Threshold Raccoon

Rafael del Pino[1], Thomas Espitau[1], Shuichi Katsumata[1,2], Mary Maller[1,3], Fabrice Mouhartem[4], Thomas Prest[1], Markku-Juhani Saarinen[1,5], and Kaoru Takemure[2]

[1] PQShield
[2] AIST
[3] Ethereum Foundation
[4] XWiki
[5] Tampere University

Abstract Threshold signatures improve both availability and security of digital signatures by splitting the signing key into N shares handed out to different parties. Later on, any subset of at least T parties can cooperate to produce a signature on a given message. While threshold signatures have been extensively studied in the pre-quantum setting, they remain sparse from quantum-resilient assumptions.

In this work, we show that the Raccoon signature scheme [dEK+23] can be easily thresholdized. More precisely, we present Threshold Raccoon, a threshold signature that is very close to Raccoon. Our scheme has signature size 13 KiB and communication cost 40 KiB per user, supporting a threshold size as large as 1024 signers.

All operations used during signing are due to symmetric primitives and simple lattice operations; in particular our scheme does not need heavy tools such as threshold fully homomorphic encryption or homomorphic trapdoor commitments as in prior constructions. Our key technical idea is to use *one-time additive masks* to mitigate the leakage of the partial signing keys through partial signatures.

1 Introduction

A threshold signature scheme [Des90, DF90] is a specific type of multiparty computation that aims at issuing digital signatures, for which any subset of T parties among N signers are able to sign a message, but $(T-1)$ cannot. This ability to distribute trust among several parties has sparked widespread interest from the blockchain ecosystem.

In their September 2022 call for additional post-quantum signatures [NIS22, Section 4.D.1], NIST listed "additional functionalities" such as threshold threshold signatures to be desirable features. In January 2023, NIST released a draft call for multi-party threshold schemes [PB23]. Quantum resistance is repeatedly listed as an important criterion [PB23, Sections 3.2 and 3.3], with a deadline for submissions expected for 2024-2025. These two documents suggest that signature schemes that are both post-quantum signatures and thresholdizable would be of great interest for NIST.

While there exist several pre-quantum threshold signatures, building efficient post-quantum threshold signatures seems to be a much more challenging task. Some solutions have been proposed, but few have been implemented, and those who have suffer from major inefficiencies, such as large signatures, slow signing times, and sometimes both. In particular, no signature scheme submitted to the 2017 and 2023 NIST PQC calls for standardization have been shown to be efficiently thresholdizable until now.

1.1 Our Contributions

We propose Threshold Raccoon: a practical three-round lattice-based threshold signature assuming the hardness of the MLWE and MSIS problems. As its name indicates, it can be

viewed as a thresholdized version of the masking-friendly signature Raccoon [dPEK+23]. To distinguish both schemes clearly, we may refer to the scheme from [dPEK+23] as Masked Raccoon.

We recall in Section 2.1 a blueprint for a (standard) lattice-based signature called Lyubashesky's signature without abort [ASY22]. Since it underlies both Masked Raccoon and Threshold Raccoon, we will also refer to it as Vanilla Raccoon. Masked Raccoon applies several algorithmic tweaks to Vanilla Raccoon in order to make it masking-friendly. Since Vanilla Raccoon can be seen as a lattice-based variant of Schnorr signatures, it is a natural idea to combine one of the many existing constructions of Schnorr-based threshold signature [Bol03, LJY14, KG20, BCK+22, CKM+23, Lin22, RRJ+22] with Vanilla Raccoon in order to obtain a lattice-based threshold signature.

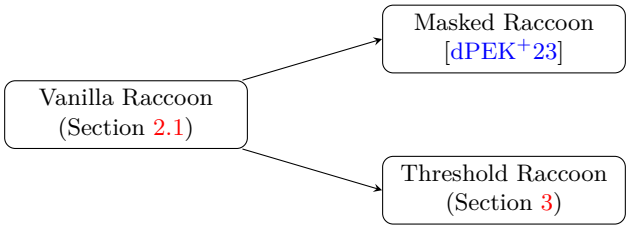

Figure 1: The Raccoon family of signature schemes.

However, as it is well-known in the lattice community, a naive translation does not work since, unlike in the classical setting, the signing key and signatures must satisfy additional size constraints. Indeed, the folklore construction ported to the lattice setting would leak too much information about the (distributed) signing keys and lead to practical attacks. The key technical ingredient we use to mitigate this leakage is the use of pairwise *one-time additive masks* that are *non-interactively* shared between each pair of users at each signing procedure and recombined in a way that allows individual users to hide their response while preserving correctness. More details are provided in Section 2.

2 Our Techniques

Schnorr's signature scheme has been a successful tool to construct threshold signature schemes in the classical setting. Our goal is to replicate this in the post-quantum setting building on (a variant of) Lyubashevsky's signature scheme [Lyu09, Lyu12], a lattice-based signature scheme based on the Fiat-Shamir transform.

2.1 Blueprint: Lyubashevsky's Signature *Without* Abort

We first recall Lyubashevsky's signature scheme without abort [ASY22]. For simplicity, this description purposedly ignores size optimizations such as bit dropping [BG14], which are mostly orthogonal to our security arguments.

Let $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n+1)$ be a polynomial ring. Let $\mathbf{A} \in \mathcal{R}_q^{k\times\ell}$ and $\mathbf{t} = \mathbf{A}\cdot\mathbf{s}+\mathbf{e}$, where $(\mathbf{s},\mathbf{e}) \in \mathcal{R}_q^{\ell}\times\mathcal{R}_q^{k}$ are "short" vectors. The verification key $\mathsf{vk} = (\mathbf{A},\mathbf{t})$ is a search-MLWE

(A.1) Sample ephemeral short randomness $(\mathbf{r}, \mathbf{e}')$ from an appropriate distribution and compute a *commitment* $\mathbf{w} = \mathbf{A} \cdot \mathbf{r} + \mathbf{e}'$.
(A.2) Generate a *challenge* $c \in \mathcal{R}_q$ using a hash function H_c as $c \leftarrow \mathsf{H}_c(\mathsf{vk}, \mathsf{msg}, \mathbf{w})$, where c has small coefficients.
(A.3) Compute a *response* $(\mathbf{z}, \mathbf{y}) = (c \cdot \mathbf{s} + \mathbf{r}, c \cdot \mathbf{e} + \mathbf{e}')$.
(A.4) To verify, check that $(\mathbf{z}, \mathbf{y})$ satisfies some size constraint and that $c = \mathsf{H}_c(\mathsf{vk}, \mathsf{msg}, \mathbf{A} \cdot \mathbf{z} + \mathbf{y} - c \cdot \mathbf{t})$.

Figure 2: Lyubashevsky's signature without abort, alias Vanilla Raccoon

instance, whereas the signing key $\mathsf{sk} = (\mathbf{s}, \mathbf{e})$ is the solution to this MLWE instance. To sign a message msg, the signature scheme proceeds as in Fig. 2.

As mentioned earlier, Fig. 2 can be seen as a transposition of Schnorr's signature to lattices. It is also similar to Masked Raccoon [dPEK+23], this is especially apparent when setting $d = 1$ in [dPEK+23] (1 share corresponding to the unmasked case).

Absence of rejection sampling. This description does not involve the so-called *rejection sampling* step [Lyu12] step, which would entail enforcing interval constraints on $(\mathbf{z}, \mathbf{y})$. Rejection sampling is difficult to mask, so Masked Raccoon made the design choice to remove it. This requires increasing parameters in order to preserve security.

Rejection sampling is also very challenging to perform in a distributed manner, i.e. to thresholdize. This is also listed as a reason by recent lattice-based threshold signatures [ASY22, GKS23] to remove it.

2.2 Naive Thresholdiszation

Due to the similarity between Schnorr's signature scheme and the scheme in Section 2.1, we can try to apply the common approach used in the classical setting [Sho00, KY02, Bol03] to build threshold signatures starting from Schnorr. We first recall Shamir's secret sharing.

