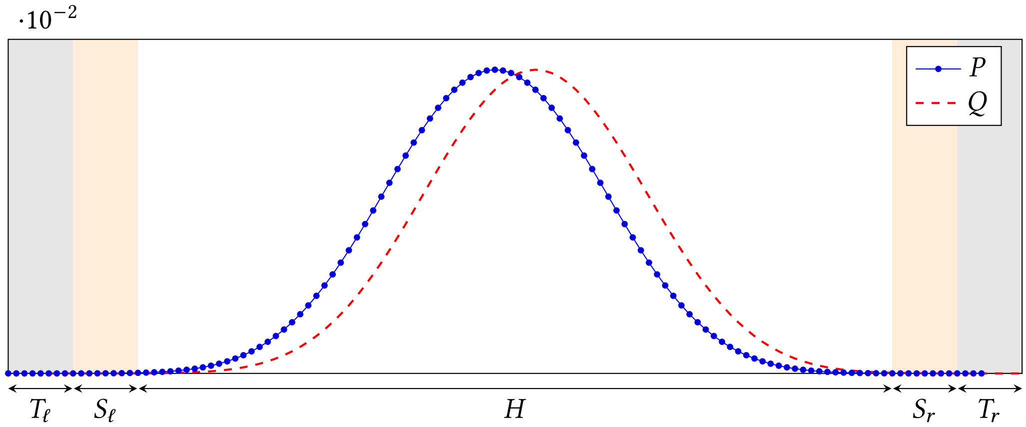


Figure 4: Two copies of $P_{\{n=15,t=8\}}$, shifted by an offset $c = 5$. The areas T_ℓ, S_ℓ, H, S_r and T_r relate to a proof in Appendix A.2

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

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

The worst-case to average-case reductions in the module lattice setting to support the confidence on MLWE and MSIS (or alternatively SelfTargetMSIS) is provided in Appendix D.1. A concrete security analysis of the lattice assumptions we use are provided in Section 4.3.

4.1.2 Smooth Rényi Divergence

We introduce the smooth Rényi divergence. It is motivated by the limitations of the usual Rényi divergence, which is undefined for distributions P, Q of supports not included in one another. This is the case of the two distributions in Figure 4, which left and right "tails" T_ℓ and T_r make the Rényi divergence undefined. The smooth Rényi divergence (Definition 4) addresses these limitations by combining the statistical distance and the Rényi divergence. The statistical distance component captures problematic sets, while the Rényi divergence component benefits from the same efficiency as the usual Rényi divergence over unproblematic parts of the supports.

Definition 4 (Smooth Rényi divergence)**.** *Let* $\epsilon \ge 0$ *and* $1 < \alpha < \infty$*. Let* P, Q *be two distributions of countable supports* $\mathrm{Supp}(P) \subseteq \mathrm{Supp}(Q) = X$*. The smooth Rényi divergence of parameters* (α, ϵ) *between* P *and* Q *is defined as:*

$$R_\alpha^\epsilon(P;Q) = \min_{\substack{\Delta_{\mathrm{SD}}(P';P)\le\epsilon \\ \Delta_{\mathrm{SD}}(Q';Q)\le\epsilon}} R_\alpha(P';Q'), \tag{10}$$

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

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

While [DFPS22] has also provided a definition of smooth Rényi divergence, we argue that our definition is more natural. Indeed, it satisfies variations of properties that are expected from classical Rényi divergences. These are listed in Lemma 13.

We provide a smooth Rényi divergence bound between the sum of uniform distribution, centered at either 0 or a small offset. The proof of the following asymptotic bounds are provided in Appendix A. Below, τ roughly denotes the size of the tails (see Appendix A.2).

Lemma 1. *Let $T, u, N \in \mathbb{N}$ and $c \in \mathbb{Z}$ such that $T \geq 4$ and $N = 2^u$. Let $P = \mathrm{SU}(u, T)$ and Q the distributions corresponding to shifting the support of P by c. Let $\alpha \geq 2$ and $\tau > 0, \epsilon > 0$ be such that:*

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

Then:

$$R_\alpha^\epsilon(P;Q) \leq \left(1 + \frac{\alpha(\alpha-1)}{2}\left(\frac{Tc}{N}\right)^2 + \frac{2}{T!}\left(\frac{T\alpha c}{N}\right)^2 + \epsilon + O\left(\left(\frac{T\alpha c}{N}\right)^3\right)\right)^{1/(\alpha-1)} \tag{11}$$

Gap with practice. In practice, Lemma 1 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 1 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 1 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 Conjecture 1, which ignores this gap and which we use when setting our concrete parameters.

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

$$R_\alpha^\epsilon(P;Q) \lesssim \exp\left(\frac{C_{\text{Rényi}} \cdot \alpha \cdot c^2\,(1 + \frac{2}{\alpha-1})}{T \cdot N^2}\right) \tag{12}$$

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

$$R_\alpha^\epsilon(\mathcal{P};\mathcal{Q}) \lesssim \exp\left(\frac{C_{\text{Rényi}} \cdot \alpha \cdot \|\mathbf{c}\|_2^2}{T \cdot N^2}\right), \tag{13}$$

$$\text{where} \quad \epsilon \approx \frac{\alpha^T\,\|\mathbf{c}\|_T^T}{N^T\,T!} \lesssim \frac{1}{\sqrt{2\,\pi\,T}}\left(\frac{\alpha\,e\,\|\mathbf{c}\|_2}{N\,T}\right)^T \tag{14}$$

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

We note that Eq. (13) is obtained by invoking the tensorization property of the smooth Rényi divergence (see Lemma 13, Item 3) on Eq. (12).

4.1.3 Security Reduction

$\mathsf{KeyGen}() \longrightarrow (\mathsf{vk}, \mathsf{sk})$

1: $\mathsf{seed} \leftarrow \{0,1\}^{\kappa}$
2: $\mathbf{A} \leftarrow \mathsf{ExpandA}(\mathsf{seed})$
3: $\mathbf{s} \leftarrow \mathrm{SU}(u_t, T)^{n\ell}$ ▷ Sample the secret key in $\mathcal{R}_q^{\ell}$.
4: $\mathbf{e} \leftarrow \mathrm{SU}(u_t, T)^{nk}$ ▷ Sample the MLWE noise in $\mathcal{R}_q^{k}$.
5: $\bar{\mathbf{t}} := \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ ▷ Compute an MLWE sample in $\mathcal{R}_q^{k}$.
6: $\mathbf{t} := \lfloor \bar{\mathbf{t}} \rceil_{\nu_t}$ ▷ Drop ν_t bits and move $\bar{\mathbf{t}}$ to $\mathcal{R}_{q_t}^{k}$.
7: $\mathsf{vk} := (\mathsf{seed}, \mathbf{t})$
8: $\mathsf{sk} := (\mathsf{vk}, \mathbf{s})$
9: **return** $(\mathsf{vk}, \mathsf{sk})$

$\mathsf{Sign}(\mathsf{sk}, \mathsf{msg}) \longrightarrow \mathsf{sig}$

1: $\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk}) \| \mathsf{msg})$
2: $\mathbf{r} \leftarrow \mathrm{SU}(u_w, T)^{n\ell}$
3: $\mathbf{e}' \leftarrow \mathrm{SU}(u_w, T)^{nk}$
4: $\mathbf{w} := \lfloor \mathbf{A} \cdot \mathbf{r} + \mathbf{e}' \rceil_{\nu_w}$ ▷ Round commitment to $\mathcal{R}_{q_w}^{k}$.
5: $c_{\mathsf{poly}} := \mathsf{G}(\mathbf{w}, \mu)$ ▷ Note that $\mathsf{G} = \mathsf{ChalPoly} \circ \mathsf{ChalHash}$.
6: $\mathbf{z} := c_{\mathsf{poly}} \cdot \mathbf{s} + \mathbf{r}$
7: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - 2^{\nu_t} \cdot c_{\mathsf{poly}} \cdot \mathbf{t}$ ▷ Lift $\mathbf{t}$ to $\mathcal{R}_q$ and compute $\mathbf{y} \in \mathcal{R}_q^{k}$.
8: $\mathbf{h} := \mathbf{w} - \lfloor \mathbf{y} \rceil_{\nu_w}$ ▷ Compute hint $\mathbf{h} \in \mathcal{R}_{q_w}^{k}$.
9: $\mathsf{sig} := (c_{\mathsf{poly}}, \mathbf{h}, \mathbf{z})$
10: **if** $\mathsf{CheckBounds}(\mathsf{sig}) = \mathsf{FAIL}$ **goto** Line 2 ▷ CheckBounds w/o L_∞- norm check.
11: **return** sig

$\mathsf{Verify}(\mathsf{sig}, \mathsf{msg}, \mathsf{vk}) \longrightarrow \{\mathsf{OK} \text{ or } \mathsf{FAIL}\}$

1: $(c_{\mathsf{poly}}, \mathbf{h}, \mathbf{z}) := \mathsf{sig}$ ▷ Deserialization.
2: **if** $\mathsf{CheckBounds}(\mathsf{sig}) = \mathsf{FAIL}$ **then return** FAIL ▷ CheckBounds w/o L_∞- norm check.
3: $\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk}) \| \mathsf{msg})$ ▷ Bind the public key with message to form $\mu \in \{0,1\}^{2\kappa}$.
4: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - 2^{\nu_t} \cdot c_{\mathsf{poly}} \cdot \mathbf{t}$ ▷ Recompute the noisy LWE commitment.
5: $\mathbf{w}' := \lfloor \mathbf{y} \rceil_{\nu_w} + \mathbf{h}$ ▷ Adjust the MLWE commitment with hint.
6: $c'_{\mathsf{poly}} := \mathsf{G}(\mathbf{w}', \mu)$ ▷ Recompute c_{poly}. Note that $\mathsf{G} = \mathsf{ChalPoly} \circ \mathsf{ChalHash}$.
7: **if** $c_{\mathsf{poly}} \neq c'_{\mathsf{poly}}$ **return** FAIL ▷ Check commitment.
8: **return** OK

Figure 5: Simplified KeyGen, Sign, Verify algorithms used in our security proof.

Here, we prove that the Raccoon signature scheme is existentially unforgeable under chosen message attacks (EUF-CMA), whose formal definition is deferred to Definition 7.

Simplifications. For clarity, we made some changes to the Raccoon's KeyGen, Sign and Verify algorithms in order to remove the instructions specific to help implementations such as serialization. Moreover, as we are not considering probing adversaries, we also directly act the decoded

---

$\mathsf{SampleSU}(L, u, d, \mathsf{rep}) \to \mathcal{R}_q^L$

1: $[\![\mathbf{v}]\!] \leftarrow L \times \mathsf{ZeroEncoding}(d)$
2: $[\![\mathbf{v}]\!] \leftarrow \mathsf{AddRepNoise}([\![\mathbf{v}]\!], u, \mathsf{rep})$
3: $\mathbf{v} := \mathsf{Decode}([\![\mathbf{v}]\!])$
4: **return** $\mathbf{v}$

Figure 6: Algorithm $\mathsf{SampleSU}(L, u, d, \mathsf{rep})$ samples from the distribution $\mathrm{SU}(u, d \cdot \mathsf{rep})^{nL}$ and views the sample as an element over $\mathcal{R}_q^L$.

version of our variables as this change preserves the semantic. The simplified algorithms are provided in Figure 5. We discuss these modifications below:

- We compose the hash functions $\mathsf{ChalHash}$ and $\mathsf{ChalPoly}$ into a single hash function $\mathsf{G} := \mathsf{ChalPoly} \circ \mathsf{ChalHash}$, modeled as a random oracle during in the security proof.
- This modification in turn implies a slight alteration of the signature sig, which becomes $(c_{\mathsf{poly}}, \mathbf{h}, \mathbf{z})$ instead of $(c_{\mathsf{hash}}, \mathbf{h}, \mathbf{z})$, which has been set this way for signature compression purpose.
- Finally, it leads to a modified Verify algorithm to take these changes into account. Moreover, we do not check the L_∞-norm of $(\mathbf{z}, 2^{\nu_{\mathrm{w}}} \cdot \mathbf{h})$, as we only need its L_2-norm in our proof, and thus remove this check in our simplified Verify algorithm. This change does not impact the security of our scheme as the signatures we deem valid are a superset of the actual valid signatures, and we prove that even with this relaxation, it is hard for any adversary to provide a forgery for this scheme.

Notations for smooth Rényi divergence. Let use define $\mathsf{SampleSU}(L, u, d, \mathsf{rep})$ as in Figure 6. It can be checked that each coefficient of $\mathbf{v} \in \mathcal{R}_q^L$ output by $\mathsf{SampleSU}$ is a sum of $d \cdot \mathsf{rep}$ uniform samples over $\{-2^{u-1}, \ldots, 2^{u-1} - 1\}$, that is, $\mathrm{SU}(u, d \cdot \mathsf{rep})$. For any $c \in C$, $\mathbf{s} \in \mathcal{R}_q^\ell$, and $\mathbf{e} \in \mathcal{R}_q^k$, we define the following two probability distributions:

$$\mathcal{P} := \mathrm{SU}(u_{\mathrm{w}}, d \cdot \mathsf{rep})^{n(\ell+k)},$$
$$Q(\mathsf{center}) := \mathsf{center} + \mathcal{P},$$

where $\mathsf{center} := c \cdot \begin{bmatrix} \mathbf{s} \\ \mathbf{e} \end{bmatrix} \in \mathcal{R}_q^{\ell+k}$. We provide some smooth Rényi divergence related terms relating to the two distributions $\mathcal{P}$ and Q. Below, we set $T = d \cdot \mathsf{rep}$.

For any $\alpha = \omega_{\mathsf{asymp}}(1)$ and $\epsilon_{\textsc{Tail}}(\mathsf{center}) = \frac{1}{\sqrt{2\pi T}} \left(\frac{\alpha\, e\, \|\mathsf{center}\|_2}{2^{u_{\mathrm{w}}} \cdot T} \right)^T$ (see Conjecture 1), we define $\epsilon_{\textsc{Tail}}$ to be any value satisfying

$$\Pr_{(\mathbf{s},\mathbf{e}) \leftarrow \mathrm{SU}(u_{\mathrm{w}}, T)^{n\ell} \times \mathrm{SU}(u_{\mathrm{w}}, T)^{nk}} \left[\epsilon_{\textsc{Tail}} \geq \max_{c \in C} \epsilon_{\textsc{Tail}}(\mathsf{center}) \right] \geq 1 - \mathsf{negl}(\kappa). \tag{15}$$

With an abuse of notation, we define $R_\alpha^{\epsilon_{\textsc{Tail}}}(\mathcal{P}; Q)$ as any value satisfying

$$\Pr_{(\mathbf{s},\mathbf{e}) \leftarrow \mathrm{SU}(u_{\mathrm{w}}, T)^{n\ell} \times \mathrm{SU}(u_{\mathrm{w}}, T)^{nk}} \left[R_\alpha^{\epsilon_{\textsc{Tail}}}(\mathcal{P}; Q) \geq \max_{c \in C} R_\alpha^{\epsilon_{\textsc{Tail}}(\mathsf{center})}(\mathcal{P}; Q(\mathsf{center})) \right] \geq 1 - \mathsf{negl}(\kappa). \tag{16}$$

For efficiency and better parameters, we set ϵ_{TAIL} and $R_\alpha^{\epsilon_{\text{TAIL}}}(\mathcal{P};\mathcal{Q})$ to be the smallest values satisfying the above inequality. A candidate asymptotic parameter selection for these values (and all other parameters) is provided in Appendix D.2.

Theorem 1. *The Raccoon signature scheme described in Section 2 is* EUF-CMA *secure under the* $\mathsf{MLWE}_{q,\ell,k,\mathrm{SU}(u_t,d\cdot\mathsf{rep})}$ *and* $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,C,\nu_w,\beta}$ *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} and $R_\alpha^{\epsilon_{TAIL}}(\mathcal{P};\mathcal{Q})$ satisfying Eqs. (15) and (16), there exists adversaries $\mathcal{B}$ and $\mathcal{B}'$ against the* $\mathsf{MLWE}_{q,\ell,k,\mathrm{SU}(u_t,d\cdot\mathsf{rep})}$ *and* $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,C,\nu_w,\beta}$ *problems such that*

$$\begin{aligned}\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}} \leq\ & 2^{-\kappa}\cdot Q_h\cdot(1+2^{-\kappa+1}\cdot Q_s)+Q_s\cdot\epsilon_{TAIL}\\ &+\left(\mathsf{Adv}_{\mathcal{B}}^{\mathsf{MLWE}}+\mathsf{Adv}_{\mathcal{B}'}^{\mathsf{SelfTargetMSIS}}+Q_s\cdot\epsilon_{TAIL}\right)^{\frac{\alpha-1}{\alpha}}\cdot\left(R_\alpha^{\epsilon_{TAIL}}(\mathcal{P};\mathcal{Q})\right)^{Q_s},\end{aligned} \tag{17}$$

where $\mathsf{Time}(\mathcal{A})\approx\mathsf{Time}(\mathcal{B})\approx\mathsf{Time}(\mathcal{B}')$ *and we can assume* $\mathsf{Time}(\mathcal{A}) > O(Q_h+Q_s)$. *Concretely, plugging in our candidate asymptotic parameters in Appendix D.2, we conclude* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}$ *is bounded by* $\mathsf{negl}(\kappa)$.

The full proof of the theorem is provided in Appendix D.3. We also show that by further assuming the MSIS problem for a similar set of parameters, we can establish sEUF-CMA security. The details are provided in Appendix D.4. Below, we provide a proof overview of Theorem 1.

Proof overview. The security proof follows a sequence of hybrids (Hybrid_0 to Hybrid_7) going from the EUF-CMA game to a final game that enjoys a reduction to the SelfTargetMSIS problem.

The first two hybrids (Hybrid_1 and Hybrid_2) manipulate the XOF ExpandA and hash function H, both modeled as random oracles. In Hybrid_1, we first sample a random matrix $\mathbf{A}\in\mathcal{R}_q^{k\times\ell}$ and program the output of ExpandA(seed) to be the matrix $\mathbf{A}$ for a random seed $\leftarrow\{0,1\}^\kappa$. This is unnoticeable from an adversary since seed is sampled uniformly. Looking ahead, this allows the reduction to embed a SelfTargetMSIS problem in $\mathbf{A}$. Then in Hybrid_2, we add a step to abort in the unlikely event that the adversary outputs a forgery on message msg^* such that $\mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg}^*)=\mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg})$ for some msg queried to the signing oracle. In particular, this takes care of the event that the adversary breaks EUF-CMA security by finding a collision in H. Since H is modeled as a random oracle, this cannot happen.

Now, in the next three hybrids (Hybrid_3 to Hybrid_5), the goal is to edit the signing oracle which initially coincides with the simplified Sign algorithm from Figure 5 to lift the dependency on secret information: it boils down to proving the honest-verifier zero-knowledge of the argument underlying the signature scheme. In Hybrid_3, we replace the way the challenge c_{poly} is generated by sampling it at random from its set C and then program the random oracle G to answer consistently.[5] Once that is done, in Hybrid_4, we replace the way the commitment $\mathbf{w}$ is computed: instead of being sampled as an MLWE sample, we introduce a new variable $\mathbf{z}':=c_{\mathsf{poly}}\cdot\mathbf{e}+\mathbf{e}'$, which essentially corresponds to the difference between $\mathbf{A}\cdot\mathbf{z}-c_{\mathsf{poly}}\cdot\bar{\mathbf{t}}$ and the commitment $\mathbf{w}$ before rounding. This rewriting effectively allows us to first sample $\mathbf{z}=c_{\mathsf{poly}}\cdot\mathbf{s}+\mathbf{r}'$ and $\mathbf{z}'$ and then set the commitment $\mathbf{w}=\mathbf{A}\cdot\mathbf{z}-c_{\mathsf{poly}}\cdot\bar{\mathbf{t}}+\mathbf{z}'$; recall that in the real game, $\mathbf{w}=\mathbf{A}\cdot\mathbf{r}+\mathbf{e}'$ is sampled and then $\mathbf{z}$ is computed. Which leads us to Hybrid_5, where $(\mathbf{z},\mathbf{z}')$ is now sampled from $\mathrm{SU}(u_w,T)^{n\ell}\times\mathrm{SU}(u_w,T)^{nk}$. Notice the distribution of $(\mathbf{z},\mathbf{z}')$ in Hybrid_4 and Hybrid_5 follow $\mathcal{Q}(\mathsf{center})$ and $\mathcal{P}$ defined above, respectively. To argue this change, we measure the difference between the distributions $\mathcal{Q}(\mathsf{center})$ and $\mathcal{P}$ using the smooth Rényi divergence presented in Section 4.1.2. It is worth noting that we cannot rely on the usual Rényi divergence as the support

[5] As the Raccoon signature scheme does not use rejection sampling, our proof of reprogramming the random oracle G is not affected by the bug in the proof of Dilithium [DKL+18b] pointed out in [DFPS23, BBD+23].

of $\mathcal{Q}(\mathsf{center})$ and $\mathcal{P}$ are different; this is in contrast to prior works where $(\mathbf{z}, \mathbf{z}')$ follows a discrete Gaussian distribution with shifted centers, having the same support regardless of the center. Concretely, we set the parameters so that the size ϵ_{TAIL} of the tails of $\mathcal{Q}(\mathsf{center})$ and $\mathcal{P}$, for which we cannot apply the usual Rényi divergence argument, is negligible. We then rely on Lemma 1 and Conjecture 1 to show that for any $\mathcal{A}$, we can set α so that $R_\alpha^{\epsilon_{\text{TAIL}}}(\mathcal{P}; \mathcal{Q})^{Q_s}$ is polynomially bounded (cf. Eq. (16)).

Then, in Hybrid_6, realizing that the rest of the algorithm does not depend on the secret $\mathbf{s}$ and $\mathbf{e}$, we can swap $\bar{\mathbf{t}}$ to a uniform vector over $\mathcal{R}_q^k$, which is possible thanks to the hardness of the MLWE assumption. We arrived to a hybrid where we can embed an SelfTargetMSIS instance $[-\bar{\mathbf{t}} \mid \mathbf{A}] \leftarrow \mathcal{R}_q^{k\times(\ell+1)}$ into the verification key. Lastly, in Hybrid_7, we modify the description of the random oracle G provided to the EUF-CMA adversary, by embedding the random oracle G' provided by the SelfTargetMSIS problem. At this point, any EUF-CMA adversary against Hybrid_7 can be used to derive a valid solution for the SelfTargetMSIS problem.

Collecting all the advantage bounds, we obtain Theorem 1. The bound is used as a base reference in Section 4.3 to set our concrete parameters. □

4.1.4 An Alternative Proof Aiming For 2^κ-Security

In Theorem 1, we aimed at the standard notion of EUF-CMA security where the advantage divided by the running time of the adversary (i.e., the inverse of the working factor $\mathsf{Time}(\mathcal{A})/\mathsf{Adv}_{\mathcal{A}}$) is upper bounded by some negligible function $f(\kappa)$. In practice, however, it is more informative for aiming at 2^κ-EUF-CMA security, meaning that $\mathsf{Adv}_{\mathcal{A}}/\mathsf{Time}(\mathcal{A})$ can be upper bounded by $C \cdot 2^{-\kappa}$ for some small constant $C > 0$. This helps assess the *concrete* hardness of our Raccoon signature, as otherwise, we would have to consider how large the security parameter κ must be before $f(\kappa)$ starts to behave well.

In Theorem 1, we can set the parameters so that

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}/\mathsf{Time}(\mathcal{A}) \lessapprox 2^{-\kappa} + \epsilon_{\text{TAIL}},$$

assuming the 2^κ-hardness of the underlying MLWE and SelfTargetMSIS problems. For all but one set of our concrete parameters in Section 2.1, we have $\epsilon_{\text{TAIL}} \lessapprox 2^{-\kappa}$. However, when $(d, \mathsf{rep}) = (1, 2)$, we have $\epsilon_{\text{TAIL}} = 2^{-64}$ for $\kappa \in \{128, 192, 256\}$. Therefore, in this specific case, Theorem 1 does not guarantee 2^κ-EUF-CMA security (while it still guarantees the standard notion of asymptotic unforgeable security). To this end, we briefly provide an alternative proof of Theorem 1 so that the Raccoon signature provides 2^κ-EUF-CMA security even if ϵ_{TAIL} is some negligible function larger than $2^{-\kappa}$.

