Loquat: A SNARK-Friendly Post-Quantum Signature based on the Legendre PRF with Applications in Ring and Aggregate Signatures

Xinyu Zhang[1,2], Ron Steinfeld[1], Muhammed F. Esgin[1], Joseph K. Liu[1], Dongxi Liu[2], and Sushmita Ruj[3]

[1] Monash University, Melbourne, Australia
{Xinyu.Zhang1,Ron.Steinfeld,Muhammed.Esgin,Joseph.Liu}@monash.edu
[2] Data61, CSIRO, Sydney, Australia
Dongxi.Liu@csiro.au
[3] University of New South Wales
Sushmita.Ruj@unsw.edu

Abstract. We design and implement a novel post-quantum signature scheme based on the Legendre PRF, named Loquat. Prior to this work, efficient approaches for constructing post-quantum signatures with comparable security assumptions mainly used the MPC-in-the-head paradigm or hash trees. Our method departs from these paradigms and, notably, is SNARK-friendly, a feature not commonly found in earlier designs. Loquat requires significantly fewer computational operations for verification than other symmetric-key-based post-quantum signature schemes that support stateless many-time signing. Notably, the performance of Loquat remains practical even when employing algebraic hash functions. Our Python-based implementations of Loquat demonstrate a signature size of 46KB, with a signing time of 5.04 seconds and a verification time of merely 0.21 seconds. Instantiating the random oracle with an algebraic hash function results in the R1CS constraints for signature verification being about 148K, 7 to 175 times smaller than those required for state-of-the-art MPC-in-the-head-based signatures and 3 to 9 times less than those for SPHINCS+ [Bernstein et al. CCS'19].

We explore two applications of Loquat. First, we incorporate it into the ID-based ring signature scheme [Buser et al. ACNS'22], achieving a significant reduction in signature size from 1.9 MB to 0.9 MB with stateless signing and practical master key generation. Our second application presents a SNARK-based aggregate signature scheme. We use the implementations of Aurora [Ben-Sasson et al. EC'19] and Fractal [Chiesa et al. EC'20] to benchmark our aggregate signature's performance. Our findings show that aggregating 32 Loquat signatures using Aurora results in a proving time of about 7 minutes, a verification time of 66 seconds, and an aggregate signature size of 197 KB. Furthermore, by leveraging the recursive proof composition feature of Fractal, we achieve an aggregate signature with a *constant* size of 145 KB, illustrating Loquat's potential for scalability in cryptographic applications.

Keywords: Post-Quantum Signature · Legendre PRF · SNARK · Aggregate Signature · ID-Based Ring Signature

1 Introduction

In the last decade, the increasing computational feasibility of quantum computing infrastructures, including quantum as a service (QaaS), has rejuvenated the cryptographic community's concern about the threats raised by quantum-enabled algorithms, notably Shor's polynomial-time quantum algorithm for factoring and computing discrete logarithms [61]. In late 2016, NIST announced a process to select quantum-resistant public key cryptographic alternatives whose security would not be undermined by quantum computers. Among the options considered by the community and stakeholders is the use of symmetric-key primitives as building blocks. Existing signature schemes based on symmetric-key primitives can be categorized into one of two paradigms: hash-based signatures [52, 22, 44, 13, 14] or MPC-in-the-head-based signatures [25, 49, 45, 59, 6, 16, 58, 30]. Compared to post-quantum digital signatures derived from lattice cryptography [31, 56], the majority of the previously mentioned signatures exhibit larger sizes and longer running times. However, symmetric-key-based approaches continue to be regarded as crucial, given the necessity of developing post-quantum cryptographic primitives using a diverse range of mathematical assumptions.

The advent of practical Succinct Non-Interactive Argument of Knowledge (SNARK) protocols shows a new direction for constructing advanced cryptographic primitives with post-quantum security, including ring signatures [23] and aggregate signatures [50]. In these schemes, the validity of signatures is proven by a SNARK protocol, which requires converting the signature verification algorithm into a language recognizable by SNARKs, such as Rank-1 Constraint System (R1CS). Consequently, the efficiency of these primitives is closely linked to the complexity of constraint systems. Despite the surge in research aimed at enhancing the performance of post-quantum ring and aggregate signatures, the area of SNARK-friendly signatures remains comparatively undeveloped. This gap serves as a key motivator for our research, which is dedicated to developing a post-quantum signature scheme that relies on symmetric-key primitives based security assumptions and is inherently SNARK-compatible (i.e., can be efficiently proven by SNARKs).

1.1 Challenges of Enabling SNARK-Compatibility using Symmetric-Key Primitives

Hash-based Paradigm. The construction of hash-based signature schemes can be traced back to the late 1970s, when Lamport [52] proposed the first hash-based one-time signature (OTS). To increase the signing capacity, an OTS can be combined with a Merkle tree [55] to obtain a hash-based many-time signature (MTS). Such constructions, culminating in XMSS [22] and its multi-tree variant [44], are quite efficient with minimal security assumptions. Nonetheless, signature schemes like XMSS suffer from stateful signing (i.e., the signer must store the used one-time secret keys somewhere and ensure they are never reused). In order to eliminate the statefulness, Bernstein et al. proposed SPHINCS [13] and

SPHINCS+ [14], which incorporated hash-based few-time signatures (FTS) into a hyper-tree (a tree of Merkle trees).

Signature verification in hash-based schemes typically involves a large number of hash function evaluations. Using a standard hash function such as SHA-256 tends to impose a substantial number of R1CS constraints (approximately 27,000 per hash evaluation). To address this issue, it becomes necessary to explore algebraic alternatives such as Poseidon [41], Rescue-prime [62], or Griffin [39]. Unfortunately, despite the optimized R1CS efficiency, algebraic hash functions suffer from slow performance when implemented in native computations.[4]

Concretely, the SageMath implementation of Griffin exhibits a running time of 7 ms per permutation for an input size of 64 bytes, while the Python implementation of SHA-256 runs in just 0.034 ms for the same input size. The performance gap stems from the fact that the round function in algebraic hash utilizes finite field multiplications, which require hundreds of CPU cycles, in contrast to the bit operations employed by traditional hash functions. Replacing the standard hash with an R1CS-friendly hash would result in impractical key generation and/or signing time for hash-based MTS schemes, including XMSS, SPHINCS, and SPHINCS+, due to the performance loss.

It is worth noting that the performance degradation caused by using algebraic hash functions is present in all signature schemes, including MPCitH-based schemes and our scheme. However, the impact on signing and/or key generation runtimes is more pronounced in hash-based schemes, as they typically require considerably more hash evaluations compared to schemes constructed with other paradigms.

MPC-in-the-Head based Paradigm. MPCitH-based post-quantum signature schemes [25, 49, 45, 59, 6, 16, 58, 30] are constructed by enabling the signer to prove to the verifier (in zero-knowledge) that he/she has a secret key k corresponding to the public key (x, y), satisfying the condition $y = f_k(x)$, where f represents a symmetric one-way function (e.g., block ciphers).

The idea is that the prover simulates an N-party computation protocol in the head, where each party is equipped with a private random tape and a secret share of the input. The simulated MPC computation results in N output shares that are sent to the verifier, together with (binding and hiding) commitments to the input shares of all parties, their random tapes, and exchanged messages (referred to as views). Upon receiving these commitments, the verifier challenges the prover to randomly open $T < N$ commitments and checks that all committed messages (and output shares) are consistent with an honest execution of the MPC protocol (i.e., by honestly re-executing the MPC protocol with T opened parties). Typically, the process is repeated M times to achieve negligible sound-

[4]An exception is Reinforced Concrete (RC) [40], which applies lookup tables in $\mathbb{F}_p$, leading to a remarkable speedup of up to 15x compared to other algebraic hash functions such as Poseidon [41]. However, this technique prevents RC from being efficiently utilized in R1CS-based SNARKs or AIR-based STARKs.

ness error, and the interaction can be eliminated by applying the Fiat-Shamir heuristic [32].

In the original version of Picnic [25], the verifier randomly opens 2-out-of-3 parties, which leads to an extensive number of parallel executions (219 for 128-bit security) as each repetition has a relatively large soundness error. Katz et al. [49] enhanced the MPCitH framework by introducing an input-independent preprocessing phase, enabling the protocol to rely on n-party MPC functionalities. Their approach involves a "cut-and-choose" mechanism, where the prover runs M preprocessing simulations, of which the verifier selects $M-\tau$ to check. The remaining τ executions are then used to run online executions of the n-party MPC protocol. Each MPC instance is then verified by revealing views of all-but-one party. This framework is generally applied in subsequent MPCitH-based schemes [45, 59, 6, 16, 58, 30].

Despite the fact that MPCitH-based signature schemes show good performance on native computers, they fall short in achieving succinct verification. In essence, the verifier undertakes similar tasks as the prover, leading to millions of R1CS constraints per verification. This inefficiency renders the framework unsuitable for SNARK-based applications.

1.2 Our Contributions

Our contributions are twofold. Firstly, we introduce a novel post-quantum signature scheme called Loquat (**L**egendre PRF-based SNARK-friendly p**o**st-**qua**ntum signa**t**ure), which stands out as the most SNARK-friendly post-quantum signature scheme constructed solely using symmetric-key primitives. We implement Loquat in Python and evaluate its performance across a range of parameters to accommodate distinct security levels.

Secondly, we demonstrate the practicality of Loquat in SNARK-based cryptographic applications, including ID-based ring signatures [23] and aggregate signatures. Our comparative analysis with existing schemes illustrates that Loquat emerges as the preferred signature scheme in scenarios where the application necessitates post-quantum security from symmetric-key primitives and stateless signing. We now provide further details of our contributions.

Contribution I. SNARK-friendly Post-Quantum Signature from the Legendre PRF. Loquat offers the following features that all prior works fail to meet these criteria on one or more counts.

- **Succinct Verification**: The verifier in Loquat has significantly fewer computational operations than the prover, making it particularly effective for proving signature validity with SNARKs.
- **Practical Key Generation and Signing**: The time required for key generation and signing remains practical when using algebraic hash functions.
- **Security Assumptions based on Symmetric-Key Primitives**: The security of Loquat solely relies on symmetric-key primitives, specifically collision-resistant hash functions and the Legendre PRF.

Our construction of Loquat does not rely on the existing MPC-in-the-Head approach or on using a *generic* zkSNARK protocol.[5] Instead, we show a novel reduction that transforms the task of designing a key identification protocol for the Legendre PRF into the task of designing a multivariate polynomial commitment. This reduction facilitates succinct verification. We further optimize the protocol in terms of SNARK-friendliness using multiple methods, including a novel approach to verify quadratic residuosity in the SNARK circuit.

We implement Loquat using Python and test its performance on a 2021 MacBook Pro with an Apple M1 Max chip and 32GB RAM. Despite the inherent performance degradation in Python programs, the signing and verification times of our scheme are 5.04 and 0.21 seconds, respectively, with a signature size of 57KB, targeting standard 128-bit security. When replacing the standard hash function with Griffin [39], the signing time extends to 105 seconds (284 times faster than the signing of SPHINCS+ when instantiated with the same algebraic hash function), while the verification process completes in 11 seconds.

Compared to Rainier [30], a state-of-the-art symmetric-key primitives-based signature scheme, our signature size is 6.8 times larger. However, Loquat's verification circuit is notably smaller, being 176 times smaller than that of Rainier. It is important to note that direct comparisons of proving and verification times between Loquat and Rainier should be approached with caution due to the differences in implementation languages (C++ for Rainier versus Python for Loquat). More comparisons can be found in Table 1.[6]

Scheme	$\|\sigma\|$ (KB)	# R1CS	# Hash (KG)	# Hash (Sign)
Picnic1 [25]	32	3,451,440	0	7,008
Picnic3 [64]	12	21,598,340	0	1,679,784
LegRoast [16]	16	1,143,351	0	10,152
Banquet [6]	12	11,804,560	0	4,147
Rainer [30]	8	26,130,303	0	3,337
SPHINCS+s [14]	7.8	458,920	305,663	2,397,129
SPHINCS+f [14]	16	1,390,209	4,775	111,353
Loquat-128	57	148,825	0	8,434

Table 1. Comparison of Post-Quantum Signature Schemes based on Symmetric-Key Primitives. $|\sigma|$ = signature size, # R1CS = number of R1CS constraints for verifying one signature, # Hash (KG)/# Hash (Sign) = the number of hash invocations in key generation/signing, respectively.

[5]By generic zkSNARKs, we mean zkSNARK protocols that prove arbitrary circuits, which suggest a naive way to replace the MPCitH paradigm. However, signature schemes based on generic zkSNARKs result in prohibitively large signatures (more than 200KB).

[6]The results in Table 1 for other signatures are estimated.

Contribution II. Applying Loquat in SNARK-based Cryptographic Primitives. We introduce two key applications for SNARK-friendly signatures. The first application is a post-quantum ID-based ring signature proposed in [23]. By incorporating Loquat into the existing framework, we achieve a notable reduction in the ring signature's size (from 1.898 MB to 0.973 MB) compared to the stateless Picnic-based scheme with a ring of size 2^6. Furthermore, in comparison to the stateful XMSS-based ring signature, the use of Loquat leads to a dramatic decrease in the time required to generate the master public key, dropping from 3,355 seconds to just 0.1 seconds, albeit with a slight increase in the ring signature's size.

Our second application focuses on SNARK-based post-quantum aggregate signatures. We first construct an aggregate signature by employing Aurora [11], a transparent post-quantum SNARK. This approach yields a signature size that is polylogarithmic and a verification time that is linear in the number of signers. Subsequently, we illustrate the feasibility of developing an aggregate signature with a *constant* signature size by leveraging SNARKs with succinct verifications like Fractal [27]. We conducted our experiments using the open-source implementations of Aurora and Fractal in libiop [1]. These experiments were carried out on an Ubuntu virtual machine (allocated 28GB memory) hosted on a system equipped with 32GB of RAM and an i7-12700KF CPU operating at 3.6GHz. We demonstrate the *maximum* number of signatures that can be aggregated using Aurora and Fractal within the memory constraints of 28GB in Table 2.[7] Moreover, owing to the quasi-linear proving time and linear verification time of Aurora, aggregating N Loquat signatures can be achieved 3 to 175 times faster than aggregating the same number of signatures based on other symmetric-key primitives.

Scheme	# Sig	# R1CS	Sig. Size (KB)	Agg. Time (s)	Ver. Time (s)
SPHINCS+s [14]	18(4)	$2^{23}(2^{21})$	207(145)	553(556)	62(0.78)
LegRoast [16]	7(1)				
Rainer [30]	0(0)				
Picnic3 [64]	0(0)				
Banquet [6]	0(0)				
Loquat	64(16)				

Table 2. Aurora and Fractal Aggregation of Different Symmetric-Key Primitives based Post-Quantum Signatures. # Sig = maximum number of signatures can be aggregated with 28GB memory, # R1CS = maximum number of R1CS constraints that can be proven with 28GB memory, red = Aurora-based aggregation, blue = Fractal-based aggregation.

[7] Since the implementation of Aurora [11] and Fractal [27] support R1CS constraints that are powers of 2, the maximum number of signatures in Table 2 is computed as the floor of 2^{23} divided by the number of R1CS constraints of the signature verification.

Remark 1 (From ROM to the URS model). While we prove the EUF-CMA of LOQUAT in the random oracle model (ROM), for the aforementioned SNARK-based applications, we need to leave the model by *heuristically* instantiating the random oracle with a (collision-resistant) cryptographic hash function and start from LOQUAT in the URS model. The reason for this is not merely motivated by implementation as we now explain. Since the verifier of LOQUAT makes calls to the random oracle, and proving the validity of LOQUAT in ID-based ring signature and aggregate signature schemes requires a SNARK which can prove the correctness of computations that involve oracle gates. Previous research has shown that such SNARKs do not exist [26]. Hence, we have to instantiate the random oracle such that the SNARK proves the computation of cryptographic hash functions instead of the oracle gates. We note that random oracles cannot be properly instantiated in general, and so we make the same heuristic assumption as in other works [43, 27, 7]: there is a secure instantiation of the random oracle for the key identification protocol produced via Theorem 1.

It is worth noting that the reliance on the heuristic instantiation of the random oracle model presents a widespread challenge for *stateless* post-quantum signature schemes in the context of SNARK-based applications. This issue has spurred a line of research aimed at addressing and potentially circumventing this theoretical obstacle.

1.3 Related Works

The first practical symmetric-key primitives-based aggregate signature scheme was introduced in [50], which compresses hash-based *one-time* signatures (i.e., Lamport+) using STARK [9], a variant of SNARK. Owing to the succinctness of STARK, the aggregate signature size grows sublinearly with the number of signers. Furthermore, due to the simplicity of the design of one-time signatures, the concrete performance approaches 130KB when aggregating 128 signatures. Nonetheless, one-time signatures are stateful, requiring the signer to keep track of all used secret keys. This requirement is often not practical in real-world applications because it is often impossible to continuously update a key pair, and storing the used secret keys can pose significant challenges.

Beyond symmetric-key primitives, the construction of aggregate signatures and multi-signatures has seen notable advancements through alternative post-quantum techniques, particularly lattice-based methods [19, 21, 20, 34, 29]. The advent of practical lattice-based multi-signatures, such as Chipmunk [33], underscores the potential of lattices in developing advanced cryptographic primitives since, unlike symmetric-key primitives, lattices offer distinct advantages owing to the underlying algebraic structures. However, the quantitative security level of lattice-based cryptography is unclear despite their reasonably efficient performance [13]. This is also evident from NIST's selection of SPHINCS+ [14] for standardization. Despite its significant efficiency disadvantages compared to lattice-based alternatives, there is a practical and theoretical need to develop cryptographic schemes based on different cryptographic assumptions. As such, exploring post-quantum cryptographic primitives based on a range of security assumptions becomes critically important.

In this work, we focus on narrowing the performance gap between advanced cryptographic primitives based on lattices and symmetric-key primitives. We achieve this by introducing a SNARK-friendly signature scheme that stands out for its reliance on distinct security assumptions compared to lattice-based alternatives. This approach not only offers practical benefits but also presents theoretical interest in the realm of cryptography.

2 Technical Overview

We start with an overview of the key identification problem for the Legendre PRF, followed by the description of the identification protocol based on MPC-in-the-head as presented in LegRoast [16]. We then introduce our approach of replacing the MPC-in-the-head with a polynomial commitment scheme, which facilitates succinct verification.

2.1 Key Identification Problem of the Legendre PRF

The Legendre PRF (Section 3.1) is a one-way function denoted by $\mathcal{L}_K(\cdot) : \mathbb{F}_p \to \mathbb{Z}_2$, which on input a random field element $a \in \mathbb{F}_p$ (and a hard-coded secret key K), outputs a bit indicating the quadratic residuosity of $(K + a)$. Owing to the limitation that the PRF only produces one bit of output, the public key should consist of many PRF evaluations denoted as $pk = (pk_1, \dots, pk_L) = (\mathcal{L}_K(I_1), \dots, \mathcal{L}_K(I_L))$ for some fixed and publicly defined arbitrary list $\mathcal{I} = (I_1, \dots, I_L)$. The key identification problem is described as follows: given a public key consists of L Legendre PRF outputs, prove the knowledge of a value K' such that $\mathcal{L}_{K'}(I_\ell) = \mathcal{L}_K(I_\ell)$ for all $\ell \in [L]$ simultaneously.

2.2 MPCitH-based Identification Protocol [16].

A straightforward way to prove the knowledge of K' would involve applying a generic zero-knowledge proof demonstrating that, for all $\ell \in [L]$, the evaluation of $\mathcal{L}$ on the secret input K' and public input I_ℓ produces the exact bit as pk_ℓ. However, this approach is prohibitively expensive due to the following reasons:

1. The circuit of evaluating $\mathcal{L}$ requires $O(\log p)$ operations, where p is a large prime.
2. L is a large integer in the case of key identification.

Grassi et al. [42] addressed the first problem by manipulating the multiplicative homomorphism of the PRF. Namely, for $\ell \in [L]$, the prover employs a zero-knowledge proof to demonstrate the correct computation of public values o_ℓ that is defined as $o_\ell = (K' + I_\ell)r_\ell$, for some random $r_\ell \in \mathbb{F}_p$. If o_ℓ is computed honestly and $K' = K$, then $\mathcal{L}_0(o_\ell)$ must equal to $pk_\ell + \mathcal{L}_0(r_\ell)$. The approach effectively reduces the number of arithmetic operations to be proven by the zero-knowledge proof from $O(\log p)$ to a single multiplication and addition per symbol.

To obtain a practical signature scheme, LegRoast [16] addressed the second problem by presenting a relaxation of the Legendre PRF relation. Instead of proving that the signer knows a K' such that $\mathcal{L}_0((K' + I_\ell)r_\ell) = pk_\ell + \mathcal{L}_0(r_\ell)$ for all $\ell \in [L]$, LegRoast lets a prover prove knowledge of a K' such that this holds for a large fraction of ℓ in $[L]$. The fraction is uniformly randomly chosen by the verifier after the prover commits to the claimed witness K' and the randomness r_ℓ. It has been proven in LegRoast that this relaxed PRF relation (Definition 2) is as hard as the original Legendre PRF relation (Definition 1), given appropriately chosen parameters [16, Theorem 1].