Shamir secret sharing. Given $s \in \mathbb{Z}_q$ with q prime, we secret-share s by (a) generating a polynomial $P \in \mathbb{Z}_q[x]$ uniformly at random in the affine space of polynomials in $\mathbb{Z}_q[x]$ of degree at most $T-1$ such that $P(0) = s$, (b) for each user $i \in \mathbb{Z}_q^*$, their share of the secret is $s_i = P(i)$. If s is shared among N users, this is called T-out-of-N (Shamir) secret sharing. Given any set $(s_i)_{i \in I}$ of T distinct shares, s can be recovered by Lagrange interpolation:

$$s = \sum_{i \in I} \lambda_i \cdot s_i, \qquad \text{where} \qquad \lambda_i = \prod_{j \in I \setminus \{i\}} \frac{-j}{i-j}. \tag{1}$$

Shamir secret-sharing can be extended to $\mathbb{Z}_q$ for non-prime q by restricting users indices to exceptional sets [ABCP23], then to rings $\mathcal{R}_q = \mathbb{Z}_q[x]/(f(x))$ and finally to $\mathcal{R}_q$-modules by coefficient-wise application.

First attempt: direct transposition. We first secret share Raccoon's signing key $(\mathbf{s}, \mathbf{e})$ of Section 2.1 using Shamir's secret sharing scheme [Sha79]. Namely, $\mathbf{s}$ is encoded as the constant term of a degree $T-1$ polynomial P, and the *partial* signing key of user $i \in [N]$ is defined as the evaluation $\mathbf{s}_i = P(i) \in \mathcal{R}_q^\ell$ along with a *freshly sampled* short vector $\mathbf{e}_i$. Each users' partial signing key (implicitly) defines a partial public key $\mathbf{t}_i = \mathbf{A} \cdot \mathbf{s}_i + \mathbf{e}_i$, which is an MLWE instance. Here, note that given any T partial signing keys $(\mathbf{s}_i, \mathbf{e}_i)_{i \in \mathsf{act}}$, where $\mathsf{act} \subset [N]$ and $|\mathsf{act}| = T$, we can use the Lagrange coefficients $(\lambda_{\mathsf{act},i})_{i \in \mathsf{act}}$ to recompute the signing key as:

$$\mathbf{s} = \sum_{i \in \mathsf{act}} \lambda_{\mathsf{act},i} \cdot \mathbf{s}_i. \tag{2}$$

For any set act of T signers, the distributed signing protocol proceeds as described in Fig. 3. a routine calculation using Eq. (2) shows that the signature is valid.

(B.1) User $i \in \mathsf{act}$ computes $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i$. To protect against rushing adversaries (see [BN06]) it initially only outputs a *hash commitment* $\mathsf{H}_{\mathsf{com}}(\mathbf{w}_i)$.

(B.2) After obtaining the hash commitment from all users in act, user i reveals $\mathbf{w}_i$ and checks the correctness of all other reveals.

(B.3) User i collects all the commitments and *locally* generates a challenge $c \leftarrow \mathsf{H}_c(\mathsf{vk}, \mathsf{msg}, \mathbf{w})$, where $\mathbf{w} = \sum_{j \in \mathsf{act}} \mathbf{w}_j$.

(B.4) User i computes a response $(\mathbf{z}_i, \mathbf{y}_i) = (c \cdot \lambda_{\mathsf{act},i} \cdot \mathbf{s}_i + \mathbf{r}_i, c \cdot \mathbf{e}_i + \mathbf{e}'_i)$ and outputs $(\mathbf{z}_i, \mathbf{y}_i)$ as its *partial* signature.

(B.5) The final signature is $(c, \mathbf{z}, \mathbf{y}) = (c, \sum_{j \in \mathsf{act}} \mathbf{z}_j, \sum_{j \in \mathsf{act}} \mathbf{y}_j)$, verified as in Raccoon by checking the equality $c = \mathsf{H}_c(\mathsf{vk}, \mathsf{msg}, \mathbf{A} \cdot \mathbf{z} + \mathbf{y} - c \cdot \mathbf{t})$.

Figure 3: Naive and insecure threshold signature

Difficulty of Handling Lagrange Coefficients. While correct, the above construction admits an attack. This stems from the fact that Lagrange coefficients are large and can be chosen adaptively by the adversary.

In more detail, looking at Item (B.4) carefully, we can alternatively view user i as generating a signature with a signing key $(\lambda_{\mathsf{act},i} \cdot \mathbf{s}_i, \mathbf{e}_i)$. Importantly, $\mathbf{s}_i$ is scaled by the Lagrange coefficient $\lambda_{\mathsf{act},i}$. Since the user i provides a valid signature — a partial signature of the threshold scheme — this allows the adversary to obtain information on the corresponding scaled partial public key $\mathbf{t}_{\mathsf{act},i} = \lambda_{\mathsf{act},i} \cdot \mathbf{A} \cdot \mathbf{s}_i + \mathbf{e}_i$.

The adversary can adaptively ask user i to sign on a scaled public key of its choice by specifying a different signer set $\mathsf{act} \subset [N]$. By collecting enough $\mathbf{t}_{\mathsf{act},i}$ with specifically crafted Lagrange coefficients $\lambda_{\mathsf{act},i}$, the partial signing key $\mathbf{s}_i$ can be recovered via simple linear algebra. In the classical setting where the noise vector $\mathbf{e}_i$ does not exist, the above attack does not apply since all the obtained scaled partial public keys are linearly dependent.

This phenomenon is not new to our work. Lattice-based cryptography has always had a hard time handling Lagrange coefficients, see for example [ABV+12, BLMR13, BGG+18]. This

has led some works to rely on an alternative secret sharing scheme know as the $\{0,1\}$-linear (or $\{-1,0,1\}$-linear) secret sharing scheme, see e.g., [LST18, BGG+18, DLN+21, ASY22, CSS+22, CCK23]. While this gets around the issue with Lagrange coefficients, the downside is that the reconstruction algorithm becomes much more complex and individual shares grow by at least $O(N^{\sqrt{2}})$ [LST18]. Alternatively, we can blow up the modulus size q to scale with $O(N!^2)$ to argue that large Lagrange coefficients become relatively small to q [ABV+12, BGG+18, CCK23]. However, it is clear that such an approach leads to impractical parameters.

2.3 Our Solution: Masking the Commitments

We sidestep all these prior hurdles by using a very simple idea, exploiting the fact that threshold signatures are *interactive*. In more detail, assume for now that every two pairs of users $i, j \in \mathsf{act}$ privately share two *one-time random masks* $(\mathbf{m}_{\mathsf{act},i,j}, \mathbf{m}_{\mathsf{act},j,i}) \in (\mathcal{R}_q^\ell)^2$. We modify the signing protocol of the naive threshold signature scheme as in Fig. 4.

(C.1) User $i \in \mathsf{act}$ computes a commitment $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i$, a (public) *row* mask $\mathbf{m}_{\mathsf{act},i} = \sum_{j \in \mathsf{act}} \mathbf{m}_{\mathsf{act},i,j}$, and outputs $(\mathsf{H}_{\mathsf{com}}(\mathbf{w}_i), \mathbf{m}_{\mathsf{act},i})$.[5]

(C.2) After obtaining the hash commitments and row masks from all users in act, user i reveals $\mathbf{w}_i$.

(C.3) User i collects all the commitments and locally generate a challenge $c := \mathsf{H}_c(\mathsf{vk}, \mathsf{msg}, \mathbf{w})$, where $\mathbf{w} = \sum_{i \in \mathsf{act}} \mathbf{w}_i$.

(C.4) User i computes a (private) *column* mask $\mathbf{m}^*_{\mathsf{act},i} = \sum_{j \in \mathsf{act}} \mathbf{m}_{\mathsf{act},j,i}$ and response $(\mathbf{z}_i, \mathbf{y}_i) = (c \cdot \lambda_{\mathsf{act},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{m}^*_{\mathsf{act},i}, c \cdot \mathbf{e}_i + \mathbf{e}'_i)$, and outputs $(\mathbf{z}_i, \mathbf{y}_i)$ as its partial signature.