How does ϵ_{TAIL} affect the concrete security? Recall the bound on ϵ_{TAIL} appears when invoking the smooth Rényi divergence to move from Hybrid_4 to Hybrid_5 in the proof of Theorem 1, where $(\mathbf{z}, \mathbf{z}')$ is set as $(c_{\mathsf{poly}} \cdot \mathbf{s} + \mathbf{r}, c_{\mathsf{poly}} \cdot \mathbf{e} + \mathbf{e}')$ in Hybrid_4 and as $(\mathbf{r}, \mathbf{e}')$ in Hybrid_5. To put it in context, if $\epsilon_{\text{TAIL}} \lessapprox 2^{-\kappa}$, the statistical distance of the distributions of $(\mathbf{z}, \mathbf{z}')$ in the two hybrids are roughly $2^{-\kappa}$-close, i.e., $(\mathbf{z}, \mathbf{z}')$ will not reside in the TAIL region of the distribution with all but probability $2^{-\kappa}$ (see Figure 4). However, when $\epsilon_{\text{TAIL}} = \mathsf{negl}(\kappa) \geq 2^{-\kappa}$, there is some chance (still negligible) that some coefficient of $(\mathbf{z}, \mathbf{z}')$ may reside in the tail of the distribution, in which case, Hybrid_4 and Hybrid_5 become distinguishable. We briefly explain an alternative hybrid sequence that does not rely on Hybrid_5 in order to establish 2^κ-unforgeable security of Theorem 1. At a high level, we argue that leaking only very few samples in TAIL does not harm security in any noticeable manner.

Bounding the number of samples in TAIL. Recall the set of responses $\{(\mathbf{z}^{(\mathsf{q_s})}, \mathbf{z}'^{(\mathsf{q_s})})\}_{\mathsf{q_s}\in[Q_s]} \in (\mathcal{R}_q^\ell \times \mathcal{R}_q^k)^{Q_s}$. We prepare a list BAD whose entry is initialized with 0 and update $\mathsf{BAD}[\mathsf{q_s}, i, j] = 1$ for $(\mathsf{q_s}, i, j) \in [Q_s] \times [\ell] \times [n]$ when the j-th coefficient of $z_i^{(\mathsf{q_s})}$ falls in TAIL, where $z_i^{(\mathsf{q_s})}$ is the i-th element of $\mathbf{z}^{(\mathsf{q_s})}$. Similarly we define BAD' for $\{\mathbf{z}'^{(\mathsf{q_s})}\}_{\mathsf{q_s}\in[Q_s]}$. We first count the number of locations in BAD with a 1 standing.

Let $N = 2^{u_w}$, let $p_0(c) = 2\epsilon_{\text{TAIL}}(c)$ be the probability that a sample $z = c + r$ falls in TAIL for $r \leftarrow \mathrm{SU}(u, T)$. According to Lemma 7, for $\tau > |c|(\alpha + 2)$ we have $\epsilon_{\text{TAIL}}(c) \leq \epsilon_{\text{TAIL},\infty} = \frac{(\tau+T)^T}{T!\,N^T}$, therefore $\tau \sim N\,(\epsilon_{\text{TAIL},\infty} \cdot T!)^{1/T}$, and since the infinity norm is smaller than the Euclidean norm $\epsilon_{\text{TAIL},\infty} \leq \epsilon_{\text{TAIL}} = \mathsf{negl}(\kappa)$. In total, there will be $M = n\,(\ell + k)\,Q_s$ samples (i.e., $M = |\mathsf{BAD}| + |\mathsf{BAD}'|$), with a probability less than $p_0 = \max_{c\in C} p_0(c)$ of each falling in TAIL. Thus, the number of samples in TAIL follows a Bernoulli distribution of parameters (p_0, M). Let X be the random variable of the number of coefficients falling in TAIL. The additive Chernoff bound tells us that

$$\Pr[X > M\,p_0 + \delta] < \exp\left(-\frac{\delta^2}{2\,M\,p_0}\right)$$

Since we want the number of BAD events to be constant with overwhelming probability, as long as $\epsilon_{\text{TAIL,inf}} \leq \frac{1}{M\kappa}$, which always holds true when $\epsilon_{\text{TAIL,inf}} = \mathsf{negl}(\kappa)$, and $\delta = \sqrt{2\,\log(2)\,p_0\,M\,\kappa} = 2\sqrt{\log(2)\epsilon_{\text{TAIL,inf}}\,M\,\kappa}$, we get:

$$\Pr\left[X > 2\left(1 + \sqrt{\ln 2}\right)\right] < 2^{-\kappa} \tag{18}$$

We have established that the number of 1s in BAD and BAD' are bounded by some constant ν with all but probability $2^{-\kappa}$ when $\epsilon_{\text{TAIL,inf}}$ is negligible.

We are now ready to define an alternative to Hybrid_5, denoted as $\mathsf{Hybrid}_{5'}$. This is defined exactly as Hybrid_5 for all the samples where BAD and BAD' are 0. The difference is that for samples where BAD and BAD' are 1, $\mathsf{Hybrid}_{5'}$ remains identical to Hybrid_4, i.e., we leak the samples that fall in TAIL. We can now change the distribution of $\mathbf{z}, \mathbf{z}'$ at all indices for which BAD and BAD' take the value 0. Additionally since we condition this change on the event BAD (or BAD') not occurring (which implies that we are not in the TAIL) we can use the (standard) Rényi divergence instead of the smooth Rényi divergence and avoid the additive ϵ_{TAIL} term. i.e. we have:

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_4} &\leq \left(\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_{5'}}\right)^{\frac{\alpha-1}{\alpha}} \cdot (R_\alpha(\mathcal{P}'; Q'))^{Q_s} \\
&\leq \left(\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_{5'}}\right)^{\frac{\alpha-1}{\alpha}} \cdot \exp\left(\frac{Q_s \cdot C_{\text{RÉNYI}} \cdot \alpha \cdot \|(c_{\mathsf{poly}} \cdot \mathbf{s}, c_{\mathsf{poly}} \cdot \mathbf{e})\|_2^2}{T \cdot N^2}\right)
\end{aligned}$$

Where $\mathcal{P}'$ (resp. Q') is the distribution obtained by cutting the tails of $\mathcal{P}$. The last inequality comes from the fact that we set $R_\alpha^\epsilon(\mathcal{P}; Q)$, by using the Rényi divergence of $\mathcal{P}', Q'$ with identical tails which is equivalent to cutting both tails. If we set $N = \omega_{\mathsf{asymp}}\left(\sqrt{\frac{Q_s \cdot \alpha}{T}}\|(c_{\mathsf{poly}} \cdot \mathbf{s}, c_{\mathsf{poly}} \cdot \mathbf{e})\|_2\right)$, then the exponential term in the above equation will be constant. It remains to discuss the two following properties to establish 2^κ-EUF-CMA security of the Raccoon signature:

Leak 1. The position of 1s in BAD and BAD' do not leak information on the secret $(\mathbf{s}, \mathbf{e})$.

Leak 2. The samples in the TAIL do not leak information on the secret $(\mathbf{s}, \mathbf{e})$.

Controlling Leak 1 via Rényi divergence. We rely on the standard Rényi divergence to argue that the position of 1s in BAD and BAD' is independent of the secret, i.e., center of the distribution. A keen reader may have noticed that we already used this argument above when counting the number of samples falling in $\textsc{Tail}$. Namely, we show that the distribution of 1 in BAD and BAD' follows a Bernoulli distribution of parameters (p_0, M). Using Lemma 7 we obtain a bound on the Rényi divergence between the distribution of the events $(\mathsf{BAD}, \mathsf{BAD}')$ in $\mathsf{Hybrid}_{5'}$ and the event of $\mathsf{Hybrid}_{6'}$ where $(\mathsf{BAD}, \mathsf{BAD}')$ is defined as a vector of iid. Bernoulli's variables independent of $\mathbf{s}, \mathbf{e}$, and c.

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_{5'}} \leq \left(\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_{6'}}\right)^{\frac{\alpha-1}{\alpha}} \cdot \exp\left(\frac{M\,T^4\,\epsilon_{\textsc{Tail},\infty}}{4\,\alpha^3}\left(1 + O((T/\alpha)^2)\right)\right),$$

When setting parameters we will use $\alpha = \omega_{\mathsf{asymp}}(\sqrt{\kappa})$ and $T = \omega_{\mathsf{asymp}}(1)$, hence if we set $N \geq M^{1/T}\left(\|(c_{\mathsf{poly}} \cdot \mathbf{s}, c_{\mathsf{poly}} \cdot \mathbf{e})\|_\infty(\alpha + 2) + T\right)$ then we have:

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_{5'}} \leq \left(\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_{6'}}\right)^{\frac{\alpha-1}{\alpha}} \cdot \exp\left(O\left(\frac{1}{\kappa^{3/2}}\right)\right).$$

Controlling Leak 2 via Extended MLWE. We rely on the 2^κ-hardness of the *extended* MLWE assumption ($\mathsf{ExtMLWE}$) to argue that we can leak a constant number of samples falling in the $\textsc{Tail}$. $\mathsf{ExtMLWE}$ roughly states that MLWE remains difficult even if some *hints* of the secret and noise is revealed to the adversary. We consider a variant of $\mathsf{ExtMLWE}$ where each hint is a coefficient of an inner product of the secret and noise vector. Formally, $\mathsf{ExtMLWE}$ is defined as follows.

Definition 5 ($\mathsf{ExtMLWE}$). *Let ℓ, k, q, η be integers, and $\mathcal{D}, \mathcal{F}$ be probability distributions over $\mathcal{R}_q$ and $\mathbb{Z}_q^{\eta\times(\ell+k)\cdot n}$ where recall $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$. The advantage of an adversary $\mathcal{A}$ against the* Extended Module Learning with Errors $\mathsf{ExtMLWE}_{q,\ell,k,\mathcal{D},\mathcal{F}}$ *problem is defined as:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{ExtMLWE}}(\kappa) = \left|\Pr\left[1 \leftarrow \mathcal{A}\left(\mathbf{A}, \mathbf{A}\cdot\mathbf{s} + \mathbf{e}, \mathbf{M}, \mathbf{M}\cdot \mathrm{coeff}\left(\begin{bmatrix}\mathbf{s}\\ \mathbf{e}\end{bmatrix}\right)\right)\right] - \Pr\left[1 \leftarrow \mathcal{A}\left(\mathbf{A}, \mathbf{b}, \mathbf{M}, \mathbf{M}\cdot \mathrm{coeff}\left(\begin{bmatrix}\mathbf{s}\\ \mathbf{e}\end{bmatrix}\right)\right)\right]\right|,$$

where $(\mathbf{A}, \mathbf{b}, \mathbf{s}, \mathbf{e}, \mathbf{M}) \leftarrow \mathcal{R}_q^{k\times\ell} \times \mathcal{R}_q^k \times \mathcal{D}^\ell \times \mathcal{D}^k \times \mathcal{F}$. Here, $\mathrm{coeff} : \mathcal{R}_q \to \mathbb{Z}_q^n$ denotes the coefficient embedding, and is naturally defined for vectors over $\mathcal{R}_q$. The $\mathsf{ExtMLWE}_{q,\ell,k,\mathcal{D},\mathcal{F}}$ assumption states that any efficient adversary $\mathcal{A}$ has negligible advantage.

For our argument, we rely on $\mathsf{ExtMLWE}$ with the following parameter selection:

- $\eta = O(1)$, i.e., the number of samples falling in $\textsc{Tail}$.
- $\mathcal{F}$ is a distribution such that $\left\{\mathbf{M}\cdot\mathrm{coeff}\left(\begin{bmatrix}\mathbf{s}\\ \mathbf{e}\end{bmatrix}\right) \mid \mathbf{M} \leftarrow \mathcal{F}\right\}$ induces the same distribution of the set of *centers* for the samples falling in $\textsc{Tail}$.

In more detail, when $\mathsf{BAD}[\mathsf{q_s}, i, j] = 1$, we leak the j-th coefficient of $z_i^{\mathsf{q_s}} = (c_{\mathsf{poly}} \cdot s_i + r_i)$ to the adversary, where the center is the j-th coefficient of $c_{\mathsf{poly}} \cdot s_i$ and (s_i, r_i) denote the i-th entry of $(\mathbf{s}, \mathbf{r})$. $\mathsf{ExtMLWE}$ says that the scheme remains secure even if we leak $\mathrm{coeff}(c_{\mathsf{poly}} \cdot s_i)_j = \mathbf{m} \cdot \mathrm{coeff}(s_i)$, where $\mathbf{m}$ denotes the j-th row of $c_{\mathsf{poly}} \in \mathcal{R}_q$ when represented as an anti-circulant matrix $\mathbb{Z}_q^{n\times n}$. Since the inner product is always defined with a challenge polynomial, any $\mathbf{M} \in \mathcal{F}$ satisfies $\|\mathbf{M}\|_\infty = 1$ and each row of $\mathbf{M}$ consists of exactly ω non-zero entries.

As a minimal background, the non-structured variant of $\mathsf{ExtMLWE}$ (i.e., extended LWE) was originally introduced by [OPW11, AP12] in the context of (bi)deniable encryption and key-dependent message security, and later used by [BLP+13] in the context of establishing classical hardness of the LWE problem. Under specific parameters, ExtLWE is known to be as hard as LWE. Since then, different variants have been considered [ALS16, AA16, LNS21, BJRW23]. We discuss the concrete hardness of our variant of ExtMLWE in Section 4.3.7. We also discuss its asymptotic hardness in Appendix C.2.

4.2 Security against Probing Adversaries

In this section, we provide an informal argument for the security of the Raccoon signature scheme in the presence of t-probing adversaries. Previous works on masking lattice-based signatures proceed by decomposing the main algorithms (key generation and signing) in subroutines, masking each of these subroutines, and arguing the security of the global scheme via composition frameworks. For example, [BBE+18, GR19, BBE+19] rely on the (Strong)-Non-Interference (NI/SNI) framework [BBD+16], whereas [ABC+22] rely on the Probe Isolating Non-Interference (PINI) framework [CS20].

Contrary to existing masked lattice-based signatures, the design of Raccoon makes a deliberate choice to allow signatures to leak a limited amount of information in a way that is well-understood and captured with a Rényi divergence analysis, similarly to what Falcon does [PFH+22, §2.5.2 and §2.6].

In more detail, all subroutines of Raccoon can be proved composable in the SNI model, except one: AddRepNoise (Algorithm 8). While this gadget performs operations share by share, the underlying distributions are not uniform. Short noise values are added together and the knowledge of any intermediate short value biases the *a posteriori* distribution of the final noise. Hence, one cannot prove the Non-Interference of such a gadget without extra computational assumption on the attacker.

We put forth a preliminary argument regarding the incorporation of AddRepNoise into prevailing masking composition frameworks, which presents certain challenges. However, these do not arise from inherent limitations of the Raccoon scheme, and rather, it seems possible to extend the current composition frameworks to capture "leaky" algorithms such as AddRepNoise and ensure the masking security with a computational advantage. This interesting problem will be fully dealt with in the near future.

4.2.1 Impact of Probing on AddRepNoise.

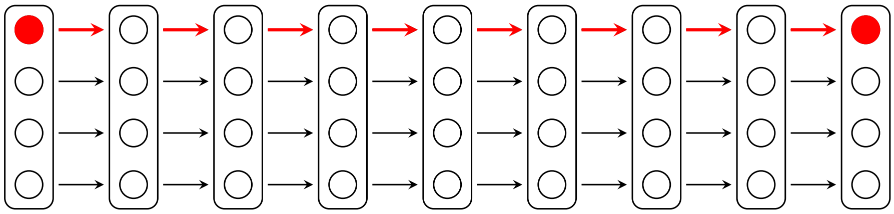
When considering unmasked coefficients, AddRepNoise is functionally equivalent to performing $a \leftarrow a + \mathrm{SU}(u, T)$ for each coefficient a, for $T = d \cdot \mathsf{rep}$. The internal use of Refresh operations does not affect this behavior but is meant to offer some resilience to t-probing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

Without Refresh, a viable strategy would be to probe individual shares of $[\![a]\!]$ at the start and at the end of AddRepNoise, allowing to learn the sum b of $\mathsf{rep} \cdot t/2$ small uniform errors. This is illustrated in Figure 7. The conditional distribution of the additive noise (conditioned on the t probed values) is now $b + \mathrm{SU}(u, T - t \cdot \mathsf{rep}/2)$.

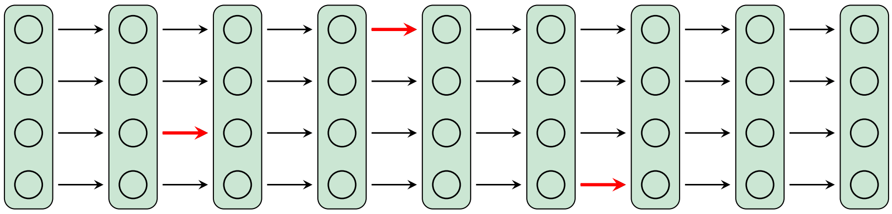
With Refresh, the previous strategy is not possible anymore but the t-probing adversary can still probe individual errors, which in the end gives out no more than the sum b_{PROBE} of t small uniform errors. This is also illustrated in Figure 7. The conditional distribution of the additive noise (conditioned on the t probed values) is now $b_{\text{PROBE}} + \mathrm{SU}(u, T - t)$, where the adversary learns b_{PROBE} but knows nothing about the realization of $\mathrm{SU}(u, T - t)$.

4.2.2 Probing Security of Key Generation

In a nutshell, key generation (KeyGen, Algorithm 1) generates an MLWE sample $(\mathbf{A}, \mathbf{t})$ as follows:



(a) Without refresh: a t-probing adversary can learn the sum of $\lfloor t/2 \rfloor \cdot \mathsf{rep}$ small uniform noises by probing the same indices (●) at the start and end of the algorithm.



(b) With refresh: a t-probing adversary can learn the sum of at most t small uniform noises.

Figure 7: Illustration of (a) an insecure algorithm for adding noise and (b) our probing-resilient algorithm AddRepNoise. Parameters are $d = 4, \mathsf{rep} = 8, t = 3$.
• Each circle represent one share of a d-sharing.
• A d-sharing is indicated in green if it is refreshed, otherwise it is in white.
• Each red arrow represent the addition of small uniform noise to the corresponding share.
• In (a) and (b), a probing adversary learns the sum of the additive noise involved in red arrows.

1. $\mathbf{t}$ is first computed in masked form: $[\![\mathbf{t}]\!] \leftarrow \mathbf{A} \cdot [\![\mathbf{s}]\!] + [\![\mathbf{e}]\!]$, where $[\![\mathbf{s}]\!], [\![\mathbf{t}]\!]$ are generated during the call to AddRepNoise in Algorithm 1 (Line 4 respectively).

2. $\mathbf{t}$ is then unmasked (Algorithm 1, Line 7) and rounded. We ignore this rounding in our analysis and assume that the adversary has access to $\mathbf{t}$ before it is rounded.

For the reasons explained above, while all the other implied gadgets are composable, a t-probing adversary can still infer some information about the output of AddRepNoise. In contexts such as trapdoor sampling, such a bias could be exploited to mount key-recovery attacks [GMRR22, ZLYW23, Pre23]. However, we argue that it has a minimal impact on the overall security of the key generation procedure.

If we ignore the effect of rounding, thus providing more information to the adversary, then the verification key is an MLWE sample $(\mathbf{A}, \mathbf{t})$ where $\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$, where each coefficient of $(\mathbf{s}, \mathbf{e})$ is sampled according to a noise distribution of variance $\sigma_{\mathbf{t}}^2 = \frac{T(2^{2u_t}-1)}{12}$, see Eq. (6), and $T = d \cdot \mathsf{rep}$. This enables a security analysis based on the MLWE assumption with standard deviation $\sigma_{\mathbf{t}}$, which is well understood.

As discussed above, a t-probing[6]adversary can slightly bias the noise distribution. More precisely, we can rewrite the verification key vector $\mathbf{t}$ as:

$$\mathbf{t} = \mathbf{A} \cdot (\mathbf{s}_{\text{PROBE}} + \mathbf{s}^*) + (\mathbf{e}_{\text{PROBE}} + \mathbf{e}^*) = \underbrace{(\mathbf{A} \cdot \mathbf{s}_{\text{PROBE}} + \mathbf{e}_{\text{PROBE}})}_{\mathbf{t}_{\text{PROBE}}} + \underbrace{(\mathbf{A} \cdot \mathbf{s}^* + \mathbf{e}^*)}_{\mathbf{t}^*} \tag{19}$$

In Eq. (19), $\mathbf{s}_{\text{PROBE}}$ and $\mathbf{e}_{\text{PROBE}}$ correspond to additive errors probed during AddRepNoise. We very conservatively assume that the adversary can probe up to t additive errors *per integer coefficient of*

[6]Recall that the italic variable t corresponds to the masking order $t = d - 1$ and the vector $\mathbf{t}$ corresponds to the public key.

$\mathbf{t}$ (i.e. obtain their exact values). Thus, the remaining secrets $(\mathbf{s}^*, \mathbf{e}^*)$ are such that each coefficient of $(\mathbf{s}^*, \mathbf{e}^*)$ is sampled according to a noise distribution of variance $\sigma_{\mathbf{t}^*}^2 = \frac{(T-t)\,(2^{2u_t}-1)}{12}$. Since $t \leq d-1$ by hypothesis:

$$\left(\frac{\sigma_{\mathbf{t}^*}}{\sigma_{\mathbf{t}}}\right)^2 = \frac{T-t}{T} > \frac{\mathsf{rep}-1}{\mathsf{rep}}. \tag{20}$$

Since $\mathsf{rep} \geq 2$ in Raccoon, Eq. (20) tells us that a t-probing adversary cannot reduce the standard deviation of the MLWE noise distribution by more than a factor $\sqrt{\frac{\mathsf{rep}-1}{\mathsf{rep}}} \leq \sqrt{2}$. We incorporate this security loss when studying the concrete pseudo-randomness of the verification key, more precisely the hardness of MLWE in Section 4.3.4.

4.2.3 Probing Security of Signing

The structure of the signing procedure (Sign, Algorithm 2) is as follows:

(S1) An MLWE commitment $\mathbf{w}$ is computed in masked form, then unmasked, in a way that is identical to the computation of $\mathbf{t}$ during key generation, only with different parameters;

(S2) A challenge c_{poly} is computed in unmasked form; a response $\mathbf{z}$ is computed in masked form, then unmasked; and the hint $\mathbf{h}$ is computed from publicly available values.

Non-composability. For the reasons discussed in Section 4.2.2, while the way we implement Item (S1) makes it extremely efficient, it also precludes it from being a composable building block in existing frameworks, as AddRepNoise leaks some information about the ephemeral randomness contained in $\mathbf{w}$. While Section 4.2.2 provides security arguments for key generation in the presence of this leakage, in the signing procedure we additionally need to study how the leakage in Item (S1) trickles down in Item (S2).

Impact of probing on Sign. The commitment $\mathbf{w}$ is of the form $\mathbf{w} = \mathbf{A}\cdot\mathbf{r}+\mathbf{e}'$, where each coefficient of $(\mathbf{r}, \mathbf{e}')$ is sampled according to a noise distribution of variance $\sigma_{\mathbf{w}}^2 = \frac{T(2^{2u_w}-1)}{12}$. Following the same reasoning as in Section 4.2.2, a t-probing adversary can learn partial information about $\mathbf{r}, \mathbf{e}'$. More precisely, if we write:

$$\mathbf{w} = \mathbf{A} \cdot \underbrace{(\mathbf{r}_{\text{PROBE}} + \mathbf{r}^*)}_{\mathbf{r}} + \underbrace{(\mathbf{e}'_{\text{PROBE}} + \mathbf{e}'^*)}_{\mathbf{e}'}, \tag{21}$$

we assume that the adversary may learn $\mathbf{r}_{\text{PROBE}}, \mathbf{e}'_{\text{PROBE}}$. The remaining randomness $\mathbf{r}^*, \mathbf{e}'^*$ have each of their coefficients sampled according to a noise distribution of variance $\sigma_{\mathbf{w}^*}^2 = \frac{(T-t)\,(2^{2u_w}-1)}{12}$. This mainly impacts our smooth Rényi divergence argument; instead of arguing that $(\mathbf{r}, \mathbf{e}')$ and $(\mathbf{r}, \mathbf{e}') + c_{\mathsf{poly}}(\mathbf{s}, \mathbf{e})$ are close in the sense of the smooth Rényi divergence (Definition 4), we now need to argue $(\mathbf{r}^*, \mathbf{e}'^*) + c_{\mathsf{poly}}(\mathbf{s}, \mathbf{e})$ is close to $(\mathbf{r}^*, \mathbf{e}'^*)$, still in the sense of Definition 4, with only a loss $\sqrt{2}$ in the standard deviation of $(\mathbf{r}^*, \mathbf{e}'^*)$. This is reflected by our concrete security analysis in Section 4.3.6.

4.3 Concrete Security

4.3.1 Modelization and Methodology

We now turn to the *concrete* security estimation of the Raccoon signature scheme and design a methodology to provide a practical set of parameters. We follow here the standard methodology and define the bit-security as

$$\kappa = \log_2\left(\mathsf{Time}(\mathcal{A})/\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}\right) \tag{22}$$

as a translation from the advantage of an adversary into concrete security measured in bits (for computations carried in the so-called Core-SVP model, as detailed in Section 4.3.3). In a word, the log-advantage is normalized by the resources $\mathsf{Time}(\mathcal{A})$ spent by the adversary. It always holds that $\mathsf{Time}(\mathcal{A}) \geq Q_s + Q_h$. Here, the adversaries considered will be the one breaking the pseudo-randomness of the verification key $\mathbf{t}$ and the one playing the EUF-CMA game, recovering the usual and more informal notions of security against respectively *key recovery* and *forgery*. Our analysis can be seamlessly extended to cover sEUF-CMA (Theorem 2) with the same parameter sets, since the constant $2\sqrt{2}$ in Theorem 2 seems to be an artifact of the proof, and may instead be set to 1 for practical purposes.