2.3 Our Approach for the Identification Scheme

Our first observation is that the objective of the MPC-in-the-head (MPCitH) in LegRoast [16] is to demonstrate that o_ℓ are computed honestly, namely, in the form of $(K' + I_\ell)r_\ell$. To achieve succinct verification, our first attempt was to replace MPCitH with a generic zkSNARK such as Fractal [27]. However, despite the small number of constraints to be proven, this approach results in large signature sizes and slow prover time. More importantly, it does not lead to sufficiently small R1CS constraints for signature verification.

We then observed that since the witness K' and the randomness r_ℓ must be committed prior to the verifier sampling the random fraction (denoted as $\mathcal{I}'$ such that $|\mathcal{I}'| = B \leq L$) of $\mathcal{I}$, this scenario can be reformulated as a *polynomial commitment* [47] problem, treating the elements $I_\ell \in \mathcal{I}'$ as unknown values and K', r_ℓ as coefficients. Thus, at the beginning of the protocol, the prover computes B linear functions $\hat{f}_\ell(x) = (K' + x)r_\ell = (K'r_\ell) + r_\ell \cdot x$ (i.e., $(K'r_\ell)$ is the constant coefficient and r_ℓ is the coefficient of the degree 1 variable in $\hat{f}_\ell(x)$). The MPCitH proof of $o_\ell = (K' + I_\ell)r_\ell$ can then be replaced by a proper polynomial commitment proof showing that $\hat{f}_\ell(I_\ell) = o_\ell$ for $I_\ell \in \mathcal{I}'$.

However, symmetric-key primitives-based post-quantum polynomial commitment schemes such as RedShift [48] are (much) more expensive than MPC-in-the-head. Hence, we followed a batching technique as follows. Instead of proving o_ℓ individually, the prover proves the random linear combination of all o_ℓ's using a *multivariate polynomial.*

Specifically, we let the prover to show that $\sum_{\ell=1}^{B} \lambda_\ell o_\ell = \sum_{\ell=1}^{B} \lambda_\ell r_\ell (K' + I_\ell) = \sum_{\ell=1}^{B} K r_\ell \cdot \lambda_\ell + r_\ell \cdot \lambda_\ell I_\ell$ for randomly chosen λ_ℓ by the verifier. We treat both verifier's challenges $(\lambda_\ell, I_\ell)_{\ell \in B}$ as variables in the first round. Therefore, the prover defines a multilinear polynomial with $2B$ variables as $\hat{f}(x_1, \ldots, x_B, y_1, \ldots, y_B) = K'r_1y_1 + r_1x_1y_1 + \cdots + K'r_By_1 + r_Bx_By_B$. Then, the prover commits to the polynomial and proves to the verifier that the evaluation of $\hat{f}$ on $(I_1, \ldots, I_B, \lambda_1, \ldots, \lambda_B)$ equals to $\sum_{\ell=1}^{B} \lambda_\ell o_\ell$. We adopted the multivaraite polynomial commitment in Virgo [65] to obtain Loquat, resulting in desired Legendre PRF-based signature scheme with succinct verification.

Efficient Verification of Quadratic Residuosity in SNARKs The aforementioned approach enables succinct signature verification. However, proving $\mathcal{L}_0(o_\ell) = pk_{I_\ell} + \mathcal{L}_0(r_\ell)$ for $\ell \in [B]$ in the SNARK circuit can incur a large overhead. We introduce the following technique to demonstrate that o_ℓ has the same quadratic residuosity as $(K + I_\ell)r_\ell$, which only requires *three multiplication gates* per symbol.

Let us fix a quadratic non-residue (QNR) $\alpha \in \mathbb{F}_p$. Then, we define

$$t_\ell = o_\ell \cdot (\alpha \cdot (pk_{I_\ell} + T_\ell) + (1 - (pk_{I_\ell} + T_\ell))). \tag{1}$$

where $T_\ell = \mathcal{L}_0(r_\ell)$. Note that, for $o_\ell \neq 0$, t_ℓ will be a quadratic residue (QR) if and only if $\mathcal{L}_0(o_\ell) = pk_{I_\ell} + T_\ell$ (i.e., non-zero o_ℓ's can be easily proved with $B + 1$ constraints in the circuit).

Therefore, if it holds true that for all $\ell \in [B]$, $\mathcal{L}_0(o_\ell) = pk_{I_\ell} + T_\ell$, we can find the square root s_ℓ of t_ℓ and prove in the circuit that $t_\ell = s_\ell \cdot s_\ell \pmod p$. It is worth emphasizing that the computation of s_ℓ values can be done *outside* of the circuit. Consequently, the total number of constraints required to verify the quadratic residuosity of o_ℓ is $4B + 1$.

3 Preliminaries

Notations Let $\mathbb{F}_p$ denote the prime field for a large prime p and $\mathbb{F} = \mathbb{F}_{p^2}$ denote the extension field of $\mathbb{F}_p$. We use bold letters to denote vectors (e.g., $\mathbf{f}$), and $\mathbf{f}[i]$ denotes the i-th element in $\mathbf{f}$. We use hatted letters to denote polynomials (e.g., $\hat{f}$). Given a set $S = (s_1, \ldots, s_n)$ and a vector $\mathbf{f} = (f_1, \ldots, f_n)$, we write $\hat{f} = \texttt{Interpolate}(S, \mathbf{f})$ the interpolation of $\hat{f}(X)$ using n points $\{(s_1, f_1), \ldots, (s_n, f_n)\}$, and $\mathbf{f} = \hat{f}|_S$ denotes the evaluation of the polynomial $\hat{f}$ over the set S.

For any $n \in \mathbb{N}$, $[n] = \{1, \ldots, n\}$, and $h \xleftarrow{\$} H$ for a set H denotes that h is chosen uniformly at random from H. The negligible function is denoted by $\texttt{negl}(\cdot)$, the empty string is $\perp$, and the security parameter is λ.

Reed-Solomon Code and Interleaved Code. Given a subset $U \subseteq \mathbb{F}$ and a rate parameter $\rho \in (0, 1]$, we denote by $RS[U, \rho] \subseteq \mathbb{F}^{|U|}$ the evaluations over U of univariate polynomials with degree less than $\rho|U|$. In other words, a codeword $\mathbf{f} \in \mathbb{F}^{|U|}$ is in $RS[U, \rho]$ if there exists a polynomial $\hat{f}$ of degree less than $\rho|U|$ and $\mathbf{f} = \hat{f}|_U$.

If there exist m linear codes $(\mathbf{f}_1, \ldots, \mathbf{f}_m)$ with alphabet $\mathbb{F}$, where $\mathbf{f}_i \in \mathbb{F}^n$ for all $i \in [m]$, the interleaved code with alphabet $\mathbb{F}^m$ is denoted by $\prod_{i=1}^m C_i \subseteq (\mathbb{F}^m)^n \equiv \mathbb{F}^{m \times n}$ that is an $m \times n$ matrix whose i-th row is $\mathbf{f}_i$.

Merkle Tree In this paper, we use Merkle tree [55] as a primitive for committing to a vector with short commitment and logarithmic-size proof. Occasionally, we require not only the commitment (i.e., Merkle root) is hiding, but also the authentication path will not reveal any information, which is called Merkle tree with privacy [12]. We will specify in the paper when we require Merkle tree with privacy. If not specified, then it runs like an original Merkle tree commitment scheme. The scheme consists of following algorithms:

- $\texttt{root}_c \leftarrow \texttt{MT.Commit}(\mathbf{c})$: On input a vector $\mathbf{c}$, the algorithm outputs a short commitment $\texttt{root}_c$.
- $(c_i, \texttt{auth}_i) \leftarrow \texttt{MT.Open}(i, \mathbf{c})$: On input an index i, a vector $\mathbf{c}$ that was committed using $\texttt{MT.Commit}$ algorithm, the deterministic algorithm outputs a leaf node $c_i \in \mathbf{c}$ and an authentication path $\texttt{auth}_i$.
- $(1/0) \leftarrow \texttt{MT.Verify}(root_c, i, c_i, \texttt{auth}_i)$: On input a Merkle root $root_c$, an index i, the leaf node c_i, and the corresponding authentication path $\texttt{auth}_i$, the deterministic algorithm outputs 1 if and only if c_i is a valid leaf node with respect to the root $root_c$. Otherwise, outputs 0.

Fast Fourier Transform Our scheme requires polynomial evaluation and interpolation, which are implemented via Fast Fourier Transform (FFT) and its inverse (IFFT), respectively. Given a multiplicative coset of size d of a finite field, the polynomial interpolation can be performed in $O(d \log d)$ field operations using IFFT. We often evaluate the polynomial on a larger subspace (of size $n > d$), which can be performed in $O(n \log d)$ field operations via FFT.

If there exist m linear codes $(\mathbf{f}_1, \ldots, \mathbf{f}_m)$ with alphabet $\mathbb{F}$, where $\mathbf{f}_i \in \mathbb{F}^n$ for all $i \in [m]$, the interleaved code with alphabet $\mathbb{F}^m$ is denoted by $\prod_{i=1}^m C_i \subseteq (\mathbb{F}^m)^n \equiv \mathbb{F}^{m \times n}$ that is an $m \times n$ matrix whose i-th row is $\mathbf{f}_i$.

3.1 Legendre PRF

The *Legendre PRF* relies on the conjectured pseudo-randomness of the Legendre symbol with a secret shift [15]. Specifically, on input $a \in \mathbb{F}_p$, the Legendre symbol outputs $\{-1, 0, 1\}$, depending on whether a is a square modulo p:

$$\left(\frac{a}{p}\right) = \begin{cases} 1, & \text{if } a = b^2 \pmod p \text{ for some } b \in \mathbb{F}_p^* \\ 0, & \text{if } a = 0 \pmod p \\ -1, & \text{otherwise} \end{cases}$$

Then, the Legendre PRF is defined as:

$$\mathcal{L}_0(a) = \left\lfloor \frac{1}{2}\left(1 - \left(\frac{a}{p}\right)\right)\right\rfloor \in \{0, 1\}$$

Namely, the PRF outputs 1 if a is a quadratic non-residue (QNR) modulo p and 0 otherwise. The Legendre PRF has multiplicative homomorphism, that is $\mathcal{L}_0(a \cdot b) = \mathcal{L}_0(a) + \mathcal{L}_0(b)$ for non-zero $a, b \in \mathbb{F}_p^*$. The keyed Legendre PRF is:

$$\mathcal{L} : \mathbb{F}_p \times \mathbb{F}_p \to \mathbb{Z}_2$$
$$\mathcal{L} : (K, a) \mapsto \mathcal{L}_0(K + a)$$

and $\mathcal{L}_0(K + a)$ is used interchangeably with $\mathcal{L}_K(a)$.

Definition 1 (Legendre PRF relation (adapted from [16])). *For an odd prime p and a list $\mathcal{I} = (I_1, \dots, I_L)$, where $I_\ell \xleftarrow{\$} \mathbb{F}_p$ for all $\ell \in [L]$, we define the Legendre PRF relation $R_{\mathcal{L}}$ as*

$$R_{\mathcal{L}} = \{(\mathcal{L}_K(\mathcal{I}), K) \in \{0,1\}^L \times \mathbb{F}_p | K \in \mathbb{F}_p\}$$

and $\mathcal{L}_K(\mathcal{I}) = (\mathcal{L}_K(I_1), \dots, \mathcal{L}_K(I_L))$.

Definition 2 (β-approximate Legendre PRF relation (adapted from [16])). *Given $\beta \in [0,1]$, an odd prime p, and a list $\mathcal{I}$ of L elements uniformly chosen from $\mathbb{F}_p$ at random, we define the β-approximate PRF relation $R_{\beta\mathcal{L}}$ as*

$$R_{\beta\mathcal{L}} = \{(s, K) \in \{0,1\}^L \times \mathbb{F}_p | \exists a \in \{0,1\} : d_H(s + (a, \dots, a), \mathcal{L}_K(\mathcal{I})) \leq \beta L\}$$

where $d_H(\cdot, \cdot)$ denotes the Hamming distance.

According to [16, Theorem 1], the β-approximate Legendre PRF relation is as hard as Legendre PRF relation with the probability at least $1 - 2p \cdot \Pr[\mathfrak{B}(L, 1/2 + 1/\sqrt{p} + 2/p) \geq (1-\beta)L]$ over the choice of $\mathcal{I}$, where $\mathfrak{B}(n, q)$ denotes the binomial distribution with n samples each with success probability q. Simply put, with the proper choice of the parameter L (sufficiently large) and β (sufficiently small) [16], if there exists a PPT algorithm that is able to find the β-approximate witness K, then K is also a witness of the exact Legendre PRF relation. For a comprehensive analysis of the quantum and classical security of the Legendre PRF, we direct interested readers to Appendix A.

3.2 Multivariate Polynomial Commitment from Univariate Sumcheck

The multivariate polynomial commitment protocol [65] allows a prover to prove the evaluation of a committed multivariate polynomial. This protocol builds upon the "univariate sumcheck" [11], an Interactive Oracle Proof (IOP) used to verify whether a given univariate polynomial $\hat{f}(x)$ sums to a specific value μ on an additive or multiplicative coset $H \subseteq \mathbb{F}$. Our construction differs from [65] in several aspects; for example, we do not use the GKR protocol [36] to reduce the verification time. Therefore, we only assume black-box access to the univariate sumcheck protocol, which can be defined as follows:

Definition 3 (Univariate Sumcheck Relation (Adapted from [11])). *The relation $R_{\mathtt{US}}$ is the set of all tuples $((\mathbb{F}, U, H, d, \mu), \hat{f})$, where $\mathbb{F}$ is a finite field, U, H are multiplicative cosets of $\mathbb{F}$, $d \in \mathbb{N}$ is the degree bound of $\hat{f}$, and $\mu \in \mathbb{F}$, such that $\mu = \sum_{a \in H} \hat{f}(a)$ and $\hat{f}|_U \in RS[L, \frac{d}{|U|}]$.*

At a high-level, the univariate sumcheck protocol allows a prover to convince a verifier that there exists a low-degree univariate polynomial $\hat{f}(x)$ such that

$\mu = \sum_{a \in H} \hat{f}(a)$ for $\mu \in \mathbb{F}$ and $H \subseteq \mathbb{F}$. The following lemma describes the foundation of the univariate sumcheck protocol.[8]

Lemma 1. *[24] Let H be a multiplicative coset of $\mathbb{F}$, and $\hat{g}(x) \in \mathbb{F}[X]$ that has degree strictly less than $|H|$. Then $\sum_{a \in H} \hat{g}(a) = \hat{g}(0) \cdot |H|$.*

We demonstrate the underlying idea of the univariate sumcheck for better readability and refer the readers to [11, Section 5] for more details. Let $\hat{f}(x) \in \mathbb{F}[X]$ of degree less than d, which can be decomposed vanto two unique polynomials $\hat{g}(x), \hat{h}(x)$ of degree less than $|H|$ and $d - |H|$, respectively, such that $\hat{f}(x) \equiv \hat{g}(x) + Z_H(x) \cdot \hat{h}(x)$, where $Z_H(x)$ is the vanishing polynomial over H (i.e., it evaluates to 0 everywhere on H). It is not hard to see that $\mu = \sum_{a \in H} \hat{f}(a) = \sum_{a \in H} \hat{g}(a)$. Then, according to Lemma 1, $\mu = |H| \cdot \hat{g}(0)$ since the degree of $\hat{g}$ is less than $|H|$. In other words, $\frac{|H|\hat{g}(x) - \mu}{|H|x}$ must be a polynomial with degree less than $|H| - 1$. Hence, checking if $\mu = \sum_{a \in H} \hat{f}(a)$ is equivalent to check:

1. $\hat{g}(x) = \hat{f}(x) - Z_H(x) \cdot \hat{h}(x)$ is computed correctly, and
2. $\frac{|H|\hat{g}(x) - \mu}{|H|x}$ is of degree less than $|H| - 1$.

Let us define the rational constraint $\hat{p}(x) \in \mathbb{F}[X]$ that can be computed as the division of a sequence of polynomials:

$$\hat{p}(x) = \frac{|H|\hat{f}(x) - |H|Z_H(x)\hat{h}(x) - \mu}{|H| \cdot x} \tag{2}$$

Note that if Condition 1 is satisfied, then the numerator of $\hat{p}(x)$ is $|H|\hat{g}(x) - \mu$. Furthermore, if the second condition is also satisfied, then $\hat{p}(x)$ is a polynomial of degree less than $|H| - 1$ since $|H|\hat{g}(x) - \mu$ should have constant term 0 if μ is indeed the sum. Therefore, the univaraite sumcheck is reduced to low-degree test of three polynomials: $\deg(\hat{p}) < |H| - 1$, $\deg(\hat{h}) < d - |H|$, and $\deg(\hat{f}) < d$.

Remark 2 (Zero-Knowledge). Univariate sumcheck can be made zero-knowledge using standard techniques [11, Section 5.1]. Namely, to mask witness polynomial $\hat{f}(x)$, the prover randomly samples a polynomial $\hat{s}(x)$ with $\deg(\hat{s}) = \deg(\hat{f})$ and computes $S = \sum_{a \in H} \hat{s}(a)$. It sends S to the verifier and commits to $\hat{s}|_U \in RS[U, \frac{|H|}{U}]$. The verifier sends a uniformly random challenge $z \xleftarrow{\$} \mathbb{F}$ to the prover, who then prove the univariate sumcheck relation with respect to the claimed sum $z\mu + S = \sum_{a \in H} z\hat{f}(a) + \hat{s}(a)$.

[8] Aurora [11] introduced univariate sumcheck over both additive and multiplicative cosets, but their protocol was mainly designed for additive cosets. We follow [65] to use univariate sumcheck over multiplicative cosets for better compatibility with the Legendre PRF.

Low-Degree Test (LDT) Univariate sumcheck uses FRI [8] for low-degree test, which allows a verifier to test whether a codeword $\mathbf{c}$ is a member of the RS-code with prescribed code rate ρ. Following previous discussion on univariate sumcheck, the prover needs to send three oracle codewords to the verifier:

1. **Witness oracle:** $\hat{f}|_U \in RS[U, \frac{d}{|U|}]$.
2. **Message oracle:** $\hat{h}|_U \in RS[U, \frac{d-|H|}{|U|}]$.
3. **Rational constraint:** $\hat{p}|_U \in RS[U, \frac{|H|-1}{|U|}]$

A straightforward way is to perform low-degree tests individually on each codeword. However, given that LDT is computationally expansive, Aurora employs a strategy that facilitates the degree test for interleaved RS codes with *different* code rates using a single LDT protocol [11, Protocol 8.2]. At a high-level, this involves multiplying each polynomial by an appropriate monomial in such a way that all of them have the same (maximum) code rate. Subsequently, a random linear combination of the interleaved code is computed. The resulting codeword, with code rate ρ^*, is then taken as the input to the low-degree test protocol.

3.3 Digital Signature

Definition 4 (Syntax). *A signature scheme consists of the following algorithms* $(\texttt{Setup}, \texttt{KeyGen}, \texttt{Sign}, \texttt{Verify})$*:*

1. $(\texttt{pp}) \leftarrow \texttt{Setup}(\lambda)$*. On input a security parameter λ, the algorithm generates a set of public parameters denoted as* $\texttt{pp}$*.*
2. $(sk, pk) \leftarrow \texttt{KeyGen}(\texttt{pp})$*: On input public parameters* $\texttt{pp}$ *generated in the setup phase, the algorithm outputs a key pair (sk, pk).*
3. $\sigma \leftarrow \texttt{Sign}(sk, m, \texttt{pp})$*: On input a secret key sk, a message m, and public parameters pp, the (randomized) algorithm outputs a signature σ.*
4. $(0/1) \leftarrow \texttt{Verify}(pk, m, \sigma, \texttt{pp})$*: On input a public key pk, a message m, a signature σ, and public parameters pp, the (deterministic) algorithm outputs 1 (accept) or 0 (reject).*

A signature scheme should be correct and existential unforgeable under adaptive chosen message attacks (EUF-CMA) [38]. We first define the correctness of the signature scheme.

Definition 5 (Correctness). *For any message m,*

$$\Pr\left[\texttt{Verify}(pk, m, \sigma) = 1 \;\middle|\; \begin{array}{l} (sk, pk) \leftarrow \texttt{KeyGen} \\ \sigma \leftarrow \texttt{Sign}(sk, m) \end{array}\right] = 1$$

We first reduce the EUF-KO (existential unforgeability against key-only attacks) of Loquat to the β-approximate Legendre PRF relation as a stepping stone, then we reduce EUF-CMA (existential unforgeability against (adaptive) chosen message attacks) to EUF-KO. We define the two security models as follows.