(C.5) The final signature is $(c, \mathbf{z}, \mathbf{y}) = (c, \sum_{j \in \mathsf{act}} (\mathbf{z}_j - \mathbf{m}_{\mathsf{act},j}), \sum_{j \in \mathsf{act}} \mathbf{y}_j)$ and is verified as in Item (A.4).

Figure 4: Construction with masked commitments but non-authenticated views.

Notice the sum of the row masks and column masks are equal: $\sum_{j \in \mathsf{act}} \mathbf{m}_{j,\mathsf{act}} = \sum_{j \in \mathsf{act}} \mathbf{m}^*_{j,\mathsf{act}}$. When all the users are honest, it can be checked that the aggregated response becomes:

$$\begin{aligned}
\mathbf{z} &= \sum_{j \in \mathsf{act}} (\mathbf{z}_j - \mathbf{m}_{\mathsf{act},j}) \\
&= \sum_{j \in \mathsf{act}} (c \cdot \lambda_{\mathsf{act},j} \cdot \mathbf{s}_j + \mathbf{r}_j + (\mathbf{m}^*_{\mathsf{act},j} - \mathbf{m}_{\mathsf{act},j})) \\
&= c \cdot \mathbf{s} + \sum_{j \in \mathsf{act}} \mathbf{r}_j
\end{aligned}$$

This gives a Raccoon signature as desired (see Item (A.3)).

[5] See Fig. 5 for why we call it row and column masks.

Intuition of the Security Proof. A typical security proof of a Lyubashevsky signature consists of invoking honest-verifier zero-knowledge of the (implicit) underlying identification protocol and programming the random oracle. At a high level, the reduction first samples a challenge c and response $(\mathbf{z}, \mathbf{y})$ distributed independently from the signing key, and simulates the commitment $\mathbf{w} = \mathbf{A}\cdot\mathbf{z}+\mathbf{y}-c\cdot\mathbf{t}$ (see Item (A.4)). Informally, if the commitment randomness $(\mathbf{r}, \mathbf{e}')$ are sufficiently larger than the scaled signing key $(c\cdot\mathbf{s}, c\cdot\mathbf{e})$, such a reduction remains indistinguishable from the real world. It is worth highlighting that $(\mathbf{r}, \mathbf{e}')$ cannot be *too* large since the response $(\mathbf{z}, \mathbf{y})$ must be "short", unlike in the classical Schnorr signature. Finally, it programs the random oracle as $\mathsf{H}_c(\mathsf{vk}, \mathsf{msg}, \mathbf{w}) \coloneqq c$.

Let us consider porting this proof to the threshold setting. To illustrate the effect of our masking idea, we explain what happens without them. Without the mask, user i outputs a partial signature $(\mathbf{z}_i, \mathbf{y}_i) = (c\cdot\lambda_{\mathsf{act},i}\cdot\mathbf{s}_i+\mathbf{r}_i, c\cdot\mathbf{e}+\mathbf{e}'_i)$. To perform the above proof strategy, the reduction must sample the response $(\mathbf{z}_i, \mathbf{y}_i)$ and simulate the commitment as $\mathbf{w}_i = \mathbf{A}\cdot\mathbf{z}_i + \mathbf{y}_i - c\cdot\mathbf{t}_{\mathsf{act},i}$ without the partial signing key $\mathbf{s}_i$, where $\mathbf{t}_{\mathsf{act},i} = \mathbf{A}\mathbf{s}_i + \mathbf{e}_i$ is the (implicit) partial public key. However, notice the above proof strategy falls apart since the scaled partial signing key $c\cdot\lambda_{\mathsf{act},i}\cdot\mathbf{s}_i$ is not guaranteed to be small compared to the commitment randomness $\mathbf{r}_i$ as the Lagrange coefficients $\lambda_{\mathsf{act},i}$ can become arbitrarily large modulo q. Moreover, we cannot just sample $\mathbf{r}_i$ random over $\mathcal{R}_q^\ell$ since this breaks the condition that the response $\mathbf{z}_i$ is short. Recall here that this is not an artifact of the proof strategy since there is a concrete attack, as we explained above.

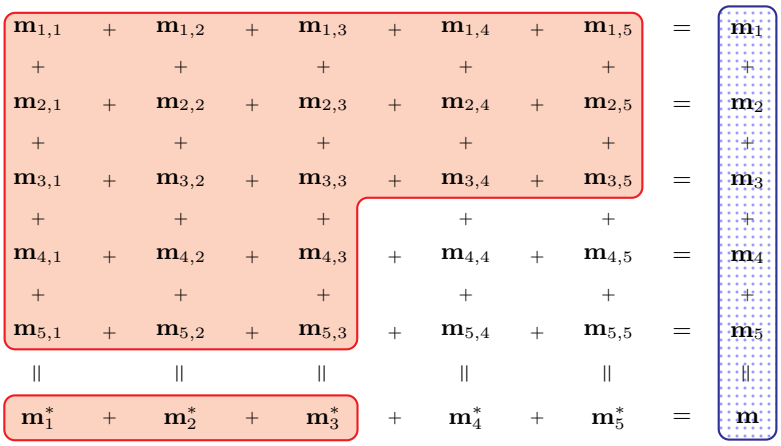

Figure 5: Relationships between $\mathbf{m}_{i,j}, \mathbf{m}_i$ and $\mathbf{m}^*_j$, where we drop the subscript $\mathsf{act} = \{1, 2, 3, 4, 5\}$ for readability.

- The row masks $\mathbf{m}_i$ (blue, dotted pattern) are all public.
- An adversary corrupting the user set $\{1, 2, 3\}$ learns the set $(\mathbf{m}_{i,j})_{\min(i,j)\le 3}$ and can infer the column masks $(\mathbf{m}^*_j)_{j\le 3}$ (red).

Additive masks. This brings us to our masking idea. At a high level, the masking allows the reduction to move around the partial signing keys around in such a way that the response

can be simulated using only the full signing key, without the partial signing key. Effectively, we can remove the Lagrange coefficients in the reduction, and arrive at a reduction similar to the standard non-thresholdised signature scheme.

Let us explain via an example. Assume the adversary queries a set $\mathsf{act} = \{1,2,3,4,5\}$ with two honest users 4 and 5 as in Fig. 5. Let us focus on the four masks $(\mathbf{m}_{\mathsf{act},i,j})_{i,j\in\{4,5\}}$ not known to the adversary. From Item (C.1), recall that the first signing round reveals the sums $\sum_{j\in\{4,5\}} \mathbf{m}_{\mathsf{act},4,j}$ and $\sum_{j\in\{4,5\}} \mathbf{m}_{\mathsf{act},5,j}$ to the adversary since all $(\mathbf{m}_{\mathsf{act},i,j})_{\min(i,j)\leq 3}$ (in red in Fig. 5) are known to the adversary. This leaves us one degree of freedom; the sums $\sum_{j\in\{4,5\}} \mathbf{m}^*_{\mathsf{act},j,4}$ and $\sum_{j\in\{4,5\}} \mathbf{m}^*_{\mathsf{act},j,5}$ are distributed uniformly random from the view of the adversary, conditioned on their sum being $\sum_{i,j\in\{4,5\}} \mathbf{m}_{\mathsf{act},i,j}$. Put differently, the column masks $\mathbf{m}^*_{\mathsf{act},4}$ and $\mathbf{m}^*_{\mathsf{act},5}$ are distributed uniformly random, conditioned on their sum being consistent with $\sum_{j\in\{4,5\}} \mathbf{m}_{\mathsf{act},j}$. Using this, in the proof, we can argue that the two responses $(\mathbf{z}_4, \mathbf{z}_5)$ generated as

$$\left(c \cdot \lambda_{\mathsf{act},4} \cdot \mathbf{s}_4 + \mathbf{r}_4 + \mathbf{m}^*_{\mathsf{act},4}, \quad c \cdot \lambda_{\mathsf{act},5} \cdot \mathbf{s}_5 + \mathbf{r}_5 + \mathbf{m}^*_{\mathsf{act},5}\right) \tag{3}$$

are distributed identically to responses generated as

$$\left(\mathbf{r}_4 + \overline{\mathbf{m}}^*_{\mathsf{act},4}, \quad c \cdot \left(\sum_{i\in\{4,5\}} \lambda_{\mathsf{act},i} \cdot \mathbf{s}_i\right) + \mathbf{r}_5 + \overline{\mathbf{m}}^*_{\mathsf{act},5}\right),$$

where $\overline{\mathbf{m}}^*_{\mathsf{act},4}$ is sampled uniformly random and $\overline{\mathbf{m}}^*_{\mathsf{act},5}$ is set as the unique value that guarantees consistency with the verification equations.