Remark 1. As detailed in Section 4.1.4, depending on the regime of parameters, we might need a finer-grained security analysis because of the relative size of ϵ_{TAIL}. This section also takes this subtlety into account by analyzing practically the impact of leaks in the ExtMLWE problem.

4.3.2 Roadmap

Recall from Section 4.1 that the blackbox security reduction of Theorem 1 yields a tight estimate of the advantage of the adversary in the EUF-CMA security game. We reproduce this estimate in Eq. (23). We map each constitutive term of the right-hand-side of Eq. (23) to the corresponding section where its *concrete* analysis and security estimate is presented.

$$\begin{aligned}\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}} \leq\ & 2^{-\kappa} \cdot Q_h \cdot (1 + 2^{-\kappa+1} \cdot Q_s) + Q_s \cdot \epsilon_{\text{TAIL}} \\ & + \Big(\underbrace{\mathsf{Adv}_{\mathcal{B}}^{\mathsf{MLWE}}}_{\S 4.3.4} + \underbrace{\mathsf{Adv}_{\mathcal{B}'}^{\mathsf{SelfTargetMSIS}}}_{\S 4.3.5} + Q_s \cdot \underbrace{\epsilon_{\text{TAIL}}}_{\S 4.3.6}\Big)^{\overbrace{\frac{\alpha-1}{\alpha}}^{\S 4.3.6}} \cdot \underbrace{R_{\alpha}^{\epsilon_{\text{TAIL}}}(\mathcal{P};\mathcal{Q})^{Q_s}}_{\S 4.3.6},\end{aligned} \tag{23}$$

After giving some details on the model of computation used to translate complexity into practical bitsec in Section 4.3.3, we propose in Section 4.3.4 a hardness analysis of the MLWE problem and discuss its relation to probing security. This will entail the resilience against *key recovery* through breaking the pseudorandomness of $\mathbf{t}$. Next, we present in Section 4.3.5 the different building blocks required to assess the advantage against EUF-CMA. Eventually we bind every piece of the puzzle together in Section 4.3.8 and discuss the interactions of parameters, as well as how to perform the global optimization.

4.3.3 Concrete Model of Lattice Reduction, GSA and Beyond

The core-SVP hardness. To accurately assess the hardness of the underlying problems and ensure a specified level of bit-security, it is necessary to establish a model that simulates the behavior of a practical oracle for approximate Shortest Vector Problem (SVP). This modeling is crucial since our hard problems involve the identification of relatively short vectors in various lattices. To achieve this, we will employ the celebrated (self-dual) Block Korkine-Zolotarev (BKZ) algorithm. Specifically, the BKZ algorithm with a block size denoted by β necessitates a polynomial number of calls to an SVP oracle in dimension β, with a heuristically expected number of calls that is approximately linear—with some implementation tricks.

To account for potential future advancements in this reduction method, we will only consider the cost of a single call to the SVP oracle. This approach, known as *core-SVP hardness*, entails a highly conservative estimation. This cautionary measure is warranted by the possibility of cost amortization for SVP calls within BKZ, particularly when sieving is employed as the SVP oracle. Notably, sieving has become the prevailing standard for larger block sizes, as exemplified in [ADH+19].

Modelization of the output of reduced bases. For the sake of clarity in the following explanations, we adopt the "Geometric series assumption" (GSA). This assumption states that the norm of the Gram-Schmidt vectors of a reduced basis decreases with a geometric decay. Specifically, in the context of the self-dual Block Korkine-Zolotarev (DBKZ) reduction algorithm proposed by Micciancio and Walter [MW16], the GSA can be instantiated as follows. Suppose we have an output basis $(\mathbf{b}_i)_{i\in[n]}$ obtained from the DBKZ algorithm with a block size denoted as β, applied to a lattice Λ of rank n. Then, the following equation holds for the i-th Gram-Schmidt vector $\mathbf{b}_i^*$ of the basis:

$$\|\mathbf{b}_i^*\| = \delta_\beta^{d-2(i-1)} \det(\Lambda)^{\frac{1}{n}}, \quad \text{where} \quad \delta_\beta = \left(\frac{(\pi\beta)^{\frac{1}{\beta}} \cdot \beta}{2\pi e} \right)^{\frac{1}{2(\beta-1)}},$$

for $\mathbf{b}_i^*$ being the i-th Gram Schmidt vector of the basis.

In order to get a finer estimate, when computing the actual figures this analysis can be refined by using the probabilistic simulation of [DDGR20] rather than this coarser GSA-based model to determine the BKZ blocksize β for a successful attack. This helps to take into account the well-known *quadratic tail* phenomenon of reduced bases [YD17].

From lattice reduction block-size to concrete bitsec. This analysis translates into concrete bit-security estimates following the methodology of NewHope [ADPS16] (so-called "core-SVP methodology"). In this model, the bit complexity of lattice sieving (which is asymptotically the best SVP oracle) is taken as $\lfloor 0.292\beta \rfloor$ in the classical setting [BDGL16] and $\lfloor 0.257\beta \rfloor$ in the quantum setting [CL21] in blocksize β.

4.3.4 Hardness of Key Recovery

Pseudorandomness of t. For Raccoon, the MLWE assumption captures the pseudorandomness of the verification key $\mathbf{t}$ (i.e. entails the security when the adversary is only provided with $\mathbf{t}$). This problem is defined over modules and is then categorized as a *structured* problem. However up to the knowledge of the authors, there is no real improvements to the practical solving of MLWE than seeing it as an *unstructured* problem and apply the classical unstructured attacks.

On MLWE and lattice reduction. More precisely, any $\mathsf{MLWE}_{q,\ell,k,\mathcal{D}}$ instance $(\mathbf{A}, \mathbf{b})$ over the ring $\mathcal{R}_q$ of degree n for some noise distribution $\mathcal{D}$ can be viewed as an LWE instance of dimensions $n \cdot \ell$ and $n \cdot k$. Indeed, the above can be rewritten as finding $\mathrm{vec}(\mathbf{s}), \mathrm{vec}(\mathbf{e}) \in \mathbb{Z}^{n\cdot\ell} \times \mathbb{Z}^{n\cdot k}$ from the instance $(\mathrm{rot}(\mathbf{A}), \mathrm{coeff}(\mathbf{b}))$, where we recall that $\mathrm{coeff} : \mathcal{R}_q \to \mathbb{Z}_q^n$ denotes the coefficient embedding, and $\mathrm{rot}(\mathbf{A}) \in \mathbb{Z}^{n\cdot k \times n\cdot\ell}$, is obtained by replacing all entries $a \in \mathcal{R}_q$ of $\mathbf{A}$ by the $n \times n$ matrix whose f-th column is $(X^{f-1} \cdot a_{ij})$.

Given an LWE instance, there are two basic lattice-based attacks: the primal attack and the dual attack. On the one hand, the former consists in finding a short non-zero vector in the lattice $\{\mathbf{x} \in \mathbb{Z}^d : \mathbf{D} \cdot \mathbf{x} = 0 \pmod{q}\}$ where

$$\mathbf{D} = (\mathrm{rot}(\mathbf{A})_{[1:m]} \,|\, \mathbf{I}_m \,|\, \mathrm{vec}(\mathbf{t})_{[1:m]}) \in \mathbb{Z}^{m\times d}$$

is a matrix which dimensions verify $d = n \cdot \ell + m + 1$ and[7] $m \leq n \cdot k$. On the other hand, the dual attack consists in finding a short non-zero vector in the lattice $\{(\mathbf{x}, \mathbf{y}) \in \mathbb{Z}^m \times \mathbb{Z}^d : \mathbf{D}^T\mathbf{x} + \mathbf{y} = 0$

[7] This description already encompasses a folklore little optimization consisting in reducing only a sublattice instead of the whole one, in order to play with the interaction between dimension and volume.

(mod q))}, where $\mathbf{D} = (\mathrm{rot}(\mathbf{A})_{[1:m]}) \in \mathbb{Z}^{m\times d}$ for $d = n\cdot\ell$ and $m \leq n\cdot k$. Again, for each value of m, we increased the value of β until the value obtained as explained above was deemed sufficiently small. From these two basic ideas, numerous optimizations and variants have been proposed and the optimal one depends greatly on the regime and distributions.

Concrete hardness. When reinstantiating the problem in the context of the Raccoon signature scheme, we are bound to estimate the concrete hardness of $\mathsf{MLWE}_{q,\ell,k,\mathrm{SU}(u_t,T)}$, with $T = d\cdot\mathsf{rep} - t$. Remark that this already encompasses a t-probing adversary, as discussed in Section 4.2.2.

To entail the set of equations just described and find the smallest block size β allowing to run the attack, we practically rely on simulation (see [APS15]) to get finer-grained results. According to this estimator, the best known attacks are the primal uSVP attack by Alkim et al. [ADPS16] and the dual/hybrid attack by Espitau et al. [EJK20]. Eventually we can apply the *dimensions for free* optimization by Ducas [Duc18] to gain a few additional bits when using a sieve-based BKZ.

4.3.5 Hardness of Direct Forgery

The problem can be restated from Definition 3 as follows. If we note $\bar{\mathbf{A}} = \begin{bmatrix}\mathbf{A} \mid \mathbf{I}\end{bmatrix}$ and $\bar{\mathbf{z}} = \begin{bmatrix}\mathbf{z}\\ \mathbf{h}\end{bmatrix}$, then the adversary needs to find an element $\vec{cz}$ such that:

$$\left(0 < \|(\mathbf{z}, 2^{v}\mathbf{h})\|^2 < B\right) \wedge \left(G(\lfloor \mathbf{A}\cdot\mathbf{z}\rceil_{\nu_w} + h - \mathbf{t}\cdot c, \mathsf{msg}) = c\right). \tag{24}$$

Following [LDK+22, §C.3], we can assume that the best way to solve (24) is either by breaking the second preimage resistance of G or by finding a short $\bar{\mathbf{z}}$ such that

$$\bar{\mathbf{A}}\cdot\bar{\mathbf{z}} = \mathbf{w} + \mathbf{t}\cdot c + \alpha, \tag{25}$$

for $\alpha = \mathbf{A}\cdot\mathbf{z} - 2^{\nu_w}\lfloor\mathbf{A}\cdot\mathbf{z}\rceil_{\nu_w}$ being a small term (of norm bounded by $2^{\nu_w - 1}$) and where $\mathbf{w}$ is a preimage of c, i.e. $G(\mathbf{w}, \mathsf{msg}) = c$. We study both problems in separate paragraphs.

MSIS. Solving Eq. (25) amounts to solving an inhomogeneous (noisy) MSIS problem which in turns (practically) amounts to finding $\bar{\mathbf{z}}$ at a bounded distance from the point $\bar{\mathbf{v}} = \mathbf{w} + \mathbf{t}\cdot c$. This BDD problem can be solved using the so-called *Nearest-Cospace* framework developed by Espitau and Kirchner in [EK20]. Under the GSA, [EK20, Theorem 3.3] states that under the condition: $\|\bar{\mathbf{z}} - \mathbf{v}\| \leq \left(\delta_\beta^{(k+\ell)n} q^{\frac{kn}{k+\ell}}\right)$, the decoding can be done in time $\mathrm{poly}(n)$ calls to a CVP oracle in dimension β. Once again here, we reduced to the unstructured equivalent problem by descending from the ring of integer of the base field to $\mathbb{Z}$ to mount the attack.

As mentioned in [CPS+20] a standard optimization of this attack consists only considering the lattice spanned by a subset of the vectors of the public basis and perform the decoding within this sublattice. The only interesting subset seems to consist in forgetting the $k \leq n$ first vectors. The dimension is of course reduced by x, at the cost of working with a lattice with covolume $q^{\frac{kn}{(k+\ell)n-x}}$ bigger. Henceforth the global condition of decoding becomes the (slightly more general) inequality $\|\bar{\mathbf{z}} - \mathbf{v}\| \leq \min_{x\leq n}\left(\delta_\beta^{(k+\ell)n-x} q^{\frac{kn}{(k+\ell)n-x}}\right)$ As such, we need to enforce the condition:

$$B_2 + 2^{\nu_w - 1}\sqrt{kn} \leq \min_{\ell n \leq m \leq (k+\ell)n}\left(q^{\frac{kn}{m}}\cdot\delta_\beta^m\right), \tag{26}$$

The term $2^{\nu_w - 1}\sqrt{k\,n}$ in Eq. (26) represents the slight wiggle room available to the adversary due to the rounding in the computation of $\mathbf{h}$. B_2 is computed in Section 2.6.2.

Challenge space. We need the hash function H to be second preimage resistant. To guarantee this we ensure that $|C| > 2^{\kappa}$. Considering how C is defined in Section 2.4.6 it is enough to set ω such that:

$$\binom{n}{\omega} \cdot 2^{\omega} \geq 2^{\kappa}.$$

4.3.6 Leakage of Signatures

For the purpose of this section, we use two different notations for the number of repetitions when sampling error distributions in the key generation and signing procedures. Indeed, when considering t-probing adversaries, these will influence security in different ways. We recall that coefficients of $(\mathbf{s}, \mathbf{e})$ and $(\mathbf{r}, \mathbf{e}')$ are sampled from $\mathrm{SU}(u_{\mathrm{t}}, T_{\mathrm{t}})$ and $\mathrm{SU}(u_{\mathrm{w}}, T_{\mathrm{w}})$, where $T_{\mathrm{t}} = d \cdot \mathsf{rep}$ and $T_{\mathrm{w}} = d \cdot \mathsf{rep}$ when no probing is done by the adversary.

Bounding the smooth Rényi divergence. By linearity of the expected value [drh17]:

$$\mathbb{E}[\|c_{\mathsf{poly}} \cdot (\mathbf{s}, \mathbf{e})\|^2] = \omega \cdot \mathbb{E}[\|(\mathbf{s}, \mathbf{e})\|^2] \leq \frac{\omega\, n\, (k + \ell) T_{\mathrm{t}}\, 2^{2u_{\mathrm{t}}}}{12} \tag{27}$$

Since Q_s is extremely large, we may take the heuristic $\sum_{i \in [Q_s]} \|c^{(i)}_{\mathsf{poly}} \cdot (\mathbf{s}, \mathbf{e})\|^2 \approx Q_s\, \omega\, \mathbb{E}[\|(\mathbf{s}, \mathbf{e})\|^2]$, where $c^{(i)}_{\mathsf{poly}}$ is the challenge polynomial in i-th signature. Combining it with Eq. (27) and Conjecture 1 allows us to bound the smooth Rényi divergence as:

$$R_{\alpha}^{\epsilon_{\mathrm{TAIL}}}(\mathcal{P}; \mathcal{Q})^{Q_s} \lesssim \exp\left(\frac{C_{\text{RÉNYI}} \cdot Q_s \cdot \alpha \cdot \omega\, n\, (k + \ell) T_{\mathrm{t}}\, 2^{2u_{\mathrm{t}}}}{12 \cdot T_{\mathrm{w}} \cdot 2^{2u_{\mathrm{w}}}}\right),$$

where $C_{\text{RÉNYI}} \approx 6$. A t-probing adversary can decrease T_{w} from $d \cdot \mathsf{rep}$ to $d \cdot \mathsf{rep} - t$, see Section 4.2.3. Taking this into account gives this closed form heuristic for the smooth Rényi divergence:

$$R_{\alpha}^{\epsilon_{\mathrm{TAIL}}}(\mathcal{P}; \mathcal{Q})^{Q_s} \lesssim \exp\left(Q_s \cdot \alpha \cdot \omega\, n\, (k + \ell)\, 2^{2(u_{\mathrm{t}} - u_{\mathrm{w}})}\right), \tag{28}$$

Number of queries Q_s. At this point, all terms of Eq. (23) are determined except for Q_s and α. We first determine the optimal value for α. Then we determine the maximal value for Q_s such that Eq. (23) provides a bit-security κ. Let us set $b, c > 0$ such that:

$$\begin{cases} \exp(-b) & = & \mathsf{Adv}_{\mathcal{B}}^{\mathsf{MLWE}} + \mathsf{Adv}_{\mathcal{B}'}^{\mathsf{SelfTargetMSIS}} + Q_s\, \epsilon_{\mathrm{TAIL}}, \\ \exp(\alpha\, c\, Q_s) & = & R_{\alpha}^{\epsilon_{\mathrm{TAIL}}}(\mathcal{P}; \mathcal{Q})^{Q_s}. \end{cases}$$

We take $c = \omega\, n\, (k + \ell)\, 2^{2(u_{\mathrm{t}} - u_{\mathrm{w}})}$ following Eq. (28), and b can be computed explicitly from Sections 4.3.4 and 4.3.5 and Eq. (14). Ignoring terms that are clearly negligible, and assuming the term $Q_s\, \epsilon_{\mathrm{TAIL}}$ is negligible in $\exp(-b)$, our goal in Eq. (23) is essentially to ensure that:

$$\exp\left(-b\, \frac{\alpha - 1}{\alpha}\right) \exp(\alpha\, c\, Q_s)/Q_s \leq 2^{-\kappa} \tag{29}$$

Let $f : x \mapsto \exp(-b + 2\sqrt{b\, c\, x})/x$. The left term in Eq. (29) is minimized for $\alpha = \sqrt{b/(c\, Q_s)}$, in which case it is equal to $f(Q_s)$.

We now establish Q_s. We require $f(x)$ to be upper bounded by $2^{-\kappa}$ over $\{1, \ldots, Q_s\}$. By computing its derivative, one can check that f is non-increasing over $\left[1, \frac{1}{bc}\right]$ and non-decreasing over $\left[\frac{1}{bc}; \infty\right)$. Since $f(1) \leq 2^{-\kappa}$, it suffices to study f over $\left[\frac{1}{bc}; \infty\right)$. Since f is non-decreasing over this set, Q_s can be computed by dichotomy over $\left[\frac{1}{bc}; \infty\right)$.

Tail bound ϵ_{TAIL} and number of leaked vectors η. We use a coarse bound on the L_{T_w} norm:

$$\begin{aligned}\|c_{\text{poly}} \cdot (\mathbf{s}, \mathbf{e})\|_{T_w}^{T_w} &\le n\,(k+\ell)\,\|c_{\text{poly}} \cdot (\mathbf{s}, \mathbf{e})\|_{\infty}^{T_w} \\ &\le n\,(k+\ell)\,(\omega\, T_t\, 2^{u_t - 1})^{T_w}\end{aligned} \tag{30}$$

Combining Eqs. (14) and (30), with $T_t \le 2\,T_w$, provides a heuristic approximation for ϵ_{TAIL}:

$$\epsilon_{\text{TAIL}} \approx \frac{\alpha^{T_w} \|c_{\text{poly}} \cdot (\mathbf{s}, \mathbf{e})\|_{T_w}^{T_w}}{2^{u_w \cdot T_w}\, T_w!} \le \frac{n\,(k+\ell)}{\sqrt{2\,\pi\, T_w}} \left(\alpha\, e\, \omega\, 2^{u_t - u_w - 2}\right)^{T_w} \tag{31}$$

Note that for our parameter sets, $T_w = \mathsf{rep} \cdot d - t$ takes the following values: $T_w = 8, 7, 5, 25, 17, 93$ for $d = 1, 2, 4, 8, 16, 32$, respectively. If $\epsilon_{\text{TAIL}} \le 2^{-\kappa}$, we can directly use Theorem 1. If $\epsilon_{\text{TAIL}} > 2^{-\kappa}$, then we argue security via ExtMLWE, as discussed in Section 4.1.4. We may approximate the Bernoulli distribution by a Poisson distribution of parameter $2\,Q_s\,\epsilon_{\text{TAIL}}$. Note that the Poisson distribution dominates the Bernoulli distribution on their tail, therefore we may safely rely on Poisson tail bounds. Except with probability $< 2^{-\kappa}$, the number of *individual* leaked coefficients will less than η as long as:

$$\frac{(2\,Q_s\,\epsilon_{\text{TAIL}})^{\eta}}{\eta!} \lesssim 2^{-\kappa} \tag{32}$$

Concretely, upper bounds on η (with overwhelming probability) for our parameter sets is given by Table 7. We study the corresponding ExtMLWE instance in Section 4.3.7.

Table 7: Upper bound on the number of leaked vectors η as a function of d and κ

	1	2	4	8	16	32
128	-	-	1	-	-	-
192	1	1	3	-	-	-
256	1	2	4	-	-	-

4.3.7 On the ExtMLWE Assumption.

In Section C.2, we present a dimension-preserving reduction from the unstructured variant of the problem (ExtLWE) to regular LWE, providing evidence of its asymptotic difficulty. The analysis of this problem is needed to take into the leaks induced by the BAD events discussed in Section 4.1.4. Now, we turn our attention to estimating the practical security of this problem by exploiting the information leakage introduced by the matrix $\mathbf{M}$. Similar to our analysis in Section 4.3.4, we rely on investigating the hardness of the integer descended problem, which corresponds to an (unstructured) ExtLWE instance.

Like previously, given an instance of $\mathsf{ExtMLWE}_{q,\ell,k,\mathcal{D},\mathcal{F}}$ as:

$$\left(\mathbf{A}, \mathbf{b}, \mathbf{M}, \mathbf{M} \cdot \text{coeff}\left(\begin{bmatrix}\mathbf{s}\\ \mathbf{e}\end{bmatrix}\right)\right),$$

our goal is to recover a short vector in the lattice $\mathcal{L} = \mathbf{x} \in \mathbb{Z}^d : \mathbf{D}\mathbf{x} = 0 \pmod{q}$. Here, the matrix $\mathbf{D} = (\text{rot}(\mathbf{A})_{[1:m]} | \mathbf{I}_m | \,\text{coeff}(\mathbf{b})_{[1:m]})$ has dimensions $\mathbf{D} \in \mathbb{Z}^{m \times d}$, where $d = n \cdot \ell + m + 1$ and $m \le n \cdot k$. Finding $\text{vec}(s1), \text{vec}(s2) \in \mathbb{Z}^{n \cdot \ell} \times \mathbb{Z}^{n \cdot k}$ from the instance $(\text{rot}(\mathbf{A}), \text{vec}(\mathbf{t}))$, where $\text{vec}(\cdot)$ maps a vector of ring elements to the vector obtained by concatenating the coefficients of its coordinates, and $\text{rot}(\mathbf{A}) \in \mathbb{Z}^{n \cdot k \times n \cdot \ell}$, is obtained by replacing all entries $a \in \mathcal{R}_q$ of $\mathbf{A}$ by the $n \times n$ matrix whose f-th column is $(X^{f-1} \cdot a_{ij})$. The hint part of the problem aligns well with this conversion to an LWE instance over $\mathbb{Z}$. In our definition, matrix $\mathbf{M}$ acts on the coefficient embedding of the vector $(\mathbf{s}, \mathbf{e})$, which, in hindsight, is the embedding used to construct $\mathcal{L}$.

To exploit the leakage, remark that each line of $\mathbf{M}$ gives rise to an equation of the form $\langle \mathbf{m}, (\mathbf{s}|\mathbf{e}) \rangle = \alpha$, where $\mathbf{m}$ is a vector from $\mathbf{M}$ and α is an integer. Consequently, we only need to search for solutions in the hyperplane coset $\ker(\langle \mathbf{m}, \cdot \rangle) - \alpha$ intersected with $\mathcal{L}$. As discussed in [DDGR20], we can avoid dealing with lattice cosets by embedding the lattice into $\{(\mathbf{u}, 1), |, \mathbf{u} \in \mathcal{L}\}$ and considering the intersection with the kernel of the map $\mathbf{x} \mapsto \langle (\mathbf{m}, -\alpha), \mathbf{x} \rangle$.

Remark 2. Overall, this technique bears striking resemblance to the framework developed in [DDGR20]. Notably, we leverage the `leaky estimator` they provide to incorporate these hints into the conventional primal LWE estimation.

4.3.8 Putting it All Together.

In order to ensure the desired level of security, we can now substitute each term in Eq. (23) and determine the appropriate parameters accordingly. However, it is worth noting that the effects of these parameters are highly intertwined, and there is no apparent straightforward order in which to optimize them. To facilitate this optimization process, we have compiled a table outlining the dependencies of each parameter, as shown below:

Table 8: Impact of parameters on security and performance metrics. Terminology: ↗↗ (resp. ↗, =, ↘, ↘↘) indicates that increasing this parameter has a very positive (resp. positive, negative, neutral, very negative) impact on the considered metric.