Definition 6 (EUF-KO). *Given a security parameter λ, we say that the signature scheme is* EUF-KO*-secure in the random oracle model if any PPT algorithm $\mathcal{A}$ has negligible advantage in the* EUF-KO *game defined as*

$$\mathbf{Adv}_{\mathcal{A}}^{\texttt{EUF-KO}} = \Pr\left[\texttt{Verify}(pk, m^*, \sigma^*) = 1 \;\middle|\; \begin{array}{l} (sk, pk) \leftarrow \texttt{KeyGen}(\lambda) \\ (m^*, \sigma^*) \leftarrow \mathcal{A}(pk) \end{array}\right]$$

Definition 7 (EUF-CMA). *Given a security parameter λ, we say that the signature scheme is* EUF-CMA*-secure in the random oracle model if any PPT algorithm $\mathcal{A}$ has negligible advantage in the* EUF-CMA *game defined as*

$$\mathbf{Adv}_{\mathcal{A}}^{\texttt{EUF-CMA}} = \Pr\left[\begin{array}{l}\texttt{Verify}(pk, m^*, \sigma^*) = 1 \\ \wedge\, m^* \notin Q \end{array} \;\middle|\; \begin{array}{l} (sk, pk) \leftarrow \texttt{KeyGen}(\lambda) \\ (m^*, \sigma^*) \leftarrow \mathcal{A}^{\texttt{Sign}(sk,\cdot)(pk)} \end{array}\right]$$

where $\mathcal{A}^{\texttt{Sign}(sk,\cdot)}$ denotes $\mathcal{A}$'s access to a signing oracle with private key sk, and Q denotes the set of messages m that are queried to the signing oracle by $\mathcal{A}$.

3.4 Aggregate Signature

Definition 8 (Syntax). *An aggregate signature scheme consists of the following algorithms* ($\texttt{Setup}, \texttt{KeyGen}, \texttt{Sign}, \texttt{Verify}, \texttt{Agg}, \texttt{AggVerify}$)*:*

1. $\texttt{Setup}, \texttt{KeyGen}, \texttt{Sign}, \texttt{Verify}$ *are defined the same as in ordinary signature schemes, except that* Setup *also generates the public parameters for the aggregate signature scheme.*
2. $\Sigma \leftarrow \texttt{Agg}(\{\sigma_i, m_i, pk_i\}_{i\in[n]}, \texttt{pp})$*: On input n signature, message, public-key pairs, where (σ_i, m_i, pk_i) denotes the pair for the i-th signer, and a set of public parameters* pp *generated in the* Setup *phase, the algorithm outputs Σ which is the aggregated signature.*
3. $(0/1) \leftarrow \texttt{AggVerify}(\Sigma, PK, M, \texttt{pp})$*: On input public parameters* pp*, the aggregated signature Σ, n public keys $PK = (pk_1, \ldots, pk_n)$ and messages $M = (m_1, \ldots, m_n)$, the algorithm accepts (outputs 1) if the aggregate signature is valid against PK and M. Otherwise, the algorithm rejects (outputs 0).*

We follow the *aggregate chosen-key* model from [18] for our aggregate signature, in which the adversary $\mathcal{A}$ is given a challenge public key and a signing oracle with respect to the challenge key. $\mathcal{A}$ can generate other key pairs and attempt to forge an aggregate signature. The definition of the aggregate chosen-key game $\texttt{Game}^{\texttt{AggSig}}_{\mathcal{H},\texttt{Sign}}(\mathcal{A}, N, q_H, q_s)$ as follows.

Definition 9 (Aggregate Chosen-Key Model [18]). *The game contains four phases as follows.*

- ***Setup**. Define* pp *to be the public parameters for the signature scheme. The adversary $\mathcal{A}$ is given a challenge public key pk_1 chosen at random and public parameters* pp.

- ***Queries***. *The adversary $\mathcal{A}$ with pk_1 can make at most q_H queries to the random oracle and at most q_s queries to the signing oracle* `Sign` *with respect to pk_1 on messages of his choice.*
- ***Response***. *$\mathcal{A}$ outputs $N-1$ distinct public keys $pk_2, \ldots, pk_N$, a tuple of messages $m_1, \ldots, m_N$ where m_1 was not queried to the signing oracle, and an aggregate signature Σ.*
- ***Output***. *Denote $PK = \{pk_1, \ldots, pk_n\}$ and $M = \{m_1, \ldots, m_n\}$. The game outputs* $\texttt{AggVerify}(\Sigma, PK, M)$.

The advantage of the adversary, which is denoted as $\mathbf{Adv}_{\mathcal{A}}^{\texttt{AggSig}}$, is defined to be the probability $\texttt{Game}_{\mathcal{H},\texttt{Sign}}^{\texttt{AggSig}}(\mathcal{A}, N, q_H, q_s)$ outputs 1. The aggregate signature is secure against any efficient adversary under the aggregate chosen-key model if $\mathbf{Adv}_{\mathcal{A}}^{\texttt{AggSig}}(N, q_H, q_s)$ is negligible in the security parameter.

3.5 Interactive Oracle Proof

Interactive oracle proofs (IOPs) [12], also known as probabilistically checkable interactive proofs (PCIPs) [57], generalise interactive proofs (IPs) [37] and probabilistically checkable proofs (PCPs) [4, 5]. In this paper, we consider only public-coin IOP protocols (i.e., the messages from the verifier are sampled uniformly at random and are independent from messages sent by the prover).

Definition 10 (Interactive Oracle Proof (adapted from [12])). *An interactive oracle proof system for relation R with round complexity $k : \{0,1\}^* \to \mathbb{N}$ and soundness $s : \{0,1\}^* \to [0,1]$ is a tuple $(\mathcal{P}, \mathcal{V})$ where $\mathcal{P}, \mathcal{V}$ are probabilistic algorithms, that satisfies the following properties.*

1. Completeness. *For every $(x, w) \in R$, $\langle P(x, w), V(x)\rangle$ is a $k(x)$-round interactive oracle protocol with accepting probability* 1.
2. Soundness. *For every x not in the language of R and $\tilde{P}$, $\langle \tilde{P}, V(x)\rangle$ is a $k(x)$-round interactive oracle protocol with accepting probability at most $s(x)$.*

Public-coin IOPs can be complied to non-interactive random oracle proofs (NIROPs) using BCS transform [12] which preserves the proof-of-knowledge and zero-knowledge properties of the underlying IOP. The soundness of the NIROP depends on the (restricted) *state-restoration soundness* which relates to the standard soundness and the round complexity of the IOP protocol.

3.6 Succinct Non-Interactive Argument of Knowledge (SNARK)

SNARKs consist of the following three algorithms, which are defined as:

Definition 11 (Syntax).

- $\texttt{SNARK-pp} \leftarrow \texttt{SNARK-Setup}(\lambda)$. *On input a security parameter, the algorithm outputs a set of public parameters that everyone can access (hence transparent).*

- $\pi_{\texttt{SNARK}} \leftarrow \texttt{SNARK-P}(\texttt{SNARK-pp}, \mathbb{w}, \mathbb{x})$. *On input a set of public parameters* SNARK-pp *and a witness-statement pair* $(\mathbb{x}, \mathbb{w})$*, the algorithm outputs a succinct proof* $\pi_{\texttt{SNARK}}$.
- $0/1 \leftarrow \texttt{SNARK-V}(\texttt{SNARK-pp}, \mathbb{x}, \pi_{\texttt{SNARK}})$. *On input a set of public parameters* SNARK-pp*, a claimed statement* $\mathbb{x}$ *and the proof* $\pi_{\texttt{SNARK}}$*, the algorithm outputs* 1 *(accept) if* $\pi_{\texttt{SNARK}}$ *is convincing.*

If the SNARK is a preprocessing SNARK (e.g., Fractal [27]), then the protocol additionally consists of an indexer algorithm, denoted as `SNARK-Indexer`, which on input an arithmetic circuit, outputs a short *ivk* and a long *ipk* for the SNARK verifier and the SNARK prover, respectively.

A SNARK protocol satisfies correctness and argument of knowledge, which are defined in

Definition 12 (Security Properties).

1. Completeness. *For every* $(\mathbb{x}, \mathbb{w}) \in R$*,*

$$\Pr[\texttt{SNARK-V}(\mathbb{x}, \texttt{SNARK-P}(\mathbb{x}, \mathbb{w}))(\texttt{SNARK-pp}) = 1] = 1$$

2. Knowledge Soundness. *For all PPT adversaries* $\mathcal{A}$ *there exists a PPT extractor* $\mathcal{E}$ *such that*

$$\Pr\left[\begin{matrix}\texttt{SNARK-V}(\texttt{SNARK-pp}, \mathbb{x}, \pi) = 1 \\ \wedge\ (\mathbb{x}, \mathbb{w}) \notin R\end{matrix}\ \middle|\ ((\mathbb{x}, \pi); \mathbb{w}) \leftarrow \mathcal{A} \parallel \mathcal{E}(\texttt{SNARK-pp})\right] = \texttt{negl}$$

4 Loquat: SNARK-Friendly Post-Quantum Signature from the Legendre PRF

4.1 IOP-based Interactive Key Identification Scheme

We first present an Interactive Oracle Proof-based key identification scheme for the Legendre PRF, which is then compiled into a signature scheme (i.e., Loquat) using the BCS transform [12].

Interactive Oracle Proof (IOP) Generally speaking, an (public coin) IOP has three phases, interaction phase, query phase, and decision phase that can be described as follows. During the interaction phase, the prover and the verifier engage in k rounds of interaction, where in each round $i \in [k]$, the verifier sends a message $\mathbf{m}_i$ to the prover, who reads in full and responds with oracle messages $\mathbf{f}_i$, along with a plaintext message $\mathbf{p}_i$. The verifier makes point queries to the oracle messages, while he/she reads the plaintext message in its entirety.[9]

After k rounds of interaction, the (public coin) verifier initiates the query phase by making queries to the oracle messages $(\mathbf{f}_1, \ldots, \mathbf{f}_k)$ and subsequently receives responses $(\mathbf{a}_1, \ldots, \mathbf{a}_k)$. Finally, the verifier decides whether to accept or reject the proof, based on the answers $(\mathbf{a}_1, \ldots, \mathbf{a}_k)$, the verifier's messages $(\mathbf{m}_1, \ldots, \mathbf{m}_k)$, and the plaintext message $(\mathbf{p}_1, \ldots, \mathbf{p}_k)$.

[9] Occasionally, the oracle message or the plaintext message may be an empty string.

Key Identification of the Legendre PRF Fix two integers L and B, where L denotes the bit length of the Legendre PRF statement, and $B \leq L$ denotes the number of challenged quadratic residuosity symbols. Define a set of publicly accessible strings denoted as $\mathcal{I} = (I_1, \ldots, I_L)$, which are uniformly selected at random from the field $\mathbb{F}_p$ for a large prime p. The prover proceeds to choose a random $K \overset{\$}{\leftarrow} \mathbb{F}_p^*$ as the PRF private key. It then computes the PRF output $\mathcal{L}_K(\mathcal{I}) = (\mathcal{L}_K(I_1), \ldots, \mathcal{L}_K(I_L)) \in \{0,1\}^L$. The objective of the prover is to convince the verifier that it has knowledge of the β-approximate PRF key K' with respect to the output $\mathcal{L}_K(\mathcal{I})$.

Moreover, the prover and the verifier agree on two disjoint multiplicative subgroups $H, U \subseteq \mathbb{F}$, where $|H| = B$ and $|U| > |H|$. We denote the interactive key identification protocol as $\langle \mathcal{P}(K, \mathcal{L}_K(\mathcal{I})), \mathcal{V}(\mathcal{L}_K(\mathcal{I}))\rangle$, which uses, as a black-box, the zero-knowledge univariate sumcheck protocol $\langle \mathcal{P}_{\texttt{US}}(x, w), \mathcal{V}_{\texttt{US}}(x)\rangle$ for the relation $R_{\texttt{US}} = (x, w)$ as shown in Definition 3. We present the IOP-based key identification protocol in Algorithm 1.

Remark 3 (Achieving Zero-Knowledge). Query answers to $\mathbf{f}_1$ may leak information about the PRF key K. We can employ a similar technique used in Vigro [65] to mask the polynomial $\hat{c}$. Let $|Q|$ be the number of queries made by $\mathcal{V}$. The prover $\mathcal{P}$ randomly selects $\hat{r}(x)$ with degree $|Q|$, and computes $\hat{c}'(x) = \hat{c}(x) + Z_H(x)\hat{r}(x)$, where $Z_H(x)$ is the vanishing polynomial (i.e., $Z_H(h) = 0$ for all $h \in H$). The prover $\mathcal{P}$ computes the witness oracle as $\mathbf{f}_1 = \hat{c}'|_U$. Note that the univariate sumcheck polynomial is now computed as $\hat{w}(x) = \hat{c}'(x) \cdot \hat{q}(x)$, with degree less than $4B + |Q|$, and the claimed sum $\mu = \sum_{b=1}^{B} \lambda_b o_B$ remains the same. Consequently, any $|Q|$ queries to $\mathbf{f}_1$ are randomly distributed, ensuring the zero-knowledge property of the key identification protocol.

Theorem 1. *Suppose there exists an IOP protocol $\langle \mathcal{P}_{\texttt{US}}(x, w), \mathcal{V}_{\texttt{US}}(x)\rangle$ for the univariate sumcheck relation $R_{\texttt{US}}$ (Definition 3), where $x = (\mathbb{F}, U, H, d, \mu)$ and $w = \hat{f}$ that is knowledge sound and zero-knowledge with following parameters [11]:*

- Knowledge error: $\epsilon_{\texttt{US}} = O\left(\frac{|U|}{|\mathbb{F}|}\right) + \texttt{negl}(\kappa)$*, where κ denotes the repetition parameter of the low-degree test,*
- Proof length: $p_{\texttt{US}} = O(|U|)$,
- Query complexity: $q_{\texttt{US}} = O(\kappa)$,
- Round complexity: $r_{\texttt{US}} = O(\log |U|)$,
- Prover complexity: $t_{\texttt{US}}^{P} = O(|U|)$,
- Verifier complexity: $t_{\texttt{US}}^{V} = O(\log^2 |H|) + O(\log |U|)$.

Then, we can construct a public-coin IOP protocol $\langle \mathcal{P}(\mathbb{w}, \mathbb{x}), \mathcal{V}(\mathbb{x})\rangle$ (Algorithm 1) where $\mathbb{w} = K$ and $\mathbb{x} = \mathcal{L}_K(\mathcal{I})$ for the β-approximate Legendre PRF relation $R_{\beta\mathcal{L}}$ (Definition 2), assume the black-box access to $\langle \mathcal{P}_{\texttt{US}}(x, w), \mathcal{V}_{\texttt{US}}(x)\rangle$. The protocol is complete, knowledge sound, and zero-knowledge with following complexity parameters.

- Knowledge error: $\epsilon(\mathbb{x}) = \epsilon_{\texttt{US}} + (1 - \beta)^B + 1/p$,

Algorithm 1: IOP-based Key Identification of the Legendre PRF

```
Interactive Phase
    Prover Round 1
        Randomly pick $(r_1, \dots, r_B) \in \mathbb{F}_p^*$ and a random polynomial
          $\hat{r}(x) \in \mathbb{F}[X]$ of degree $|Q|$.
        Compute witness vector $\mathbf{c} = (Kr_1, r_1, \dots, Kr_B, r_B)$ and witness
          polynomial $\hat{c} = \texttt{Interpolate}(H, \mathbf{c})$.
        Compute $\hat{c}'(x) = \hat{c}(x) + Z_H(x)\hat{r}(x)$, where $Z_H(x)$ is the vanishing
          polynomial on set $H$.                       // $\deg(\hat{c}') \leq 2B + |Q|$
        for $b = 1$ to $B$ do
            Compute $T_b = \mathcal{L}_0(r_b)$.
        Send the witness oracle $\mathbf{f}_1 = \hat{c}'|_U$ and plaintext message
          $\mathbf{p}_1 = (T_1, \dots, T_B)$ to $\mathcal{V}$.
    Verifier Round 1
        Randomly sample $\mathcal{I}' \subset \mathcal{I}$, with $|\mathcal{I}'| = B$ and send $\mathbf{m}_1 = \mathcal{I}'$ to $\mathcal{P}$.
    Prover Round 2
        for $b = 1$ to $B$ do
            Compute $o_b = (K + I_b)r_b$ for $I_b \in \mathcal{I}'$.
        Send the plaintext message $\mathbf{p}_2 = (o_1, \dots, o_B)$ to $\mathcal{V}$.          // The
          oracle message in this round is empty.
    Verifier Round 2
        Randomly samples $(\lambda_b)_{b \in [B]} \xleftarrow{\$} \mathbb{F}_p$.
        Send $\mathbf{m_2} = (\lambda_1, \dots, \lambda_B)$ to $\mathcal{P}$.
    Prover Round 3
        Compute $\mathbf{q} = (\lambda_1, \lambda_1 I_1, \dots, \lambda_B, \lambda_B I_B)$ and the evaluation
          polynomial $\hat{q}(x) = \texttt{Interpolate}(H, \mathbf{q})$.          // $\deg(\hat{q}) < 2B$
        Compute the polynomial for univariate sumcheck
          $\hat{w}(x) = \hat{c}'(x) \cdot \hat{q}(x)$.                 // $\deg(\hat{w}) < 4B + |Q|$
        Compute the claimed sum $\mu = \sum_{b=1}^{B} \lambda_b o_b$.
    Univariate Sumcheck Round
        Invoke $\langle \mathcal{P}_{\texttt{US}}(x, \hat{w}), \mathcal{V}_{\texttt{US}}(x) \rangle$, where $x = (\mathbb{F}, U, H, 4B + |Q|, \mu)$.
Query Phase
    $\mathcal{V}$ samples a random set $Q \subset U$ and send $Q$ to $\mathcal{P}$, who responds with
      $(\mathbf{a}_1, \mathbf{a}_{\texttt{US}})$, where $\mathbf{a}_1[i] = \mathbf{f}_1[i]$ for $i \in [Q]$ and $\mathbf{a}_{\texttt{US}}$ is the query answer
      to oracles generated in the univariate sumcheck.
Decision Phase
    $\mathcal{V}$ outputs 1 (i.e., accept) if and only if
      1. $\mathcal{V}_{\texttt{US}}(x)$ accepts.
      2. For all $b \in [B]$, $\mathcal{L}_0(o_b) = \mathcal{L}_K(I_b) + T_b$ where $o_b \neq 0 \pmod p$ and $I_b \in \mathcal{I}'$.
    Otherwise, $\mathcal{V}$ outputs 0 (i.e., reject).
```

- Proof length: $p(\mathbb{x}) = |U| \cdot |\mathbb{F}| + (\log p + 1)B + p_{\mathtt{US}}$,
- Query complexity $q(\mathbb{x}) = \kappa \cdot 2^{\eta} + q_{\mathtt{US}}$, *where* η *denotes the localization parameter in the low-degree test,*
- Round complexity $r(\mathbb{x}) = 2 + r_{\mathtt{US}}$,
- Prover complexity $t_P(\mathbb{x}) = 2 \cdot FFT(IFFT(|H|), |U|) + O(\log p) + t^{p}_{\mathtt{US}}$,
- Verifier complexity $t_V(\mathbb{x}) = B \cdot O(\log p) + t^{V}_{\mathtt{US}}$.

Where FFT denotes the complexity of running one FFT to evaluate the polynomial and $IFFT$ denotes the complexity of running one $IFFT$ to interpolate the polynomial.

We show a formal proof and analysis for Theorem 1 in Appendix, Section B.1.

According to [12, Lemma 5.7], an IOP protocol with $k(\mathbb{x})$ rounds of interaction has (restricted) state-restoration soundness error (knowledge error) $\bar{\epsilon}(b, \mathbb{x}) \leq \binom{b}{k(\mathbb{x})}\epsilon(\mathbb{x})$, where b denotes the maximum number of interactions allowed between the state restoration prover and the verifier. However, the general upper bound of the state-restoration soundness is not enough for our key identification protocol. We analyze a tighter upper bound of the (restricted) state-restoration soundness as shown in Lemma 2, and provide the formal proof in Appendix B.2.

Lemma 2. *The restricted state-restoration soundness error (knowledge error) of the IOP-based key identification protocol for the Legendre PRF against a b-bound malicious prover $\tilde{\mathcal{P}}$ is $\bar{\epsilon}(b, \mathbb{x}) = \Pr[X + Y + Z = B]$ where $X = \max(X_1, \ldots, X_{b_1})$, $Y = \max(Y_1, \ldots, Y_{b_2})$, and $Z = \max(Z_1, \ldots, Z_{b_3})$ such that $b_1 + b_2 + b_3 \leq b$, X_i are i.i.d. as $\mathfrak{B}(B, (1-\beta))$, Y_i are i.i.d. $\mathfrak{B}(B-X, 1/p)$, and Z_i are i.i.d. as $\mathfrak{B}(B - X - Y, \epsilon_{\mathtt{US}})$, where $\mathfrak{B}(\cdot, \cdot)$ denotes the binomial distribution.*

Remark 4. Note that the security of the Legendre PRF is well-studied in the prime field $\mathbb{F}_p$ while the univaraite sumcheck works in the extension field $\mathbb{F} = \mathbb{F}_{p^2}$. To solve the inconsistency between the fields, we will naturally lift the elements in $\mathbb{F}_p$ to $\mathbb{F}_{p^2}$ for vectors $\mathbf{c}, \mathbf{q}$, where $\mathbb{F}_{p^2}$ only affects the statistical soundness of the univariate sumcheck, and it is independent from the Legendre PRF.