Lastly, we use the fact that $\sum_{j\in\{4,5\}} \lambda_{\mathsf{act},j} \cdot \mathbf{s}_j = \mathbf{s} - \sum_{j\in\mathsf{corrupt}} \lambda_{\mathsf{act},j} \cdot \mathbf{s}_j$ (see Eq. (2)), where the adversary (and the reduction) controls the secrets for all users in $\mathsf{corrupt} = \mathsf{act}\backslash\{4,5\}$. Plugging this into the above, the reduction can instead generate the responses as

$$\left(\mathbf{r}_4 + \overline{\mathbf{m}}^*_{\mathsf{act},4}, \quad c \cdot \mathbf{s} - c \cdot \sum_{j\in\mathsf{corrupt}} \lambda_{\mathsf{act},j} \cdot \mathbf{s}_j + \mathbf{r}_5 + \overline{\mathbf{m}}^*_{\mathsf{act},5}\right).$$

Since the reduction can now simulate the response $c \cdot \mathbf{s} + \mathbf{r}_5$ of the base signature scheme only using the full signing key $\mathbf{s}$, we can rely on prior proof techniques at this point to complete the proof.

Subtle Issue with the Proof and a Fix. While the intuition is simple, the concrete proof requires much care. One important point we glossed over was how we guarantee Eq. (3). Recall users 4 and 5 only *locally* generate the challenge $c := \mathsf{H}_c(\mathsf{vk}, \mathsf{msg}, \mathbf{w})$, where $\mathbf{w} = \sum_{i\in\mathsf{act}} \mathbf{w}_i$ is the aggregated commitment (see Item (C.3)).

In particular, a malicious adversary could send users 4 and 5 inconsistent commitments (e.g., malicious user 1 provides distinct $\mathbf{w}_1$ and $\mathbf{w}'_1$ to users 4 and 5), in which case, the locally derived challenges c and c' by users 4 and 5 may differ. Against such an adversary, the reduction cannot argue Eq. (3), and incidentally, the proof breaks down. In fact, we can turn this idea into a concrete attack, similarly to those explained prior.

This brings us to our final construction, Threshold Raccoon, where we fix this issue by modifying the users to authenticate their views in the second round. One way we achieve this is to let the users add a signature to the hash commitments it received in Item (C.2). Another way is to let the users add a MAC instead. Our construction is detailed in Section 3.

Sharing the Masks. Lastly, we explain how pairs of users $(i, j) \in \mathsf{act}$ share the masks $\mathbf{m}_{\mathsf{act},i,j}$ and $\mathbf{m}_{\mathsf{act},j,i}$ during the signing protocol. We simply generate seeds $(\mathsf{seed}_{i,j})_{i,j\in[N]}$ during the key generation phase and give $(\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j\in[N]}$ to user i as part of their partial signing key. Once the set act is defined, user i can locally compute the random masks $\mathbf{m}_{\mathsf{act},i,j}$ and $\mathbf{m}_{\mathsf{act},j,i}$ by using a PRF on $\mathsf{seed}_{i,j}$ and $\mathsf{seed}_{j,i}$ respectively. For the masks to never be repeated, we assume each signing session has a unique identifier for which the PRF is called upon.

3 Threshold Raccoon: Our Threshold Signature Scheme

Our 3-round threshold signature, named Threshold Raccoon, is given formally in Figs. 6 and 7 and a security reduction is stated in Theorem 4.1. We assume the presence of a trusted centralised party to run the key generation algorithm KeyGen. This can also be achieved with a distributed key generation algorithm. The design of a suitable DKG, while important, is outside of the scope of this work. The key generation runs Shamir's Secret Sharing algorithm in order to derive a Raccoon public key along with N secret key shares such that any T are sufficient to sign.[6] It takes as inputs the system parameters $\mathsf{pp}(\kappa)$, a threshold T, and a total number of parties N.

3.1 Key Generation

Algorithm 1: $\mathsf{KeyGen}(\mathsf{pp}, T, N)$

$\mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}$ ▷ Sample matrix
$(\mathbf{s}, \mathbf{e}) \leftarrow \mathcal{D}_{\mathbf{t}}^{\ell} \times \mathcal{D}_{\mathbf{t}}^{k}$ ▷ Small secret and noise
$\mathbf{t} \coloneqq \lfloor \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \rceil_{\nu_{\mathbf{t}}}$ ▷ Part of public key in $\mathcal{R}_{q_{\mathbf{t}}}^{k}$
$\mathsf{vk} \coloneqq (\mathbf{A}, \mathbf{t})$
$\mathbf{P} \leftarrow \mathcal{R}_q^{\ell}[X]$ with $\deg(\mathbf{P}) = T - 1$, $\mathbf{P}(0) = \mathbf{s}$ ▷ Shamir Secret Sharing
$(\mathbf{s}_i)_{i\in[N]} \coloneqq (\mathbf{P}(i))_{i\in[N]}$ ▷ Secret shares
for $i \in [N]$ **do**
  $(\mathsf{vk}_{\mathsf{sig},i}, \mathsf{sk}_{\mathsf{sig},i}) \leftarrow \mathsf{KeyGen}_{\mathsf{sig}}(1^{\kappa})$ ▷ Standard signature keys for each user
  for $j \in [N]$ **do**
    $\mathsf{seed}_{i,j} \leftarrow \{0,1\}^{\kappa}$ ▷ Pairwise-shared seeds
for $i \in [N]$ **do**
  $\mathsf{sk}_i \coloneqq (\mathbf{s}_i, (\mathsf{vk}_{\mathsf{sig},i})_{i\in[N]}, \mathsf{sk}_{\mathsf{sig},i}, (\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j\in[N]})$
return $(\mathsf{vk}, (\mathsf{sk}_i)_{i\in[N]})$

Figure 6: Centralised key generation for Threshold Raccoon. In above, we assume the key generation algorithm initialises the state of each user j.

The key generation algorithm is defined formally in Fig. 6. As a threshold version of the plain Raccoon signature, the key generation algorithm generates the public key in the same

[6] Strictly speaking, the underlying signature scheme is not Raccoon as we use discrete Gaussians instead of sum of uniforms. However, we attribute Raccoon as the core features (i.e., removing rejection sampling and optimisations) are the same.

manner. A short secret $(\mathbf{s}, \mathbf{e}) \leftarrow \mathcal{D}_{\mathbf{t}}^{\ell} \times \mathcal{D}_{\mathbf{t}}^{k}$ is sampled and the public key is $(\mathbf{A}, \lfloor \mathbf{As} + \mathbf{e} \rceil_{\nu_{\mathbf{t}}})$, where $\lfloor \mathbf{As} + \mathbf{e} \rceil_{\nu_{\mathbf{t}}}$ is the modulus rounding operation, which essentially drops the $\nu_{\mathbf{t}}$ least significant bits on each entry of the input vector. The main changes are the use of secret sharing and pairwise shared seeds.

Pairwise Shared Seeds. To ensure the unforgeability of the signing procedure, the users' individual responses are additively hidden with private column mask vectors $(\mathbf{m}_i^*)_{i \in \mathsf{act}}$. These are later subtracted from the aggregated response with the publicly computable row mask vectors $\sum_{i \in \mathsf{act}} \mathbf{m}_i$, see see Fig. 5 for an illustration.