	Key rec. (§4.3.4) (MLWE)	Forgery (§4.3.5) (SelfTargetMSIS)	Leakage (§ 4.3.6) ϵ_{TAIL}	$R_\alpha^{\epsilon_{\text{TAIL}}}$	Size of vk	Size of sig
q	↘	↗↗	=	=	↘	↘
u_t	↗↗	=	↘↘	↘↘	↗	↘
u_w	=	↘↘	↗↗	↗↗	=	↘
$d \cdot \mathsf{rep}$	↗	↘	↗↗	=	=	↘
ν_t	=	↘	=	=	↗↗	=
ν_w	=	↘	=	=	=	↗↗
n	↗↗	↗↗	↘	↘	↘↘	↘↘
ℓ	↗↗	↗	↘	↘	=	↘↘
k	=	↗↗	↘	↘	↘↘	↘
ω	=	↘	↘	↘	=	=

The selection of parameters becomes an optimization problem involving these ten parameters. The objective is to ensure the desired level of security while minimizing the overall size, represented by the sum of the sizes of $|\mathsf{vk}| + |\sigma|$. To address this problem practically, the conditions are hard-encoded, and an almost exhaustive exploration of the search space is performed. Practical figures are given in Table 9, where we *very conservatively* estimated the securities with the maximum possible number of leakage vectors.

Table 9: Security of the Raccoon family (bitsec are given for the classical/quantum regime)

	Raccoon-128-x	Raccoon-192-x	Raccoon-256-x
Key recovery (MLWE) [bits]	134/115	193/166	284/243
Forgery (MSIS) [bits]	134/114	214/183	292/250
Number of queries	2^{55}	2^{51}	2^{53}
Leaked vectors	≤ 1	≤ 3	≤ 4

Remark 3. Since in practice the conditions detailed in are verified by the parameters we also get the strong sEUF-CMA security guarantee as byproduct of our conservative choices of design. A discussion on the asymptotic realization of this property is given in Appendix D.4.

4.4 Additional "BUFF" Security Properties

Cramers et al. [CDF+21] discuss three additional security properties that go beyond the security requirement of existentially unforgeable digital signatures with respect to an adaptive chosen message attack (EUF-CMA). These properties are not necessarily implied by EUF-CMA or by each other. However, it is easy to see that Raccoon has these "BUFF" properties:

Proposition. *Raccoon provides exclusive ownership (M-S-UEO), message-bound signatures (MBS), and non re-signability (NR), assuming that the hash functions used are collision-resistant and non-malleable.*

Due to structural similarities between Raccoon and Dilithium, we refer to [CDF+21, Proposition V.1][8] and related lemmas for detailed security arguments for all three properties. To see that the necessary conditions are met for Raccoon, we observe that the c_{hash} component of Raccoon signature (Algorithm 2) is composed as $c_{\mathsf{hash}} = \mathsf{ChalHash}(\mathbf{w}, \mu)$ (Line 10) where $\mu = \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg})$ (Line 2). Since both ChalHash and H are 2κ-bit collision-resistant hashes, the composition c_{hash} in a Raccoon signature is collision-resistant in relation to both vk and msg.

We can also examine the BUFF transformation [CDF+21, Figure 5] and its lemmas; we see that Raccoon's μ is functionally equivalent to the signed message digest h in BUFF and that μ is contained in the signature via c_{hash}.

[8] The Dilithium security argument is Proposition 6.1. in the January 2023 IACR e-Print version of [CDF+21].

Bibliography

[AA16] Jacob Alperin-Sheriff and Daniel Apon. Dimension-preserving reductions from LWE to LWR. Cryptology ePrint Archive, Report 2016/589, 2016. https://eprint.iacr.org/2016/589.

[AAC+22] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, and Yi-Kai Liu. NISTIR 8413 – Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process, 2022. https://doi.org/10.6028/NIST.IR.8413.

[ABC+22] Melissa Azouaoui, Olivier Bronchain, Gaëtan Cassiers, Clément Hoffmann, Yulia Kuzovkova, Joost Renes, Markus Schönauer, Tobias Schneider, François-Xavier Standaert, and Christine van Vredendaal. Leveling Dilithium against leakage: Revisited sensitivity analysis and improved implementations. Cryptology ePrint Archive, Report 2022/1406, 2022. https://eprint.iacr.org/2022/1406.

[ABD+14] Massimo Alioto, Simone Bongiovanni, Milena Djukanovic, Giuseppe Scotti, and Alessandro Trifiletti. Effectiveness of leakage power analysis attacks on DPA-resistant logic styles under process variations. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(2):429–442, 2014. doi:10.1109/TCSI.2013.2278350.

[ADH+19] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 717–746. Springer, Heidelberg, May 2019. doi:10.1007/978-3-030-17656-3_25.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, August 2016.

[ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 333–362. Springer, Heidelberg, August 2016. doi:10.1007/978-3-662-53015-3_12.

[AP12] Jacob Alperin-Sheriff and Chris Peikert. Circular and KDM security for identity-based encryption. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 334–352. Springer, Heidelberg, May 2012. doi:10.1007/978-3-642-30057-8_20.

[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. URL: https://doi.org/10.1515/jmc-2015-0016 [cited 2022-08-02], doi:doi:10.1515/jmc-2015-0016.

[ARM22] ARM. PSA cryptography API 1.1. Arm Document number: IHI 0086, February 2022. URL: https://arm-software.github.io/psa-api/crypto/1.1/IHI0086-PSA_Certified_Crypto_API-1.1.2.pdf.

[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. doi:10.4230/LIPIcs.ICALP.2022.8.

[BAA+17] Nina Bindel, Sedat Akleylek, Erdem Alkim, Paulo S. L. M. Barreto, Johannes Buchmann, Edward Eaton, Gus Gutoski, Juliane Kramer, Patrick Longa, Harun Polat, Jefferson E. Ricardini, and Gustavo Zanon. qTESLA. 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.

[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. doi:10.1145/2976749.2978427.

[BBD+23] Manuel Barbosa, Gilles Barthe, Christian Doczkal, Jelle Don, Serge Fehr, Benjamin Grégoire, Yu-Hsuan Huang, Andreas Hülsing, Yi Lee, and Xiaodi Wu. Fixing and mechanizing the security proof of fiat-shamir with aborts and dilithium. Cryptology ePrint Archive, Paper 2023/246, 2023. https://eprint.iacr.org/2023/246. URL: https://eprint.iacr.org/2023/246.

[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, Heidelberg, April / May 2018. doi:10.1007/978-3-319-78375-8_12.

[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. doi:10.1145/3319535.3363223.

[BDE+18] Jonathan Bootle, Claire Delaplace, Thomas Espitau, Pierre-Alain Fouque, and Mehdi Tibouchi. LWE without modular reduction and improved side-channel attacks against BLISS. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 494–524. Springer, Heidelberg, December 2018. doi:10.1007/978-3-030-03326-2_17.

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th SODA*, pages 10–24. ACM-SIAM, January 2016. doi:10.1137/1.9781611974331.ch2.

[BG14] Shi Bai and Steven D. Galbraith. An improved compression technique for signatures based on learning with errors. In Josh Benaloh, editor, *CT-RSA 2014*, volume 8366 of *LNCS*, pages 28–47. Springer, Heidelberg, February 2014. doi:10.1007/978-3-319-04852-9_2.

[BJRW23] Katharina Boudgoust, Corentin Jeudy, Adeline Roux-Langlois, and Weiqiang Wen. On the hardness of module learning with errors with short distributions. *Journal of Cryptology*, 36(1):1, January 2023. doi:10.1007/s00145-022-09441-3.

[BK15] Elaine Barker and John Kelsey. Recommendation for Random Number Generation Using Deterministic Random Bit Generators. NIST Special Publication SP 800-90A Revision 1, June 2015. doi:10.6028/NIST.SP.800-90Ar1.

[BKM+22] Elaine Barker, John Kelsey, Kerry McKay, Allen Roginsky, and Meltem Sönmez Turan. Recommendation for Random Bit Generator (RBG) Constructions (3rd Draft). Draft NIST Special Publication SP 800-90C, September 2022. doi:10.6028/NIST.SP.800-90C.3pd.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 575–584. ACM Press, June 2013. doi:10.1145/2488608.2488680.

[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. doi:10.1145/1180405.1180453.

[CAD+20] David A. Cooper, Daniel C. Apon, Quynh H. Dang, Michael S. Davidson, Morris J. Dworkin, and Carl A. Miller. Recommendation for Stateful Hash-Based Signature Schemes. NIST Special Publication SP 800-208, October 2020. doi:10.6028/NIST.SP.800-208.

[CDF+21] Cas Cremers, Samed Düzlü, Rune Fiedler, Marc Fischlin, and Christian Janson. BUFFing signature schemes beyond unforgeability and the case of post-quantum signatures. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1696–1714. IEEE, 2021. URL: https://eprint.iacr.org/2020/1525, doi:10.1109/SP40001.2021.00093.

[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, Heidelberg, March 2015. doi:10.1007/978-3-662-48116-5_7.

[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, Heidelberg, September 2014. doi:10.1007/978-3-662-44709-3_11.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999. doi:10.1007/3-540-48405-1_26.

[CL21] André Chailloux and Johanna Loyer. Lattice sieving via quantum random walks. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 63–91. Springer, Heidelberg, December 2021. doi:10.1007/978-3-030-92068-5_3.

[Cor99] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Çetin Kaya Koç and Christof Paar, editors, *CHES'99*, volume 1717 of *LNCS*, pages 292–302. Springer, Heidelberg, August 1999. doi:10.1007/3-540-48059-5_25.

[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, Shiuh-Pyng Shieh, Guofei Gu, and Giuseppe Ateniese, editors, *ASIACCS 20*, pages 853–866. ACM Press, October 2020. doi:10.1145/3320269.3384758.

[Cri22] Common Criteria. Common criteria for information technology security evaluation. part 5: Pre-defined packages of security requirements. CCMB-2022-11-005, November 2022. https://www.commoncriteriaportal.org/files/ccfiles/CC2022PART5R1.pdf.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020. doi:10.1109/TIFS.2020.2971153.

[Csi63] Imre Csiszár. Eine informationstheoretische Ungleichung und ihre Anwendung auf den Beweis der Ergodizitat von Markoffschen Ketten. *Magyar. Tud. Akad. Mat. Kutató Int. Közl*, 8:85–108, 1963.

[DDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 329–358. Springer, Heidelberg, August 2020. doi:10.1007/978-3-030-56880-1_12.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 40–56. Springer, Heidelberg, August 2013. doi:10.1007/978-3-642-40041-4_3.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *Journal of Cryptology*, 34(3):33, July 2021. doi:10.1007/s00145-021-09398-9.

[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, Heidelberg, December 2022. doi:10.1007/978-3-031-22972-5_2.

[DFPS23] Julien Devevey, Pouria Fallahpour, Alain Passelègue, and Damien Stehlé. A detailed analysis of fiat-shamir with aborts. Cryptology ePrint Archive, Report 2023/245, 2023. https://eprint.iacr.org/2023/245.

[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. doi:10.1007/s00145-018-9277-0.

[DKL+18a] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR TCHES*, 2018(1):238–268, 2018. https://tches.iacr.org/index.php/TCHES/article/view/839. doi:10.13154/tches.v2018.i1.238-268.

[DKL+18b] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):238–268, 2018. doi:10.13154/tches.v2018.i1.238-268.

[dPPRS23] Rafaël del Pino, Thomas Prest, Mélissa Rossi, and Markku-Juhani O. Saarinen. High-order masking of lattice signatures in quasilinear time. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, 22-25 May 2023*, pages 1152–1169. IEEE, 2023. doi:10.1109/SP46215.2023.00160.

[drh17] drhab. Expected value of square of euclidean norm of a gaussian random vector. Mathematics Stack Exchange, 2017. Author profile: https://math.stackexchange.com/users/75923/drhab. Version: 2017-11-22. URL: https://math.stackexchange.com/q/2530567.

[Duc18] Léo Ducas. Shortest vector from lattice sieving: A few dimensions for free. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 125–145. Springer, Heidelberg, April / May 2018. doi:10.1007/978-3-319-78381-9_5.

[EFGT17] Thomas Espitau, Pierre-Alain Fouque, Benoît Gérard, and Mehdi Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongSwan and electromagnetic emanations in microcontrollers. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1857–1874. ACM Press, October / November 2017. doi:10.1145/3133956.3134028.

[EJK20] Thomas Espitau, Antoine Joux, and Natalia Kharchenko. On a dual/hybrid approach to small secret LWE - A dual/enumeration technique for learning with errors and application to security estimates of FHE schemes. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 440–462. Springer, Heidelberg, December 2020. doi:10.1007/978-3-030-65277-7_20.

[EK20] Thomas Espitau and Paul Kirchner. The nearest-colattice algorithm. Cryptology ePrint Archive, Report 2020/694, 2020. https://eprint.iacr.org/2020/694.

[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. doi:10.1007/978-3-030-60939-9_19.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987. doi:10.1007/3-540-47721-7_12.

[Glo21] GlobalPlatform. TEE internal core API specification – public release v1.3.1. Document Reference: GPD_SPE_010, July 2021. URL: https://globalplatform.org/specs-library/tee-internal-core-api-specification/.

[GLP12] Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 530–547. Springer, Heidelberg, September 2012. doi:10.1007/978-3-642-33027-8_31.

[GMRR22] Morgane Guerreau, Ange Martinelli, Thomas Ricosset, and Mélissa Rossi. The hidden parallelepiped is back again: Power analysis attacks on falcon. *IACR TCHES*, 2022(3):141–164, 2022. doi:10.46586/tches.v2022.i3.141-164.

[Gou01] Louis Goubin. A sound method for switching between Boolean and arithmetic masking. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES 2001*, volume 2162 of *LNCS*, pages 3–15. Springer, Heidelberg, May 2001. doi:10.1007/3-540-44709-1_2.

[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. doi:10.1007/978-3-030-42068-0_5.

[HT19] Michael Hutter and Michael Tunstall. Constant-time higher-order Boolean-to-arithmetic masking. *Journal of Cryptographic Engineering*, 9(2):173–184, June 2019. doi:10.1007/s13389-018-0191-z.

[ISO19] ISO. It security techniques – test tool requirements and test tool calibration methods for use in testing non-invasive attack mitigation techniques in cryptographic modules – part 1: Test tools and techniques. Standard ISO/IEC 20085-1:2019(E), International Organization for Standardization, 2019. URL: https://www.iso.org/standard/70081.html.

[ISO20] ISO. IT security techniques – test tool requirements and test tool calibration methods for use in testing non-invasive attack mitigation techniques in cryptographic modules – part 2: Test calibration methods and apparatus. Standard ISO/IEC 20085-2:2020(E), International Organization for Standardization, 2020. URL: https://www.iso.org/standard/70082.html.

[ISO22] ISO. Information technology – security techniques – security requirements for cryptographic modules. Standard ISO/IEC WD 19790:2022(E), International Organization for Standardization, 2022.

[ISO23] ISO. Information technology – security techniques – testing methods for the mitigation of non-invasive attack classes against cryptographic modules. Draft International Standard ISO/IEC DIS 17825:2022(E), International Organization for Standardization, 2023.

[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, Heidelberg, August 2003. doi:10.1007/978-3-540-45146-4_27.

[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. doi:10.1145/3548606.3560579.

[KA21] Emre Karabulut and Aydin Aysu. FALCON Down: Breaking FALCON Post-Quantum Signature Scheme through Side-Channel Attacks. In *58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021*, pages 691–696. IEEE, 2021. doi:10.1109/DAC18074.2021.9586131.

[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. doi:10.1109/HOST49136.2021.9702284.

[KGB+18] Matthias J. Kannwischer, Aymeric Genêt, Denis Butin, Juliane Krämer, and Johannes Buchmann. Differential power analysis of XMSS and SPHINCS. In Junfeng Fan and Benedikt Gierlichs, editors, *COSADE 2018*, volume 10815 of *LNCS*, pages 168–188. Springer, Heidelberg, April 2018. doi:10.1007/978-3-319-89641-0_10.

[KLS18] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 552–586. Springer, Heidelberg, April / May 2018. doi:10.1007/978-3-319-78372-7_18.

[LDK+22] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède 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.

[LNS21] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. Shorter lattice-based zero-knowledge proofs via one-time commitments. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 215–241. Springer, Heidelberg, May 2021. doi:10.1007/978-3-030-75245-3_9.

[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, Heidelberg, May 2013. doi:10.1007/978-3-642-38348-9_3.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015. doi:10.1007/s10623-014-9938-4.

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Heidelberg, December 2009. doi:10.1007/978-3-642-10366-7_35.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 738–755. Springer, Heidelberg, April 2012. doi:10.1007/978-3-642-29011-4_43.

[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 19*, volume 11464 of *LNCS*, pages 344–362. Springer, Heidelberg, June 2019. doi:10.1007/978-3-030-21568-2_17.

[MR02] Silvio Micali and Leonid Reyzin. Improving the exact security of digital signature schemes. *Journal of Cryptology*, 15(1):1–18, January 2002. doi:10.1007/s00145-001-0005-8.

[MRS22] Loïc Masure, Olivier Rioul, and François-Xavier Standaert. A nearly tight proof of duc et al.'s conjectured security bound for masked implementations. In Ileana Buhan and Tobias Schneider, editors, *Smart Card Research and Advanced Applications - 21st International Conference, CARDIS 2022, Birmingham, UK, November 7-9, 2022, Revised Selected Papers*, volume 13820 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2022. URL: https://eprint.iacr.org/2022/600, doi:10.1007/978-3-031-25319-5_4.

[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. URL: https://math.stackexchange.com/q/1140573.

[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. https://eprint.iacr.org/2022/106.

[MW16] Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 820–849. Springer, Heidelberg, May 2016. doi:10.1007/978-3-662-49890-3_31.

[NIS15] NIST. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Federal Information Processing Standards Publication FIPS 202, August 2015. doi:10.6028/NIST.FIPS.202.

[NIS19] NIST. Security Requirements for Cryptographic Modules. Federal Information Processing Standards Publication FIPS 140-3, March 2019. doi:10.6028/NIST.FIPS.140-3.

[NIS22] NIST. Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf, 2022.

[NIS23a] NIST. Digital Signature Standard (DSS). Federal Information Processing Standards Publication FIPS 186-5, February 2023. doi:10.6028/NIST.FIPS.186-5.

[NIS23b] NIST. NIST IR 8214C ipd: NIST First Call for Multi-Party Threshold Schemes (Initial Public Draft). https://doi.org/10.6028/NIST.IR.8214C.ipd, 2023. doi:doi:10.6028/NIST.IR.8214C.ipd.

[OO98] Kazuo Ohta and Tatsuaki Okamoto. On concrete security treatment of signatures derived from identification. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 354–369. Springer, Heidelberg, August 1998. doi:10.1007/BFb0055741.

[OPW11] Adam O'Neill, Chris Peikert, and Brent Waters. Bi-deniable public-key encryption. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 525–542. Springer, Heidelberg, August 2011. doi:10.1007/978-3-642-22792-9_30.

[PBY17] Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. To BLISS-B or not to be: Attacking strongSwan's implementation of post-quantum signatures. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1843–1855. ACM Press, October / November 2017. doi:10.1145/3133956.3134023.

[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, Heidelberg, December 2017. doi:10.1007/978-3-319-70694-8_13.

[Pre23] Thomas Prest. A Key-Recovery Attack Against Mitaka in the t-Probing Model. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *Public-Key Cryptography – PKC 2023*, pages 205–220, Cham, 2023. Springer Nature Switzerland.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000. doi:10.1007/s001450010003.

[Saa23] Markku-Juhani O. Saarinen. WrapQ: Side-channel secure key management for post-quantum cryptography. IACR ePrint 2022/1499, April 2023. URL: https://eprint.iacr.org/2022/1499.

[SOG22] SOGIS. Application of attack potential to smartcards and similar devices. Joint Interpretation Library – Version 3.2, November 2022. https://www.sogis.eu/documents/cc/domains/sc/JIL-Application-of-Attack-Potential-to-Smartcards-v3.2.pdf.

[TBK+18] Meltem Sönmez Turan, Elaine Barker, John Kelsey, Kerry A. McKay, Mary L. Baish, and Mike Boyle. Recommendation for the Entropy Sources Used for Random Bit Generation. NIST Special Publication SP 800-90B, January 2018. doi:10.6028/NIST.SP.800-90B.

[YD17] Yang Yu and Léo Ducas. Second order statistical behavior of LLL and BKZ. In Carlisle Adams and Jan Camenisch, editors, *SAC 2017*, volume 10719 of *LNCS*, pages 3–22. Springer, Heidelberg, August 2017. doi:10.1007/978-3-319-72565-9_1.

[ZLYW23] Shiduo Zhang, Xiuhan Lin, Yang Yu, and Weijia Wang. Improved Power Analysis Attacks on Falcon. Cryptology ePrint Archive, Paper 2023/224, 2023. URL: https://eprint.iacr.org/2023/224.

[ZSS+21] Sara Zarei, Aein Rezaei Shahmirzadi, Hadi Soleimany, Raziyeh Salarifard, and Amir Moradi. Low-latency keccak at any arbitrary order. *IACR TCHES*, 2021(4):388–411, 2021. https://tches.iacr.org/index.php/TCHES/article/view/9070. doi:10.46586/tches.v2021.i4.388-411.

A Rényi Divergence Arguments for Sums of Discrete Uniform Variables

This section provides a collection of results related to sums of discrete uniform variables.

A.1 The Sum of Discrete Uniform Variables

Definition 6. *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 2 links the cumulative distribution function (CDF) of $P_{N,T}$ and the probability distribution function (PDF) of $P_{N,T+1}$.

Lemma 2. *For any $x \geq 0$, $P_{N,T+1}(x) = \frac{1}{N} P_{N,T}(\{\max(0, x-N+1), \ldots, x\})$.*

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:

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

□

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

Lemma 3. *For $x \in [N]$:*

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

By Lemma 2, this implies:

$$P_{N,T}(x) = \binom{x+T-1}{T-1} \frac{1}{N^T} \tag{33}$$

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 2, $(T+1)$ 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:

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

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

Monotony of $x \to P_{N,T}(x+c)/P_{N,T}(x)$.

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 4. *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 4 in the special case $c = 1$. The general case follows from the telescopic product:

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

For the rest of the proof, let $c = 1$. For $x < N$, the statement can be verified using Lemma 3. 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]. □

A.2 Smooth Rényi Divergence Between Shifted Copies of $P_{N,T}$

The goal of this section is to prove Lemma 1. We first partition $\mathrm{Supp}(P) \cup \mathrm{Supp}(Q)$ in five sections, as illustrated in Figure 4:

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.

Informally speaking, our proof strategy is to separately bound the statistical distance over the tails (Appendix A.2.1), and the partial Rényi divergence over the sides (Appendix A.2.2) and head (Appendix A.2.3). The smooth Rényi divergence between P and Q is obtained (Appendix A.2.4) as a simple consequence of these separate bounds.

A.2.1 Selecting P' and Q'

Let $\tau > 0$ such that $\frac{(\tau+T)^T}{T!\,N^T} \leq \epsilon$. By Lemma 3, 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 $\Delta_{\mathrm{SD}}(P', P) \leq \epsilon$ and $\Delta_{\mathrm{SD}}(Q', Q) = 0$.

A.2.2 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 5. *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:*

$$\sum_{x \in S_\ell \sqcup S_r} \left(\frac{P(x)}{Q(x)}\right)^\alpha Q(x) \leq \frac{1}{T!}\left(2 + \frac{T}{T-2}\left(\frac{a}{N}\right)^2 + O((a/N)^3)\right). \tag{34}$$

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:

$$x \mapsto \left(\frac{x+c}{x}\right)^\alpha x \qquad \text{is non-decreasing over } [(\alpha-1)c; +\infty) \tag{35}$$

$$f_{a,T} : x \mapsto \exp\left(\frac{a}{x}\right) x^T \qquad \text{is non-decreasing over } [\alpha/T; +\infty) \tag{36}$$

In particular, since $\max((\alpha-1)\,c, \alpha/T) \leq \tau - c$, they are non-decreasing over $[\tau - c; +\infty)$. Since Lemma 12, Eq. (33) 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 $x \in S_\ell$:

$$\begin{aligned}
\left(\frac{P(x)}{Q(x)}\right)^\alpha Q(x) &= \frac{1}{(T-1)!\,N^T} \prod_{u=x-c}^{x-c+T-1} \left(\frac{u+c}{u}\right)^\alpha u \\
&\leq \frac{1}{(T-1)!\,N^T} \left(\frac{x+T-1}{x-c+T-1}\right)^{(T-1)\alpha} (x-c+T-1)^{T-1} \qquad (37) \\
&\leq \frac{1}{(T-1)!\,N^T} \exp\left(\frac{c(T-1)\alpha}{x-c+T-1}\right) (x-c+T-1)^{T-1} \qquad (38)
\end{aligned}$$

(37) follows from the non-decreasingness of (35), while (38) follows from Bernoulli's inequality $1 + x \leq \exp(x)$. We now bound the partial Rényi divergence sum over the left side S_ℓ:

$$\begin{aligned}
\sum_{x \in S_\ell} \left(\frac{P(x)}{Q(x)}\right)^\alpha Q(x) &\leq \frac{1}{(T-1)!\,N^T} \sum_{x=\tau-c+T-2}^{n-c+T-1} \exp\left(\frac{c(T-1)\alpha}{x}\right) x^{T-1} \\
&\leq \frac{1}{(T-1)!\,N^T} \sum_{x=\tau}^{N-1} f_{a,T-1}(x) \qquad (39) \\
&\leq \frac{1}{(T-1)!\,N^T} \int_\tau^N f_{a,T-1}(u) du, \qquad (40)
\end{aligned}$$

where (39) is implied by $f_{a,T-1}$ being non-decreasing, see (36), and (40) 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 $\tau \le a$ and $a = o(N)$, we can approximate $F_{a,T}$ using Taylor series as follows:

$$\begin{aligned} F_{a,T} &= \frac{\exp(a/N)}{T+1}\left(N^{T+1} + \frac{a}{T} N^T + \frac{a^2}{T(T-1)} N^{T-1} + O(a^3 N^{T-2})\right) \\ &= \frac{N^{T+1}}{T+1}\left(1 + \left(1 + \frac{1}{T}\right)\frac{a}{N} + \frac{T+1}{2(T-1)}\left(\frac{a}{N}\right)^2 + O((a/N)^3)\right) \end{aligned} \tag{41}$$

Combining (40) and (41) gives the partial sum on the left tail:

$$\sum_{x \in S_\ell} \left(\frac{P(x)}{Q(x)}\right)^\alpha Q(x) \le \frac{1}{T!}\left(1 + \left(1 + \frac{1}{T-1}\right)\frac{a}{N} + \frac{T}{2(T-2)}\left(\frac{a}{N}\right)^2 + O((a/N)^3)\right) \tag{42}$$

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

$$\sum_{x \in S_r} \left(\frac{P(x)}{Q(x)}\right)^\alpha Q(x) \le \frac{1}{T!}\left(1 - \left(1 + \frac{1}{T-1}\right)\frac{a}{N} + \frac{T}{2(T-2)}\left(\frac{a}{N}\right)^2 + O((a/N)^3)\right) \tag{43}$$

Adding (42) and (43) gives the result.