4.2 Loquat: Post-Quantum Signature Scheme from the Legendre PRF

Loquat (Algorithm 2 - 7) is constructed by applying BCS transform [12] to the IOP-based key identification scheme of the Legendre PRF presented in Section 4.1. Note that in key generation (Algorithm 3), we require $K + I_{\ell} \neq 0 \pmod{p}$ for enabling efficient residuosity check in SNARK-based applications, which reduces the key space from p to $p - L$. This remains secure since $L \ll p$.

We first show the EUF-KO (ROM) security of Loquat in Theorem 2, which follows directly from [12, Theorem 7.1]. Then, we prove the EUF-CMA (ROM) of Loquat, using EUF-KO as the stepping stone, in Theorem 3.

Algorithm 2: LOQUAT Setup

```
Input: Security Parameter $\lambda$
Output: Public Parameter L-pp
1  L-Setup
2      Public Parameters for Legendre PRF
3          $\mathbb{F}_p$: prime field for sufficiently large $p$.
4          $L \in \mathbb{N}$: the number of bits in the public key.
5          $B \in \mathbb{N}$: the number of challenged residuosity symbols and $B \leq L$.
6          $\mathcal{I} = \{I_1, \dots, I_L\}$ where $I_\ell \xleftarrow{\$} \mathbb{F}_p$ for all $\ell \in [L]$.
7          $m, n \in \mathbb{N}$: degree bound and the number of parallel executions
             such that $m \times n = B$ with $m$ being a power of 2.
8      Public Parameters for Univariate Sumcheck and LDT
9          $\mathbb{F} = \mathbb{F}_p^2$: extension field of $\mathbb{F}_p$ that contains large enough smooth
             multiplicative subgroups.
10         $H \subseteq \mathbb{F}$: a multiplicative coset with $|H| = 2m$.
11         $\eta$: the localisation parameter of LDT.
12         $\kappa$: the query repetition parameter of LDT.
13         $U$: a smooth multiplicative coset $U \subseteq \mathbb{F}$ such that $|U| > |H|$ and
             $H \cap U = \emptyset$.
14         $\rho^*$: maximum rate that is the closest power of 2, where
             $\rho^* > \frac{4m+\kappa \cdot 2^\eta}{|U|}$.
15         $r = \left\lfloor \frac{\log_2(|U|) - \log_2(1/\rho^*)}{\eta} \right\rfloor$: the round complexity of LDT.
16         $U^{(1)}, \dots, U^{(r)}$: $r$ multiplicative subgroups where $U^{(0)} = U$:
17         for $i = 1$ to $r$ do
18             for $x \in U^{(i-1)}$ do
19                 $y = x^{2^\eta}$.
20                 $U^{(i)}$.append($y$)
21         $\mathcal{H}_1, \dots, \mathcal{H}_{5+r}, \mathcal{H}_{\mathtt{MT}}$: collision-resistant hash functions.
22         Expand : $\mathbb{F} \to \mathbb{F}^*$: an expand function.
23     Output L-pp :=
         $(\mathbb{F}_p, \mathbb{F}, p, L, B, \mathcal{I}, m, n, H, U, (U^{(k)})_{k \in [r]}, \rho^*, r, \kappa, \eta, (\mathcal{H}_k)_{k \in [5+r]}, \mathcal{H}_{\mathtt{MT}}, \mathtt{Expand})$.
```

Algorithm 3: Loquat Key Generation

```
Input: L-pp
Output: $(sk, pk)$
1 L-KeyGen
2     On input public parameters L-pp, the algorithm performs the
      following tasks.
3     Generate the secret key. Randomly pick a field element
      $K \xleftarrow{\$} \mathbb{F}_p^*/\{-I_1, \dots, -I_L\}$ and set $sk := K$.
4     Generate the public key. Compute
      $pk := \mathcal{L}_K(\mathcal{I}) = (\mathcal{L}_K(I_1), \dots, \mathcal{L}_K(I_L))$. The bit length of the public
      key is $L$.
5     Output $(sk, pk)$.
```

Remark 5. Decomposing a large polynomial with degree $2B$ to n polynomials with degree $2m$ introduces an additional soundness error $1/|\mathbb{F}|$ owing to randomly linear combination of n polynomials for univariate sumcheck. The increment is *negligible* and is not reflected in our proof of Theorem 1 and Lemma 2.

Theorem 2. *Let $\Pi_{\texttt{Loquat}}$ be the signature scheme that is constructed in Section 4.2. $\Pi_{\texttt{Loquat}}$ is secure in EUF-KO as defined in Definition 6 under the assumption that finding β-approximate witnesses for a given public key is hard. Specifically, for a given security parameter λ, if there exists a PPT adversary $\mathcal{A}$ that makes at most m queries to the random oracles with success forgery probability in the EUF-KO model ε, then there exists a PPT algorithm $\mathcal{B}$ which, on receive the challenge public key $pk = (\mathcal{L}_K(I_1), \dots, \mathcal{L}_K(I_L))$ for a random key $K \in \mathbb{F}_p^*$, outputs a β-approximate witness for pk with probability at least $\varepsilon - \bar{\epsilon}(pk, m) - 3(m^2+1)2^{-2\lambda}$, where $\bar{\epsilon}(pk, m)$ is the restricted state-restoration soundness of the IOP-based key identification scheme defined in Lemma 2.*

Proof. The proof of Theorem 2 is same as the proof of [12, Theorem 7.1]. Namely, the restricted state restoration soundness (knowledge) error $\bar{\epsilon}(pk, m)$ is the error probability of $\mathcal{A}$ forges a signature by successfully cheating on the protocol, which is negligible with appropriate parameters. $(3(m^2+1)2^{-\lambda})$ is the probability that $\mathcal{B}$ aborts in the IOP protocol owing to collisions, as analyzed in [12]. □

Theorem 3. *Assume that $\mathcal{H}_1$ is modeled as a random oracle. Then, if there exists an adversary $\mathcal{A}$ that wins the EUF-CMA security game with advantage ε, then there exists an adversary $\mathcal{B}$ with a run-time within a constant factor of the run-time of $\mathcal{A}$ wins the EUF-KO security game with probability at least $\varepsilon - (q_s(q_s + q_{\mathcal{H}_1})) \cdot 2^{-2\lambda}$ by making at most $q_{\mathcal{H}_1}$ random oracle queries and at most q_s signing oracle queries.*

The security proof of Theorem 3 is shown in Appendix B.3. To enable SNARK-based applications, we define Loquat in the URS model by *heuristically* instantiating the random oracle via cryptographic hash functions (see

Algorithm 4: Loquat Sign (Part I)

```
Input: Public parameter L-pp, secret key $sk$, and message $M$
Output: Signature $\sigma$
L-Sign
    Phase 1. Commit to secret key and randomness
        for $j = 1$ to $n$ do
            Randomly pick $(r_{1,j}, \dots, r_{m,j}) \xleftarrow{\$} \mathbb{F}_p^*$.
            for $i = 1$ to $m$ do
                Compute $T_{i,j} \leftarrow \mathcal{L}_0(r_{i,j})$.
            Assign $\mathbf{c}_j \leftarrow (Kr_{1,j}, r_{1,j}, \dots, Kr_{m,j}, r_{m,j}) \in \mathbb{F}_p^{2m}$ and lift $\mathbf{c}_j$ to
             $\mathbb{F}^{2m}$.
            $\hat{c}_j(x) \leftarrow$ Interpolate$(H, \mathbf{c}_j)$.    // $\hat{c}_j \in \mathbb{F}[X]$ and $\deg(\hat{c}_j) < 2m$
            Randomly sample $\hat{r}(x) \in \mathbb{F}[X]$ with degree $\kappa \cdot 2^\eta$.
            Compute $\hat{c}'_j(x) \leftarrow \hat{c}_j(x) + Z_H(x)\hat{r}(x)$.  // $\deg(\hat{c}'_j) < 2m + \kappa \cdot 2^\eta$
        for $e = 1$ to $|U|$ do
            leaf$_e = \mathcal{H}_c(\hat{c}'_1(U[e]), \dots, \hat{c}'_n(U[e]))$
        Commit to **leaf** $=$ (leaf$_1, \dots,$ leaf$_{|U|}$):
         root$_c \leftarrow$ MT.Commit(**leaf**).
        Set $\sigma_1 \leftarrow ($root$_c, (T_{1,j}, \dots, T_{m,j})_{j \in [n]})$.
    Phase 2. Compute residuosity symbols
        Compute $h_1 \leftarrow \mathcal{H}_1(\sigma_1, M)$ and
         $(I_{1,j}, \dots, I_{m,j})_{j \in [n]} \leftarrow$ Expand$(h_1)$, where $I_{i,j} \in \mathcal{I}$.
        for $i = 1$ to $m$ do
            for $j = 1$ to $n$ do
                Compute $o_{i,j} \leftarrow (K + I_{i,j})r_{i,j}$.
        Set $\sigma_2 \leftarrow (o_{1,j}, \dots, o_{m,j})_{j \in [n]}$.
    Phase 3. Compute witness vector for univariate sumcheck
        Compute $h_2 \leftarrow \mathcal{H}_2(\sigma_2, h_1)$ and
         $((\lambda_{1,j}, \dots, \lambda_{m,j}), \epsilon_j)_{j \in [n]} \leftarrow$ Expand$(h_2)$, where $\lambda_{i,j} \in \mathbb{F}_p$ and
         $\epsilon_j \in \mathbb{F}$.
        for $j = 1$ to $n$ do
            Assign $\mathbf{q}_j \leftarrow (\lambda_{1,j}, \lambda_{1,j}I_{1,j}, \dots, \lambda_{m,j}, \lambda_{m,j}I_{m,j}) \in \mathbb{F}_p^{2m}$ and lift
             $\mathbf{q}_j$ to $\mathbb{F}^{2m}$.
            $\hat{q}_j(x) \leftarrow$ Interpolate$(H, \mathbf{q}_j)$.    // $\hat{q}_j \in \mathbb{F}[X]$ and $\deg(\hat{q}_j) < 2m$
            Define $\hat{f}_j(x) \leftarrow \hat{c}'_j(x) \cdot \hat{q}_j(x)$.            // $\deg(\hat{f}_j) < 4m + \kappa \cdot 2^\eta$
        Define $\hat{f}(x) = \sum_{j=1}^n \epsilon_j \hat{f}_j(x)$        // $\deg(\hat{f}) = \deg(\hat{f}_j) < 4m + \kappa \cdot 2^\eta$
        Execute the (non-interactive) zero-knowledge univaraite
         sumcheck protocol with respect to
         $((\mathbb{F}, U, H, 4m + \kappa \cdot 2^\eta, \mu), \hat{f}(x))$, where $\mu \in \mathbb{F}$ is lifted from
         $\sum_{j=1}^n \epsilon_j (\sum_{i=1}^m \lambda_{i,j} o_{i,j})$. Obtain partial signature $\pi_{\mathtt{US}}$.
    Output the signature $\sigma := ((T_{i,j}, o_{i,j})_{i \in [m], j \in [n]}, \pi_{\mathtt{US}}, \pi_{\mathtt{LDT}})$.
```

Algorithm 5: Loquat Sign (Part II) - Univariate Sumcheck

```
L-Sign - Part II
    Phase 3 (cont'd). Enable ZK of Univariate Sumcheck
        Randomly sample $\hat{s}(x)$ with degree $4m + \kappa \cdot 2^\eta - 1$.
        Compute $S \leftarrow \sum_{a \in H} \hat{s}(a)$.
        Commit $\texttt{root}_s \leftarrow \texttt{MT.Commit}(\hat{s}|_U)$.
        Set $\sigma_3 \leftarrow (\texttt{root}_s, S)$.
    Phase 4. Univariate Sumcheck
        Compute $h_3 \leftarrow \mathcal{H}_3(\sigma_3, h_2)$ and $z \leftarrow \texttt{Expand}(h_3)$.
        Define $\hat{f}' = z\hat{f}(x) + \hat{s}(x)$.        // $\deg(\hat{f})' < 4m + \kappa \cdot 2^\eta$
        Compute $\hat{g}(x)$ and $\hat{h}(x)$ such that $\hat{f}'(x) = \hat{g}(x) + Z_H(x)\hat{h}(x)$.
         // $\deg(\hat{g}) < 2m$ and $\deg(\hat{h}) < 2m + \kappa \cdot 2^\eta$
        Commit $\texttt{root}_h \leftarrow \texttt{MT.Commit}(\hat{h}|_U)$.
        Set $\sigma_4 \leftarrow (\texttt{root}_h)$.
    Phase 5. Stacking Codeword for LDT
        Compute $h_4 \leftarrow \mathcal{H}_4(\sigma_4, h_3)$ and $\mathbf{e} \leftarrow \texttt{Expand}(h_4)$, where $\mathbf{e} \in \mathbb{F}^8$.
        Define rational constraint $\hat{p}(x) = \frac{|H|\hat{f}' - |H|Z_H(x)\hat{h}(x) - (z\mu + S)}{|H|x}$.
         // $\deg(\hat{p}) < 2m - 1$
        Stack vertically $\hat{c}'|_U, \hat{s}|_U, \hat{h}|_U, \hat{p}|_U$, denote the resulting matrix as
         $\mathit{\Pi}_0 \in \mathbb{F}^{4 \times |U|}$.
        Define $\rho_1 = \frac{2m+\kappa \cdot 2^\eta + 1}{|U|}$, $\rho_2 = \frac{4m+\kappa \cdot 2^\eta}{|U|}$, $\rho_3 = \frac{2m+\kappa \cdot 2^\eta}{|U|}$, and
         $\rho_4 = \frac{2m-1}{|U|}$.
        for $i = 1$ to $4$ do
            Compute $(\mathit{\Pi}_1)_{i,y} = y^{(\rho^* - \rho_i)|U|} \cdot (\mathit{\Pi}_0)_{i,y}$.
        Obtain $\mathit{\Pi} = \binom{\mathit{\Pi}_0}{\mathit{\Pi}_1} \in \mathbb{F}^{8 \times |U|}$.
        Compute $\mathbf{f}^{(0)} \leftarrow \mathbf{e}^\top \cdot \mathit{\Pi}$ such that $\mathbf{f}^{(0)} \in RS[U, \rho^*]$.
```

Algorithm 6: LOQUAT Sign (Part III) - Low-Degree Test

```
1  L-Sign - Part III
2      Phase 6. LDT Folding
3          Define $\hat{l}(x) = x^{2^\eta}$ and denote $U^{(0)} = U$.
4          for $i = 0$ to $r$ do
5              Compute $\texttt{root}_{f^{(i)}} \leftarrow \texttt{MT.Commit}(\mathbf{f}^{(i)})$.
6              Define $\sigma_{5+i} \leftarrow \texttt{root}_{f^{(i)}}$ and $h_{5+i} \leftarrow \mathcal{H}_{5+i}(\sigma_{5+i}, h_{4+i})$.
                $x^{(i)} \leftarrow \texttt{Expand}(h_{5+i})$, where $x^{(i)} \in \mathbb{F}$.
7              for $y \in U^{(i+1)}$ do
8                  Define $S_y^{(i)} = \{x \in U^{(i)} | \hat{l}(x) = y\}$        // $S_y^{(i)}$ is the
                    multiplicative coset of $U^{(i)}$ with $|S_y^{(i)}| = 2^\eta$
9                  Set $\hat{P}_y^{(i)}(x) = \texttt{Interpolate}(S_y^{(i)}, \hat{f}^{(i)}|_{S_y^{(i)}})$        // $\hat{P}_y^{(i)}(x)$
                    agrees with $\hat{f}^{(i)}$ on the multiplicative coset $S_y^{(i)}$ and
                    $\deg(\hat{P}_y^{(i)}) = 2^\eta - 1$
10                 $\mathbf{f}^{(i+1)}.\texttt{append}(\hat{P}_y^{(i)}(x^{(0)}))$
11             if $i + 1 = r$ then
12                 Let $\mathbf{f}^{(r)} = \mathbf{f}^{(i+1)}$ be defined in previous steps, and
                    $\hat{f}^{(r)}(x) = \texttt{Interpolate}(U^{(r)}, \mathbf{f}^{(r)})$.
13                 Let $d = \rho^* \cdot |U^{(r)}| - 1$
14                 Commits to the first $d + 1$ coefficients $\mathbf{a} = (a_0, \ldots, a_d)$ of
                    $\hat{f}^{(r)}(x)$: $h_{5+r} \leftarrow \mathcal{H}_{5+r}(\mathbf{a}, h_{4+r})$.
15                 End the iteration
16             else
17                 Continue the iteration with $\mathbf{f}^{(i+1)}$.
18     Phase 7. LDT Query
19         Compute $(S_1, \ldots, S_\kappa) \leftarrow \texttt{Expand}(h_{5+r})$, where $|S_j| = 2^\eta$ and
            $S_j \subset U^{(0)}$ such that there exists a $y \in U^{(1)}$, $\hat{l}(s) = y$ for all
            $s \in S_j$.
20         Reveal leaf nodes for trees $\texttt{root}_c, \texttt{root}_s, \texttt{root}_h$, denote the query
            answers, corresponding Merkle roots, and auxiliary information
            in univariate sumcheck as $\pi_{\texttt{US}}$.
21         for $i = 1$ to $r - 1$ do
22             Reveal $\kappa \cdot (2^\eta - 1)$ leaves for $\mathbf{f}^{(i)}$.        // Using $\kappa \cdot 2^\eta$ query
                answers of $\mathbf{f}^{(i-1)}$ can compute $\kappa$ leaves of $\mathbf{f}^{(i)}$. Then,
                reveal the remaining leaves in the same coset would be
                enough
23         Denote the query answers as $\pi_{\texttt{LDT}}$.
24         Append coefficients $\mathbf{a}$ of $\hat{f}^{(r)}$ Merkle roots
            $\texttt{root}_{f^{(0)}}, \ldots, \texttt{root}_{f^{(r-1)}}$ in LDT phase to $\pi_{\texttt{LDT}}$.
25     The signature $\sigma := ((T_{i,j}, o_{i,j})_{i \in [m], j \in [n]}, \pi_{\texttt{US}}, \pi_{\texttt{LDT}})$.
```

Algorithm 7: Loquat Verify

```
Input: Signature $\sigma$, public key $pk$, and message $M$
Output: 0/1
L-Verify
    Step 1. Recompute Challenges
        There exists a hash chain from the beginning to the end of the
          signature protocol, where the verifier first obtains all Merkle
          roots and plaintext messages from the signature, then computes
          the hash output round-by-round. It then expand each round's
          hash output to obtain corresponding challenges if required.
    Step 2. Recompute Leaf Nodes
        for $j = 1$ to $n$ do
            Define $\mathbf{q}_j \leftarrow (\lambda_{1,j}, \lambda_{1,j} I_{1,j}, \ldots, \lambda_{m,j}, \lambda_{m,j} I_{m,j}) \in \mathbb{F}_p^{2m}$ and lift
              $\mathbf{q}_j$ to $\mathbb{F}^{2m}$.
            $\hat{q}_j(x) \leftarrow \mathtt{Interpolate}(H, \mathbf{q}_j)$.
            Define $\hat{f}_j(x) \leftarrow \hat{c}'_j(x) \cdot \hat{q}_j(x)$.
        for $s \in Q$ do
            Recompute $(\hat{q}_j(s))_{j \in [n]}$.                // $Q$ being the query set
            Recompute $\hat{f}_j(s) \leftarrow \hat{c}'_j(s) \cdot \hat{q}_j(s)$ and $\hat{f}(s) = \sum_{j=1}^{n} \epsilon_j \hat{f}_j^{(} s)$.
            Recompute $\hat{p}(s) = \frac{|H|(z\hat{f}(s)+\hat{s}(s)) - |H| Z_H(s)\hat{h}(s) - (z\mu + S)}{|H| s}$, where
              $\mu \in \mathbb{F}$ is lifted from $\sum_{j=1}^{n} \epsilon_j (\sum_{i=1}^{m} \lambda_{i,j} o_{i,j})$.
            Recompute $\mathbf{f}^{(0)}(s)$.     // By reconstructing the interleaved
              code $\Pi$ and takes the random linear combination
    Step 3. Checking Proofs
        for $i \in [m], j \in [n]$ do
            if $o_{i,j} = 0 \vee \mathcal{L}_0(o_{i,j}) \neq pk_{I_{i,j}} + T_{i,j}$ then
                Reject the signature with output 0.
        If there exists any authentication path that is not a valid opening
          of the Merkle commitment, reject and output 0.
        If the low-degree test is inconsistent, reject and output 0.
    Otherwise, accept and output 1.
```

Remark 1). It is worth noting that the security of Loquat public-private keys depends on the security of the Legendre PRF (analyzed in Appendix A), while the EUF-KO and EUF-CMA security of Loquat solely rely on collision-resistant hash functions.