These mask vectors can be viewed as a T-out-of-T shared secret that is computed *on-the-fly* and *non-interactively* during the individual ShareSign protocols, and its shared values are recomputed by the combine algorithm using the communication transcript. These shares are pairwise shared between users and have to be unique between sessions. To achieve this, we generate them as the result of a pseudorandom function PRF from a seed and the session id: $\mathsf{PRF}(\mathsf{seed}_{i,j}, \mathsf{sid})$ with the seeds $(\mathsf{seed}_{i,j})_{(i,j) \in [N] \times [N]}$ that are generated and given to the corresponding users during the key generation.

Signing Keys. To ensure that users agree on the view of the signing session in Round 2, they sign their view under a personal signing key. The key generation chooses a personal verification and signing key for all parties, $(\mathsf{vk}_{\mathsf{sig},i}, \mathsf{sk}_{\mathsf{sig},i})$. Alternatively, key generation can generate N^2 pairwise symmetric keys so that all parties are pairwise linked. Then the view is authenticated using T MACs per party. This is more efficient when T is small because MACs are much smaller than post quantum digital signatures. Such symmetric keys may be derived from the pairwise shared seeds explained above.

3.2 Distributed Signing Procedure

Signing proceeds in 3 rounds. In essence, we use two T-out-of-T secret sharings. The first one is of the commitment $\mathbf{w}$ and the second is a masking term $\mathbf{m}$ that is used to mask the distributions of the partial responses in the Fiat-Shamir transform underlying Raccoon. Over the first two rounds, this commitment $\mathbf{w}$ is exchanged in a commit-reveal manner to prevent potential attacks from rushing adversaries. We note that some important yet tedious consistency checks (e.g., check whether session for sid exists) in our signing protocol is outsourced to Section 7, Fig. 8 for better readability.

First round. Every party j inside the signing set act generates their (rounded) MLWE commitment share $\mathbf{w}_j$ encoding the ephemeral randomness $\mathbf{r}_j$. In parallel, they use their pairwise-shared seeds $(\mathsf{seed}_{j,i})_{i \in \mathsf{act}}$ and the session id sid to compute a public *row* mask $\mathbf{m}_j = \sum_{i \in \mathsf{act}} \mathsf{PRF}(\mathsf{seed}_{j,i}, \mathsf{sid})$. We recall Fig. 5 for a pictorial explanation of the mask term. They then publish $\mathbf{m}_j$, as well as a hash commitment cmt_j of $\mathbf{w}_j$.

Second round. Each party j reveals their MLWE commitment share $\mathbf{w}_j$. Additionally they sign their current view of the signing session under their personal signing keys $\mathsf{sk}_{\mathsf{sig},j}$ (or alternatively, using MAC keys). The commit-reveal is a standard technique so that the adversary does not generate its commitments in accordance with those of the honest users (see for instance [BN06]).

Algorithm 2: $\mathsf{ShareSign}_1(\mathsf{state}, \mathsf{sid}, \mathsf{act}, \mathsf{msg})$

$\textbf{assert}\{\ \mathsf{ConsistCheck}_1(\mathsf{state}, \mathsf{sid}, \mathsf{act}, \mathsf{msg})\ \}$ ▷ Consistency checks, see Fig. 8
$(\mathbf{r}_j, \mathbf{e}'_j) \leftarrow \mathcal{D}^{\ell}_{\mathbf{w}} \times \mathcal{D}^{k}_{\mathbf{w}}$ ▷ Sample small ephemeral randomness and small noise
$\mathbf{w}_j \coloneqq \mathbf{A} \cdot \mathbf{r}_j + \mathbf{e}'_j$ ▷ MLWE commitment in $\mathcal{R}^k_q$ without rounding
$\mathsf{cmt}_j \coloneqq \mathsf{H}_{\mathsf{com}}(\mathsf{sid}, \mathsf{act}, \mathsf{msg}, \mathbf{w}_j)$ ▷ Hash commitment
Fetch $(\mathsf{seed}_{j,i})_{i \in \mathsf{act}}$ from $\mathsf{state.sk}$
$\mathbf{m}_j \coloneqq \sum_{i \in \mathsf{act}} \mathsf{PRF}(\mathsf{seed}_{j,i}, \mathsf{sid})$ ▷ Compute row blinder in $\mathcal{R}^{\ell}_q$
$\mathsf{state.session}[\mathsf{sid}] \coloneqq \{\mathsf{sid}, \mathsf{act}, \mathsf{msg}, 1, \{\mathbf{r}_j, \mathbf{w}_j, \mathsf{cmt}_j, \mathbf{m}_j\}, \emptyset\}$ ▷ New session state
return $\mathsf{contrib}_1[j] \coloneqq (\mathsf{cmt}_j, \mathbf{m}_j)$

Algorithm 3: $\mathsf{ShareSign}_2(\mathsf{state}, \mathsf{sid}, \mathsf{contrib}_1)$

$\textbf{assert}\{\ \mathsf{ConsistCheck}_2(\mathsf{state}, \mathsf{sid}, \mathsf{contrib}_1)\ \}$ ▷ Consistency checks, see Fig. 8
Fetch $\mathsf{sk}_{\mathsf{sig},j}$ from $\mathsf{state.sk}$
$\sigma_j \leftarrow \mathsf{Sign}_{\mathsf{sig}}(\mathsf{sk}_{\mathsf{sig},j}, \mathsf{sid} \,||\, \mathsf{act} \,||\, \mathsf{msg} \,||\, \mathsf{contrib}_1)$ ▷ Sign first-round contribution with standard signature
Fetch $\mathbf{w}_j$ from $\mathsf{state.sessions}[\mathsf{sid}].\mathsf{internal}$ ▷ Recall $\mathbf{w}_j$ from $\mathsf{ShareSign}_1$
$\mathsf{state.session}[\mathsf{sid}] \coloneqq \{\mathsf{sid}, \mathsf{act}, \mathsf{msg}, 2, \{\mathbf{r}_j, \mathbf{w}_j, \mathsf{cmt}_j, \mathbf{m}_j\}, \mathsf{contrib}_1\}$ ▷ Update session state
return $\mathsf{contrib}_2[j] \coloneqq (\mathbf{w}_j, \sigma_j)$

Algorithm 4: $\mathsf{ShareSign}_3(\mathsf{state}, \mathsf{sid}, \mathsf{contrib}_2)$

$\textbf{assert}\{\ \mathsf{ConsistCheck}_3(\mathsf{state}, \mathsf{sid}, \mathsf{contrib}_2)\ \}$ ▷ Consistency checks, see Fig. 8
Let $\mathsf{session} = \mathsf{state.sessions}[\mathsf{sid}]$
Fetch $(\mathsf{sid}, \mathsf{act}, \mathsf{msg})$ from $\mathsf{session}$
Fetch $\mathbf{r}_j$ from $\mathsf{session.internal}$ and $\mathbf{s}_j, (\mathsf{vk}_{\mathsf{sig},i})_{i \in [N]}, (\mathsf{seed}_{i,j})_{i \in \mathsf{act}}$ from $\mathsf{state.sk}$
Fetch $\mathsf{contrib}_1 = (\mathsf{cmt}_i, \mathbf{m}_i)_{i \in \mathsf{act}}$ from $\mathsf{session.contrib}_1$
Parse $\mathsf{contrib}_2 = (\mathbf{w}_i, \sigma_i)_{i \in \mathsf{act}}$
for $i \in \mathsf{act}$ **do**
  $\textbf{assert}\{\ \mathsf{cmt}_i = \mathsf{H}_{\mathsf{com}}(\mathsf{sid}, \mathsf{msg}, \mathsf{act}, \mathbf{w}_i)\ \}$ ▷ Check consistency of hash commitments
  $\textbf{assert}\{\ \mathsf{Verify}_{\mathsf{sig}}(\mathsf{vk}_{\mathsf{sig},i}, \mathsf{sid} \,||\, \mathsf{act} \,||\, \mathsf{msg} \,||\, \mathsf{contrib}_1, \sigma_i) = 1\ \}$ ▷ Check same first-round contribution used
$\mathbf{w} \coloneqq \lfloor \sum_{i \in \mathsf{act}} \mathbf{w}_i \rceil_{\nu_{\mathbf{w}}}$ ▷ Aggregated rounded commitment in $\mathcal{R}^k_{q_{\mathbf{w}}}$
$c \coloneqq \mathsf{H}_c(\mathsf{state.vk}, \mathsf{msg}, \mathbf{w})$ ▷ Global challenge
$\mathbf{m}^*_j \coloneqq \sum_{i \in \mathsf{act}} \mathsf{PRF}(\mathsf{seed}_{i,j}, \mathsf{sid})$ ▷ Compute column blinder in $\mathcal{R}^{\ell}_q$
$\mathbf{z}_j \coloneqq c \cdot \lambda_{\mathsf{act},j} \cdot \mathbf{s}_j + \mathbf{r}_j + \mathbf{m}^*_j$ ▷ Individual response in $\mathcal{R}^{\ell}_q$
return $\mathsf{contrib}_3[j] \coloneqq \mathbf{z}_j$