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

□

A.2.3 Partial Sum Over the Head

Lemma 6. *Let $cT = o(N)$. Then:*

$$\sum_{x \in H} \left(\frac{P(x)}{Q(x)}\right)^\alpha Q(x) \le 1 + \frac{\alpha(\alpha-1)}{2}\left(\frac{(cT)^2}{N^2} + O\left(\frac{(cT)^3}{N^3}\right)\right) - \frac{2}{T!}\left(1 + 4(c/N)^2 + O((c/N)^4)\right) \tag{44}$$

Proof. Our goal is to apply [Pre17, Lemma 3]. This lemma requires us to bound the ratio $P(x)/Q(x)$ over $H = \{N, \ldots, (T-1)(N-1) + c\}$. Thanks to the monotonicity of this ratio (Lemma 4), we know that it suffices to bound it at the extremities of H.

$$\frac{P(N)}{Q(N)} = \prod_{x=N}^{N+T-1} \frac{x}{x-c} \le \left(\frac{N}{N-c}\right)^T \le \exp\left(\frac{cT}{N-c}\right) \tag{45}$$

Similarly, by symmetry:

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

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 $P(\mathbb{Z}\backslash H) = Q(\mathbb{Z}\backslash H)$. 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:

$$\sum_{x \in H} \left(\frac{P(x)}{Q(x)}\right)^{\alpha} Q(x) \leq 1 + \frac{\alpha(\alpha-1)\delta^2}{2(1-\delta)^{\alpha+1}} - Q(\mathbb{Z}\backslash H) \tag{46}$$

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

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

□

A.2.4 Proof of Lemma 1

Proof. It suffices to prove Lemma 1 for $c \geq 0$ since $R_\alpha^\epsilon(P;Q) = R_\alpha^\epsilon(Q;P)$. We apply Lemma 3 to compute ϵ, and Lemmas 5 and 6 to compute the Rényi divergence sum. □

A.3 Distribution of Extreme Events

Lemma 7 informally states that the distribution of coefficients for which $(c_{\mathsf{poly}}\mathbf{s} + \mathbf{r}, c_{\mathsf{poly}}\mathbf{e} + \mathbf{e}')$ is "too large" is independent of the secret $(\mathbf{s}, \mathbf{e})$. In the context of Raccoon, $M = n\,(\ell + k)\,Q_s$ and $\mathbf{c}$ is the concatenation of all the vectors $c_{\mathsf{poly}}^{[i]}(\mathbf{s}, \mathbf{e})$, for all Q_s values $c_{\mathsf{poly}}^{[i]}$ taken during the game.

Lemma 7. *Consider the following:*

- $\mathbf{c} = (c_i)_i \in \mathbb{Z}^M$ *is a vector of integer values;*
- $\alpha, \tau, N, T \in \mathbb{N}$ *satisfy* $\alpha = \omega(1)$, $\alpha T = O(\tau)$, $\tau = \|\mathbf{c}\|_\infty(\alpha + 2)$ *and* $T \geq 2$*;*
- $\textsc{Tail} = \{0, \ldots, \tau\} \sqcup \{(N-1)T - \tau, \ldots, (N-1)T\}$*;*

For each $i \in [M]$*, we generate* r_i *as the sum of* T *discrete uniform variables in* $[N]$*, and set* $b_i = 1$ *if* $(c_i + r_i) \in \textsc{Tail}$*, otherwise we set* $b_i = 0$*. Let* $\mathcal{P}_{\mathbf{c}}$ *be the distribution of the vector* $\mathbf{b} = (b_i)_i$*.*

Let $p_0 = \frac{2\tau^T}{T!\,N^T} = O((T/\alpha)^2)$ *and let* $\mathcal{Q}$ *be the distribution of the* M*-dimensional vector for which each coefficient is sampled independently according to the Bernoulli distribution* Bern_{p_0}*. Note that* $p_0 \leq 2\,\epsilon$*, where* $\epsilon = \frac{(\tau+T)^T}{T!\,N^T}$*. We have:*

$$\begin{aligned}\log R_\alpha(\mathcal{P}_{\mathbf{c}}; \mathcal{Q}) &\leq \frac{\alpha\, T^4\, \|\mathbf{c}\|_4^4\, \epsilon}{4\,\tau^4}\left(1 + O((T/\alpha)^2)\right)\\ &\leq \frac{M\, T^4\, \epsilon}{4\,\alpha^3}\left(1 + O((T/\alpha)^2)\right).\end{aligned}$$

Proof. We start by studying the one-dimensional case. Let P_c be the distribution of $c_i + r_i$ when $c_i = c$.

$$P_c(\textsc{Tail}) = \binom{\tau - c + T}{T}\frac{1}{N^T} + \binom{\tau + c + T}{T}\frac{1}{N^T} \tag{47}$$

$$= \frac{1}{T!\,N^T}\left(\prod_{x=\tau-c+1}^{\tau-c+T} x + \prod_{y=\tau+c+1}^{\tau+c+T} y\right) \tag{48}$$

$$= q\left(1 + \frac{T(T-1)c^2}{2\tau^2} + O((c/\tau)^4)\right) \tag{49}$$

$\mathsf{Game}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}(\kappa) \to \{\mathsf{OK}\text{ or }\mathsf{FAIL}\}$

1: $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$
2: $\mathcal{Q}_{\mathsf{Sign}} := \emptyset$
3: $(\mathsf{msg}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathsf{OSgn}(\cdot)}(\mathsf{vk})$
4: **if** $\exists \mathsf{sig}'$ s.t. $(\mathsf{msg}^*, \mathsf{sig}') \in \mathcal{Q}_{\mathsf{Sign}}$ **then**
5: **return** FAIL
6: **return** $\mathsf{Verify}(\mathsf{sig}^*, \mathsf{msg}^*, \mathsf{vk})$

$\mathsf{Game}_{\mathcal{A}}^{\mathsf{sEUF\text{-}CMA}}(\kappa) \to \{\mathsf{OK}\text{ or }\mathsf{FAIL}\}$

1: $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\kappa)$
2: $\mathcal{Q}_{\mathsf{Sign}} := \emptyset$
3: $(\mathsf{msg}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathsf{OSgn}(\cdot)}(\mathsf{vk})$
4: **if** $(\mathsf{msg}^*, \mathsf{sig}^*) \in \mathcal{Q}_{\mathsf{Sign}}$ **then**
5: **return** FAIL
6: **return** $\mathsf{Verify}(\mathsf{sig}^*, \mathsf{msg}^*, \mathsf{vk})$

$\mathsf{OSgn}(\mathsf{msg}) \to \mathsf{sig}$

1: $\mathsf{sig} \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$
2: $\mathcal{Q}_{\mathsf{Sign}} := \mathcal{Q}_{\mathsf{Sign}} \cup \{(\mathsf{msg}, \mathsf{sig})\}$
3: **return** sig

Figure 8: Existential (EUF-CMA) and strong-existential (sEUF-CMA) unforgeability under chosen message attacks security games for digital signatures. In both games, the signing oracle OSgn remains the same.

Eq. (47) is immediate from Lemma 3, then Eq. (48) is a simple re-arrangement of the terms. Finally, Eq. (49) uses Taylor series and the fact that $\alpha c = O(\tau)$ and $\alpha T = O(\tau)$. Let us note $x = \frac{T(T-1)c^2}{2\tau^2} + O((c/\tau)^4)$ and $p = P_c(\textsc{Tail})$. We have $p = p_0(1 + x)$ and therefore:

$$\begin{aligned} R_\alpha(\mathsf{Bern}_p; \mathsf{Bern}_{p_0})^{\alpha-1} &= \left(\frac{1-p}{1-p_0}\right)^\alpha (1-p_0) + \left(\frac{p}{p_0}\right)^\alpha p_0 \\ &= (1 - p_0\,(1+x))^\alpha\,(1-p_0)^{1-\alpha}\,(1+x)^\alpha\,p_0 \end{aligned}$$

Using Taylor series at order 3 with respect to x and p_0 gives the following:

$$\log R_\alpha(\mathsf{Bern}_p; \mathsf{Bern}_{p_0}) = \frac{\alpha\,x^2\,p_0\,(1 + O(x, p_0))}{2}$$

Therefore if we note $\mathbf{c} = (c_i)_{i\in[M]}$, the tensorization property of the Rényi divergence gives:

$$\log R_\alpha(\mathcal{P}_{\mathbf{c}}; \mathcal{Q}) \le \frac{\alpha\,T^4\,\|\mathbf{c}\|_4^4\,q\,(1 + O(x, p_0))}{8\tau^4}$$

Since $q \le 2\,\epsilon$, we can conclude. □

B Deferred Definitions

B.1 Digital Signatures

We provide the formal definition of a digital signature scheme.

Definition 7 (Digital signature)**.** *A digital signature is a triple of algorithms* (KeyGen, Sign, Verify) *such that:*

$\mathsf{KeyGen}(1^\kappa) \to (\mathsf{vk}, \mathsf{sk})$*: This algorithm, from public parameters such as the security parameter κ outputs a signing key* sk *and a verification key* vk*. Moreover, it initializes any hash functions that may be used during the signature.*

$\mathsf{Sign}(\mathsf{sk}, \mathsf{msg}) \to \mathsf{sig}$: *From a signing key* sk *and a message* msg*, this algorithm derives a signature* sig *or returns an error message* $\perp$.

$\mathsf{Verify}(\mathsf{sig}, \mathsf{msg}, \mathsf{vk}) \to \mathsf{OK}/\mathsf{FAIL}$: *From a verification key* vk*, a message* msg *and a signature* sig*, this deterministic algorithm outputs* OK *if the signature is accepted and* FAIL *otherwise.*

Correctness. *A digital signature is said to be correct if for any valid message* $\mathsf{msg} \in \mathcal{M}$*, it holds that*

$$\Pr[\mathsf{Verify}(\mathsf{Sign}(\mathsf{sk}, \mathsf{msg}), \mathsf{msg}, \mathsf{vk}) = \mathsf{OK} \mid (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()] \leftarrow 1 - \mathsf{negl}(\kappa).$$

Security. *A digital signature is* existentially unforgeable under chosen message attacks (EUF-CMA) *if the following advantage of any efficient adversary* $\mathcal{A}$ *in winning the unforgeability game described in Figure 8 is negligible:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}} := \Pr[\mathsf{Game}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}(\kappa) = 1].$$

We also define strong *existentially unforgeable under chosen message attacks* (sEUF-CMA) *if the game is relaxed so that the adversary wins as long as* $(\mathsf{msg}^*, \mathsf{sig}^*) \notin Q_{\mathsf{Sign}}$.

Note that in this document, we consider a signature scheme that is Q_s-bounded for $Q_s = \mathsf{poly}(\kappa)$, where Q_s is the maximal signing query an adversary can perform. Specifically, the scheme parameters are set with respect to the upper bound Q_s. We set $Q_s \approx 2^{50}$ in our concrete parameter selection. See Section 2.1 for more details.

C More Detail on Hardness Assumptions

C.1 Hardness of SelfTargetMSIS

As discussed in Section 4.1.1, SelfTargetMSIS is known to be as difficult as MSIS. For completeness, we provide a reduction from SelfTargetMSIS to MSIS and display the asymptotic relations of the parameters.

Lemma 8 (Hardness of SelfTargetMSIS)**.** *For any adversary* $\mathcal{A}$ *against the* $\mathsf{SelfTargetMSIS}_{q,\ell,k,C,\nu,\beta}$ *problem making at most* Q_h *random oracle queries, there exists an adversary* $\mathcal{B}$ *against the* $\mathsf{MSIS}_{q,\ell,k,\beta'}$ *problem with* $\beta' = 4\beta + 2^{\nu+2} \cdot \sqrt{nk}$ *such that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SelfTargetMSIS}} \leq \sqrt{Q_h \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{MSIS}}} + \frac{Q_h}{|C|},$$

where $\mathsf{Time}(\mathcal{B}) \approx 2 \cdot \mathsf{Time}(\mathcal{A})$.

Proof Sketch. To construct $\mathcal{B}$, we simply invoke the standard forking lemma [BN06] to run $\mathcal{A}$ twice. In particular, when $\mathcal{B}$ receives $\mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}$ as input, it invokes $\mathcal{A}$ on input $\mathbf{A}$. $\mathcal{B}$ simulates the random oracle H on the fly by sampling a random $c \leftarrow C$. Eventually, $\mathcal{A}$ outputs $(\mathsf{msg}, \mathbf{s}, \mathbf{h}) \in \{0,1\}^{2\kappa} \times \mathcal{R}_q^{\ell+k} \times \mathcal{R}_{q_\nu}^k$ that breaks SelfTargetMSIS. That is

$$\left(\mathbf{s} = \begin{bmatrix} c \\ \mathbf{s}' \end{bmatrix}\right) \wedge (0 < \|(\mathbf{s}, 2^\nu \cdot \mathbf{h})\|_2 \leq \beta) \ \wedge\ \mathsf{G}\left(\lfloor [\mathbf{A} \mid \mathbf{I}] \cdot \mathbf{s}\rceil_\nu + \mathbf{h},\ \mathsf{msg}\right) = c.$$

Using the forking lemma, we can argue that when $\mathcal{B}$ rewinds $\mathcal{A}$, $\mathcal{A}$ outputs $(\overline{\mathsf{msg}}, \bar{\mathbf{s}}, \bar{\mathbf{h}})$ with a different $\bar{c} \neq c$ such that

$$\left(\bar{\mathbf{s}} = \begin{bmatrix} \bar{c} \\ \bar{\mathbf{s}}' \end{bmatrix}\right) \wedge \left(0 < \|(\bar{\mathbf{s}}, 2^\nu \cdot \bar{\mathbf{h}})\|_2 \leq \beta\right) \ \wedge\ \mathsf{G}\left(\lfloor [\mathbf{A} \mid \mathbf{I}] \cdot \bar{\mathbf{s}}\rceil_\nu + \bar{\mathbf{h}},\ \overline{\mathsf{msg}}\right) = \bar{c}.$$

with the specified probability in the statement. Moreover, since the forking lemma programs the random oracle on the same input, we have

$$\left\lfloor [\mathbf{A} \mid \mathbf{I}] \cdot \mathbf{s} \right\rceil_\nu + \mathbf{h} = \left\lfloor [\mathbf{A} \mid \mathbf{I}] \cdot \bar{\mathbf{s}} \right\rceil_\nu + \bar{\mathbf{h}} \mod q_\nu,$$

where recall that the equality holds over $\mathcal{R}_{q_\nu}$ for $q_\nu = \lfloor q/2^\nu \rceil$ as the input space of G is $\mathcal{R}_{q_\nu}^k \times \{0,1\}^{2\kappa}$. Now, since the output of $\lfloor \cdot \rceil_\nu$ is over $\mathcal{R}_{q_\nu}$ and $\mathbf{h} \in \mathcal{R}_{q_\nu}^k$, we have the equality over $\mathcal{R}_q$ for some $\boldsymbol{\delta} \in \mathcal{R}_q^k$ such that $\|\boldsymbol{\delta}\|_\infty \leq 1$:

$$\left\lfloor [\mathbf{A} \mid \mathbf{I}] \cdot \mathbf{s} \right\rceil_\nu + \mathbf{h} = \left\lfloor [\mathbf{A} \mid \mathbf{I}] \cdot \bar{\mathbf{s}} \right\rceil_\nu + \bar{\mathbf{h}} + q_\nu \cdot \boldsymbol{\delta} \mod q. \tag{50}$$

Define $\mathbf{d} \in \mathcal{R}_q^k$ as $\mathbf{d} := [\mathbf{A} \mid \mathbf{I}] \cdot \mathbf{s} - 2^\nu \cdot \left\lfloor [\mathbf{A} \mid \mathbf{I}] \cdot \mathbf{s} \right\rceil_\nu$. By definition, we have $\|\mathbf{d}\|_\infty \leq 2^{\nu-1}$. We define $\bar{\mathbf{d}}$ similarly. Then, by multiplying both sides of Eq. (50) by 2^ν and plugging in $\mathbf{d}$, we have

$$[\mathbf{A} \mid \mathbf{I}] \cdot \mathbf{s} + 2^\nu \cdot \mathbf{h} - \mathbf{d} = [\mathbf{A} \mid \mathbf{I}] \cdot \bar{\mathbf{s}} + 2^\nu \cdot \bar{\mathbf{h}} - \bar{\mathbf{d}} + 2^\nu \cdot q_\nu \cdot \boldsymbol{\delta} \mod q.$$

Define $\zeta \in \mathcal{R}_q$ as $\zeta := q - 2^\nu \cdot q_\nu$. Then, by definition, we have $|\zeta| \leq 2^{\nu-1}$. Therefore, we can rewrite the above equation as

$$[\mathbf{A} \mid \mathbf{I}] \cdot \mathbf{s} + 2^\nu \cdot \mathbf{h} - \mathbf{d} = [\mathbf{A} \mid \mathbf{I}] \cdot \bar{\mathbf{s}} + 2^\nu \cdot \bar{\mathbf{h}} - \bar{\mathbf{d}} - \zeta \cdot \boldsymbol{\delta} \mod q.$$

Equivalently, we have

$$[\mathbf{A} \mid \mathbf{I}] \cdot \underbrace{\left(\begin{bmatrix} c - \bar{c} \\ \mathbf{s}' - \bar{\mathbf{s}}' \end{bmatrix} + \begin{bmatrix} \mathbf{0}_\ell \\ 2^\nu \cdot (\mathbf{h} - \bar{\mathbf{h}}) \end{bmatrix} - \begin{bmatrix} \mathbf{0}_\ell \\ \boldsymbol{\delta}' \end{bmatrix} \right)}_{=:\mathbf{s}^*} = \mathbf{0}_k \mod q,$$

where $\boldsymbol{\delta}' = \zeta \cdot \boldsymbol{\delta} + \bar{\mathbf{d}} - \mathbf{d}$, and $\mathbf{0}_a$ denotes the zero-vector of length a. Moreover, $\mathbf{s}^*$ is bounded by

$$\begin{aligned}
\|\mathbf{s}^*\|_2^2 &\leq \left\| \begin{bmatrix} 2 \cdot c \\ 2 \cdot (\mathbf{s}' + 2^\nu \cdot \mathbf{h}) - \boldsymbol{\delta}' \end{bmatrix} \right\|_2^2 \\
&\leq \left\| \begin{bmatrix} 2 \cdot c \\ 2 \cdot \mathbf{s}' \\ 2^{\nu+1} \cdot \mathbf{h} \\ \boldsymbol{\delta}' \end{bmatrix} \right\|_2^2 + 8 \cdot \left| \left\langle \begin{bmatrix} c \\ \mathbf{s}' \end{bmatrix}, \begin{bmatrix} \mathbf{0}_\ell \\ 2^\nu \cdot \mathbf{h} \end{bmatrix} \right\rangle \right| + 8 \cdot \left| \left\langle \begin{bmatrix} c \\ \mathbf{s}' \end{bmatrix}, \begin{bmatrix} \mathbf{0}_\ell \\ \boldsymbol{\delta}' \end{bmatrix} \right\rangle \right| + 8 \cdot \left| \left\langle \begin{bmatrix} \mathbf{0}_\ell \\ 2^\nu \cdot \mathbf{h} \end{bmatrix}, \begin{bmatrix} \mathbf{0}_\ell \\ \boldsymbol{\delta}' \end{bmatrix} \right\rangle \right| \\
&\leq \left\| \begin{bmatrix} 2 \cdot c \\ 2 \cdot \mathbf{s}' \\ 2^{\nu+1} \cdot \mathbf{h} \\ \boldsymbol{\delta}' \end{bmatrix} \right\|_2^2 + 12 \cdot \left\| \begin{bmatrix} c \\ \mathbf{s}' \\ 2^\nu \cdot \mathbf{h} \\ \boldsymbol{\delta}' \end{bmatrix} \right\|_2^2 \leq 16 \cdot \beta^2 + 39 \cdot 2^{2 \cdot (\nu - 1)} \cdot nk.
\end{aligned}$$

where the second inequality follows from $\|\mathbf{a} + \mathbf{b}\|_2 \leq \sqrt{\|\mathbf{a}\|_2^2 + \|\mathbf{b}\|_2^2 + 2\langle \mathbf{a}, \mathbf{b} \rangle}$ for any vectors $\mathbf{a}, \mathbf{b}$, and the third inequality follows from the arithmetic–geometric mean inequality. Thus, we have $\|\mathbf{s}^*\|_2 \leq 4\beta + 2^{\nu+2} \cdot \sqrt{nk}$ as desired. We complete the proof by noticing that $\mathbf{s}^* \neq \mathbf{0}_{\ell+k}$ since $c \neq \bar{c}$. This completes the proof. □

C.2 Hardness of ExtMLWE

Here we discuss the asymptotic hardness of ExtMLWE. While there are some works establishing the hardness of ExtMLWE on MLWE [AA16, BJRW23], they do not cover our variant where the hints are given in the form of polynomial coefficients. Indeed, if we try to adapt their proofs, we incur a reduction loss of at least $2^{n\eta}$, where n is the lattice dimension and η is the number of hints. Ideally, we want the reduction loss to only scale with η. As handling the module case turns out non-trivial, we indirectly establish the asymptotic hardness of our ExtMLWE by showing that the non-structured variant reduces to MLWE with a polynomial reduction loss. The concrete hardness of ExtMLWE is analyzed in Section 4.3.7.

Formally, we have the following, which is an extension of the reduction by [AP12] to the multi-hint setting. We note that when $\mathcal{F}$ is distributed as a discrete Gaussian, we can extend the reduction by [BLP+13] to the multi-hint setting without incurring an exponential loss in η, in which case we can set $\eta = \omega_{\mathsf{asymp}}(1) \lessapprox n\ell$.

Lemma 9 (Hardness of ExtLWE). *Let $\eta = O(1)$, $B = \mathsf{poly}(\kappa)$, and $q = \mathsf{poly}(\kappa)$ a prime such that $B < (q-1)/4$. Let $\mathcal{D}$ and $\mathcal{F}$ be distributions over $\mathbb{Z}_q$ and $\mathbb{Z}_q^{\eta\times n(\ell+k)}$ such that we have $\Pr[\mathbf{v} \leftarrow \mathcal{D}^{n(\ell+k)}, \mathbf{M} \leftarrow \mathcal{F} : \|\mathbf{M}\cdot\mathbf{v}\|_\infty \leq B] \geq 1 - \mathsf{negl}(\kappa)$. Then, for any adversary $\mathcal{A}$ against the $\mathsf{ExtLWE}_{q,n\ell,nk,\mathcal{D},\mathcal{F}}$ problem, we can construct an adversary $\mathcal{B}$ against the $\mathsf{LWE}_{q,n\ell,nk,\mathcal{D}}$ problem such that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{ExtLWE}}(\kappa) \geq \frac{1}{(2B+1)^\eta}\cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{LWE}}(\kappa) - \mathsf{negl}(\kappa).$$

We also have $\mathsf{Time}(\mathcal{B}) \approx \mathsf{Time}(\mathcal{A})$.