Definition 13 (Loquat in the URS model). *Given a security parameter λ,* Loquat *in the URS model satisfies following security properties.*

- ***Correctness.*** *For any message m,*

$$\Pr\left[\texttt{Verify}(\texttt{urs}, pk, m, \sigma) = 1 \;\middle|\; \begin{array}{l} \texttt{urs} \leftarrow \{0,1\}^{\texttt{poly}(\lambda)} \\ (sk, pk) \leftarrow \texttt{KeyGen}(\texttt{urs}) \\ \sigma \leftarrow \texttt{Sign}(\texttt{urs}, sk, m) \end{array}\right] = 1$$

- ***EUF-CMA.*** *For any PPT adversary $\mathcal{A}$, it has negligible advantage in the* `EUF-CMA` *game defined as:*

$$\mathbf{Adv}_{\mathcal{A}}^{\texttt{EUF-CMA}} = \Pr\left[\begin{array}{l} \texttt{Verify}(\texttt{urs}, pk, m^*, \sigma^*) = 1 \\ \wedge\, m^* \notin Q \end{array} \;\middle|\; \begin{array}{l} \texttt{urs} \leftarrow \{0,1\}^{\texttt{poly}(\lambda)} \\ (sk, pk) \leftarrow \texttt{KeyGen}(\texttt{urs}) \\ (m^*, \sigma^*) \leftarrow \mathcal{A}^{\texttt{Sign}(\texttt{urs}, sk, \cdot)} \end{array}\right]$$

  where $\mathcal{A}^{\texttt{Sign}(\texttt{urs}, sk, \cdot)}$ denotes $\mathcal{A}$'s access to a signing oracle with private key sk and Q denotes the set of messages m that were queried to the signing oracle by $\mathcal{A}$.

4.3 Optimizing Loquat Protocol for SNARK-based Applications

We commence this section by presenting several improvements aimed at minimizing the R1CS complexity of the verification algorithm of Loquat.

Decomposing Large Polynomials. We leverage the modular design of our key identification scheme to decompose large polynomials, which enables the univariate sumcheck to work on a smaller amortised polynomial.

Specifically, let m and n be two integers such that $m \times n = B$, with m being a power of 2. For $j \in [n]$, the prover computes the witness polynomial $\hat{f}_j(x_{1,j}, y_{1,j}, \ldots, x_{m,j}, y_{m_j}) = Kr_{1,j}x_{1,j} + r_{1,j}x_{1,j}y_{1,j} + \cdots + Kr_{m,j}x_{m,j} + r_{m,j}x_{m,j}y_{m,j}$. Note that each $\hat{f}_j$ has $2m$ coefficients where $m = B/n$. The decomposition step does not influence the soundness as the total number of residuosity symbols being checked is still B.

Nonetheless, running n zkVPD protocols concurrently on small polynomials does not yield efficiency improvements. To reduce the number of zkVPD protocols, we explore the amortisation property of univariate sumcheck [11, Section 5.2], wherein n univariate sumchecks relations can be amortised through random linear combinations (with additional soundness error of $1/|\mathbb{F}|$).

Efficient Verification of Quadratic Residuosity. Recall that to verify σ_e for ($e \in [N]$), the SNARK prover needs to prove, in the circuit, that for all $i \in [m], j \in [n]$, $\mathcal{L}_0(o^{(e)}_{i,j}) = pk^{(e)}_{I_{i,j}} + T^{(e)}_{i,j}$. Notably, computing Legendre PRF in the circuit can be relatively expensive. To circumvent the problem, we let the SNARK prover perform the following tasks *outside of the circuit*.

1. For the Loquat signature $\sigma_e = (\ldots, (o^{(e)}_{i,j})_{i\in[m],j\in[n]}, \ldots)$, the SNARK prover computes $\mathcal{L}_0(o^{(e)}_{i,j})$. There are two possible outputs 0 or 1 implying that $o^{(e)}_{i,j}$ is a quadratic residue (QR) or quadratic non-residue (QNR) modulo p.
   (a) If $\mathcal{L}_0(o^{(e)}_{i,j}) = 0$, then either $o^{(e)}_{i,j} = 0$ (this case is avoided by checking if $o^{(e)}_{i,j}$ is a non-zero element), or $o^{(e)}_{i,j}$ is a QR. Then, the SNARK prover can find a $s^{(e)}_{i,j} \in \mathbb{F}_p$ such that $(s^{(e)}_{i,j})^2 = o^{(e)}_{i,j}$ (actually $s^{(e)}_{i,j}$ and $-s^{(e)}_{i,j}$ are both candidates that can be squared to $o^{(e)}_{i,j}$, here we let the SNARK prover pick the smallest $s^{(e)}_{i,j}$).
   (b) If $\mathcal{L}_0(o^{(e)}_{i,j}) = 1$, then $o^{(e)}_{i,j}$ must be a QNR. Then, we let the SNARK prover compute $t^{(e)}_{i,j} = o^{(e)}_{i,j} \cdot \alpha$ for some fixed QNR α that is defined in public parameters. Thus, $t^{(e)}_{i,j}$ is a QR, which implies that there exists a corresponding $s^{(e)}_{i,j}$ such that $(s^{(e)}_{i,j})^2 = t^{(e)}_{i,j}$.
2. After finding $\mathbf{s}^{(e)} = (s^{(e)}_{i,j})_{i\in[m],j\in[n]}$, the SNARK prover obtains σ'_e by appending $\mathbf{s}^{(e)}$ to σ_e.

The witness is then defined as $\mathbb{w} = (\sigma'_1, \ldots, \sigma'_N)$.

To prevent the SNARK prover cheating on $s^{(e)}_{i,j}$, we let it prove *in the circuit* that $s^{(e)}_{i,j} \cdot s^{(e)}_{i,j} = t^{(e)}_{i,j}$, where (also in the circuit) $t^{(e)}_{i,j} = o^{(e)}_{i,j} \cdot (\alpha \cdot (pk^{(e)}_{I_{i,j}} + T^{(e)}_{i,j}) + (1 - (pk^{(e)}_{I_{i,j}} + T^{(e)}_{i,j})))$ for all $e \in [N]$, $i \in [m], j \in [n]$. Note that the SNARK prover can always check the validity of the Loquat signature before proving the validity of Loquat to prevent DoS attack. Therefore, if it holds true that for all $b \in [B]$, $\mathcal{L}_0(o_b) = pk_{I_b} + T_b$, we can find the square root s_b of t_b and prove in the circuit that $t_b = s_b \cdot s_b \pmod p$. It is worth emphasising that the computation of s_b values can be done *outside* of the circuit. Consequently, the total number of constraints required to verify the quadratic residuosity of o_b is $4B + 1$.

Hash by Subset and Tree Cap. Since the answers to the queries for low-degree test involve $\kappa \cdot 2^\eta$ leaf nodes, instead of treat each element as an individual leaf node, the prover can hash 2^η elements and use the single element as the Merkle leaf. This reduces the number of leaves for each Merkle tree by a factor of 2^η. Furthermore, by the pigenhole principle, any set of authentication paths must overlap towards the top of the tree. To take advantage of this, we modify the Merkle tree by hashing the nodes in layer t together to form the root, instead of computing the tree.

5 Applications of Loquat

5.1 ID-based Ring Signature

Buser et al. [23] introduced an ID-based ring signature whose security solely relies on symmetric-key primitives. In this scheme, the Public Key Generator (PKG) initiates the process by generating a master public-private key pair using the key generation algorithm of a post-quantum signature scheme. A user wishing to register sends their ID (i.e., the public key) to the PKG, which signs this ID with its master private key, and returns the signature to the user to serve as their signing key. To construct an ID-based ring signature, a user undertakes the following steps:

1. The user selects $n-1$ IDs and inserts their own ID at a random position to form a ring of size n with the user's index denoted by π.
2. The user accumulates n IDs using a Merkle tree and records the root.
3. Subsequently, the user produces a Non-Interactive Zero-Knowledge Proof of Knowledge (NIZKPoK) demonstrating that:
   - There exists a valid signing key with respect to ID_π in the ring. In other words, the user proves that there exists a valid signature of ID_π under the master public key.
   - ID_π is accumulated in the Merkle tree.

The unforgeability is guaranteed by the (knowledge) soundness while the anonymity is ensured by the zero-knowledge property of the employed NIZKPoK protocol. By adopting Ligero++ [17] as the underlying NOIZKPoK, the ID-based ring signature has size ranging between 0.885 and 1.898 MB for rings containing 2^6 members.

The original proposal highlighted two symmetric-key primitives based signature schemes for PKG: XMSS [22] and Picnic [64]. Their analysis showed that verifying an XMSS signature requires 685 hash function calls, whereas a Picnic signature verification demands 7,008 hash function calls and 2,680,560 multiplication gates. As discussed earlier (Section 1.1), employing zero-knowledge protocols to prove signature validity mandates substituting the hash function with algebraic alternatives due to the prohibitively large number of constraints associated with standard hash functions. Unfortunately, such a substitution significantly extends the key generation time of XMSS, particularly for large tree size. In the case of Picnic, the scheme is burdened with an excessively high number of R1CS constraints. Compared to these schemes, Loquat appears as the optimal solution for the PKG in the proposed ID-based ring signature framework.

5.2 SNARK-based Aggregate Signature

We assume there exists a SNARK protocol with post-quantum security and transparent setup (e.g., [2, 11, 17, 27]) denoted as $\Pi_{\texttt{SNARK}}$. The definition of SNARKs can be found in Section 3.6.

The aggregator computes $\pi_{\texttt{SNARK}} = \texttt{SNARK-P}(\texttt{SNARK-pp}, \mathbb{w}, \mathbb{x})$ which states the validity of all signatures being aggregated. We define the aggregate signature scheme that consists of the following six algorithms, where the setup algorithm `AggSetup` is shown in Algorithm 8, while $(\texttt{KeyGen}, \texttt{Sign}, \texttt{Verify})$ are defined the same as LOQUAT (Section 4.2). Algorithm 9 presents both `Agg` (aggregate signature generation) and `AggVerify` (aggregate signature verification).

Algorithm 8: Aggregate Signature (Transparent) Setup

```
   Input: Security parameter $\lambda$
   Output: A-pp
1  AggSetup
2      Fix a QNR $\alpha \in \mathbb{F}_p$.
3      Generate public parameters for LOQUAT: $\texttt{L-pp} \leftarrow \texttt{L-Setup}(\lambda)$ and
         public parameters for the SNARK protocol:
         $\texttt{SNARK-pp} \leftarrow \texttt{SNARK-Setup}(\lambda)$.
4      Output $\texttt{A-pp} = (\alpha, \texttt{L-pp}, \texttt{SNARK-pp})$.
```

Theorem 4. *Let $\Pi_{\texttt{A}}$ be the aggregate signature protocol shown in Algorithm 9. Assume that the knowledge error, $e_{\texttt{SNARK}}$, of the SNARK protocol used by $\Pi_{\texttt{A}}$ is negligible (in the random oracle model). If there exists an $\mathcal{A}$ that wins the aggregate chosen-key security game (Definition 9) with advantage ϵ, then there exists a PPT adversary $\mathcal{B}$, who runs in time $t + q_s \cdot t_{\texttt{L-Sign}} + t_{\texttt{SNARK}}$, that breaks the unforgeability of* LOQUAT *in the URS model with probability at least $\epsilon - e_{\texttt{SNARK}}$ while making at most q_s queries to the signing oracle of* LOQUAT *in the URS model. We denote t as the running time of $\mathcal{A}$, $t_{\texttt{L-Sign}}$ as the running time of the signing oracle of* LOQUAT, *and $t_{\texttt{SNARK}}$ as the running time of the SNARK extractor $\mathcal{E}_{\texttt{SNARK}}$.*

The security proof of Theorem 4 is presented in Appendix B.4.

SNARK-based Aggregate Signature with Proof Recursion Given the black-box access to a SNARK protocol with succinct verification (i.e., the number of R1CS constraints to represent the SNARK verifier is less than the R1CS constraints that the SNARK verifier checks), we can construct an aggregate signature using *SNARK proof recursive composition.* The setup phase is the same as Algorithm 8, while `Agg` and `AggVerify` with proof recursion is shown in Algorithm 10 and 11. Note that we omit the indexer step if the underlying SNARK is a preprocessing SNARK for the sake of simplicity.

Theorem 5. *Let Π'_A be the aggregate signature protocol shown in Algorithm 10. Assume the knowledge error of the SNARK protocol with succinct verification is $e_{\texttt{SNARK}}$, which is negligible (in the URS model). If there exists an adversary $\mathcal{A}$ that*

Algorithm 9: Aggregate Signature Generation and Verification

```
1  Agg
     Input: Public parameters A-pp, arithmetic circuit of signature
            verification algorithm $\mathfrak{C}$, signatures to be aggregated
            $\{(\sigma_e, M_e, pk_e)\}_{e \in [N]}$
     Output: $\Sigma$
2    Step 1. Prepare square roots (outside of the circuit)
3      for $e = 1$ to $N$ do
4        for $i \in [m], j \in [n]$ do
5          if $\mathcal{L}_0(o_{i,j}^{(e)}) = 0$ then
6            Define the smallest $s_{i,j}^{(e)}$ such that $(s_{i,j}^{(e)})^2 = o_{i,j}^{(e)}$.
7          else
8            Define the smallest $s_{i,j}^{(e)}$ such that $(s_{i,j}^{(e)})^2 = o_{i,j}^{(e)} \cdot \alpha$.
9        Define $\sigma'_e = (\sigma_e, (s_{i,j}^{(e)})_{i \in [m], j \in [n]})$.
10   Step 2. Define witness and statement for SNARK prover
11     Define $\mathbb{w} = (\sigma'_1, \ldots, \sigma'_N)$ and
       $\mathbb{x} = (\mathfrak{C}^N, (M_1, pk_1), \ldots, (M_N, pk_N))$, where $\mathfrak{C}^N$ denotes the
       concatenation of $N$ circuits with multiplication gates.
12   Step 3. Construct SNARK proof
13     Obtain $\pi_{\mathtt{SNARK}} = (\mathtt{SNARK\text{-}pp}, \mathbb{w}, \mathbb{x})$ and set $\Sigma = (\pi_{\mathtt{SNARK}}, o)$, where
       $o = \prod_{i=1}^{n} \left( \prod_{b=1}^{B} o_{i,j}^{(e)} \right)$ is computed in the circuit.
14   Output $\Sigma$.
15 Verify
     Input: Public parameters A-pp, arithmetic circuit of signature
            verification algorithm $\mathfrak{C}$, an aggregate signature $\Sigma$, and a list
            of message, public key pairs $((M_1, pk_1), \ldots, (M_N, pk_N))$.
     Output: 0/1
16   Step 1. Check zero
17     Parse $\Sigma = (\pi_{\mathtt{SNARK}}, o)$. If $o = 0$, then reject and output 0.
18   Step 2. Prepare the statement
19     Define $\mathbb{x} = (\mathfrak{C}, (M_1, pk_1), \ldots, (M_N, pk_N))$.
20   3. Verify SNARK proof
21     Compute $b \leftarrow \mathtt{SNARK\text{-}V}(\mathtt{SNARK\text{-}pp}, \Sigma, \mathbb{x})$.
22   Output $b$.
```

Algorithm 10: Aggregate Signature Generation

```
Input: A-pp, $\mathfrak{C}$, $\{(\sigma_e, M_e, pk_e)\}_{e\in[N]}$
Output: $\Sigma$
1  Agg
2      On input a set of public parameters A-pp, an R1CS instance $\mathfrak{C}$ that
        represents the arithmetic circuit of the LOQUAT verification for $s$
        signatures, and a list of signature, message, public key pairs
        $\{(\sigma_e, m_e, pk_e)\}_{e\in[N]}$, the algorithm performs the following tasks.
3      Step 1. Split the Inputs.
4          Split $N$ signatures into $\gamma$ chunks $(\Gamma_1, \dots, \Gamma_\gamma)$ such that:
5          for $k = 1$ to $\gamma$ do
6              if $k = 1$ then
7                  $|\Gamma_k| \cdot |\mathfrak{C}| > T$
8              else
9                  $|\Gamma_k| \cdot |\mathfrak{C}| + \mathfrak{R}_{k'-1} > T$
10         where $T$ denotes the recursion threshold.
11         Denote each chunk of the message public key pairs as $(M, PK)_k$,
            and the circuit for verifying each chunck of circuit as $\mathfrak{C}_k$ for
            $k \in [\gamma]$.
12     Step 2. Prepare Square roots (Outside of the Circuit).
13         Same as in Algorithm 9.
14     Step 3. First Aggregation
15         Define $\mathbb{w}_1 = \Gamma_1$ and $\mathbb{x}_1 = (\mathfrak{C}_1, (M, PK)_1)$.
16         Compute $\pi_1$ = SNARK-P(SNARK-pp, $\mathbb{w}_1, \mathbb{x}_1$).
17     4. Recursive Aggregation
18         for $k = 2$ to $\gamma$ do
19             Define $\mathbb{w} = (\Gamma_k, \pi_{k-1})$ and
                $\mathbb{x} = ((\mathfrak{C}_k, \mathfrak{R}_{k-1}), ((M, PK)_k), x_{k-1}))$, where $\mathfrak{R}_{k-1}$ denotes
                the circuit presentation of the $(k-1)$-th SNARK verifier:
                SNARK-V(SNARK-pp, $\pi_{k-1}, x_{k-1}$)).
20             Compute $\pi_k$ = SNARK-P(SNARK-pp, $\mathbb{w}_k, \mathbb{x}_k$).
21     Define $\Sigma = (\pi_\gamma, o)$ and output $\Sigma$.
```

Algorithm 11: Aggregate Signature Verification

```
Input: $\texttt{A-pp}, \Sigma, \mathfrak{C}_\gamma, \{(M_e, pk_e)\}_{e\in[N]}$
Output: 0/1
1  Verify
2    On input a set of public parameters $\texttt{A-pp}$, an aggregate signature $\Sigma$,
       an R1CS instance represents the LOQUAT verifier, and a list of
       message, public key pairs $((M_1, pk_1), \ldots, (M_N, pk_N))$, the algorithm
       does the following.
3    Step 1. Check Zero
4      Parse $\Sigma = (\pi_\gamma, o)$.
5      If $o = 0$, then reject and output 0.
6    Step 2. Prepare the statement
7      Define $\mathbb{x} = ((\mathfrak{C}_\gamma, \mathfrak{R}_\gamma), (M_1, pk_1), \ldots, (M_N, pk_N))$.
8    2. Verify SNARK proof
9      Compute $b \leftarrow \texttt{SNARK-V}(\texttt{SNARK-pp}, \Sigma, \mathbb{x})$.
10   Output $b$.
```

wins the aggregate chosen-key security game (Definition 9), then there exists a PPT adversary $\mathcal{B}$, who runs in time $t + q_s \cdot t_{\texttt{L-Sign}} + k \cdot t_{\texttt{SNARK}}$, that breaks the unforgeability of LOQUAT *in the URS model with probability at least $\epsilon - k \cdot e_{\texttt{SNARK}}$, while making at most q_s queries to the signing oracle of* LOQUAT *in the URS model. We denote t as the running time of $\mathcal{A}$, $t_{\texttt{L-Sign}}$ as the running time of the signing oracle of* LOQUAT*, $t_{\texttt{SNARK}}$ as the running time of the SNARK extractor $\mathcal{E}_{\texttt{SNARK}}$, and k as the total number of the recursion.*

The proof of Theorem 5 is shown in Appendix B.5.

6 Evaluation

6.1 Choice of Parameters

Choice of the field. We choose a prime field such that finding the β-approximate witness for the Legendre PRF relation is hard. In particular, we follow the choice in LegRoast [16] to set $p = 2^{127} - 1$. This allows for 128-bit security level for the Legendre PRF. The corresponding field choice of univariate sumcheck and low-degree test is $\mathbb{F}_{p^2}$, which contains sufficiently large smooth multiplicative subgroup for the code domain U. We note that a smaller extension field such as $\mathbb{F}_{q^2}$ for $q = 2^{61} - 1$ can achieve 100+ bit security [65]. However, owing to the field inconsistency problem, we can only choose $\mathbb{F}_{p^2}$ that enables simple and efficient field lifting.

Choice of parameters for relaxed Legendre PRF relation. In order to make sure β-approximate Legendre PRF relation is as hard as Legendre PRF relation, we need to choose L and β such that $2p \cdot \Pr[\mathfrak{B}(L, 1/2 + 1/\sqrt{p} + 2/p) > (1-\beta)L] \leq 2^{-\lambda}$, where $\mathfrak{B}(\cdot,\cdot)$ denotes the binomial distribution [16]. Since $p = 2^{127} - 1$ and for $\lambda = 128$, we can roughly re-write the inequality so that $\Pr[\mathfrak{B}(L, 1/2 + 1/\sqrt{p} + 2/p) > (1-\beta)L] \leq 2^{-128}$. In the binomial distribution, the winning probability is dominated by $1/2$, thus, the inequality becomes $\Pr[\mathfrak{B}(L, 1/2) > (1-\beta)L] \leq 2^{-128}$. Then, following cumulative distribution function, $\Pr[\mathfrak{B}(L, 1/2) > (1-\beta)L] = 1 - \sum_{i=0}^{(1-\beta)L} \left(\binom{L}{i} \cdot \left(\frac{1}{2}\right)^L\right)$. We observe that with the same choice of $\beta = 0.449$ and $L = 32768 = 2^{15}$ as in LegRoast, the inequality holds. This implies our choice of $B = 128$ for 128-bit security. For the choice of m and n such that $m \times n = B$, the lower bound on m is 16 since LDT must run on polynomials with degree larger than 16.