Algorithm 5: $\mathsf{Combine}(\mathsf{vk}, \mathsf{sid}, \mathsf{msg}, \mathsf{contrib}_1, \mathsf{contrib}_2, \mathsf{contrib}_3)$

Parse $\mathsf{contrib}_1 = (\mathsf{cmt}_i, \mathbf{m}_i)_{i \in \mathsf{act}}$, $\mathsf{contrib}_2 = (\mathbf{w}_i, \sigma_i)_{i \in \mathsf{act}}$, $\mathsf{contrib}_3 = (\mathbf{z}_i)_{i \in \mathsf{act}}$
Parse $\mathsf{vk} = (\mathbf{A}, \mathbf{t})$
$\mathbf{w} \coloneqq \lfloor \sum_{i \in \mathsf{act}} \mathbf{w}_i \rceil_{\nu_{\mathbf{w}}}$ ▷ Aggregated rounded commitment in $\mathcal{R}^k_{q_{\mathbf{w}}}$
$\mathbf{z} \coloneqq \sum_{i \in \mathsf{act}} (\mathbf{z}_i - \mathbf{m}_i)$ ▷ Aggregated response shifted by column blinders in $\mathcal{R}^{\ell}_q$
$c \coloneqq \mathsf{H}_c(\mathsf{vk}, \mathsf{msg}, \mathbf{w})$ ▷ Global challenge
$\mathbf{y} \coloneqq \lfloor \mathbf{A} \cdot \mathbf{z} - 2^{\nu_{\mathbf{t}}} \cdot c \cdot \mathbf{t} \rceil_{\nu_{\mathbf{w}}}$ ▷ Intermediate value in $\mathcal{R}^k_{q_{\mathbf{w}}}$
$\mathbf{h} \coloneqq \mathbf{w} - \mathbf{y}$ ▷ Hint in $\mathcal{R}^k_{q_{\mathbf{w}}}$
return $\mathsf{sig} \coloneqq (c, \mathbf{z}, \mathbf{h})$

Figure 7: Signing procedure for Threshold Raccoon. In above, we omit the subscript and assume state is the state of party $j \in \mathsf{act}$. Consistency checks are described in Fig. 8.

Third round. All parties checks that the received commitment share $\mathbf{w}_j$ is consistent with the hash commitments in the first round and that all the signatures from the second round verify. It then computes the resulting global commitment $\mathbf{w}$ as $\lfloor \sum_{i \in \mathsf{act}} \mathbf{w}_i \rceil_{\nu_\mathbf{w}}$ using the messages they received from the first two rounds. Then, parties compute the signature challenge $c = \mathsf{H}_c(\mathsf{vk}, \mathsf{msg}, \mathbf{w})$ for themselves.

The parties further compute a secret *column* mask $\mathbf{m}_j^* = \sum_{i \in \mathsf{act}} \mathsf{PRF}(\mathsf{seed}_{i,j}, \mathsf{sid})$ using their pairwise-shared seeds $(\mathsf{seed}_{i,j})_{i \in \mathsf{act}}$ and the session id sid. They use this to define their response share $\mathbf{z}_j = c \cdot \lambda_{\mathsf{act},j} \cdot \mathbf{s}_j + \mathbf{r}_j + \mathbf{m}_j^*$ for $\mathbf{s}_j$ their secret share and $\lambda_{\mathsf{act},j}$ a Lagrange coefficient corresponding to the active signing set act. Since $\sum_{i \in \mathsf{act}} \lambda_{\mathsf{act},i} \cdot \mathbf{s}_i = \mathbf{s}$ sums to the full secret $\mathbf{s}$, these shares sum to a valid response shifted by the column masks. Here, the main observation is that $\sum_{i \in \mathsf{act}} \mathbf{m}_i = \sum_{i \in \mathsf{act}} \mathbf{m}_i^*$ (see Fig. 5). Lastly, they return the response share $\mathbf{z}_j$.

Combination. Once all parties have completed all rounds, the coordinator runs a combine algorithm to compute the signature. This algorithm simply rounds the sum of the MLWE commitments to get the full commitment $\mathbf{w} = \lfloor \sum_{i \in \mathsf{act}} \mathbf{w}_i \rceil_{\nu_\mathbf{w}}$. The challenge is $c = \mathsf{H}_c(\mathsf{vk}, \mathsf{msg}, \mathbf{w})$. The response is the sum of the response shares, subtracted with the sum of the public column masks: $\mathbf{z} = \sum_{i \in \mathsf{act}} (\mathbf{z}_i - \mathbf{m}_i)$. Finally, the hint is computed as $\mathbf{h} = \mathbf{w} - \lfloor \mathbf{A} \cdot \mathbf{z} - 2^{\nu_\mathbf{t}} \cdot c \cdot \mathbf{t} \rceil_{\nu_\mathbf{w}}$ where $\mathsf{vk} = (\mathbf{A}, \mathbf{t})$. It returns a signature $(c, \mathbf{z}, \mathbf{h})$ of Raccoon.

Verification. We do not explicitly define the verification algorithm since it is identical to those of the plain Raccoon signature scheme.

Remark 3.1 (Statefulness). The signing algorithm requires signers never to respond with respect to the same session ID twice. They must store all session IDs that they have used previously and abort if they receive a repeated request.

4 Security Reduction

The security of our threshold signature Threshold Raccoon is summarised in Theorem 4.1.

Theorem 4.1. *The threshold signature scheme Threshold Raccoon described in Fig. 7 is unforgeable under the unforgeability of the (non-thresholdised) signature scheme, pseudorandomness of* PRF*, the* $\mathsf{Hint\text{-}MLWE}_{q,\ell,k,Q_{\mathsf{Sign}},\sigma_\mathbf{t},\sigma_\mathbf{w},\mathcal{C}}$ *and* $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,\mathcal{C},B_{\mathsf{stmsis}}}$ *assumptions.*

Formally, for any adversary $\mathcal{A}$ *against the unforgeability game making at most* Q_H *and* Q_Sign *queries to the random oracles* $\mathsf{H}_c, \mathsf{H}_\mathsf{com}$ *and the signing oracle, respectively, there exists adversaries* $\mathcal{B}_\mathsf{Sign}$*,* $\mathcal{B}_\mathsf{PRF}$*,* $\mathcal{B}$*, and* $\mathcal{B}'$ *against the unforgeability of the signature scheme, pseudorandomness of* PRF*, and* $\mathsf{Hint\text{-}MLWE}_{q,\ell,k,Q_{\mathsf{Sign}},\sigma_\mathbf{t},\sigma_\mathbf{w},\mathcal{C}}$ *and* $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,\mathcal{C},B_{\mathsf{stmsis}}}$ *problems, respectively, such that*

$$\begin{aligned}\mathsf{Game}^{\mathsf{ts\text{-}uf}}_{\mathcal{A}}(\kappa) \leq N \cdot \mathsf{Adv}^{\mathsf{sig\text{-}uf}}_{\mathcal{B}_\mathsf{Sign}}(\kappa) + \mathsf{Adv}^{\mathsf{PRF}}_{\mathcal{B}_\mathsf{PRF}}(\kappa) + \frac{(Q_\mathsf{H}+1) \cdot Q_\mathsf{Sign}}{2^{n-1}} \\ + \frac{Q_\mathsf{H} + Q_\mathsf{H}^2}{2^{2\kappa}} + \mathsf{Adv}^{\mathsf{Hint\text{-}MLWE}}_{\mathcal{B}}(\kappa) + \mathsf{Adv}^{\mathsf{SelfTargetMSIS}}_{\mathcal{B}'}(\kappa)\end{aligned}$$

where $\mathsf{Time}(\mathcal{B}_\mathsf{Sign}), \mathsf{Time}(\mathcal{B}_\mathsf{PRF}), \mathsf{Time}(\mathcal{B}), \mathsf{Time}(\mathcal{B}') \approx \mathsf{Time}(\mathcal{A})$.