Proof. Assume $\mathcal{B}$ receive $(\mathbf{A},\mathbf{b}) \in \mathbb{Z}_q^{nk\times n\ell}\times\mathbb{Z}_q^{nk}$ as the LWE instance. We describe how $\mathcal{B}$ simulates an ExtLWE instance to $\mathcal{A}$. $\mathcal{B}$ first samples $\mathbf{V} \leftarrow \mathbb{Z}_q^{nk\times\eta}$, $(\mathbf{s}^*,\mathbf{e}^*) \leftarrow \mathcal{D}^{n\ell}\times\mathcal{D}^{nk}$, $\mathbf{M} \leftarrow \mathcal{F}$, and sets $(\mathbf{M}_\mathbf{s},\mathbf{M}_\mathbf{e}) \in \mathbb{Z}_q^{\eta\times n\ell}\times\mathbb{Z}_q^{\eta\times nk}$ as the first $n\ell$ and last nk columns of $\mathbf{M}$, respectively. It then runs through $c \in [\eta+1]$ and finds $\mathbf{T}_c = \mathbf{I}_{nk} + c\cdot\mathbf{V}\mathbf{M}_\mathbf{e} \in \mathbb{Z}_q^{nk\times nk}$ such that $\det(\mathbf{T}_c) \neq 0$, where $\mathbf{I}_{nk}$ is an identity matrix of size nk. We show below that such c can always be found.

It sets such matrix $\mathbf{T}_c$ as $\mathbf{T}$ and further computes

$$\mathbf{A}' := \mathbf{T}\mathbf{A} - \mathbf{V}\mathbf{M}_\mathbf{s} \ \wedge\ \mathbf{b}' = \mathbf{T}\mathbf{b} - \mathbf{V}\mathbf{M}\begin{bmatrix}\mathbf{s}^*\\ \mathbf{e}^*\end{bmatrix}.$$

Finally, it provides $(\mathbf{A}',\mathbf{b}',\mathbf{M},\mathbf{M}\begin{bmatrix}\mathbf{s}^*\\ \mathbf{e}^*\end{bmatrix})$ to $\mathcal{A}$ as the ExtLWE instance. $\mathcal{B}$ then outputs whatever $\mathcal{A}$ outputs.

Let us analyze the advantage of $\mathcal{B}$. We first prove that there exists $c \in [\eta+1]$ such that $\det(\mathbf{T}_c) \neq 0$. Using the Weinstein–Aronszajn identity, we have

$$\det(\mathbf{T}_c) = \det(\mathbf{I}_{nk} + c\cdot\mathbf{V}\mathbf{M}_\mathbf{e}) = \det(\mathbf{I}_\eta + c\cdot\mathbf{M}_\mathbf{e}^\top\mathbf{V}^\top).$$

The right hand side is a polynomial of degree η in the variable c, meaning that it equals 0 mod q on at most η values of $c \in [\eta+1]$. Since q is a prime, there exists at least one invertible matrix such that $\det(\mathbf{T}_c) \neq 0$ over $\mathbb{Z}_q$. Therefore, when $\mathbf{A}$ is uniform, $\mathbf{A}'$ is uniform as desired regardless of $\mathbf{b}$ being random or not. Note that such c can be computed in polynomial time since computing the determinant of the matrix can be performed in polynomial time and we have $\eta = O(1)$. We next see what happens to $\mathbf{b}'$.

We first consider the case $\mathbf{b} \leftarrow \mathbb{Z}_q^{nk}$. In this case, following the same argument, since $\mathbf{b}$ is uniform, $\mathbf{b}^*$ is uniform. Hence, the ExtLWE instance given to $\mathcal{A}$ is a valid random MLWE instance.

We consider the other case $\mathbf{b} = \mathbf{As} + \mathbf{e} \in \mathbb{Z}_q^{nk}$ for $(\mathbf{s}, \mathbf{e}) \leftarrow \mathcal{D}^{n\ell} \times \mathcal{D}^{nk}$. In this case, we have

$$\begin{aligned}
\mathbf{b}' &= \mathbf{T}(\mathbf{As} + \mathbf{e}) - \mathbf{VM} \begin{bmatrix} \mathbf{s}^* \\ \mathbf{e}^* \end{bmatrix} \\
&= (\mathbf{A}' + \mathbf{VM_s}) \cdot \mathbf{s} + (\mathbf{I} + \mathbf{VM_e}) \cdot \mathbf{e} - \mathbf{VM} \begin{bmatrix} \mathbf{s}^* \\ \mathbf{e}^* \end{bmatrix} \\
&= \mathbf{A}'\mathbf{s} + \mathbf{e} + \mathbf{V} \underbrace{\mathbf{M} \begin{bmatrix} \mathbf{s} - \mathbf{s}^* \\ \mathbf{e} - \mathbf{e}^* \end{bmatrix}}_{=:\mathbf{d}}.
\end{aligned}$$

If $\mathbf{d} = \mathbf{0}$, then $\mathbf{b}' = \mathbf{A}'\mathbf{s} + \mathbf{e}$ and the instance given to $\mathcal{A}$ is distributed exactly as a valid MLWE instance. We first bound the probability that $\mathbf{d} = \mathbf{0}$. With an overwhelming choice of $\mathbf{M} \leftarrow \mathcal{F}$, we have the following over the random choice of $(\mathbf{s}, \mathbf{e}, \mathbf{s}^*, \mathbf{e}^*)$ due to our assumption in the statement:

$$\begin{aligned}
\Pr[\mathbf{d} = \mathbf{0}] &= \Pr\left[\mathbf{M} \begin{bmatrix} \mathbf{s} \\ \mathbf{e} \end{bmatrix} = \mathbf{M} \begin{bmatrix} \mathbf{s}^* \\ \mathbf{e}^* \end{bmatrix}\right] \\
&= \sum_{\mathbf{w} \in \mathbb{Z}_q^{\eta} : \|\mathbf{w}\|_\infty \le B} \Pr\left[\mathbf{M} \begin{bmatrix} \mathbf{s} \\ \mathbf{e} \end{bmatrix} = \mathbf{w}\right]^2 \ge \frac{1}{(2\beta + 1)^\eta} \cdot \sum_{\mathbf{w} \in \mathbb{Z}_q^{\eta} : \|\mathbf{w}\|_\infty \le B} \Pr\left[\mathbf{M} \begin{bmatrix} \mathbf{s} \\ \mathbf{e} \end{bmatrix} = \mathbf{w}\right],
\end{aligned}$$

where the second equality follows since the pair $(\mathbf{s}, \mathbf{e})$ and $(\mathbf{s}^*, \mathbf{e}^*)$ are identically and independently distributed, and the last inequality follows from Cauchy–Schwarz. Since the right hand side equals 1 with an overwhelming probability, we can bound $\Pr[\mathbf{d} = \mathbf{0}]$ by $\frac{1}{(2B+1)^\eta}$.

Otherwise, when $\mathbf{d} \neq \mathbf{0}$, assume without loss of generality that $m_1 \neq 0$, i.e., the first entry of $\mathbf{d}$ is non-zero. Then, we can rewrite $\mathbf{Vd} = \mathbf{v}_1 \cdot d_1 + \mathbf{V}_{\neq 1}\mathbf{d}_{\neq 1}$, where $\mathbf{v}_1$ is the first column of $\mathbf{V}$, and $\mathbf{V}_{\neq 1}$ and $\mathbf{d}_{\neq 1}$ are the matrix and vector by removing $\mathbf{v}_1$ and d_1 from $\mathbf{V}$ and $\mathbf{d}$, respectively. Due to the assumption in our statement, with an overwhelming probability, we have $\|\mathbf{d}\|_\infty \le 2B$. Combining this with $d_1 \neq 0$, $4B + 1 < q$, and q being a prime, $\mathbf{v}_1 \cdot d_1$ is distributed uniformly at random over $\mathbb{Z}_q^{nk}$. Since $\mathbf{v}_1$ is independent from $\mathbf{V}_{\neq 1}$, $\mathbf{Vd}$ is distributed uniformly at random as well. Hence, the instance given to $\mathcal{A}$ is distributed as a random MLWE instance.

Combining everything, in case $\mathcal{B}$ is given a random MLWE instance, then $\mathcal{A}$ is given a random ExtMLWE instance. In the other case, with probability at least $\frac{1}{(2B+1)^\eta} - \mathsf{negl}(\kappa)$, $\mathcal{A}$ is given a valid ExtMLWE instance, and otherwise a random ExtMLWE instance. This completes the proof. □

D Full Detail on Black-box Security Reduction

In this section, we provide all the missing details to establish EUF-CMA security of the Raccoon signature scheme.

D.1 Omitted Tools for Security Reduction

Min-entropy of MLWE for the sum of uniform distribution. In the security proofs, we use the $\mathsf{MLWE}_{q,\ell,k,\mathcal{D}}$ distribution with bit dropping: BD-MLWE, formally defined as follows.

Definition 8. *Let* BD-MLWE *be the* $\mathsf{MLWE}_{q,\ell,k,\mathcal{D}}$ *distribution with* ν_{w} *dropped bits. Namely, given* $\mathbf{A} \in \mathcal{R}_q^{k\times\ell}$, BD-MLWE *is defined as the ensemble* $\{\lfloor \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \rceil_{\nu_{\mathrm{w}}} \mid (\mathbf{s}, \mathbf{e}) \leftarrow \mathcal{D}^{\ell+k}\}$.

Conjecture 2. *For the parameters of our scheme in Section 4.1.3, i.e.,* $\mathsf{MLWE}_{q,\ell,k,\mathrm{SU}(u_t, d\cdot \mathsf{rep})}$, *we have*

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

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 $\mathbf{As} + \mathbf{e}$ is "well distributed", i.e. if we assume that the distributions $\lfloor \mathbf{As} + \mathbf{e} \rceil_{\nu_\mathrm{w}}$ and $\mathbf{As} + \mathbf{e} \bmod 2^{\nu_\mathrm{w}}$ are independent, then we have:

$$\begin{aligned} H_\infty(\mathbf{As} + \mathbf{e}) &= H_\infty(\lfloor \mathbf{As} + \mathbf{e} \rceil_{\nu_\mathrm{w}}, \mathbf{As} + \mathbf{e} \bmod 2^{\nu_\mathrm{w}}) \\ &\leq H_\infty(\mathsf{BD\text{-}MLWE}) + kn\nu_\mathrm{w} \end{aligned}$$

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

$$H_\infty(\mathsf{BD\text{-}MLWE}) \geq H_\infty(\mathcal{D}^{\ell+k}) - kn\nu_\mathrm{w}$$

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

$$H_\infty(\mathsf{BD\text{-}MLWE}) \geq (\ell + k)nu_\mathrm{w} - kn\nu_\mathrm{w}$$

For all parameters we consider we will have $H_\infty(\mathsf{BD\text{-}MLWE}) > 100\kappa$, 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.

In the following, we recall the worst-case to average-case reductions in the module lattice setting to support the confidence on MLWE and MSIS (or alternatively SelfTargetMSIS). We note that the Raccoon signature scheme relies on the hardness of MLWE with the sums of uniform distributions, not the discrete Gaussian distribution as in the lemma statement below. In our security proof, we plug in the standard deviation σ of the sum of uniform distribution (see Eq. (6)), in place of the standard deviation of the discrete Gaussian distribution to set the asymptotic parameters. A concrete security analysis of the lattice assumptions we use are provided in Section 4.3.

Lemma 10 (Hardness of MLWE ([LS15])). *Let $k(\kappa), \ell(\kappa), q(\kappa), n(\kappa), \sigma(\kappa)$ such that $q \leq \mathsf{poly}(\ell \cdot n)$, $k \leq \mathsf{poly}(\ell)$, and $\sigma \geq \sqrt{\ell} \cdot \omega_{\mathsf{asymp}}(\sqrt{\log n})$. If $\mathcal{D}$ is a discrete Gaussian distribution with standard deviation σ, then the $\mathsf{MLWE}_{q,\ell,k,\mathcal{D}}$ problem is as hard as the worst-case lattice Generalized-Independent-Vector-Problem (GIVP) in dimension $N = \ell n$ with approximation factor $\sqrt{8 \cdot N\ell} \cdot \omega_{\mathsf{asymp}}(\sqrt{\log \ell}) \cdot q/\sigma$.*

Lemma 11 (Hardness of MSIS ([LS15])). *For any $k(\kappa), \ell(\kappa), q(\kappa), n(\kappa), \beta(\kappa)$ such that $q > \beta\sqrt{\ell n} \cdot \omega_{\mathsf{asymp}}(\log(\ell n))$, $k \leq \mathsf{poly}(\ell)$, and $\log q \leq \mathsf{poly}(\ell n)$. The $\mathsf{MSIS}_{q,\ell,k,\beta}$ problem is as hard as the worst-case lattice Generalized-Independent-Vector-Problem (GIVP) in dimension $N = \ell n$ with approximation factor $\beta\sqrt{N} \cdot \omega_{\mathsf{asymp}}(\sqrt{\log N})$.*

Tail cut bounds. We provide some norm bounds regarding the sum of uniform distribution. In practice, Lemma 12 is not tight. Therefore, for our *concrete* security analysis, we will rely on sharper but heuristic bounds, see Section 4.3.6.

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 $\mathrm{SU}(u, T)$ 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. (4) with the union bound:

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

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

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

Lemma 13. *The smooth Rényi divergence satisfies the following properties.*

1. ***Data processing inequality.*** *Let P, Q be two distributions, let $\epsilon \geq 0$, and g be a randomized function over (a superset of) $\mathrm{Supp}(P) \cup \mathrm{Supp}(Q)$.*

$$R_\alpha^\epsilon(g(P); g(Q)) \leq R_\alpha^\epsilon(P; Q). \tag{51}$$

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

$$P(E) \leq (Q(E) + \epsilon)^{(\alpha-1)/\alpha} \cdot R_\alpha^\epsilon(P; Q) + \epsilon. \tag{52}$$

3. ***Tensorization.*** *Let $(P_i)_{i \in I}$, $(Q_i)_{i \in I}$ be two finite families of distributions, let $\epsilon_i \geq 0$ for $i \in I$, and let $\epsilon = \sum_{i \in I} \epsilon_i$.*

$$R_\alpha^\epsilon\left(\prod_{i \in I} P_i; \prod_{i \in I} Q_i\right) \leq \prod_{i \in I} R_\alpha^{\epsilon_i}(P_i; Q_i). \tag{53}$$

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 data processing inequalities of general 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. □

For the sum of uniform distribution, we have a nice symmetry of the Rényi divergence.

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

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 $P_{\mathrm{SU}}(x) = Q_{\mathrm{SU}}(C - x)$. □

D.2 Asymptotic Parameter Selection

Here, we provide a set of candidate asymptotic parameters for the Raccoon signature that are used in Theorem 1.

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

$$C = \{c \in \mathcal{R}_q \mid \|c\|_\infty = 1 \wedge \|c\|_1 = \omega\}. \tag{54}$$

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:

- $B^{\mathsf{sRD}}_{u_t,\infty}, B^{\mathsf{sRD}}_{u_t,2}$ bounds on the L_∞, L_2-norm, respectively, of $c \cdot (\mathbf{s}, \mathbf{e})$ for any $c \in C$ and $(\mathbf{s}, \mathbf{e}) \leftarrow \mathrm{SU}(u_t, T)^{n(k+\ell)}$,
- $B^{\mathsf{sRD}}_{u_w,\infty}, B^{\mathsf{sRD}}_{u_w,2}$ bounds on the L_∞, L_2-norm, respectively, of $(\mathbf{r}, \mathbf{e}') \leftarrow \mathrm{SU}(u_w, T)^{n(k+\ell)}$,
- $\beta = \sqrt{\omega} + B_2 + 2^{\nu_t - 1}\sqrt{\omega n k}$,
- α the order used in the smooth Rényi divergence,
- ϵ_{TAIL} the statistical component that will be used in the smooth Rényi divergence argument,
- $\varepsilon_{\mathsf{Adv}} = \mathsf{Adv}^{\mathsf{MLWE}}_{\mathcal{B}} + \mathsf{Adv}^{\mathsf{SelfTargetMSIS}}_{\mathcal{B}'} + \epsilon_{\mathsf{negl}}$, for Lemma 16 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:

- $\mathsf{Adv}^{\mathsf{MLWE}}_{\mathcal{B}} = \mathsf{negl}(\kappa)$. I.e. $\frac{T(2^{2u_t}-1)}{12} \geq \sqrt{\ell} \cdot \omega_{\mathsf{asymp}}(\sqrt{\log n})$, using Eq. (6) and Lemma 10.
- $\mathsf{Adv}^{\mathsf{SelfTargetMSIS}}_{\mathcal{B}'} = \mathsf{negl}(\kappa)$. I.e. $\beta' \sqrt{n(\ell+1)} \cdot \omega_{\mathsf{asymp}}(\sqrt{\log(n\ell)}) \leq q$ where $\beta' = 4\beta + 2^{\nu_w+1} \cdot \sqrt{nk}$, using Lemmas 8 and 11.
- $\alpha B^{\mathsf{sRD}}_{u_t,\infty} = o\left(\frac{2^{u_w}}{T-1}\right)$, $\alpha = \omega_{\mathsf{asymp}}(1)$ and $\epsilon_{\text{TAIL}} = \frac{(\alpha B^{\mathsf{sRD}}_{u_t,\infty}+T)^T}{2^{u_w \cdot T} T!} = \mathsf{negl}(\kappa)$. So that we can use the smooth Rényi divergence as per Lemma 1 and Conjecture 1.
- $\alpha = \frac{2^{u_w}}{B^{\mathsf{sRD}}_{u_t,2}} \sqrt{\frac{-\log(\varepsilon_{\mathsf{Adv}})T}{C_{\text{Rényi}} Q_s}}$, $\frac{B^{\mathsf{sRD}}_{u_t,2}}{2^{u_w}} \cdot \sqrt{\frac{-C_{\text{Rényi}} \cdot \log(\varepsilon_{\mathsf{Adv}}) \cdot Q_s}{T}} = O(\log \kappa)$, and $Q_s \cdot \epsilon_{\text{TAIL}} \leq \epsilon_{\mathsf{negl}}$. So we can use Lemma 16.
- $B^{\mathsf{sRD}}_{u_t,\infty} = \omega \cdot (\frac{T}{2} + \delta_{u_t})$, $B^{\mathsf{sRD}}_{u_t,2} = \sqrt{n(\ell+k)} \cdot B^{\mathsf{sRD}}_{u_t,\infty}$, 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.
- $B^{\mathsf{sRD}}_{u_w,\infty} = 2^{u_w} \cdot T$, and $B^{\mathsf{sRD}}_{u_w,2} = 2^{u_w}\sqrt{3T \cdot n(\ell+k)}$ for Eq. (5).
- $B_2 = B^{\mathsf{sRD}}_{u_w,\infty} + B^{\mathsf{sRD}}_{u_w,2} + 2^{\nu_w} \cdot \sqrt{nk}$ for overwhelming correctness. For this bound note that for an honest user $\mathbf{h} = c \cdot \mathbf{e} + \mathbf{e}' + \delta$ where δ is the sum of two rounding errors (hence $\|\delta\|_2 \leq 2^{\nu_w} \cdot \sqrt{nk}$.

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 $\varepsilon_{\mathsf{Adv}}$ used in the security proof. All other parameters are scheme specific and defined independently of $\mathcal{A}$.

- $n, \ell, k = \mathsf{poly}(\kappa)$ such that $n \geq \kappa$,
- $Q_h = \mathsf{poly}(\kappa)$: the maximum number of hash queries is any unbounded polynomial,
- $Q_s = \mathsf{poly}(\kappa)$: 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 $Q_s \geq \kappa$,
- $T = d \cdot \mathsf{rep} = \omega_{\mathsf{asymp}}(1)$, e.g., $T = \sqrt{\log \kappa}$,
- $\omega = \omega_{\mathsf{asymp}}(1)$, e.g., $\omega = \log \kappa$,
- $\epsilon_{\mathsf{negl}} = 2^{-\kappa}$,
- $\nu_{\mathrm{t}}, \nu_{\mathrm{w}} = O(\log \kappa)$. From which, we get $\beta, \beta' = \mathsf{poly}(\kappa)$, and set polynomially sized modulus q such that $\beta' \sqrt{n(\ell+1)} \cdot \omega_{\mathsf{asymp}}(\log(n\ell)) \leq q$,
- $2^{u_{\mathrm{t}}} = \sqrt[4]{\ell \cdot \log \kappa}$,
- $2^{u_{\mathrm{w}}} = B^{\mathsf{sRD}}_{u_{\mathrm{t}},2} \sqrt{\frac{C_{\text{Rényi}} Q_s}{T} \cdot \sqrt{\frac{\kappa}{\log \kappa}} \cdot n(\ell+k)}$. From which we get the condition on Lemma 16, as well as $\epsilon_{\text{TAIL}} = \mathsf{negl}(\kappa)$ since

$$\epsilon_{\text{TAIL}} \leq \left(\frac{\alpha \mathcal{B}^{\mathsf{sRD}}_{u_{\mathrm{t}},\infty}}{2^{u_{\mathrm{w}}}} + \frac{1}{\sqrt{\kappa}} \right)^T \leq \left(\sqrt{\frac{-\log(\epsilon_{\mathsf{Adv}}) \cdot T}{Q_s \cdot n(\ell+k)}} + \frac{1}{\sqrt{\kappa}} \right)^T \leq \left(\frac{2}{\sqrt[4]{\kappa}} \right)^{\sqrt{\log \kappa}} = \mathsf{negl}(\kappa).$$

Where the first inequality comes from $T \cdot \kappa^{1/2} \leq 2^{u_{\mathrm{w}}}$ and the second inequality comes from $\mathcal{B}^{\mathsf{sRD}}_{u_{\mathrm{t}},2} = \sqrt{n(\ell+k)} \cdot \mathcal{B}^{\mathsf{sRD}}_{u_{\mathrm{t}},\infty}$. The last fact comes from $Q_s \geq \kappa$, $T = \sqrt{\log \kappa}$, and the fact that we can assume $\varepsilon_{\mathsf{Adv}} \geq 2^{-\sqrt{\frac{\kappa}{\log \kappa}} \cdot n(\ell+k)}$ as any adversary against SelfTargetMSIS with $\beta = \mathsf{poly}(\kappa)$ can achieve better advantage than $\varepsilon_{\mathsf{Adv}}$ by random guessing.[9] Following a similar computation, $\frac{B^{\mathsf{sRD}}_{u_{\mathrm{t}},2}}{2^{u_{\mathrm{w}}}} \cdot \sqrt{\frac{-C_{\text{Rényi}} \cdot \log(\varepsilon_{\mathsf{Adv}}) \cdot Q_s}{T}} \leq 1$. Lastly, we have $Q_s \cdot \epsilon_{\text{TAIL}} \leq \epsilon_{\mathsf{negl}}$,

- Using how we set $2^{u_{\mathrm{w}}}$, $\alpha = \frac{2^{u_{\mathrm{w}}}}{B^{\mathsf{sRD}}_{u_{\mathrm{t}},2}} \sqrt{\frac{-\log(\varepsilon_{\mathsf{Adv}})T}{C_{\text{Rényi}} Q_s}} = \sqrt{-\log(\varepsilon_{\mathsf{Adv}})} \leq \sqrt[4]{\kappa} \cdot \sqrt{n(\ell+k)}$. From this we get $\alpha B^{\mathsf{sRD}}_{u_{\mathrm{t}},\infty} = o\left(\frac{2^{u_{\mathrm{w}}}}{T-1}\right)$. Moreover, assuming the hardness of MLWE and SelfTargetMSIS, we can bound $\varepsilon_{\mathsf{Adv}} \leq \kappa^{-1}$, which establishes $\alpha \geq \log(\kappa) = \omega_{\mathsf{asymp}}(1)$.

D.3 Omitted Security Reduction

Here we provide the full proof of Theorem 1. Refer to Appendix D.2 for the parameters used in the proof.

Proof. Let $\mathcal{A}$ be an adversary against the EUF-CMA security game. 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 to the SelfTargetMSIS problem. We relate the advantage of $\mathcal{A}$ for each adjacent hybrids.