Choice of parameters for univariate sumcheck and low-degree test. We set the maximum code rate to $\rho^* = 1/16$ and explore various query complexities in low-degree testing, tailored to the desired security level. Drawing from the soundness conjecture proposed in FRI [8, Conjecture 1.5], we introduce Loquat, which rejects codes that maximally deviate from the desired code rate (a property holds with high probability for random code). Additionally, we present Loquat*, which builds upon the proven soundness bounds of FRI, as detailed in [10, Theorem 8.2]. Loquat* can be considered as a more conservative approach compared to Loquat. For both protocols, performance evaluations at security levels of 80, 100, and 128 are provided, as illustrated in Table 3.

6.2 Performance of Loquat

In our implementation of Loquat, we offer two options for hash functions: the standard SHA3-256 and SHAKE-128, as well as the algebraic hash function Griffin [39]. For Griffin, we opt for a state size of 4 for compression and 3 for expansion, both configurations featuring a capacity of 2. Notably, the time required for key generation remains constant at 0.1 seconds, regardless of the selected security level or hash function. The setup time varies depending on the hash function choice: utilising standard hash functions results in a setup time of 0.35 seconds, whereas using the algebraic hash function Griffin extends the setup time to 1.34 seconds. Our implementation leverages Python and the Sagemath library. All performance benchmarks were conducted on a 2021 MacBook Pro equipped with an M1 Max chip and 32GB of RAM using a single thread.

Given our choice of Griffin parameters, the R1CS constraints for each compression permutation are 110, while the constraints for each expansion permutation are 88. We primarily consider Loquat with the conjectured security of low-degree testing for computing R1CS constraints. Specifically, verifying a single Loquat-80 signature requires 102,089 R1CS constraints, approximately 3.5 times larger than a SHA256 circuit. Among these, 95,480 constraints stem from evaluating hash functions, while the remaining 6,609 arise from algebraic operations. In Fig. 1, we show the required R1CS constraints for verifying Loquat-100

Security Level	κ	$\lvert\sigma\rvert$ (KB)	t_P(s)	t_V(s)	Hash
Loquat-80	20	37	4.64	0.16	SHA/SHAKE
Loquat-100	25	46	4.77	0.19	
Loquat-128	32	57	5.04	0.21	
Loquat-80	20	37	104	7.40	Griffin
Loquat-100	25	46	105	9.15	
Loquat-128	32	57	105	11	
Loquat*-80	40	75	11.51	0.27	SHA/SHAKE
Loquat*-100	50	90	12.08	0.31	
Loquat*-128	64	114	13.22	0.37	
Loquat*-80	40	75	209	15	Griffin
Loquat*-100	50	90	210	18	
Loquat*-128	64	114	214	25	

Table 3. Performance Evaluation of Loquat. κ is the query complexity, $|\sigma|$ is the signature size, t_P and t_V are the proving and verification times, respectively.

signatures as an example. Detailed analysis of the R1CS constraints can be found in Appendix C.

6.3 Performance of Loquat in ID-based Ring Signature

We compare Loquat with the of these three schemes in Table 4. To ensure a fair comparison, the R1CS constraints for Loquat-100 are calculated using the Poseidon [41] hash function with 600 constraints per permutation, as employed in the original paper.

Sig	t_{KG}(s)	$\lvert SID\rvert$(KB)	R1CS	Max N	$\lvert\sigma\rvert$ (MB) (est.)		
					$N = 2^6$	$N = 2^{12}$	$N = 2^{20}$
Picnic [64]	≈ 0	167	6.88M	unli.	1.898	1.899	1.900
XMSS [22]	3355	4.899	431,480	2^{20}	0.885	0.889	0.893
Loquat-100	0.1	46	648,549	unli.	0.973	0.977	0.982

Table 4. Instantiate IDRS [23] with Various Signatures. t_{KG}= Master Key Generation Time, $|SID|$ = User Signing Key Size, N = ring size, $|\sigma|$ = ID-based Ring Signature Size.

6.4 Performance of SNARK-based Aggregate Signature

The performance of our aggregate signature varies depending on the underlying SNARKs. For instance, with the state-of-the-art transparent SNARK named

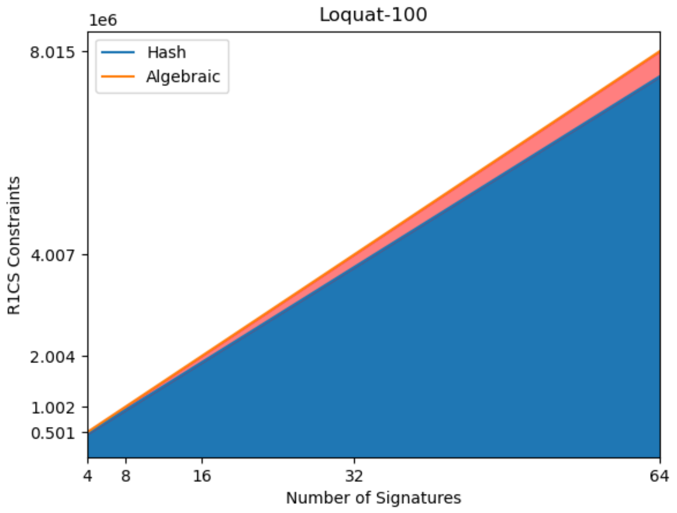

Fig. 1. R1CS Constraints for Verifying Loquat Signatures

Ligero++ [17], proving 2^7 SHA256 functions (equal to aggregate 35 Loquat-100 signatures) takes approximately 100 seconds for proving and 15 seconds for verifying, with a proof size of 158 KB, as reported in the original Ligero++ paper. We further instantiate the underlying SNARK with Aurora [11] and Fractal [27] and utilize the open-source implementation in libiop [1] to evaluate the performance. All tests are done locally on a Virtual Machine with 28GB memory restrictions.

	Aurora					Fractal			Fractal-R			(est.)
	$n = 4$	8	16	32	64	4	8	16	4	8	16	32
t_I (s)	-	-	-	-	-	33	88	218	5K	13K	32K	32K
t_P (s)	14	33	67	160	553	119	251	556	3K	6K	14K	14K
t_V (s)	2.4	4.3	10	20	62	0.2	0.47	0.78	2.4	5.64	9.6	9.6
$\lvert\sigma\rvert$ (KB)	126	137	182	197	207	131	140	145	131	140	145	145

Table 5. Performance of Aggregating Loquat Using Aurora, Fractal, and Recursive Fractal (Fractal-R). t_I = indexer time, t_P = proving time, t_V = verification time, n = the number of signatures being aggregated.

We compare the aggregate signature with OTS/STARK [50] and Chipmunk [33], the most related work with ours. To facilitate comparison, we will make the following assumption: the performances of the Aurora-based and Fractal-based aggregate signature schemes follow the asymptotic result. This assumption enables us to extrapolate the existing performance results and make a meaningful comparison.

	OTS/STARK		Chipmunk		Loquat/A		Loquat/FR	
	$n =$ 128	1024	128	1024	128	1024	128	1024
t_I (s)	-	-	-	-	-	-	32K	32K
t_P (s)	2.5	19.7	-	0.57	796	7K	14K	14K
t_V (s)	-	-	-	0.0077	61	487	9.6	9.6
$\|\sigma\|$ (KB)	68	83	-	118	212	235	145	145

Table 6. Comparison between Various Aggregate/Multi- Signature Schemes. t_I = indexer time, t_P = proving time, t_V = verification time, n = the number of signatures being aggregated.

We address that the comparison between our aggregate signature and the OTS/STARK [50] is not entirely apples-to-apples, given that one-time signatures do not adhere to the standard definition of signatures, and the task of constant key updating can be challenging to implement in real-world applications. Furthermore, Chipmunk is a multi-signature scheme that can be seen as a special type of the aggregate signature, where the signers are limited to sign on the same message. While an aggregate signature can function as a multi-signature, the reverse is not true. Although our aggregate signature's performance may not surpass existing solutions, it benefits from distinct security assumptions compared to Chipmunk and offers flexibility for signers to endorse distinct messages.

However, as demonstrated in Table 5 and 6, the practicality of our SNARK-based aggregate signatures is limited to scenarios involving a small number of signatures. These scenarios have certain real-world applications. For example, in a public key infrastructure, users' signatures are often accompanied by a chain of n certificates. Since n tends to be relatively small, aggregating these signatures can reduce the storage requirement in the user end. Similarly, in the secure BGP protocol, routers receive a list of n signatures verifying a specific network path. Here, an aggregate signature scheme could effectively decrease the bandwidth usage.

7 Conclusion

In this paper, we presented Loquat, a post-quantum signature scheme based on the Legendre PRF. We demonstrated a novel protocol design method which reduces the task of designing key identification protocol for the Legendre PRF to the task of designing multivariate polynomial commitment, a commitment scheme that can be constructed using univariate sumcheck [11]. Our method results in a SNARK-friendly signature scheme that can be applied in SNARK-based applications such as ID-based ring signatures [23] and SNARK-based aggregate signatures.

Future Work We propose the following interesting directions for future works:

1. The signature size and signing/verification time can be improved by replacing the Legendre PRF with more generalized power-residue PRFs. However, the efficient circuit evaluation of the Legendre PRF presented in Section 4.3 cannot be naively applied to power-residue PRFs.
2. The SNARK-based aggregate signature proposed in this paper is rather generic. Tailoring the SNARK for faster Loquat aggregation is also an interesting direction for future work.
3. Applying more efficient low-degree tests, such as STIR [3], to replace FRI [8] in Loquat could potentially lead to better signature sizes. We leave this as future work.

Acknowledgement.

We sincerely thank Veronika Kuchta and Raphael Phan for their help with the early draft of this paper. We also wish to express our gratitude to the anonymous reviewers for their invaluable feedback, which greatly helped improve this paper. This paper is supported by Australian Research Council (ARC) Discovery Projects DP18180102199 and DP220101234.

References

1. libiop. `https://github.com/scipr-lab/libiop` (2014)
2. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: Proceedings of the 2017 ACM CCS. pp. 2087–2104 (2017)
3. Arnon, G., Chiesa, A., Fenzi, G., Yogev, E.: Stir: Reed–solomon proximity testing with fewer queries. Cryptology ePrint Archive (2024)
4. Arora, S., Safra, S.: Probabilistic checking of proofs: A new characterization of np. JACM **45**(1), 70–122 (1998)
5. Babai, L., Fortnow, L., Levin, L.A., Szegedy, M.: Checking computations in polylogarithmic time. In: Proceedings of the twenty-third annual ACM ToC. pp. 21–32 (1991)
6. Baum, C., de Saint Guilhem, C.D., Kales, D., Orsini, E., Scholl, P., Zaverucha, G.: Banquet: Short and fast signatures from aes. In: PKC. pp. 266–297 (2021)
7. Belling, A., Soleimanian, A., Bégassat, O.: Recursion over public-coin interactive proof systems; faster hash verification. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 1422–1436 (2023)
8. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Fast reed-solomon interactive oracle proofs of proximity. In: 45th icalp. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2018)
9. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive (2018)

10. Ben-Sasson, E., Carmon, D., Ishai, Y., Kopparty, S., Saraf, S.: Proximity gaps for reed–solomon codes. Journal of the ACM (2023)
11. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for r1cs. In: Eurocrypt. pp. 103–128. Springer (2019)
12. Ben-Sasson, E., Chiesa, A., Spooner, N.: Interactive oracle proofs. In: TCC. pp. 31–60. Springer (2016)
13. Bernstein, D.J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., Wilcox-O'Hearn, Z.: Sphincs: practical stateless hash-based signatures. In: Eurocrypt. pp. 368–397. Springer (2015)
14. Bernstein, D.J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P.: The sphincs+ signature framework. In: Proceedings of the 2019 ACM CCS. pp. 2129–2146 (2019)
15. Beullens, W., Beyne, T., Udovenko, A., Vitto, G.: Cryptanalysis of the legendre prf and generalizations. IACR Transactions on Symmetric Cryptology pp. 313–330 (2020)
16. Beullens, W., de Saint Guilhem, C.D.: Legroast: Efficient post-quantum signatures from the legendre prf. In: PQCrypto. pp. 130–150. Springer (2020)
17. Bhadauria, R., Fang, Z., Hazay, C., Venkitasubramaniam, M., Xie, T., Zhang, Y.: Ligero++: a new optimized sublinear iop. In: Proceedings of the 2020 ACM CCS. pp. 2025–2038 (2020)
18. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Eurocrypt. pp. 416–432. Springer (2003)
19. Boneh, D., Kim, S.: One-time and interactive aggregate signatures from lattices. preprint **4** (2020)
20. Boschini, C., Takahashi, A., Tibouchi, M.: Musig-l: lattice-based multi-signature with single-round online phase. In: Annual International Cryptology Conference. pp. 276–305. Springer (2022)
21. Boudgoust, K., Takahashi, A.: Sequential half-aggregation of lattice-based signatures. Cryptology ePrint Archive (2023)
22. Buchmann, J., Dahmen, E., Hülsing, A.: Xmss-a practical forward secure signature scheme based on minimal security assumptions. In: PQCrypto. pp. 117–129. Springer (2011)
23. Buser, M., Liu, J.K., Steinfeld, R., Sakzad, A.: Post-quantum id-based ring signatures from symmetric-key primitives. In: ACNS. pp. 892–912. Springer (2022)
24. Byott, N.P., Chapman, R.J.: Power sums over finite subspaces of a field. Finite Fields and Their Applications **5**(3), 254–265 (1999)
25. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: Proceedings of the 2017 ACM CCS. pp. 1825–1842 (2017)
26. Chiesa, A., Liu, S.: On the impossibility of probabilistic proofs in relativized worlds. Cryptology ePrint Archive (2019)
27. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: Eurocrypt. pp. 769–793. Springer (2020)
28. van Dam, W., Hallgren, S.: Efficient quantum algorithms for shifted quadratic character problems. arXiv preprint quant-ph/0011067 (2000)
29. Damgård, I., Orlandi, C., Takahashi, A., Tibouchi, M.: Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. Journal of Cryptology **35**(2), 14 (2022)

30. Dobraunig, C., Kales, D., Rechberger, C., Schofnegger, M., Zaverucha, G.: Shorter signatures based on tailor-made minimalist symmetric-key crypto. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 843–857 (2022)
31. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-dilithium: A lattice-based digital signature scheme. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 238–268 (2018)
32. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Conference on the theory and application of cryptographic techniques. pp. 186–194. Springer (1986)
33. Fleischhacker, N., Herold, G., Simkin, M., Zhang, Z.: Chipmunk: Better synchronized multi-signatures from lattices. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 386–400 (2023)
34. Fleischhacker, N., Simkin, M., Zhang, Z.: Squirrel: efficient synchronized multi-signatures from lattices. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 1109–1123 (2022)
35. Frixons, P., Schrottenloher, A.: Quantum security of the legendre prf. Mathematical Cryptology **1**(2), 52–69 (2021)
36. Goldwasser, S., Kalai, Y.T., Rothblum, G.N.: Delegating computation: interactive proofs for muggles. JACM **62**(4), 1–64 (2015)
37. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM JoC **18**(1), 186–208 (1989)
38. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM JoC **17**(2), 281–308 (1988)
39. Grassi, L., Hao, Y., Rechberger, C., Schofnegger, M., Walch, R., Wang, Q.: Horst meets fluid-spn: Griffin for zero-knowledge applications. In: Annual International Cryptology Conference. pp. 573–606. Springer (2023)
40. Grassi, L., Khovratovich, D., Lüftenegger, R., Rechberger, C., Schofnegger, M., Walch, R.: Reinforced concrete: A fast hash function for verifiable computation. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 1323–1335 (2022)
41. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A new hash function for {Zero-Knowledge} proof systems. In: 30th USENIX Security. pp. 519–535 (2021)
42. Grassi, L., Rechberger, C., Rotaru, D., Scholl, P., Smart, N.P.: Mpc-friendly symmetric key primitives. In: Proceedings of the 2016 ACM CCS. pp. 430–443 (2016)
43. Hsiang, J.H., Fu, S., Kuo, P.C., Cheng, C.M.: Pqscale: A post-quantum signature aggregation algorithm (2023)
44. Hülsing, A., Rausch, L., Buchmann, J.: Optimal parameters for xmss mt. In: International conference on availability, reliability, and security. pp. 194–208. Springer (2013)
45. Kales, D., Zaverucha, G.: Improving the performance of the picnic signature scheme. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 154–188 (2020)
46. Kaludjerović, N., Kleinjung, T., Kostic, D.: Improved key recovery on the legendre prf. Cryptology ePrint Archive (2020)
47. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Asiacrypt. pp. 177–194. Springer (2010)
48. Kattis, A., Panarin, K., Vlasov, A.: Redshift: transparent snarks from list polynomial commitment iops. Cryptology ePrint Archive (2019)

49. Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: Proceedings of the 2018 ACM CCS. pp. 525–537 (2018)
50. Khaburzaniya, I., Chalkias, K., Lewi, K., Malvai, H.: Aggregating and thresholdizing hash-based signatures using starks. In: AsiaCCS. ACM (2022)
51. Khovratovich, D.: Key recovery attacks on the legendre prfs within the birthday bound. Cryptology ePrint Archive (2019)
52. Lamport, L.: Constructing digital signatures from a one way function (1979)
53. Lin, S.J., Chung, W.H., Han, Y.S.: Novel polynomial basis and its application to reed-solomon erasure codes. In: 2014 ieee 55th FOCS. pp. 316–325. IEEE (2014)
54. May, A., Zweydinger, F.: Legendre prf (multiple) key attacks and the power of preprocessing. In: 2022 IEEE 35th Computer Security Foundations Symposium (CSF). pp. 428–438. IEEE (2022)
55. Merkle, R.C.: A certified digital signature. In: Asiacrypt. pp. 218–238. Springer (1989)
56. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon. Post-Quantum Cryptography Project of NIST (2020)
57. Reingold, O., Rothblum, G.N., Rothblum, R.D.: Constant-round interactive proofs for delegating computation. In: Proceedings of the 48th ACM ToC. pp. 255–340 (2016)
58. Delpech de Saint Guilhem, C., Orsini, E., Tanguy, T.: Limbo: Efficient zero-knowledge mpcith-based arguments. In: ACM CCS. pp. 3022–3036 (2021)
59. de Saint Guilhem, C.D., De Meyer, L., Orsini, E., Smart, N.P.: Bbq: using aes in picnic signatures. In: SAC. pp. 669–692. Springer (2019)
60. Seres, I.A., Horváth, M., Burcsi, P.: The legendre pseudorandom function as a multivariate quadratic cryptosystem: security and applications. Applicable Algebra in Engineering, Communication and Computing pp. 1–31 (2023)
61. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review **41**(2), 303–332 (1999)
62. Szepieniec, A., Ashur, T., Dhooghe, S.: Rescue-prime: a standard specification (sok). Cryptology ePrint Archive (2020)
63. Van Dam, W., Hallgren, S., Ip, L.: Quantum algorithms for some hidden shift problems. SIAM Journal on Computing **36**(3), 763–778 (2006)
64. Zaverucha, G., Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Katz, J., Wang, X., Kolesnikov, V., Kales, D.: Picnic. Technical report, National Institute of Standards and Technology (2020)
65. Zhang, J., Xie, T., Zhang, Y., Song, D.: Transparent polynomial delegation and its applications to zero knowledge proof. In: 2020 IEEE S&P. pp. 859–876. IEEE (2020)

Appendix

A Security of the Legendre PRF

Let us first define the Shifted Legendre Symbol Problem (SLSP) as follows.

Problem 1. (Shifted Legendre Symbol Problem [28]) Let p be an odd prime and K be a uniformly random value in $\mathbb{F}_p$. Given access to an oracle $\mathcal{O}$ that on input $a \in \mathbb{F}_p$, computes $\mathcal{O}(a) = \left(\frac{K+a}{p}\right)$, find K.

It is known that if the adversary is allowed to query $\mathcal{O}$ in a quantum mechanism superposition, then a polynomial-time quantum algorithm can determine K and break the pseudo-randomness of the PRF, using a single query [28, 63]. If the oracle can only be queried classically, Khovratovich [51] showed a memoryless collision search algorithm with $O(\sqrt{p}\log p)$ Legendre evaluations when $\sqrt{p}\log p$ queries are available. Recent works concurrently improved the attack presented in [51], either by exploiting the multiplicative property of the Legendre function [15, 46] or by allowing input-independent pre-computations [54].

In particular, if the number of oracle queries is bounded by $L \le \sqrt[4]{p}$, [15] presented a table-based collision search algorithm that recovers the hidden shift with a time complexity of $O(\frac{p\log^2 p}{L^2})$ Legendre symbol evaluations and a memory cost of $O(L^2)$. [46] shows a similar table-based collision attack that further reduces the run time complexity to $O(\sqrt{p\log\log p})$ on a $\Theta(\log p)$-bit machine by using $\sqrt[4]{p\log^2 p\log\log p}$ queries and $O(\sqrt{p\log\log p}\log p)$ memory. By allowing pre-computation [54], the query complexity can be brought down (from $\sqrt{p}\log p$) to $O(\sqrt[3]{p})$, with $O(\sqrt[3]{p^2})$ (offline) run time and $O(\sqrt[3]{p})$ memory. There also exists a multiple-key attack against the Legendre PRF with $O(\sqrt{mp})$ time/query complexity to recover m hidden shifts [54]. Another work [60] shows that recovering the hidden shift of the Legendre PRF is equivalent to solving a specific family of multivariate quadratic (MQ) equation systems, and algebraic cryptanalysis does not improve previous results.