Our security reduction relies on the hardness of $\mathsf{Hint\text{-}MLWE}_{q,\ell,k,Q_{\mathsf{Sign}},\sigma_{\mathbf{t}},\sigma_{\mathbf{w}},\mathcal{C}}$. This new assumption was formalized and studied in a recent paper [KLSS23]. A significant advantage of Hint-MLWE is the existence of a very efficient reduction to the standard MLWE assumption. Moreover, this reduction covers the parameters used in Threshold Raccoon. In our context, it states that the public key $(\mathbf{A}, \mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e})$ remains pseudorandom even if Q_{Sign} hints[7] $(c_i, \bar{\mathbf{z}}_i)$ are provided to the adversary, where $c_i \leftarrow \mathcal{C}$ corresponds to the challenge and $\bar{\mathbf{z}}_i = c_i \cdot (\mathbf{s}, \mathbf{e}) + (\mathbf{r}_i, \mathbf{e}'_i)$ is a natural by-product of the signing process.

5 Instantiation and Implementation

5.1 Parameter Sets

Table 1 presents admissible parameter sets for NIST levels I, III and V. We take the noise distributions $\mathcal{D}_{\mathbf{t}}$ and $\mathcal{D}_{\mathbf{w}}$ to be Gaussian distributions with parameters $\sigma_{\mathbf{t}}$ and $\sigma_{\mathbf{w}}$, respectively. The challenge c is sampled from the set of polynomials such that $\|c\|_\infty = 1$ and $\|c\|_1 = \omega$, which is the same set as in Masked Raccoon or Dilithium. We note $|\mathsf{vk}|$, $|\mathsf{sig}|$ and $|\mathsf{trans}|$ the size in bytes of the verification key, the signature and the transcript of the signing procedure (Fig. 7), respectively.

Table 1: Parameter sets. The sizes $|\mathsf{vk}|$ and $|\mathsf{sig}|$ are provided in kilobytes. All parameter sets satisfy $(\lfloor \log_2 q \rceil, n, \sigma_{\mathbf{t}}, \max T) = (49, 512, 2^{20}, 1024)$.

NIST level	κ	Q_{Sign}	$\sigma_{\mathbf{w}}\sqrt{T}$	$\nu_{\mathbf{t}}$	$\nu_{\mathbf{w}}$	ℓ	k	ω	$\lvert\mathsf{vk}\rvert$	$\lvert\mathsf{sig}\rvert$	$\lvert\mathsf{trans}\rvert/T$
I	128	2^{60}	2^{42}	37	40	4	5	19	3.9	12.7	40.8
III	192	2^{64}	2^{42}	36	40	6	7	31	5.8	18.9	59.6
V	256	2^{60}	2^{42}	35	41	7	8	44	7.2	21.6	69.1

Comparison with Masked Raccoon. We can see that the parameters of Threshold Raccoon and Masked Raccoon [dEK+23] are similar. The modulus q and the degree n are identical in both cases, and many other parameters are close if not identical.

The main difference is the Masked Raccoon uses sums of uniforms, while Threshold Raccoon uses Gaussians. The main motivation in [dEK+23] for using sums of uniforms is that they are easier to sample securely in the context of side-channels, while still being usable in Rényi divergence-based arguments in a way that almost guarantees the same tightness as Gaussians, see [dEK+23, Appendix A] for more details.

In this work we prefer to use Gaussians, as they make our security proofs simpler and we are not concerned with side-channel attacks. In addition, Hint-MLWE with Gaussians distributions can be reduced to standard MLWE using a reduction from [KLSS23]. This reduction has better tightness than what would be provided by a Rényi divergence-based, and allows us to increase the maximum number of queries Q_{Sign} by a factor $O(\sqrt{\kappa \cdot n \cdot (k + \ell)})$. This is why Table 1 shows higher values for Q_{Sign} than [dEK+23], despite the similar parameters.

[7] The term "hint" in Hint-MLWE [KLSS23] is not the same as the hint in the signing process of Dilithium [LDK+22] ($\mathbf{h}$ in Threshold Raccoon). This is an unfortunate collision of terminology.

5.2 Implementation and Experiments

We have developed a high-performance implementation of Threshold Raccoon which can easily accomodate $T = 1024$ simulated signers (with the parameters in Table 1.) If we ignore possible communication latencies and enable 4.5 gHz turbo on an i7-12700, creating a signature (the three steps of ShareSign, $\kappa = 128$) requires from 11.1 ms ($T = 4$) to 116 ms ($T = 1024$) of single-core computation from each signer. The verification function is independent of T and N, and requires approximately 0.230 ms. Table 2 contains more detailed benchmarking results. [8]

Table 2: Threshold Raccoon ($\kappa = 128$) Cycle counts on a single core of an Intel i7-12700 CPU with "turbo boost" disabled. The units are millions of cycles; divide by 2.1 (fixed clock frequency in gHz) to obtain millisecond numbers. Measurements for KeyGen, Combine, and Verify are for the entire process, while $\mathsf{ShareSign}_i$ is per signer (when signing is a parallel process, this is equivalent to the elapsed time).

T	KeyGen	$\mathsf{ShareSign}_1$	$\mathsf{ShareSign}_2$	$\mathsf{ShareSign}_3$	Combine	Verify
4	0.592	20.092	0.539	1.588	1.128	1.094
16	0.417	20.076	2.102	5.559	1.209	1.093
64	0.817	21.830	8.216	21.350	1.579	1.100
256	2.838	33.549	32.788	84.333	3.186	1.095
1024	11.491	67.213	131.887	338.614	11.571	1.106

This implementation recycles components such as NTT and signature serialization from the Raccoon NIST submission [dPEK+23]. It uses κ-bit MACs keyed with pairwise $\mathsf{seed}_{i,j}$ (and sid) to authenticate contributions, as discussed in Section 3.1. The Uniform and Gaussian random samplers, MACs and PRFs are built from the SHAKE128 [NIS15] extensible output function. This function (or, more precisely, its Keccak permutation component) dominates the overall running time, requiring up to 80% of cycles. This is despite the code utilizing an AVX2 SIMD Keccak that computes four permutations at the same time.

At $\kappa = 128$, public key is $|\mathsf{vk}| = 3856$ bytes. Due to non-uniform distributions, the actual signature encoding size is variable, but can be (with high probability) upper bounded at $|\mathsf{sig}| \leq 12736$ bytes. Communicating the secret key shares and PRF/MAC seed pairs to each of the N potential signers requires $12556 + 32N$ bytes with this implementation. The signing bandwidth requirements (in bytes) are $\frac{1}{T}|\mathsf{contrib}_1| = 12576$, $\frac{1}{T}|\mathsf{contrib}_2| = 15680 + 16T$, and $\frac{1}{T}|\mathsf{contrib}_3| = 12544$, bringing the total per-signer contribution to $40800 + 16T$ bytes. If asymmetric signatures rather than pairwise MACs were used, each signer contribution would have a size asymptotically independent of T.

6 Full Version

The full version of this work can be found at:

https://tprest.github.io/pdf/pub/threshold-raccoon-anonymous.pdf

[8] The Threshold Raccoon implementation used to generate these benchmarking results is available to reviewers: https://anonymous.4open.science/r/ec24-thrc-F64C

References

ABCP23. Shahla Atapoor, Karim Baghery, Daniele Cozzo, and Robi Pedersen. Vss from distributed zk proofs and applications. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023*, pages 405–440, Singapore, 2023. Springer Nature Singapore.