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

Hybrid_0

1: $\mathsf{seed} \leftarrow \{0,1\}^{\kappa}$
2: $\mathbf{A} \leftarrow \mathsf{ExpandA}(\mathsf{seed})$
3: $\mathbf{s} \leftarrow \mathrm{SU}(u_t, T)^{n\ell}$
4: $\mathbf{e} \leftarrow \mathrm{SU}(u_t, T)^{nk}$
5: $\bar{\mathbf{t}} := \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$
6: $\mathbf{t} := \lfloor \bar{\mathbf{t}} \rceil_{\nu_t}$
7: $\mathsf{vk} := (\mathsf{seed}, \mathbf{t})$
8: $\mathcal{Q}_{\mathsf{Sign}} := \emptyset$
9: $(\mathsf{msg}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathsf{OSgn}()}(\mathsf{vk})$
10: **if** $\exists \mathsf{sig}'$ s.t. $(\mathsf{msg}^*, \mathsf{sig}') \in \mathcal{Q}_{\mathsf{Sign}}$ **return** FAIL
11: **return** $\mathsf{Verify}(\mathsf{sig}^*, \mathsf{msg}^*, \mathsf{vk})$

$\mathsf{OSgn}(\mathsf{msg})$

1: $\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk}) \| \mathsf{msg})$
2: $\mathbf{r} \leftarrow \mathrm{SU}(u_w, T)^{n\ell}$
3: $\mathbf{e}' \leftarrow \mathrm{SU}(u_w, T)^{nk}$
4: $\mathbf{w} := \lfloor \mathbf{A} \cdot \mathbf{r} + \mathbf{e}' \rceil_{\nu_w}$
5: $c_{\mathsf{poly}} := \mathsf{G}(\mathbf{w}, \mu)$
6: $\mathbf{z} := c_{\mathsf{poly}} \cdot \mathbf{s} + \mathbf{r}$
7: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - 2^{\nu_t} \cdot c_{\mathsf{poly}} \cdot \mathbf{t}$
8: $\mathbf{h} := \mathbf{w} - \lfloor \mathbf{y} \rceil_{\nu_w}$
9: $\mathsf{sig} := (c_{\mathsf{poly}}, \mathbf{h}, \mathbf{z})$
10: **if** $\mathsf{CheckBounds}(\mathsf{sig}) = \mathsf{FAIL}$ **goto** Line 2
11: $\mathcal{Q}_{\mathsf{Sign}} := \mathcal{Q}_{\mathsf{Sign}} \cup \{(\mathsf{msg}, \mathsf{sig})\}$
12: **return** sig

Figure 9: First hybrid game for the security proof. It corresponds to $\mathsf{Game}^{\mathsf{EUF\text{-}CMA}}$ described in Figure 8. We assume $\mathcal{A}$ is given access to the random oracles $(\mathsf{H}, \mathsf{G}, \mathsf{ExpandA})$.

Hybrid_0: This is the original EUF-CMA security game. In particular, since the adversary only has access to the output of the KeyGen and Sign algorithms, we can collapse lines 4 to 7 of Algorithm 1 by sampling $(\mathbf{s}, \mathbf{e}) \leftarrow \mathrm{SU}(u_t, T)^{n\ell} \times \mathrm{SU}(u_t, T)^{nk}$ and setting $\mathbf{t} := \lfloor \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \rceil_{\nu_t}$; in the following, we will denote by $\bar{\mathbf{t}}$ the value $\mathbf{A} \cdot \mathbf{s} + \mathbf{e}$. Note that throughout the proof, we implicitly view the n-dimensional vector output by $\mathrm{SU}(u_t, T)^n$ as an element over $\mathcal{R}_q$. Similarly we collapse lines 4 to 8 of Algorithm 2 by sampling $(\mathbf{r}, \mathbf{e}') \leftarrow \mathrm{SU}(u_w, T)^{n\ell} \times \mathrm{SU}(u_w, T)^{nk}$ and setting $\mathbf{w} := \lfloor \mathbf{A} \cdot \mathbf{r} + \mathbf{e}' \rceil_{\nu_w}$. For ease of reading, as stated in the preparation step, we use the hash function G that corresponds to $\mathsf{ChalPoly} \circ \mathsf{ChalHash}$ to sample $c_{\mathsf{poly}} := \mathsf{G}(\mathbf{w}, \mu)$.

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

Hybrid_1: In this hybrid, the challenger samples $\mathbf{A}$ uniformly at random from its target set $\mathcal{R}_q^{k \times \ell}$ and programs $\mathsf{ExpandA}(\mathsf{seed}) := \mathbf{A}$. 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

Hybrid_1

1: $\mathsf{seed} \leftarrow \{0,1\}^\kappa$
2: $\mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}$
3: $\mathsf{ExpandA}(\mathsf{seed}) := \mathbf{A}$
4: $\mathbf{s} \leftarrow \mathrm{SU}(u_\mathrm{t}, T)^{n\ell}$
5: $\mathbf{e} \leftarrow \mathrm{SU}(u_\mathrm{t}, T)^{nk}$
6: $\bar{\mathbf{t}} := \mathbf{A}\cdot\mathbf{s}+\mathbf{e}$
7: $\mathbf{t} := \lfloor \bar{\mathbf{t}} \rceil_{\nu_t}$
8: $\mathsf{vk} := (\mathsf{seed}, \mathbf{t})$
9: $Q_{\mathsf{Sign}} := \emptyset$
10: $(\mathsf{msg}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathsf{OSgn}()}(\mathsf{vk})$
11: **if** $\exists \mathsf{sig}'$ s.t. $(\mathsf{msg}^*, \mathsf{sig}') \in Q_{\mathsf{Sign}}$ **return** FAIL
12: **return** $\mathsf{Verify}(\mathsf{sig}^*, \mathsf{msg}^*, \mathsf{vk})$

Hybrid_2

1: $\mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}$
2: $\mathbf{s} \leftarrow \mathrm{SU}(u_\mathrm{t}, T)^{n\ell}$
3: $\mathbf{e} \leftarrow \mathrm{SU}(u_\mathrm{t}, T)^{nk}$
4: $\bar{\mathbf{t}} := \mathbf{A}\cdot\mathbf{s}+\mathbf{e}$
5: $\mathbf{t} := \lfloor \bar{\mathbf{t}} \rceil_{\nu_t}$
6: $\mathsf{vk} := (\mathbf{A}, \mathbf{t})$
7: $Q_{\mathsf{Sign}} := \emptyset$
8: $(\mathsf{msg}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathsf{OSgn}()}(\mathsf{vk})$
9: **if** $\exists(\mathsf{msg}, \cdot) \in Q_{\mathsf{Sign}} : \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg}^*) = \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg})$ **return** FAIL
10: **if** $\exists \mathsf{sig}'$ s.t. $(\mathsf{msg}^*, \mathsf{sig}') \in Q_{\mathsf{Sign}}$ **return** FAIL
11: **return** $\mathsf{Verify}(\mathsf{sig}^*, \mathsf{msg}^*, \mathsf{vk})$

Hybrid_3

$\mathsf{OSgn}(\mathsf{msg})$

1: $\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg})$
2: $\mathbf{r} \leftarrow \mathrm{SU}(u_\mathrm{w}, T)^{n\ell}$
3: $\mathbf{e}' \leftarrow \mathrm{SU}(u_\mathrm{w}, T)^{nk}$
4: $\mathbf{w} := \lfloor \mathbf{A}\cdot\mathbf{r}+\mathbf{e}' \rceil_{\nu_\mathrm{w}}$
5: $c_{\mathsf{poly}} \leftarrow \mathcal{C}$
6: $\mathbf{z} := c_{\mathsf{poly}}\cdot\mathbf{s}+\mathbf{r}$
7: $\mathbf{y} := \mathbf{A}\cdot\mathbf{z} - 2^{\nu_t}\cdot c_{\mathsf{poly}}\cdot\mathbf{t}$
8: $\mathbf{h} := \mathbf{w} - \lfloor \mathbf{y} \rceil_{\nu_\mathrm{w}}$
9: $\mathsf{G}(\mathbf{w}, \mu) := c_{\mathsf{poly}}$ ▷ Abort if already programmed
10: $\mathsf{sig} := (c_{\mathsf{poly}}, \mathbf{h}, \mathbf{z})$
11: **if** $\mathsf{CheckBounds}(\mathsf{sig}) = \mathsf{FAIL}$ **goto** Line 2
12: $Q_{\mathsf{Sign}} := Q_{\mathsf{Sign}} \cup \{(\mathsf{msg}, \mathsf{sig})\}$
13: **return** sig

Hybrid_4

$\mathsf{OSgn}(\mathsf{msg})$

1: $\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg})$
2: $\mathbf{r} \leftarrow \mathrm{SU}(u_\mathrm{w}, T)^{n\ell}$
3: $\mathbf{e}' \leftarrow \mathrm{SU}(u_\mathrm{w}, T)^{nk}$
4: $c_{\mathsf{poly}} \leftarrow \mathcal{C}$
5: $\mathbf{z} := c_{\mathsf{poly}}\cdot\mathbf{s}+\mathbf{r}$
6: $\mathbf{z}' := c_{\mathsf{poly}}\cdot\mathbf{e}+\mathbf{e}'$ ▷ Note $\mathbf{e} = \bar{\mathbf{t}} - \mathbf{A}\mathbf{s}$
7: $\mathbf{w} := \lfloor \mathbf{A}\cdot\mathbf{z} - c_{\mathsf{poly}}\cdot\bar{\mathbf{t}} + \mathbf{z}' \rceil_{\nu_\mathrm{w}}$
8: $\mathbf{y} := \mathbf{A}\cdot\mathbf{z} - 2^{\nu_t}\cdot c_{\mathsf{poly}}\cdot\mathbf{t}$
9: $\mathbf{h} := \mathbf{w} - \lfloor \mathbf{y} \rceil_{\nu_\mathrm{w}}$
10: $\mathsf{G}(\mathbf{w}, \mu) := c_{\mathsf{poly}}$ ▷ Abort if already programmed
11: $\mathsf{sig} := (c_{\mathsf{poly}}, \mathbf{h}, \mathbf{z})$
12: **if** $\mathsf{CheckBounds}(\mathsf{sig}) = \mathsf{FAIL}$ **goto** Line 2
13: $Q_{\mathsf{Sign}} := Q_{\mathsf{Sign}} \cup \{(\mathsf{msg}, \mathsf{sig})\}$
14: **return** sig

Figure 10: The hybrid games 1 to 4 used in the proof of Theorem 1. Differences from Hybrid_{i-1} to Hybrid_i are highlighted. We assume $\mathcal{A}$ is given access to the random oracles $(\mathsf{H}, \mathsf{G}, \mathsf{ExpandA})$.

Hybrid_5

$\mathsf{OSgn}(\mathsf{msg})$

1: $\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg})$
2: $\mathbf{z} \leftarrow \mathrm{SU}(u_{\mathrm{w}}, T)^{n\ell}$
3: $\mathbf{z}' \leftarrow \mathrm{SU}(u_{\mathrm{w}}, T)^{nk}$
4: $c_{\mathsf{poly}} \leftarrow C$
5: $\mathbf{w} := \lfloor \mathbf{A} \cdot \mathbf{z} - c_{\mathsf{poly}} \cdot \bar{\mathbf{t}} + \mathbf{z}' \rceil_{\nu_{\mathrm{w}}}$
6: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - 2^{\nu_t} \cdot c_{\mathsf{poly}} \cdot \mathbf{t}$
7: $\mathbf{h} := \mathbf{w} - \lfloor \mathbf{y} \rceil_{\nu_{\mathrm{w}}}$
8: $\mathsf{G}(\mathbf{w}, \mu) := c_{\mathsf{poly}}$ ▷ Abort if already programmed
9: $\mathsf{sig} := (c_{\mathsf{poly}}, \mathbf{h}, \mathbf{z})$
10: **if** $\mathsf{CheckBounds}(\mathsf{sig}) = \mathsf{FAIL}$ **goto** Line 2
11: $Q_{\mathsf{Sign}} := Q_{\mathsf{Sign}} \cup \{(\mathsf{msg}, \mathsf{sig})\}$
12: **return** sig

Hybrid_6

$\mathsf{seed} \leftarrow \{0,1\}^{\kappa}$
$\mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}$
$\mathsf{ExpandA}(\mathsf{seed}) := \mathbf{A}$
$\bar{\mathbf{t}} \leftarrow \mathcal{R}_q^{k}$
$\mathbf{t} := \lfloor \bar{\mathbf{t}} \rceil_{\nu_t}$
$\mathsf{vk} := (\mathsf{seed}, \mathbf{t})$
$Q_{\mathsf{Sign}} := \emptyset$
$(\mathsf{msg}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathsf{OSgn}()}(\mathsf{vk})$
if $\exists(\mathsf{msg}, \cdot) \in Q_{\mathsf{Sign}} : \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg}^*) = \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg})$ **return** FAIL
if $\exists\mathsf{sig}'$ s.t. $(\mathsf{msg}^*, \mathsf{sig}') \in Q_{\mathsf{Sign}}$ **return** FAIL
return $\mathsf{Verify}(\mathsf{sig}^*, \mathsf{msg}^*, \mathsf{vk})$

Hybrid_7

$\mathsf{seed} \leftarrow \{0,1\}^{\kappa}$
$\mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}$
$\mathsf{ExpandA}(\mathsf{seed}) := \mathbf{A}$
$\bar{\mathbf{t}} \leftarrow \mathcal{R}_q^{k}$
$\mathbf{t} := \lfloor \bar{\mathbf{t}} \rceil_{\nu_t}$
$\mathsf{vk} := (\mathsf{seed}, \mathbf{t})$
$Q_{\mathsf{Sign}} := \emptyset$
$L_{\mathsf{SimT}} := \emptyset$
$(\mathsf{msg}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathsf{OSgn}()}(\mathsf{vk})$
if $\exists(\mathsf{msg}, \cdot) \in Q_{\mathsf{Sign}} : \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg}^*) = \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg})$ **return** FAIL
if $\exists\mathsf{sig}'$ s.t. $(\mathsf{msg}^*, \mathsf{sig}') \in Q_{\mathsf{Sign}}$ **return** FAIL
return $\mathsf{Verify}(\mathsf{sig}^*, \mathsf{msg}^*, \mathsf{vk})$

$\mathsf{OSgn}(\mathsf{msg})$

1: $\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg})$
2: $\mathbf{z} \leftarrow \mathrm{SU}(u_{\mathrm{w}}, T)^{n\ell}$
3: $\mathbf{z}' \leftarrow \mathrm{SU}(u_{\mathrm{w}}, T)^{nk}$
4: $c_{\mathsf{poly}} \leftarrow C$
5: $\mathbf{w} := \lfloor \mathbf{A} \cdot \mathbf{z} - c_{\mathsf{poly}} \cdot \bar{\mathbf{t}} + \mathbf{z}' \rceil_{\nu_{\mathrm{w}}}$
6: $\mathbf{y} := \mathbf{A} \cdot \mathbf{z} - 2^{\nu_t} \cdot c_{\mathsf{poly}} \cdot \mathbf{t}$
7: $\mathbf{h} := \mathbf{w} - \lfloor \mathbf{y} \rceil_{\nu_{\mathrm{w}}}$
8: $L_{\mathsf{SimT}} \leftarrow L_{\mathsf{SimT}} \cup \{(\mathbf{w}, \mu, c_{\mathsf{poly}})\}$ ▷ Abort if $\exists c'_{\mathsf{poly}} \in C \cup \{\bot\}$ s.t. $(\mathbf{w}, \mu, c'_{\mathsf{poly}}) \in L_{\mathsf{SimT}}$
9: $\mathsf{sig} := (c_{\mathsf{poly}}, \mathbf{h}, \mathbf{z})$
10: **if** $\mathsf{CheckBounds}(\mathsf{sig}) = \mathsf{FAIL}$ **goto** Line 2
11: $Q_{\mathsf{Sign}} := Q_{\mathsf{Sign}} \cup \{(\mathsf{msg}, \mathsf{sig})\}$
12: **return** sig

$\mathsf{G}(\mathbf{w}, \mathsf{msg})$

$\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg})$
if $\exists c_{\mathsf{poly}}$ s.t. $(\mathbf{w}, \mu, c_{\mathsf{poly}}) \in L_{\mathsf{SimT}}$
  return c_{poly}
$L_{\mathsf{SimT}} \leftarrow L_{\mathsf{SimT}} \cup \{(\mathbf{w}, \mu, \bot)\}$
return $\mathsf{G}'(\mathbf{w}, \mathsf{msg})$

Figure 11: Last three Hybrid games for the proof of Theorem 1. The differences between Hybrid_{i-1} and Hybrid_i are highlighted. Note that in Hybrid_6, the signing oracle $\mathsf{OSgn}(\mathsf{msg})$ 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 $(\mathsf{H}, \mathsf{G}, \mathsf{ExpandA})$.

oracle fail is bounded by $Q_h \cdot 2^{-\kappa}$. Thus, we have

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

Hybrid_2: In this hybrid, the challenger adds a winning condition. Namely, when the adversary produces a forgery on a message msg that provokes a collision in $\mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg}^\star)$ for a message $\mathsf{msg}^\star$ previously queried to the signing oracle, the challenger does not view this as a valid forgery. Since $\mathsf{H} : \{0,1\}^* \longrightarrow \{0,1\}^{2\kappa}$ is modeled as a random oracle, this event happens with probability at most $Q_s \cdot 2^{-2\kappa}$:

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

While $\mathsf{EUF}\text{-}\mathsf{CMA}$ security is guaranteed by setting the hash output length as κ, we chose 2κ for additional security properties. See Section 4.4 for more detail.

Hybrid_3: 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 C. Then it programs the hash function to consistently return this value c_{poly} on input $(\mu, \mathbf{w})$ during further interactions with the adversary.

Note that the signing responses in Hybrid_2 are identically distributed to Hybrid_1 unless $\mathsf{OSgn}(\cdot)$ is required to program a value that has already been queried by the adversary. As $\mathbf{w}$ is sampled randomly following the $\mathsf{BD}\text{-}\mathsf{MLWE}$ distribution as in Definition 8, this happens with probability at most $Q_h \cdot 2^{-H_\infty(\mathsf{BD}\text{-}\mathsf{MLWE})}$ in each signing query. Thus it follows that

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

where we have used Bernoulli's inequality and $Q_h < 2^{H_\infty(\mathsf{BD}\text{-}\mathsf{MLWE})}$ from Conjecture 2. For completeness, recall that Bernoulli's inequality implies $(1+x)^r \geq 1 + rx$ for every integer $r \geq 0$ and real number $x > -1$.

Hybrid_4: In this hybrid, the challenger computes the commitment $\mathbf{w}$ using the public key before rounding $\bar{\mathbf{t}}$ instead of an ephemeral LWE sample $\mathbf{A} \cdot \mathbf{r} + \mathbf{e}'$. As the challenger computes $\mathbf{w} = \lfloor \mathbf{A} \cdot \mathbf{r} + \mathbf{e}' \rceil_{\nu_\mathbf{w}} = \lfloor \mathbf{A} \cdot \mathbf{z} - c_{\mathsf{poly}} \cdot \mathbf{A} \cdot \mathbf{s} + \mathbf{e}' \rceil_{\nu_\mathbf{w}}$ in the previous game, one can verify that

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

which yields the equation in Hybrid_3.

As it is simply a rewriting of $\mathbf{w}$, it remains indistinguishable from Hybrid_3:

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

Hybrid_5: In this hybrid, the challenger computes the response $(\mathbf{z}, \mathbf{z}')$ without using the secret key $\mathbf{s}$ or the noise $\mathbf{e}$. 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 $\mathcal{Q}(\mathsf{center})$ defined in Section 4.1.3. Let $\mathcal{P}$ be the distribution $\mathrm{SU}(u_{\mathrm{w}}, T)^{n(\ell+k)}$ and $\mathcal{Q}(\mathsf{center}_{\mathsf{q_s}})$ be the distribution $\mathsf{center}_{\mathsf{q_s}} + \mathcal{P}$, where $c_{\mathsf{q_s}} \leftarrow C$ is the $\mathsf{q_s}$-th ($\mathsf{q_s} \in [Q_s]$) challenge used to respond to the signing oracle OSgn and $\mathsf{center}_{\mathsf{q_s}} := c_{\mathsf{q_s}} \cdot \begin{bmatrix} \mathbf{s} \\ \mathbf{e} \end{bmatrix} \in \mathcal{R}_q^{\ell+k}$. Define $\mathcal{P}^* := \mathcal{P}^{Q_s}$ and let $\mathcal{Q}^*_{\mathsf{centers}}$ be the tensored distribution $\otimes_{\mathsf{q_s} \in [Q_s]} \mathcal{Q}(\mathsf{center}_{\mathsf{q_s}})$. Then, $\mathcal{P}^*$ and $\mathcal{Q}^*_{\mathsf{centers}}$ correspond to the distributions of $(\mathbf{z}, \mathbf{z}')$ in Hybrid_5 and Hybrid_4, respectively. Lastly, let $\epsilon_{\mathrm{TAIL},\mathsf{centers}} := \sum_{\mathsf{q_s} \in [Q_s]} \epsilon_{\mathrm{TAIL}}(\mathsf{center}_{\mathsf{q_s}})$ where recall Section 4.1.3 for the definition of $\epsilon_{\mathrm{TAIL}}(\mathsf{center}_{\mathsf{q_s}})$.

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 13, we have the following with overwhelming probability:

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

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

Hybrid_6: In this hybrid, the verification key $\mathsf{vk} = (\mathbf{A}, \lfloor \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \rceil_{\nu_t})$ is replaced with $(\mathbf{A}, \lfloor \bar{\mathbf{t}} \rceil_{\nu_t})$ where $\bar{\mathbf{t}}$ 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 $\mathsf{MLWE}_{q,\ell,k,\mathrm{SU}(u_t,T)}$ problem:

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

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

Hybrid_7: 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, $(\mathbf{w}, \mu, \perp) \in L_{\mathsf{SimT}}$ 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

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

We show in Lemma 15 that there exists an adversary $\mathcal{B}'$ solving the $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,C,\nu_{\mathrm{w}},\beta}$ problem such that

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

Before providing the proof of Lemma 15, we finish the proof of Theorem 1.

Collecting the bounds, we obtain

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

where $Q_s \cdot \epsilon_{\text{Tail}} \leq \epsilon_{\mathsf{negl}} = \mathsf{negl}(\kappa)$ due to our parameter selection in Appendix D.2. Relying on Conjecture 1, we can bound $(R_{\alpha}^{\epsilon_{\text{Tail}}}(\mathcal{P}; \mathcal{Q}))^{Q_s} \leq \exp\left(\frac{C_{\text{Rényi}} \cdot Q_s \cdot \alpha \cdot (B_{u_t,2}^{\mathsf{sRD}})^2}{T \cdot 2^{2 \cdot u_w}}\right)$. Hence, plugging in our choice of α, i.e., $\alpha = \frac{2^{u_w}}{B_{u_t,2}^{\mathsf{sRD}}} \cdot \sqrt{-\frac{\log(\varepsilon_{\mathsf{Adv}}) \cdot T}{C_{\text{Rényi}} \cdot Q_s}}$ with $\varepsilon_{\mathsf{Adv}} = \mathsf{Adv}_{\mathcal{B}}^{\mathsf{MLWE}} + \mathsf{Adv}_{\mathcal{B}'}^{\mathsf{SelfTargetMSIS}} + \epsilon_{\mathsf{negl}}$, we obtain

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

We finally show in Lemma 16 that $\Lambda = \mathsf{negl}(\kappa)$, assuming the hardness of the MLWE and SelfTargetMSIS assumptions. This completes the proof of Theorem 1. □

It remains to prove the following two Lemmas 15 and 16.

Lemma 15. *There exists an adversary $\mathcal{B}'$ solving the* $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,C,\nu_w,\beta}$ *problem with*

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

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

Proof. Let $\mathcal{A}$ be an adversary against the EUF-CMA security game in Hybrid_7. 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 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 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 11). Furthermore, $\mathcal{B}'$ sets $-\bar{\mathbf{t}} \in \mathcal{R}_q^k$ 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 $\bar{\mathbf{t}}$ are distributed uniformly in their respective sets. At the end of the game, the adversary $\mathcal{A}$ outputs a forgery $(c_{\mathsf{poly}}^*, \mathbf{h}^*, \mathbf{z}^*)$ for a message msg^*. $\mathcal{B}'$ sets $\mu^* = \mathsf{H}(\mathsf{H}(\mathsf{vk}) \| \mathsf{msg}^*)$, $\mathbf{s}_1 := \mathbf{z}^* \in \mathcal{R}_q^{\ell}$, $\mathbf{s}_2 := c_{\mathsf{poly}}^* \cdot (\bar{\mathbf{t}} - 2^{\nu_t} \cdot \lfloor \bar{\mathbf{t}} \rceil_{\nu_t}) \in \mathcal{R}_q^k$, and $\mathbf{s} = \begin{bmatrix} c_{\mathsf{poly}}^* \\ \mathbf{s}_1 \\ \mathbf{s}_2 \end{bmatrix} \in \mathcal{R}_q^{\ell+k+1}$. It then outputs $(\mu^*, \mathbf{s}, \mathbf{h}^*)$ as the solution to the SelfTargetMSIS problem.