With classical queries to the PRF, if we allow the model of classical memory with quantum random access (QRACM), the query complexity, time complexity and memory complexity (in QRACM) of [46] can be improved to $O(\sqrt[6]{p})$, $O(\sqrt[3]{p})$, and $O(\sqrt[3]{p})$, respectively. However, since there is no practical and scalable implementation of QRACM at the moment, [35] improved the attack by eliminating the requirement of QRACM. They showed that there exists a quantum attack on the SLSP with $2^{O(\sqrt{\log_2 p})}\sqrt[3]{p}$ time complexity, while using $\sqrt[3]{p}$ classical queries and a subexponential number of qubits memory.

Problem 2. (Key Recovery of Loquat) Let p be an odd prime. Given L random, uncontrolled PRF evaluations (i.e., public key) to the function $\mathcal{L}_K(\cdot) : \mathbb{F}_p \mapsto \{0,1\}$ for some secret $K \in \mathbb{F}_p$, find K.

It is easy to see that the key recovery of Loquat is at least as hard as the shifted Legendre symbol problem. Furthermore, the key recovery of Loquat imposes additional constraints upon the adversary, making many attacks discussed previously more unpractical. Notably, the polynomial-time quantum attacks introduced in [28, 63] become infeasible in the context of Loquat key recovery. This is due to the adversary's inability to perform quantum superposition queries on the PRF.

Moreover, if the number of oracle calls is bounded by L, the complexities of key recovery attacks in [51, 15, 46] become $O(\frac{p\log p}{L})$, $O(\frac{p\log^2 p}{L^2})$ and $O(\frac{p\log p\log\log p}{L^2})$, respectively. These complexities are exponential in relation to the security parameter, especially when L is sufficiently small. It is crucial to emphasise that the enhancements proposed in [15, 46] are reliant on the sequential characteristics of Legendre PRF queries. Consequently, it remains an open question whether these attacks can be extended to random point queries. In addition, it is worth noting that the attacks described in [35, 54] are not applicable, as both of them rely on a substantial number of oracle queries.

B Security Proofs

B.1 Proof of Theorem 1

We provide formal security analysis to our main theorem (Theorem 1) in this section. Note that the soundness analysis provided here is for IOP *standard* soundness. For state-restoration soundness, we refer the readers to

Proof. **Completeness.** We begin by showing the completeness of the protocol. If the polynomials are computed honestly in the protocol, then $(x, w) \in R_{\mathtt{US}}$ where $x = (\mathbb{F}, U, H, 4B+\kappa\cdot 2^{\eta}, \sum_{b=1}^{B} \lambda_b o_b)$ and $w = \hat{c}'(x)\cdot\hat{q}(x)$. It is not hard to observe that the degree of $\hat{f}(x) = \hat{c}'(x)\cdot\hat{q}(x)$ is indeed less than $4B+\kappa\cdot 2^{\eta}$ since $\deg(\hat{c}') = 2B + \kappa \cdot 2^{\eta}$ and $\deg(\hat{q}) = 2B - 1$, where $\hat{c}'(x) = \hat{c}(x) + Z_H(x)\hat{r}(x)$. Moreover, $\sum_{a\in H} \hat{c}'(a) \cdot \hat{q}(a) = \sum_{a\in H} \hat{c}(a) \cdot \hat{q}(a)$ since $0 = \sum_{a\in H} Z_H(a)\hat{r}(a)$. Recall that $\hat{c}|_H = (Kr_1, \dots, Kr_B, r_1, \dots, r_B)$ and $\hat{q}|_H = (\lambda_1, \dots, \lambda_B, \lambda_1 I_1, \dots, \lambda_B I_B)$, for challenges $I_b \in \mathcal{I}'$. Then we can write $\sum_{a\in H} \hat{c}(a) \cdot \hat{q}(a)$ as the sum of the entry-wise product between two vectors. That is,

$$\begin{aligned}
\sum_{a\in H} \hat{c}'(a) \cdot \hat{q}(a) &= \sum_{a\in H} \hat{c}(a) \cdot \hat{q}(a) \\
&= (Kr_1, r_1, \dots, Kr_B, r_B) \circ (\lambda_1, \lambda_1 I_1, \dots, \lambda_B, \lambda_B I_B) \\
&= Kr_1\lambda_1 + r_1\lambda_1 I_1 + \cdots + Kr_B\lambda_B + r_B\lambda_B I_B \\
&= \sum_{b=1}^{B} \lambda_b (K + I_b) r_b \\
&= \sum_{b=1}^{B} \lambda_b o_b
\end{aligned}$$

provided that $o_b = (K+I_b)r_b$ for all $b \in [1, B]$, and $\circ$ denotes the entry-wise product between vectors. Finally, $\mathcal{L}_0(o_b) = \mathcal{L}_0(K + I_b) + \mathcal{L}_0(r_b)$ owing to the multiplicative homomorphism of the Legendre PRF, it holds that $\mathcal{L}_0(o_b) = \mathcal{L}_K(I_b)+T_b$ provided that $T_b = \mathcal{L}_0(r_b)$ for all $b \in [B]$. The completeness holds.

Knowledge Soundness We now show the (standard) proof-of-knowledge of the IOP protocol for β-approximate Legendre PRF relation $R_{\beta\mathcal{L}}$. Given an instance $\mathcal{L}_K(\mathcal{I})$, if $\langle \tilde{\mathcal{P}}(\mathcal{L}_K(\mathcal{I})), \mathcal{V}(\mathcal{L}_K(\mathcal{I}))\rangle$ outputs 1 (i.e., $\mathcal{V}$ accepts) with non-negligible probability ε, then there exists an extractor $\mathcal{E}$ which extracts a β-approximate Legendre PRF witness K^* such that $(K^*, \mathcal{L}_K(\mathcal{I})) \in R_{\beta\mathcal{L}}$ with probability at least $\varepsilon - \epsilon_{\mathtt{US}} - (1-\beta)^B - 1/p$, where $\epsilon_{\mathtt{US}}$ is the knowledge error of the zero-knowledge univariate sumcheck.

If the verifier accepts, then there exists an extractor of the univariate sumcheck protocol that extracts a valid witness polynomial $\hat{c}'^*$ with probability at least $\varepsilon - \epsilon_{\mathtt{US}}$ such that $((\mathbb{F}, U, H, 4B + \kappa \cdot 2^\eta, \sum_{b=1}^{B} \lambda_b o_b^*), (\hat{c}'^* \cdot \hat{q})) \in R_{\mathtt{US}}$, where $\hat{q}(x) = \mathtt{Interpolate}(H, \mathbf{q})$ for vector $\mathbf{q} = (\lambda_1, \lambda_1 I_1, \ldots, \lambda_B, \lambda_B I_B)$ that is computed from $\mathcal{V}$'s messages.

Namely, $\sum_{b=1}^{B} \lambda_b o_b^* = \sum_{a\in H} \hat{c}'^*(a) \cdot \hat{q}(a)$ and $\deg(\hat{c}'^* \cdot \hat{q}) < 4B + \kappa \cdot 2^\eta$. According to the protocol, we have following equations:

$$\begin{aligned}
\sum_{b=1}^{B} \lambda_b o_b^* &= \sum_{a\in H} \hat{c}'^*(a) \cdot \hat{q}(a) \\
&= (c_1^*, r_1^*, \ldots c_B^*, r_B^*) \circ (\lambda_1, \lambda_1 I_1, \ldots, \lambda_B, \lambda_B I_B) \\
&= \lambda_1 c_1^* + \lambda_1 I_1 r_1^* + \cdots + \lambda_B c_B^* + \lambda_B I_B r_B^* \\
&= \lambda_1 (c_1^* + I_1 r_1^*) + \cdots + \lambda_B (c_B^* + I_B r_B^*)
\end{aligned}$$

Since $(\lambda_1, \ldots, \lambda_B)$ are random scalars sampled uniformly from $\mathbb{F}_p$ *after* the prover sends $(o_1^*, \ldots, o_B^*)$ in the second round, then $\sum_{b=1}^{B} \lambda_b o_b^* = \lambda_1(c_1^* + I_1 r_1^*) + \cdots + \lambda_1(c_B^* + I_B r_B^*)$ implies that for all $b \in [B]$, $\lambda_b o_b^* = \lambda_b(c_b^* + I_b r_b^*)$. In other words, for each $b \in [B]$, $o_b^* = c_b^* + I_b r_b^*$ with probability at least $\varepsilon - \epsilon_{\mathtt{US}} - 1/p$.

$\mathcal{E}$ computes for $b \in [B]$, $K_b^* = \frac{c_b^*}{r_b^*}$. We assume that for all $b \in [B]$, $(K_b^*, \mathcal{L}_K(\mathcal{I})) \notin R_{\beta\mathcal{L}}$ and show the probability that $\mathcal{L}_0(o_b^*) = \mathcal{L}_K(I_b) + T_b^*$ holds for all $b \in [B]$ is upper-bounded by $(1-\beta)^B$.

Given $K_b^* = \frac{c_b^*}{r_b^*}$, we can re-write o_b^* as $o_b^* = (K_b^* + I_b) r_b^*$. Since $\mathcal{L}_0(o_b^*) = \mathcal{L}_K(I_b) + T_b^*$ for all $b \in [B]$, we have $\mathcal{L}_{K_b^*}(I_B) + \mathcal{L}_0(r_b^*) = \mathcal{L}_K(I_b) + T_b^*$. If K_b^* is not a β-approximate witness, then we have the following inequality for all $b \in [B]$,

$$\beta_b L > \beta L \tag{3}$$

where $\beta_b L = d_H\left(\left(\mathcal{L}_{K_b^*}(\mathcal{I}) + \mathcal{L}_0(r_b^*), \ldots, \mathcal{L}_0(r_b^*)\right), (T_b^*, \mathcal{L}_K(I_B))\right)$ and d_H denotes the Hamming distance.

In other words, $\mathcal{L}_0(o_b^*) = T_b^* + \mathcal{L}_K(I_b)$ and $(K_b^*, \mathcal{L}_K(\mathcal{I})) \notin R_{\beta\mathcal{L}}$ implies that the challenge set $\mathcal{I}'$ is chosen from the fraction of $(1-\beta_b)L$ for all $b \in [B]$. Recall that $\mathcal{I}'$ is uniformly chosen after $\tilde{\mathcal{P}}$ sends the oracle codeword $\hat{c}'^*|_U$, then the probability that for all $b \in [1, B]$, $I_b \in \mathcal{I}'$ is from the fraction $(1-\beta_b)L$ is $\prod_{b=1}^{B}(1-\beta_b)$ which is strictly less than $(1-\beta)^B$. Thus, the probability that $\mathcal{E}$ fails to extract a β-approximate witness from a valid transcript generated by $\langle \tilde{\mathcal{P}}(\mathcal{L}_K(\mathcal{I})), \mathcal{V}(\mathcal{L}_K(\mathcal{I}))\rangle$ is at most $\epsilon_{\mathtt{US}} + 1/p + (1-\beta)^B$.

Proof Length. The proof length is the total number of bits sent from the prover $\mathcal{P}$ to the verifier $\mathcal{V}$. In the first round, the prover sends an oracle codeword $\mathbf{f}_1 = \hat{c}'|_U$ and a plaintext message $\mathbf{p}_1 = (T_1, \ldots, T_B)$ which has $|U| \cdot |\mathbb{F}| + B$ bits (i.e., each T_b is one bit since it is the output of the Legendre PRF), where

$|\mathbb{F}|$ denotes the bit-length of the field elements. Then, in the second round, the prover sends a plaintext message $\mathbf{p}_2 = (o_1, \ldots, o_B)$ where $o_b \in \mathbb{F}_p$. Thus, the total bit length for the second round is $B \cdot \log p$. From the third round, the prover and the verifier start to simulate the univariate sumcheck protocol, which has total bit length $p_{\mathtt{US}}$. Hence, the total proof length is $|U| \cdot |\mathbb{F}| + B + B \cdot \log p + p_{\mathtt{US}}$ bits.

Query Complexity. The query complexity denotes the total number of locations queried by the verifier across all of the prover's messages. In the first round, the prover sends an oracle codeword f_1 which will later be queried at $\kappa \cdot 2^{\eta}$ locations in order to perform the low-degree test. Hence, the query complexity is $\kappa \cdot 2^{\eta} + q_{\mathtt{US}}$.

Prover Complexity and Verifier Complexity. The prover in Algorithm 1 needs to interpolate and evaluate polynomials $\hat{c}'$ and $\hat{q}$, which has costs of $2 \cdot FFT(IFFT(|H|), |U|)$. It computes B Legendre PRFs (i.e., $T_b = \mathcal{L}_0(r_b)$ for $b \in [B]$), which has $B \cdot O(\log p)$ arithmetic complexity. The verifier additionally computes B Legendre PRF (i.e., checks if $\mathcal{L}_0(o_b) = \mathcal{L}_0(I_b) + T_b$ for all $b \in [B]$), which costs $B \cdot O(\log p)$. Note that we omits the operations required to compute $Z_H(x)$ since for a fixed multiplicative coset H, it can be computed in an offline phase.

Zero-Knowledge. We present a construction of the simulator $\mathcal{S}$ which is given the straight-line access (i.e., no rewinding) to a malicious verifier $\tilde{\mathcal{V}}$, it perfectly simulates $\tilde{\mathcal{V}}$'s view in the real protocol.

The simulator $\mathcal{S}$ picks $2B$ field elements $\mathbf{c} = (c_1, \ldots, c_{2B})$ uniformly at random and computes $\hat{c} = \mathtt{Interpolate}(H, \mathbf{c})$. It outputs $\mathbf{f}_1^* = \hat{c}|_U$. Given the second round random challenge $\mathcal{I}'$ from $\tilde{\mathcal{V}}$, it computes $o_b^* = c_b + c_{b+1} I_b$ and $T_b^* = \mathcal{L}_0(o_b^*) - \mathcal{L}_K(I_b)$. Outputs the first round message $\mathbf{f}_1^*$ and $p_1^* = (T_1, \ldots, T_B)$ and follows the description of the protocol.

With $\kappa \cdot 2^{\eta}$ queries, the verifier cannot distinguish between $\mathbf{f}_1^*$ (a random polynomial) and $\mathbf{f}_1$ (masked by a random polynomial $\hat{r}(x)$ with degree $\kappa \cdot 2^{\eta}$). $T_b^* = \mathcal{L}_0(o_b^*) - \mathcal{L}_K(I_b)$ for $b \in [B]$ are indistinguishable from $T_b = \mathcal{L}_0(r_b)$ since r_b are chosen uniformly at random from $\mathbb{F}_p$. For the second round, $o_b^* = c_b + c_{b+1} I_b$ and $o_b = K r_b + I_b r_b$ are indistinguishable owing to $r_b \overset{\$}{\leftarrow} \mathbb{F}_p$. The rest of the proof follows from the zero-knowledge univariate sumcheck. $\square$

B.2 Proof of Lemma 2

We show the restricted state-restoration soundness of our key identification protocol as shown in Algorithm 1 (with zero-knowledge enabled from Remark 3).

Proof. In the restricted state-restoration attack, the malicious prover $\tilde{\mathcal{P}}$ commits to the messages in initial round (i.e., $\mathbf{f}_1 = \hat{c}'^*|_U$ and $\mathbf{p}_1 = (T_1^*, \ldots, T_B^*)$) and starts its interaction with $\mathcal{V}$. Then, in all later rounds, $\tilde{\mathcal{P}}$ can rewind the verifier to any round (except for the initial round), and the verifier continue with fresh randomness. For any b-bound $\tilde{\mathcal{P}}$ to convince the verifier (without the knowledge of K), it must pass the following checks:

1. $\mathcal{V}_{\mathtt{US}}(\mathbb{F}, U, H, 4B + \kappa \cdot 2^\eta, \sum_{b=1}^{B} \lambda_b o_b^*)$ accepts, and
2. $\mathcal{L}_0(o_b^*) = \mathcal{L}_K(I_b) + T_b^*$ for all $b \in [B]$ and non-zero o_b^*.

Cheat on the second round. Suppose $\tilde{\mathcal{P}}$ rewinds $\mathcal{V}$ to the second round b_1 times, where $b_1 \le b - k_{\mathtt{US}}$. It finds the "best" set $\mathcal{I}'$ such that there exists a subset $\bar{\mathcal{I}}' \subseteq \mathcal{I}'$ with maximum size. For all $I_b \in \bar{\mathcal{I}}'$, $o_b^* = c_b^* + r_b^* I_b$ and $\mathcal{L}_0(o_b^*) = pk_{I_b} + T_b^*$. We denote $X = \max(X_1, \ldots, X_{b_1})$ where X_i are i.i.d. as $\mathfrak{B}(B, (1-\beta))$.
Cheat on the third round. Suppose $\tilde{\mathcal{P}}$ rewinds $\mathcal{V}$ to the third round b_2 times, where $b_2 \le b - b_1 - (k_{\mathtt{US}} - 1)$. It finds the "best" set of $(\lambda_1, \ldots, \lambda_B)$ such that for the remaining $B - |\bar{\mathcal{I}}'|$, $I'_b \in \mathcal{I}/\bar{\mathcal{I}}'$ (i.e., those has $\mathcal{L}_0(o_b^*) = pk_{I'_b} + T_b^*$ but $o_b^* \ne c_b^* + r_b^* I'_b$), $\sum_{I'_b \in \mathcal{I}'/\bar{\mathcal{I}}'} \lambda_b o_b^* = \sum_{I'_b \in \mathcal{I}'/\bar{\mathcal{I}}'} \lambda_b (c_b^* + r_b^* I'_b)$. We denote $Y = \max(Y_1, \ldots, Y_{b_2})$ where Y_i are i.i.d. as $\mathfrak{B}(B - X, 1/p)$.
Cheat on the univariate sumcheck. For the remaining indices, $\tilde{P}$ has to cheat on the univariate sumcheck within $b_3 \le b - b_1 - b_2$ rewinds. We denote by $Z = \max(Z_1, \ldots, Z_{b_3})$ where Z_i are i.i.d. as $\mathfrak{B}(B - X - Y, \epsilon_{\mathtt{US}})$. □

B.3 Proof of Theorem 3

Proof. Let $\mathcal{A}$ be an adversary against EUF-CMA security of Loquat, we construct an adversary $\mathcal{B}$ against EUF-KO. When $\mathcal{B}$ is run on input pk, it starts $\mathcal{A}$ on input pk. We first define the construction of the signing oracle and random oracle, then argue that the simulated signature is indistinguishable from real ones, and finally show that EUF-KO implies EUF-CMA security.

- **Simulating random oracles.** $\mathcal{B}$ maintains a list of input-output pairs for random oracle queries and a list of "bad" outputs denoted as $\mathtt{Bad}_{\mathcal{H}}$ which stores all random oracle outputs. If the query was made before by $\mathcal{A}$, then $\mathcal{B}$ responses with the same answer. Otherwise, $\mathcal{B}$ returns a uniformly random output and record the new input-output pair in the list.
- **Simulating signing oracle.** When $\mathcal{A}$ queries the signing oracle with message M, $\mathcal{B}$ simulates the signature σ and program the output of the $\mathcal{H}$ of the second round to generate the desired challenge indices. $\mathcal{B}$ simulates the signing oracle as follows:
  1. On input a message M, $\mathcal{B}$ randomly samples a fake key $K' \xleftarrow{\$} \mathbb{F}_p^*$ and for $j \in [n]$, $(r_{1,j}, \ldots, r_{m,j}) \xleftarrow{\$} \mathbb{F}_p^*$. Then it follows the protocol to compute the vector $\mathbf{c}_j = (K' r_{1,j}, r_{1,j}, \ldots, K' r_{m,j}, r_{m,j})$. $\mathcal{B}$ follows the protocol to obtain the Merkle root $\mathtt{root}_c$.
  2. $\mathcal{B}$ generates a random output $h_1 \in \{0,1\}^{2\lambda}$, if $h_1 \in \mathtt{Bad}_{\mathcal{H}}$, $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ computes $(I_{1,j}, \ldots, I_{m,j})_{j \in [n]} \leftarrow \mathtt{Expand}(h_1)$ and $T_{i,j} = \mathcal{L}_0((K' + I_{i,j}) r_{i,j}) - pk_{I_{i,j}}$, for $i \in [m], j \in [n]$.
  3. $\mathcal{B}$ programs the output of $\mathcal{H}$ to be h_1 on input $(\mathtt{root}_c, (T_{1,j}, \ldots, T_{m,j})_{j \in [n]}, M)$.
  4. $\mathcal{B}$ computes $o_{i,j} = (K' + I_{i,j}) r_{i,j}$ for all $b \in [B]$.
  5. $\mathcal{B}$ follows the rest of the protocol.

6. Finally, $\mathcal{B}$ outputs $((T_{i,j}, o_{i,j})_{i\in[m],j\in[n]}\pi_{\texttt{US}}, \pi_{\texttt{LDT}})$.

When $\mathcal{A}$ outputs a forgery for the EUF-CMA game, $\mathcal{B}$ forwards the forgery as its forgery for the EUF-KO game.