ABV+12. Shweta Agrawal, Xavier Boyen, Vinod Vaikuntanathan, Panagiotis Voulgaris, and Hoeteck Wee. Functional encryption for threshold functions (or fuzzy ibe) from lattices. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 280–297. Springer, Heidelberg, May 2012.

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.

BCK+22. Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 517–550. Springer, Heidelberg, August 2022.

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.

BGG+18. Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596. Springer, Heidelberg, August 2018.

BLMR13. Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 410–428. Springer, Heidelberg, August 2013.

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.

Bol03. Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Heidelberg, January 2003.

CCK23. Jung Hee Cheon, Wonhee Cho, and Jiseung Kim. Improved universal thresholdizer from threshold fully homomorphic encryption. Cryptology ePrint Archive, Paper 2023/545, 2023. https://eprint.iacr.org/2023/545.

CKM+23. Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Snowblind: A threshold blind signature in pairing-free groups. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 710–742. Springer, Heidelberg, August 2023.

CSS+22. Siddhartha Chowdhury, Sayani Sinha, Animesh Singh, Shubham Mishra, Chandan Chaudhary, Sikhar Patranabis, Pratyay Mukherjee, Ayantika Chatterjee, and Debdeep Mukhopadhyay. Efficient threshold FHE with application to real-time systems. Cryptology ePrint Archive, Report 2022/1625, 2022. https://eprint.iacr.org/2022/1625.

dEK+23. Rafael del Pino, Thomas Espitau, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, Mélissa Rossi, and Markku-Juhani Saarinen. Raccoon. Technical report, National Institute of Standards and Technology, 2023. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

Des90. Yvo Desmedt. Abuses in cryptography and how to fight them. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 375–389. Springer, Heidelberg, August 1990.

DF90. Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, Heidelberg, August 1990.

DLN+21. Julien Devevey, Benoît Libert, Khoa Nguyen, Thomas Peters, and Moti Yung. Non-interactive CCA2-secure threshold cryptosystems: Achieving adaptive security in the standard model without pairings. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 659–690. Springer, Heidelberg, May 2021.

dPEK+23. Rafaël del Pino, Thomas Espitau, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, Mélissa Rossi, and Markku-Juhani Saarinen. Raccoon. Technical report, National Institute

of Standards and Technology, 2023. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

GKS23. Kamil Doruk Gur, Jonathan Katz, and Tjerand Silde. Two-round threshold lattice signatures from threshold homomorphic encryption. Cryptology ePrint Archive, Paper 2023/1318, 2023. https://eprint.iacr.org/2023/1318.

KG20. Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, Heidelberg, October 2020.

KLSS23. Duhyeong Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Toward practical lattice-based proof of knowledge from hint-MLWE. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 549–580. Springer, Heidelberg, August 2023.

KY02. Jonathan Katz and Moti Yung. Threshold cryptosystems based on factoring. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 192–205. Springer, Heidelberg, December 2002.

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.

Lin22. Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. Cryptology ePrint Archive, Report 2022/374, 2022. https://eprint.iacr.org/2022/374.

LJY14. Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: fully distributed non-interactive adaptively-secure threshold signatures with short shares. In Magnús M. Halldórsson and Shlomi Dolev, editors, *33rd ACM PODC*, pages 303–312. ACM, July 2014.

LST18. Benoît Libert, Damien Stehlé, and Radu Titiu. Adaptively secure distributed PRFs from LWE. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 391–421. Springer, Heidelberg, November 2018.

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.

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.

NIS15. NIST. SHA-3 standard: Permutation-based hash and extendable-output functions. Federal Information Processing Standards Publication FIPS 202, August 2015.

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.

PB23. René Peralta and Luís T.A.N. Brandão. Nist first call for multi-party threshold schemes. National Institute of Standards and Technology, 2023. https://doi.org/10.6028/NIST.IR.8214C.ipd.

RRJ+22. Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: Robust asynchronous schnorr threshold signatures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2551–2564. ACM Press, November 2022.

Sha79. Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

Sho00. Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 207–220. Springer, Heidelberg, May 2000.

7 Omitted Consistency Check Algorithms in Threshold Raccoon

Here we include the deferred consistency checks from the signing protocol in Threshold Raccoon (see Fig. 7). Note that by $\mathsf{ConsistCheck}_1$, we always have $j \in \mathsf{act}$ and $\mathsf{act} \subseteq [N]$ for any user index j in state and act in $\mathsf{state.session}[\mathsf{sid}]$, if a $\mathsf{session}$ for sid exists. In particular, this check will be omitted from $\mathsf{ConsistCheck}_2$ and $\mathsf{ConsistCheck}_3$.

Algorithm 6: $\mathsf{ConsistCheck}_1(\mathsf{state}, \mathsf{sid}, \mathsf{act}, \mathsf{msg})$

1: **assert**{ sid contains $(\mathsf{act}, \mathsf{msg})$ } ▷ Check that session id sid is in correct form
2: **assert**{ $\mathsf{state.session}[\mathsf{sid}] = \bot$ } ▷ Check that user never signed using sid
3: Fetch user index j from state
4: **assert**{ $j \in \mathsf{act} \ \wedge \ \mathsf{act} \subseteq [N]$ }

Algorithm 7: $\mathsf{ConsistCheck}_2(\mathsf{state}, \mathsf{sid}, \mathsf{contrib}_1)$

1: **assert**{ $\mathsf{state.session}[\mathsf{sid}] \neq \bot$ } ▷ The user already created a $\mathsf{session}$ with sid
2: Let $\mathsf{session} = \mathsf{state.session}[\mathsf{sid}]$
3: Fetch user index j from state
4: Fetch $(\mathsf{sid}', \mathsf{act}, \mathsf{cmt}_j, \mathbf{m}_j)$ from $\mathsf{session}$
5: **assert**{ The set of keys (i.e. indices) in $\mathsf{contrib}_1$ is exactly act }
▷ Check $\mathsf{contrib}_1$ includes $|\mathsf{act}|$ number of first round messages
6: **for** $i \in \mathsf{act}$ **do**
7: **assert**{ $\mathsf{contrib}_1[i]$ is of the form $\mathsf{contrib}_1[i] = (\mathsf{cmt}_i, \mathbf{m}_i)$ }
▷ Check $\mathsf{contrib}_1$ is defined over the indices in act and of a valid form
8: **assert**{ $\mathsf{session}$ is of the form $\mathsf{session} = \{\mathsf{sid}', \mathsf{act}, \mathsf{msg}, 1, \{\mathbf{r}_j, \mathbf{w}_j, \mathsf{contrib}_1[j]\}, \emptyset\}$ }
▷ Check sid and $\mathsf{contrib}_1[j]$ is consistent with internal state

Algorithm 8: $\mathsf{ConsistCheck}_3(\mathsf{state}, \mathsf{sid}, \mathsf{contrib}_2)$

1: **assert**{ $\mathsf{state.session}[\mathsf{sid}] \neq \bot$ } ▷ The user already opened a $\mathsf{session}$ with sid
2: Let $\mathsf{session} = \mathsf{state.session}[\mathsf{sid}]$
3: Fetch user index j from state
4: Fetch $(\mathsf{sid}', \mathsf{act})$ from $\mathsf{session}$
5: **assert**{ The set of keys in $\mathsf{contrib}_2$ is exactly act }
▷ Check $\mathsf{contrib}_2$ includes $|\mathsf{act}|$ number of second round messages
6: **for** $i \in \mathsf{act}$ **do**
7: **assert**{ $\mathsf{contrib}_2[i]$ is of the form $\mathsf{contrib}_2[i] = (\mathbf{w}_i, \sigma_i)$ }
▷ Check $\mathsf{contrib}_2$ is defined over the indices in act and of a valid form
8: **assert**{ $\mathsf{session}$ is of the form $\mathsf{session} = \{\mathsf{sid}', \mathsf{act}, \mathsf{msg}, 2, \{\mathbf{r}_j, \mathbf{w}_j, \mathsf{contrib}_1[j]\}, \mathsf{contrib}_1\}$ }
▷ Check sid is consistent with internal state

Figure 8: Consistency checks for Threshold Raccoon