Let us analyze the success probability of $\mathcal{B}'$. Conditioning on $\mathcal{A}'$ breaking EUF-CMA security, no $(\mathbf{w}', c'_{\mathsf{poly}})$ such that $c'_{\mathsf{poly}} \neq \perp$ and $(\mathbf{w}', \mu^*, c'_{\mathsf{poly}}) \in L_{\mathsf{SimT}}$ exists due to the modification we made in Hybrid_2. Since the forgery is valid, this implies $c_{\mathsf{poly}}^* = \mathsf{G}'\left(\left\lfloor \mathbf{A} \cdot \mathbf{z}^* - 2^{\nu_t} \cdot c_{\mathsf{poly}}^* \cdot \lfloor \bar{\mathbf{t}} \rceil_{\nu_t} \right\rceil_{\nu_w} + \mathbf{h}^*, \mu^*\right)$

and $\|(\mathbf{z}^*, 2^{\nu_\mathrm{w}} \cdot \mathbf{h}^*)\|_2 \leq B_2$. Now, notice that

$$[\mathbf{M} \mid \mathbf{I}] \cdot \mathbf{s} = [\mathbf{M} \mid \mathbf{I}] \cdot \begin{bmatrix} c^*_{\mathsf{poly}} \\ \mathbf{s}_1 \\ \mathbf{s}_2 \end{bmatrix} = -c^*_{\mathsf{poly}} \cdot \bar{\mathbf{t}} + \mathbf{A} \cdot \mathbf{s}_1 + \mathbf{s}_2 = \mathbf{A} \cdot \mathbf{z}^* - 2^{\nu_\mathrm{t}} \cdot c^*_{\mathsf{poly}} \cdot \lfloor \bar{\mathbf{t}} \rceil_{\nu_\mathrm{t}} .$$

In particular, $c^*_{\mathsf{poly}} = \mathsf{G}'\left(\lfloor [\mathbf{M} \mid \mathbf{I}] \cdot \mathbf{s} \rceil_{\nu_\mathrm{w}} + \mathbf{h}^*, \mu^*\right)$. Finally, we have

$$\begin{aligned} \|\mathbf{s}\|_2 &= \|(c_{\mathsf{poly}}, \mathbf{s}_1, \mathbf{s}_2, 2^{\nu_\mathrm{w}} \cdot \mathbf{h})\|_2 \\ &\leq \|c^*_{\mathsf{poly}}\|_2 + \|(\mathbf{z}, 2^{\nu_\mathrm{w}} \cdot \mathbf{h}^*)\|_2 + \|c^*_{\mathsf{poly}}\|_2 \cdot \|\bar{\mathbf{t}} - 2^{\nu_\mathrm{t}} \cdot \lfloor \bar{\mathbf{t}} \rceil_{\nu_\mathrm{t}} \|_2 \\ &\leq \sqrt{\omega} + B_2 + 2^{\nu_\mathrm{t}-1}\sqrt{\omega \cdot n \cdot k} \\ &= \beta, \end{aligned}$$

where we use the fact that $\|\bar{\mathbf{t}} - 2^{\nu_\mathrm{t}} \cdot \lfloor \bar{\mathbf{t}} \rceil_{\nu_\mathrm{t}} \|_\infty \leq 2^{\nu_\mathrm{t}-1}$ by the definition of the $\lfloor \cdot \rceil_{\nu_\mathrm{t}}$ function. Since $\mathbf{s} \neq \mathbf{0}$ as c^*_{poly} has ω non-zero coefficients, we conclude that $(\mu^*, \mathbf{s}, \mathbf{h}^*)$ is a valid solution for the $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,C,\nu_\mathrm{w},\beta}$ problem. It is clear that $\mathsf{Time}(\mathcal{B}') \approx \mathsf{Time}(\mathcal{A}')$. This completes the proof. □

Lemma 16. *Under the assumption that* $\mathsf{MLWE}_{q,\ell,k,\mathrm{SU}(u_\mathrm{t},T)}$ *and* $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,C,\nu_\mathrm{w},\beta}$ *are hard, we have the following according to our parameter selection in Appendix D.2:*

$$\Lambda = \varepsilon_{\mathsf{Adv}} \cdot \exp\left(\frac{2 \cdot B^{\mathsf{sRD}}_{u_\mathrm{t},2}}{2^{u_\mathrm{w}}} \sqrt{\frac{-C_{\textit{Rényi}} \cdot \log(\varepsilon_{\mathsf{Adv}}) \cdot Q_s}{T}} \right) = \mathsf{negl}(\kappa).$$

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

D.4 Discussion on Strong EUF-CMA Security.

In some applications, having strong EUF-CMA (sEUF-CMA) security may be better. We show that the Raccoon signature scheme is sEUF-CMA secure if we add one more condition on the parameters to those in Appendix D.2 and further rely on the $\mathsf{MSIS}_{q,\ell,k,\beta''}$ assumption as defined as follows:

- $\beta'' = 2\sqrt{2} \cdot (B_2 + 2^{\nu_\mathrm{w}-1} \cdot nk)$
- $\mathsf{Adv}^{\mathsf{MSIS}}_{\mathcal{B}''} = \mathsf{negl}(\kappa)$. I.e. $\beta''\sqrt{n\ell} \cdot \omega_{\mathsf{asymp}}(\sqrt{\log(n\ell)}) \leq q$, using Lemma 11.

We can use the same asymptotic parameters in Appendix D.2, where we may slightly enlarge q to satisfy the above additional constraint.

Formally, the following establishes the sEUF-CMA security of the Raccoon signature scheme.

Theorem 2. *The Raccoon signature scheme described in Section 2 is* sEUF-CMA *secure under the* $\mathsf{MLWE}_{q,\ell,k,\mathrm{SU}(u_\mathrm{w},d\cdot\mathsf{rep})}$, $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,C,\nu_\mathrm{w},\beta}$, *and* $\mathsf{MSIS}_{q,\ell,k,\beta''}$ *assumptions.*

Formally, for any adversary $\mathcal{A}$ against the sEUF-CMA *security game making at most Q_h random oracle queries and Q_s signing queries, and $\epsilon_{\textit{Tail}}$ and $R^{\epsilon_{\textit{Tail}}}_\alpha(\mathcal{P};\mathcal{Q})$ satisfying Eqs. (15) and (16), there exists adversaries $\mathcal{B}$, $\mathcal{B}'$, $\mathcal{B}''$ against the* $\mathsf{MLWE}_{q,\ell,k,\mathrm{SU}(u_\mathrm{w},d\cdot\mathsf{rep})}$, $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,C,\nu_\mathrm{w},\beta}$, *and* $\mathsf{MSIS}_{q,\ell,k,\beta''}$ *problems such that*

$$\begin{aligned} \mathsf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathcal{A}} \leq\ & 2^{-\kappa} \cdot Q_h \cdot (1 + 2^{-\kappa+1} \cdot Q_s) + Q_s \cdot \epsilon_{\textit{Tail}} \\ & + \left(\mathsf{Adv}^{\mathsf{MLWE}}_{\mathcal{B}} + \mathsf{Adv}^{\mathsf{SelfTargetMSIS}}_{\mathcal{B}'} + \mathsf{Adv}^{\mathsf{MSIS}}_{\mathcal{B}''} + Q_s \cdot \epsilon_{\textit{Tail}}\right)^{\frac{\alpha-1}{\alpha}} \cdot \left(R^{\epsilon_{\textit{Tail}}}_\alpha(\mathcal{P};\mathcal{Q})\right)^{Q_s}, \end{aligned} \tag{55}$$

where $\mathsf{Time}(\mathcal{A}) \approx \mathsf{Time}(\mathcal{B}) \approx \mathsf{Time}(\mathcal{B}') \approx \mathsf{Time}(\mathcal{B}'')$ *and we can assume* $\mathsf{Time}(\mathcal{A}) > O(Q_h + Q_s)$. *Concretely, plugging in our candidate asymptotic parameters in Appendix D.2, we conclude* $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{sEUF\text{-}CMA}}$ *is bounded by* $\mathsf{negl}(\kappa)$.

Proof. The security proof mostly follows the same hybrids as for the proof of EUF-CMA security in Appendix D.3. In particular, following the same hybrid argument, we arrive at an identical Hybrid_7, modulo the difference in the winning condition. For completeness, we provide Hybrid_7 in Figure 12.

Hybrid_8 : In this hybrid, the challenger adds a winning condition. Namely, when the adversary outputs a forger $(\mathsf{msg}^*, \mathsf{sig}^*)$, it recovers the $(\mu^*, \mathbf{y}^*, \mathbf{w}^*)$ as it would be done by the verification algorithm and checks if there exists $(\mathbf{w}, \mu, c_{\mathsf{poly}}) \in L_{\mathsf{SimT}}$ with $c_{\mathsf{poly}} \neq \perp$ such that $(\mathbf{w}^*, \mu^*) \neq (\mathbf{w}, \mu)$. If not, the challenger does not count the forgery to be valid. Notice if an adversary outputs a forgery that triggers this condition, then it means that $(\mathbf{w}^*, \mu^*)$ was never generated by the challenger as there exists no $c_{\mathsf{poly}} \in C$ such that $(\mathbf{w}^*, \mu^*, c_{\mathsf{poly}}) \in L_{\mathsf{SimT}}$. Put differently, an adversary that can trigger this condition satisfies winning condition of EUF-CMA security, which we established in Appendix D.3. Specifically, using the exact same argument, we can show that there exists an adversary $\mathcal{B}'$ solving the $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,C,\nu_{\mathrm{w}},\beta}$ problem such that

$$\left|\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_8} - \mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_7}\right| \leq \mathsf{Adv}_{\mathcal{B}'}^{\mathsf{SelfTargetMSIS}}.$$

It remains to bound the advantage of an adversary against the sEUF-CMA security game in Hybrid_8. We show in Lemma 15 that there exists an adversary $\mathcal{B}''$ solving the $\mathsf{MSIS}_{q,\ell,k,\beta''}$ problem such that

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_8} \leq \mathsf{Adv}_{\mathcal{B}''}^{\mathsf{MSIS}}.$$

Collecting the bounds, we obtain the inequality in the problem statement. Moreover, adding the $\mathsf{MSIS}_{q,\ell,k,\beta''}$ assumption to the set of assumptions made in Appendix D.3, we conclude that $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_0} = \mathsf{negl}(\kappa)$ under the same set of asymptotic parameters. It remains to prove the following Lemma 17. □

Lemma 17. *There exists an adversary* $\mathcal{B}''$ *solving the* $\mathsf{MSIS}_{q,\ell,k,\beta''}$ *problem with*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hybrid}_8} \leq \mathsf{Adv}_{\mathcal{B}''}^{\mathsf{MSIS}}.$$

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

Proof. Let $\mathcal{A}$ be an adversary against the sEUF-CMA security game in Hybrid_8. We construct an adversary $\mathcal{B}''$ solving the MSIS problem having the same advantage as $\mathcal{A}$. Assume $\mathcal{B}''$ is given $\mathbf{A} \in \mathcal{R}_q^{k\times\ell}$ as the MSIS problem. $\mathcal{B}''$ simply simulates the challenger in Hybrid_8, which it can perfectly perform. At the end of the game, the adversary $\mathcal{A}$ outputs a forgery $(c^*_{\mathsf{poly}}, \mathbf{h}^*, \mathbf{z}^*)$ for a message msg^*. Let $(\mu^*, \mathbf{y}^*, \mathbf{w}^*)$ be the corresponding values computed in Figure 12. Due to the condition we add in Hybrid_8, conditioning on the forgery being valid, there exists some signature $\mathsf{sig} = (c_{\mathsf{poly}}, \mathbf{h}, \mathbf{z})$ on message msg signed by $\mathcal{B}''$ such that $(\mathbf{w}, \mu) = (\mathbf{w}^*, \mu^*)$, where $(\mu, \mathbf{y}, \mathbf{w})$ are defined similarly to above. $\mathcal{B}''$ retrieves such $\mathsf{sig} = (c_{\mathsf{poly}}, \mathbf{h}, \mathbf{z})$. It then computes $\mathbf{d} = \left(\mathbf{A}\cdot\mathbf{z} - 2^{\nu_t}\cdot c_{\mathsf{poly}}\cdot\lfloor\bar{\mathbf{t}}\rceil_{\nu_t}\right) - 2^{\nu_{\mathrm{w}}}\cdot\left\lfloor\mathbf{A}\cdot\mathbf{z} - 2^{\nu_t}\cdot c_{\mathsf{poly}}\cdot\lfloor\bar{\mathbf{t}}\rceil_{\nu_t}\right\rceil_{\nu_{\mathrm{w}}} \in \mathcal{R}_q^k$ such that $\|\mathbf{d}\|_\infty \leq 2^{\nu_{\mathrm{w}}-1}$ and similarly for $\mathbf{d}^*$. Finally, it sets $\mathbf{s}_1 := \mathbf{z} - \mathbf{z}^* \in \mathcal{R}_q^\ell$, $\mathbf{s}_2 := 2^{\nu_{\mathrm{w}}}\cdot(\mathbf{h} - \mathbf{h}^*) - (\mathbf{d} - \mathbf{d}^*)$, $\mathbf{s} = \begin{bmatrix}\mathbf{s}_1\\ \mathbf{s}_2\end{bmatrix} \in \mathcal{R}_q^{\ell+k}$,

Hybrid_7

$\mathsf{seed} \leftarrow \{0,1\}^{\kappa}$
$\mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}$
$\mathsf{ExpandA}(\mathsf{seed}) := \mathbf{A}$
$\bar{\mathbf{t}} \leftarrow \mathcal{R}_q^{k}$
$\mathbf{t} := \lfloor \bar{\mathbf{t}} \rceil_{\nu_t}$
$\mathsf{vk} := (\mathsf{seed}, \mathbf{t})$
$Q_{\mathsf{Sign}} := \emptyset$
$L_{\mathsf{SimT}} := \emptyset$
$(\mathsf{msg}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathsf{OSgn}()}(\mathsf{vk})$
if $\exists(\mathsf{msg}, \cdot) \in Q_{\mathsf{Sign}} : \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg}^*) = \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg})$ **return** FAIL
if $(\mathsf{msg}^*, \mathsf{sig}^*) \in Q_{\mathsf{Sign}}$ **return** FAIL
return $\mathsf{Verify}(\mathsf{sig}^*, \mathsf{msg}^*, \mathsf{vk})$

$\mathsf{OSgn}(\mathsf{msg})$

1: $\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg})$
2: $\mathbf{z} \leftarrow \mathsf{SU}(u_{\mathbf{w}}, T)^{n\ell}$
3: $\mathbf{z}' \leftarrow \mathsf{SU}(u_{\mathbf{w}}, T)^{nk}$
4: $c_{\mathsf{poly}} \leftarrow C$
5: $\mathbf{w} := \lfloor \mathbf{A}\cdot\mathbf{z} - c_{\mathsf{poly}}\cdot\bar{\mathbf{t}} + \mathbf{z}' \rceil_{\nu_{\mathbf{w}}}$
6: $\mathbf{y} := \mathbf{A}\cdot\mathbf{z} - 2^{\nu_t}\cdot c_{\mathsf{poly}}\cdot\mathbf{t}$
7: $\mathbf{h} := \mathbf{w} - \lfloor \mathbf{y} \rceil_{\nu_{\mathbf{w}}}$
8: $L_{\mathsf{SimT}} \leftarrow L_{\mathsf{SimT}} \cup \{(\mathbf{w}, \mu, c_{\mathsf{poly}})\}$ ▷ Abort if $\exists c'_{\mathsf{poly}} \in C \cup \{\perp\}$ s.t. $(\mathbf{w}, \mu, c'_{\mathsf{poly}}) \in L_{\mathsf{SimT}}$
9: $\mathsf{sig} := (c_{\mathsf{poly}}, \mathbf{h}, \mathbf{z})$
10: **if** $\mathsf{CheckBounds}(\mathsf{sig}) = \mathsf{FAIL}$ **goto** Line 2
11: $Q_{\mathsf{Sign}} := Q_{\mathsf{Sign}} \cup \{(\mathsf{msg}, \mathsf{sig})\}$
12: **return** sig

Hybrid_8

$\mathsf{seed} \leftarrow \{0,1\}^{\kappa}$
$\mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}$
$\mathsf{ExpandA}(\mathsf{seed}) := \mathbf{A}$
$\bar{\mathbf{t}} \leftarrow \mathcal{R}_q^{k}$
$\mathbf{t} := \lfloor \bar{\mathbf{t}} \rceil_{\nu_t}$
$\mathsf{vk} := (\mathsf{seed}, \mathbf{t})$
$Q_{\mathsf{Sign}} := \emptyset$
$L_{\mathsf{SimT}} := \emptyset$
$(\mathsf{msg}^*, \mathsf{sig}^*) \leftarrow \mathcal{A}^{\mathsf{OSgn}()}(\mathsf{vk})$
if $\exists(\mathsf{msg}, \cdot) \in Q_{\mathsf{Sign}} : \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg}^*) = \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg})$ **return** FAIL
if $(\mathsf{msg}^*, \mathsf{sig}^*) \in Q_{\mathsf{Sign}}$ **return** FAIL
$(c^*_{\mathsf{poly}}, \mathbf{h}^*\mathbf{z}^*) := \mathsf{sig}^*$
$\mu^* := \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg}^*)$
$\mathbf{y}^* := \mathbf{A}\cdot\mathbf{z}^* - 2^{\nu_t}\cdot c^*_{\mathsf{poly}}\cdot\mathbf{t}$
$\mathbf{w}^* := \lfloor \mathbf{y}^* \rceil_{\nu_{\mathbf{w}}} + \mathbf{h}^*$
if $\forall(\mathbf{w}, \mu, c_{\mathsf{poly}}) \in L_{\mathsf{SimT}}$ s.t. $c_{\mathsf{poly}} \neq \perp$, $(\mathbf{w}^*, \mu^*) \neq (\mathbf{w}, \mu)$ **return** FAIL
return $\mathsf{Verify}(\mathsf{sig}^*, \mathsf{msg}^*, \mathsf{vk})$

$\mathsf{G}(\mathbf{w}, \mathsf{msg})$

$\mu := \mathsf{H}(\mathsf{H}(\mathsf{vk})\|\mathsf{msg})$
if $\exists c_{\mathsf{poly}}$ s.t. $(\mathbf{w}, \mu, c_{\mathsf{poly}}) \in L_{\mathsf{SimT}}$ **return** c_{poly}
$L_{\mathsf{SimT}} \leftarrow L_{\mathsf{SimT}} \cup \{(\mathbf{w}, \mu, \perp)\}$
return $\mathsf{G}'(\mathbf{w}, \mathsf{msg})$

Figure 12: Hybrid_7 and Hybrid_8 for the proof of Theorem 2. Hybrid_7 is exactly the same as those defined in the proof of Theorem 1 modulo the part highlighted. Both hybrids share the same signing oracle and random oracle G description. The highlighted part in Hybrid_8 illustrates the difference between Hybrid_7. We assume $\mathcal{A}$ is given access to the random oracles (H, G, ExpandA).

and outputs $\mathbf{s}$ as the solution to the MSIS problem.

Let us analyze the success probability of $\mathcal{B}''$. Conditioning on $\mathcal{A}'$ breaking EUF-CMA security, we have $\mathsf{sig} \neq \mathsf{sig}^*$, implying $(c_{\mathsf{poly}}, \mathbf{h}, \mathbf{z}) \neq (c^*_{\mathsf{poly}}, \mathbf{h}^*, \mathbf{z}^*)$. On the other hand, due to the winning condition we add in Hybrid_8, we have $(\mathbf{w}, \mu) = (\mathbf{w}^*, \mu^*)$. Since the forgery is valid, we have $c_{\mathsf{poly}} = \mathsf{G}(\mathbf{w}, \mu) = \mathsf{G}(\mathbf{w}^*, \mu^*) = c^*_{\mathsf{poly}}$. Combining the two, we establish $(\mathbf{h}, \mathbf{z}) \neq (\mathbf{h}^*, \mathbf{z}^*)$. If we have $\mathbf{z} \neq \mathbf{z}^*$, then $\mathbf{s}_1 \neq \mathbf{0}$. Otherwise, if $\mathbf{h} \neq \mathbf{h}^*$, we first observe that $\mathbf{d} = \mathbf{d}^*$ as $(c_{\mathsf{poly}}, \mathbf{z}) = (c^*_{\mathsf{poly}}, \mathbf{z}^*)$. This implies that $\mathbf{s}_2 \neq \mathbf{0}$. In either case, we establish that $\mathbf{s} \neq \mathbf{0}$.

Next, notice that from $\mathbf{w} = \mathbf{w}^*$, we have

$$\left\lfloor \mathbf{A} \cdot \mathbf{z} - 2^{\nu_t} \cdot c_{\mathsf{poly}} \cdot \lfloor \bar{\mathbf{t}} \rceil_{\nu_t} \right\rceil_{\nu_w} + \mathbf{h} = \left\lfloor \mathbf{A} \cdot \mathbf{z}^* - 2^{\nu_t} \cdot c^*_{\mathsf{poly}} \cdot \lfloor \bar{\mathbf{t}} \rceil_{\nu_t} \right\rceil_{\nu_w} + \mathbf{h}^*.$$

Multiplying both side by 2^{ν_w} and plugging in $(\mathbf{d}, \mathbf{d}^*)$, we have

$$\left(\mathbf{A} \cdot \mathbf{z} - 2^{\nu_t} \cdot c_{\mathsf{poly}} \cdot \lfloor \bar{\mathbf{t}} \rceil_{\nu_t}\right) + 2^{\nu_w} \cdot \mathbf{h} - \mathbf{d} = \left(\mathbf{A} \cdot \mathbf{z}^* - 2^{\nu_t} \cdot c^*_{\mathsf{poly}} \cdot \lfloor \bar{\mathbf{t}} \rceil_{\nu_t}\right) + 2^{\nu_w} \cdot \mathbf{h}^* - \mathbf{d}^*.$$

Using the fact that $c_{\mathsf{poly}} = c^*_{\mathsf{poly}}$, we can rewrite the equation as

$$\mathbf{A} \cdot \underbrace{(\mathbf{z} - \mathbf{z}^*)}_{=\mathbf{s}_1} + \underbrace{2^{\nu_w} \cdot (\mathbf{h} - \mathbf{h}^*) - (\mathbf{d} - \mathbf{d}^*)}_{=\mathbf{s}_2} = \mathbf{0},$$

which in particular implies $\mathbf{A} \cdot \mathbf{s} = \mathbf{0}$ as desired. Lastly, we have

$$\begin{aligned}
\|\mathbf{s}\|_2^2 &= \|(\mathbf{s}_1, \mathbf{s}_2)\|_2^2 \\
&\leq 4 \cdot \left\| \begin{bmatrix} \mathbf{z} \\ 2^{\nu_w} \cdot \mathbf{h} \\ \mathbf{d} \end{bmatrix} \right\|_2^2 + 8 \cdot \left| \left\langle \begin{bmatrix} \mathbf{z} \\ 2^{\nu_w} \cdot \mathbf{h} \end{bmatrix}, \begin{bmatrix} \mathbf{0}_\ell \\ \mathbf{d} \end{bmatrix} \right\rangle \right| \\
&\leq 8 \cdot \left\| \begin{bmatrix} \mathbf{z} \\ 2^{\nu_w} \cdot \mathbf{h} \\ \mathbf{d} \end{bmatrix} \right\|_2^2 \\
&\leq 8 \cdot (B_2^2 + 2^{2(\nu_w - 1)} \cdot nk),
\end{aligned}$$

where the second inequality follows from $\|\mathbf{a} + \mathbf{b}\|_2 \leq \sqrt{\|\mathbf{a}\|_2^2 + \|\mathbf{b}\|_2^2 + 2\langle \mathbf{a}, \mathbf{b} \rangle}$ for any vectors $\mathbf{a}, \mathbf{b}$, and the third inequality follows from the arithmetic–geometric mean inequality. Therefore, $\|\mathbf{s}\|_2 \leq \beta'' = 2\sqrt{2} \cdot (B_2 + 2^{\nu_w - 1} \cdot nk)$ and it is a valid solution for the $\mathsf{MSIS}_{q,\ell,k,\beta''}$ problem. It is clear that $\mathsf{Time}(\mathcal{B}'') \approx \mathsf{Time}(\mathcal{A}')$. This completes the proof. □

E NIST requirements

This section maps the requirements from NIST's call for additional digital signature schemes [NIS22] to the corresponding parts of this document and of the submission.

2.A Cover Sheet

The cover sheet following [NIS22, §2.A] is included in this submission.

2.B.1

- The formal specification according to [NIS22, §2.B.1] is in Section 2;
- The design rationale according to [NIS22, §2.B.1] is in Section 1.2; various design choices are also discussed throughout Section 2;
- Besides the security level, Raccoon admits a single tunable parameter: the number of shares d. An analysis of how this impacts security and performances is provided in Section 4, in particular Section 4.2;
- The provenance of constants is provided throughout Section 2. For example, all domain separation headers are constructed as in Section 2.4.3, NTT-related constants are explained in Section 2.7, and norm bounds are explained in Section 2.6.2.

2.B.2

Raccoon's estimated computational efficiency and memory requirements for the NIST PQC Reference Platform are provided in Section 3. In order to facilitate reproducibility, the platform and settings used for benchmarking are described in Section 3.2.

2.B.3

Known Answer Test (KAT) values are provided as part of this submission. In addition, Section 2.9 provides SHA-256 hashes of the KAT files and explains how they were generated.

2.B.4

A security analysis is conducted in Section 4. First, a black-box security reduction to well-understood assumptions is conducted in Section 4.1. Then, the impact of probing on the security of Raccoon is discussed in Section 4.2. Finally, based on the two previous sections and on the state-of-the-art, Section 4.3 discusses the security of concrete parameter sets.

2.B.5

Known cryptanalytic attacks are discussed in Section 4.3.

2.B.6

Advantages and limitations are listed in Section 1.3.

2.B.6

Intellectual property statements are attached in separate documents as part of this submission.