We show that the simulation is not distinguishable from the real signature. If $\mathcal{B}$ did not abort, then the simulation of the random oracle is perfect. $\mathcal{A}$ can only distinguish the simulated signature if $\mathcal{B}$ failed to program the output of $\mathcal{H}$ on input $(\texttt{root}_c, (T_{1,j}, \ldots, T_{m,j})_{j\in[n]}, M)$.

In other words, $(\texttt{root}_c, (T_{1,j}, \ldots, T_{m,j})_{j\in[n]}, M)$ was queried before. Otherwise, the simulated signature follows exactly the same distribution as the real signature.

We now establish $\mathcal{B}$'s advantage against EUF-KO. There are two situations that $\mathcal{B}$ may abort during the simulation of the signing oracle, which are shown as follows.

1. In step 2, if h_1 was used in one of the previous simulated signatures, which happens with probability $q_s^2 2^{-2\lambda}$.
2. $\mathcal{B}$ failed to program the output of the $\mathcal{H}$ to generate desired challenges, that is, $(\texttt{root}_c, (T_{1,j}, \ldots, T_{m,j})_{j\in[n]}, M)$ was queried before, which happens with probability $(q_s q_{\mathcal{H}_1})2^{-2\lambda}$.

Therefore, if $\mathbf{Adv}_{\mathcal{A}}^{\texttt{EUF-CMA}}$ wins the game with probability ε, $\mathcal{B}$ wins the EUF-KO with the probability $\varepsilon - (q_s(q_s + q_{\mathcal{H}_1})) \cdot 2^{-2\lambda}$. □

B.4 Proof of Theorem 4

Proof. Let $\mathcal{A}$ be an adversary against the aggregate chosen-key security of $\Pi_{\texttt{Angus}}$, we construct an adversary $\mathcal{B}$ against the unforgeability of $\Pi_{\texttt{Loquat}}$ in the URS model. When $\mathcal{B}$ runs on input pk_1 that is generated using the $\texttt{KeyGen}$ algorithm, it starts $\mathcal{A}$ on input pk_1. Denote the extractor of the SNARK protocol as $\mathcal{E}_{\texttt{SNARK}}$ with extracting probability $e_{\texttt{SNARK}}$. After $\mathcal{A}$ terminates, $\mathcal{E}_{\texttt{SNARK}}$ extracts a valid witness $\mathbb{w} = (\sigma'_1, \sigma'_2, \ldots, \sigma'_N)$.

We now argue that if $\mathbb{w} = (\sigma'_1, \sigma'_2, \ldots, \sigma'_N)$ is a valid witness against $\mathbb{x} = ((M_1, pk_1), \ldots, (M_N, pk_N))$, then every signature σ'_e for $e \in [N]$ is a valid Loquat signature. We can rewrite $\sigma'_e = (\sigma_e, (s_{i,j}^{(e)})_{i\in[m],j\in[n]})$. Then, we prove that for all $e \in [N]$, $1 \leftarrow \texttt{L-Verify}(\sigma_e, M_e, pk_e, \texttt{L-pp})$. We first split the tasks of Loquat verification to three parts:

1. verifying hash chain and Merkle tree;
2. verifying non-interactive univariate sumcheck proof;
3. verifying quadratic residuosity of o_b against $pk_{I_{i,j}} + T_b$ for all $b \in [1, B]$.

Since the hash chain, Merkle tree, and non-interactive univariate sumcheck are directly proven in the circuit, then they are valid for each signature with the same probability $\epsilon - e_{\texttt{SNARK}}$. Now we argue that if there exists a set of valid $(s_{i,j}^{(e)})_{i\in[m],j\in[n]}$ such that $\left(s_{i,j}^{(e)}\right)^2 = o_{i,j}^{(e)} \cdot \left(\alpha \cdot \left(pk_{I_{i,j}}^{(e)} + T_{i,j}^{(e)}\right) + \left(1 - \left(pk_{I_{i,j}}^{(e)} + T_{i,j}^{(e)}\right)\right)\right)$, then the verification of quadratic residuosity of $o_{i,j}^{(e)}$ against $pk_{I_{i,j}}^{(e)} + T_{i,j}^{(e)}$ for all $i \in [n], j \in [M]$ and $e \in [N]$ must also pass.

Indeed, since the circuit proves that the computation of $t_{i,j}^{(e)}$ is correct for every $b \in [1, B]$ and $i \in [1, n]$, where $t_{i,j}^{(e)} = o_{i,j}^{(e)} \cdot \left(\alpha \cdot \left(pk_{I_{i,j}}^{(e)} + T_{i,j}^{(e)}\right) + \left(1 - \left(pk_{I_{i,j}}^{(e)} + T_{i,j}^{(e)}\right)\right)\right)$, and it proves that $t_{i,j}^{(e)}$ is a quadratic residue by finding some $s_{i,j}^{(e)}$ (outside of the circuit) such that $\left(s_{i,j}^{(e)}\right)^2 = t_{i,j}^{(e)}$, it implies that $\mathcal{L}_0(o_{i,j}^{(e)}) = pk_{I_{i,j}}^{(e)} + T_{i,j}^{(e)}$ for non-zero $o_{i,j}^{(e)}$. Specifically, we can argue the following two cases:

1. $\mathcal{L}_0(o_{i,j}^{(e)}) = 0$ and $pk_{I_{i,j}}^{(e)} + T_{i,j}^{(e)} = 1$. We prove that in this case, $s_{i,j}^{(e)}$ doesn't exist. Since $t_{i,j}^{(e)}$ is computed in the form of $o_{i,j}^{(e)} \cdot (\alpha \cdot 1 + (1-1)) = o_{i,j}^{(e)}\alpha$, where α is a public quadratic non-residue. If $o_{i,j}^{(e)} = 0$, then $o_{i,j}^{(e)}$ is a quadratic residue, and $\alpha \cdot o_{i,j}^{(e)}$ is a quadratic non-residue, and there does not exist a valid $s_i^{(e)}$ such that $\left(s_{i,j}^{(e)}\right)^2 = \alpha \cdot o_{i,j}^{(e)}$. Thus the SNARK proof fails that contradicts with the assumption where Σ is a valid aggregate signature.
2. $\mathcal{L}_0(o_{i,j}^{(e)}) = 1$ and $pk_{I_{i,j}}^{(e)} + T_{i,j}^{(e)} = 0$. In this case, $s_{i,j}^{(e)}$ doesn't exist as well. Since $t_{i,j}^{(e)} = o_{i,j}^{(e)} \cdot (\alpha \cdot 0 + (1-0)) = o_{i,j}^{(e)}$ and $\mathcal{L}_0(o_{i,j}^{(e)}) = 1$ implies that $o_{i,j}^{(e)}$ is a quadratic non-residue, then there does not exist a valid $s_{i,j}^{(e)}$ such that $\left(s_{i,j}^{(e)}\right)^2 = o_{i,j}^{(e)}$. Thus the SNARK proof fails in this case, which contradicts with the assumption where Σ is a valid aggregate signature.

Therefore, for any non-zero $o_{i,j}^{(e)}$, Σ is a valid aggregate signature implies that $\mathcal{L}_0(o_{i,j}^{(e)}) = pk_{I_{i,j}}^{(e)} + T_{i,j}^{(e)}$ for all $b \in [1, B]$ and $i \in [1, n]$. To sum up, the signature σ_i for all $i \in [1, n]$ is valid. Thus, $\mathcal{B}$ is able to forge the signature against Loquat in the URS model using $\mathcal{A}$. □

B.5 Proof of Theorem 5

Proof. The security proof is similar to the the proof of Theorem 4, where given a challenge public key pk_1, if $\mathcal{A}$ outputs a valid SNARK proof π_k after k times of the proof recursion, by making at most q_s queries to the signing oracle, then the algorithm $\mathcal{B}$ can break the unforgeability of Loquat in the URS model running $\mathcal{E}_{\texttt{SNARK}}$. Namely, after $\mathcal{A}$ terminates, $\mathcal{E}_{\texttt{SNARK}}$ is able to extract a valid witness $\mathbb{w} = (\sigma'_1, \ldots, \sigma'_N)$ recursively.

As we have proved in Section B.4, where if the adversary $\mathcal{A}$ who outputs a valid (ordinary) SNARK proof, the extractor can extract the corresponding witness. We now show the process of recursive extraction.

If π_γ is a valid SNARK proof, then $1 \leftarrow \texttt{SNARK-V}((\mathfrak{C}_\gamma, \mathfrak{R}_\gamma), \pi_\gamma, \mathbb{x})$, where $\mathbb{x} = ((M, PK)_\gamma, \mathbb{x}_{\gamma-1})$. $\mathcal{E}_{\texttt{SNARK}}$ runs in time $t_{\texttt{SNARK}}$ to extracts $\mathbb{w}$ which is a valid SNARK witness $\mathbb{w} = (\Gamma_\gamma, \pi_{\gamma-1})$ with probability at least $\epsilon_\gamma = 1 - e_{\texttt{SNARK}}$. Hence, all signatures in Γ_γ are valid (i.e., the last chunk of signatures) and the previous proof $\pi_{\gamma-1}$ is valid against $\mathbb{x}_{\gamma-1}$. Since $\pi_{\gamma-1}$ is a valid SNARK proof, then the extractor extracts a witness $\mathbb{w}_{\gamma-1} = (\mathit{\Gamma}_{\gamma-1}, \pi_{\gamma-2})$ with probability at least $\epsilon_{\gamma-1} = 1 - \epsilon_\gamma - e_{\mathtt{SNARK}}$. In other words, signatures in $\mathit{\Gamma}_{\gamma-1}$ are all valid and $\pi_{\gamma-2}$ is a valid SNARK proof against $\mathbb{x}_{\gamma-2}$. The recursive process repeats γ times, with the final probability $\epsilon_1 = 1 - \gamma \cdot e_{\mathtt{SNARK}}$, the extractor extracts a list of valid signatures $\mathbb{w}_1 = \mathit{\Gamma}_1$, which contains a valid signature against $(M, PK)_1$. □

C Analysis of Signatures' SNARK-Friendliness

C.1 Loquat

We first analyse the number of arithmetic operations (mainly multiplications) and hash evaluations involved in Loquat verification algorithm.

Hash Evaluations There are three parts in Loquat verification that requires hash evaluations, including

- Recompute hash chain.
- Expand the challenges.
- Verifying Merkle Commitments.

Recompute Hash Chain. We denote the security parameter to be λ.

1. Recompute $h_1 \leftarrow \mathcal{H}_1(\mathtt{root}_c, (T_{i,j})_{i\in[m],j\in[n]}, M) : \mathcal{H}_1 : \{0,1\}^{4\lambda} \to \{0,1\}^{2\lambda}$.
2. Recompute $h_2 \leftarrow \mathcal{H}_2((o_{i,j})_{i\in[m],j\in[n]}, h_1)$: $\mathcal{H}_2 : \{0,1\}^{(B+2)\lambda} \to \{0,1\}^{2\lambda}$.
3. Recompute $h_3 \leftarrow \mathcal{H}_3(\mathtt{root}_s, S, h_2)$: $\mathcal{H}_3 : \{0,1\}^{6\lambda} \to \{0,1\}^2\lambda$.
4. For $i = [4, 5 + r]$, recompute $h_i \leftarrow \mathcal{H}_i(\mathtt{root}, h_{i-1})$: $\mathcal{H}_i : \{0,1\}^{4\lambda} \to \{0,1\}^{2\lambda}$.

Overall, the verifier computes $3 + r$ hash evaluations with input length 4λ, 1 evaluation with input length $(B + 2)\lambda$, and 1 evaluation with input length 6λ.

Expand the Challenges. For expanding h_1, instead of getting $(I_{i,j})_{i\in[m],j\in[n]} \in \mathcal{I}$, we let the prover and the verifier expand h_1 to get indices $(I_{i,j})_{i\in[m],j\in[n]} \in [L]$. This change effectively reduces the required output length for expanding h_1. Similarly, expanding h_{5+r} to obtain indices in $[|U|]$. Since 2^η code elements are hashed together, the number of leaf nodes in each tree decreases to $|U|/2^\eta$.

1. $(I_{i,j})_{i\in[m],j\in[n]} \leftarrow \mathtt{Expand}(h_1)$: output length $\{0,1\}^{B\times\log_2 L}$.
2. $((\lambda_{i,j})_{i\in[m]}, \epsilon_j)_{j\in[n]}$: output length $\{0,1\}^{B\lambda+2n\lambda}$.
3. $\mathsf{e} \leftarrow \mathtt{Expand}(h_4)$: output length $\{0,1\}^{16\lambda}$.
4. $(S_1, \ldots, S_\kappa) \leftarrow \mathtt{Expand}(h_{5+r})$: output length $\{0,1\}^{(\log|U|-2)\kappa}$.
5. For $i = 3$ and $i \in [5, 4 + r]$, output length $\{0,1\}^{2\lambda}$. If using algebraic hash functions and set output length to be 2λ, then we can omit the expansions in these steps.

Verifying Merkle Commitments. We first introduce several optimisations in order to reduce the total number of constraints. We choose $t = \lceil \log_2(\kappa) - 1 \rceil$ as in Fractal [27].

In the following analysis, we mainly consider the verification of main codeword (i.e., the polynomials that are evaluated over the code domain U). This is because the hash calls of other rounds in LDT can be small with the optimisation.

1. Verify $4 \cdot \kappa$ Merkle tree authentication paths for $\texttt{root}_c, \texttt{root}_s, \texttt{root}_h, \texttt{root}_{f^{(0)}}$, with the number of leaf nodes $|U|/2^\eta$ and tree cap t: $4 \times \kappa \times (\log_2(|U|) - \eta - t)$ invocation to $\mathcal{H}_{\texttt{MT}} : \{0,1\}^{4\lambda} \to \{0,1\}^{2\lambda}$.
2. Recompute κ leaf nodes for each tree: $4 \times \kappa$ invocations to $\mathcal{H}_{\texttt{leaf}} : \{0,1\}^{2^{\eta+1}\lambda} \to \{0,1\}^{2\lambda}$.
3. Recompute tree cap: 4 calls to $\mathcal{H}_{\texttt{cap}} : \{0,1\}^{2^{t-1}\lambda} \to \{0,1\}^{2\lambda}$.

Hash-based Computations	**Number of Hashes**
Recompute Hash Chain	$5 + r$
Expand Challenges	4
Verify Merkle Commitments	$4\kappa(\log_2(\|U\|) - \eta - t + 1) + 4$
Total	$4\kappa(\log_2(\|U\|) - \eta - t + 1) + r + 13$

Table 7. Number of Hash Evaluations in LOQUAT Verification

Algebraic Operations The algebraic operations of LOQUAT verifier mainly involves

1. Interpolate the challenge polynomial $(\hat{q}_j(x))_{j \in [n]}$ and evaluate the polynomial on $\kappa \cdot 2^\eta$ points.
2. Recompute code elements for rational constraints and interleaved code.
3. Verify polynomials to check round consistency in the low-degree test.
4. Check if $\mathcal{L}_0(o_b) = pk_{I_{i,j}} + T_b$ for all $b \in [B]$.

For interpolating the polynomial $\hat{q}_j(x)_{j \in [n]}$, the verifier is given vectors $\mathbf{q}_j = (\lambda_{1,j}, \lambda_{1,j} I_{1,j}, \ldots, \lambda_{m,j}, \lambda_{m,j} I_{m,j})$ and a multiplicative coset $H \subset \mathbb{F}$ with $|H| = 2m$. According to [53], polynomial evaluation over a smooth multiplicative subgroup of size n can be performed in $O(n \log n)$ field operations via FFT and IFFT respectively. Hence, interpolating $(\hat{q}_j(x))_{j \in [n]}$ requires $2mn \times (1 + \log m)$ operations. Then, the verifier evaluates $(\hat{q}_j(x))_{j \in [n]}$ on $\kappa \cdot 2^\eta$, with $\kappa \cdot 2^\eta n(1 + \log m)$ operations. Hence, the total number of interpolating and evaluating polynomial $n(2m + \kappa \cdot 2^\eta)(1 + \log m)$ operations.

The verifier next re-computes code elements for rational constraint $\hat{p}(x)$ which is defined as $\hat{p}(s) = \frac{|H|(z\hat{f}(s) + \hat{s}(s)) - |H| Z_H(s)\hat{h}(s) - (z\mu + S)}{|H| s}$, for $s \in Q$, where Q

denotes the LDT queries ($|Q| = \kappa \cdot 2^\eta$). To recompute $\hat{f}(s) = \sum_{j=1}^{n} \epsilon_j \hat{f}_j(s)$, the number of multiplications is $n \cdot \kappa \cdot 2^\eta$, while the number of addition is $(n-1) \cdot \kappa \cdot 2^\eta$. To recompute $\hat{f}_j(s) = \hat{c}_j(s) \cdot \hat{q}_j(s)$ requires $n \cdot \kappa \cdot 2^\eta$ multiplicaitons. To recompute $\mu = \sum_{i=1}^{m}(\sum_{j=1}^{n} \lambda_{i,j} o_{i,j})$, which requires B multiplications and $B-1$ additions. To recompute $Z\mu + S$ requires 1 multiplication and 1 addition. Thus, the recomputation of $\hat{p}(s)$ requires $5 \cdot \kappa \cdot 2^\eta$ multiplications and $3 \cdot \kappa \cdot 2^\eta$ additions.

The verifier also needs to recompute the interleaved code according to the following equation:

$$\begin{aligned}\hat{f}^{(0)}(x) = e_1 \cdot \sum_{j=1}^{n}(\hat{c}'_j(x)) + e_2 \cdot \hat{s}(x) + e_3 \cdot \hat{h}(x) + e_4 \cdot \hat{p}(x) \\ + e_5 \cdot x^{(\rho^* - \rho_1)|U|} \cdot \hat{c}'(x) + e_6 \cdot x^{(\rho^* - \rho_2)|U|} \cdot \hat{s}(x) \\ + e_7 \cdot x^{(\rho^* - \rho_3)|U|} \cdot \hat{h}(x) + e_8 \cdot x^{(\rho^* - \rho_4)|U|} \cdot \hat{p}(x)\end{aligned}$$

Since $x^{(\rho^* - \rho_i)}|U|$ where $i \in \{1, \ldots, 4\}$ can be computed outside of the circuit, the total number of multiplications to recompute codeword for $\hat{f}^{(0)}$ is $8 \cdot \kappa \cdot 2^\eta$, while the total number of additions is $(n+6) \cdot \kappa \cdot 2^\eta$.

To check round consistency in the low-degree test, we can use the constraint system designed in Fractal [27, Section 12.2.2]. That is, given an identifier of a coset $S \subset U$, checking round consistency costs $2|S| + O(\log(|S|))$, where $|S| = 2^\eta$ Since we choose the localisation parameter $\eta = 2$, $|S| = 2^\eta = 4$. Since we have r rounds, with each round, the verifier interpolate and evaluation κ polynomials with degree $2^\eta - 1$, the total cost is $r \cdot \kappa \cdot (2^{\eta+1} + \eta)$.

Finally, we analyse the number of constraints to check $\mathcal{L}_0(o_{i,j}) = pk_{I_{i,j}} + T_{i,j}$ for all $i \in [m], j \in [n]$. In order to avoid heavy computation of Legendre PRF in the circuit, we let the verifier computes $t_{i,j} = o_{i,j} \cdot (\alpha \cdot (pk_{I_{i,j}} + T_{i,j}) + 1 - (pk_{I_{i,j}} + T_{i,j}))$ and then checks if $s_{i,j}^2 = t_{i,j}$ (i.e., check if $t_{i,j}$ is a quadratic residue). This step costs $3B$ multiplications and $4B$ additions. We summarise the analysis in Table 8.

Algebraic Computations	**Number of Operations**
Interpolate and Evaluate $\hat{q}(x)$	$n(2m + \kappa \cdot 2^\eta)(1 + \log m)$.
Recompute Missing Code Elements	$(2n + 13)\kappa \cdot 2^\eta + B + 1$
Verify Round Consistency of LDT	$r \cdot \kappa \cdot (2^{\eta+1} + \eta)$
Check Quadratic Residues	$3B$
Total	$\kappa \cdot 2^\eta(3n + n \log m + 13 + 2r) +$ $2mn(1 + \log m) + nr\kappa + 4B + 1$

Table 8. Number of Algebraic Operations in Loquat Verification

C.2 Other Signature Schemes

We analyse the algebraic operations and hash operations involved in Picnic [25, 45], LegRoast [16], Banquet [6], Rainer[30], and SHPINCS+[14] using the following method.

Picnic1 (without preprocessing) [25] Choose the MPC-in-the-head parameter $n = 3$ and the repetition parameter $M = 438$ with security level 128, the number of multiplications is 2,680,560, while the number of hash operations is 7,008 in the signature verification algorithm.

Picnic3 (with preprocessing) [45] Choose the MPC-in-the-head parameter $n = 16$, the repetition parameter $M = 604$, and the preprocessing parameter $\tau = 68$, with security level 128. The number of multiplications is 3,121,200 while the number of total hash operations is 1,679,784 in the signature verification algorithm.

LegRoast [16] Choose the MPC-in-the-head parameter $n = 16$, the repetition parameter $M = 54$, and the Legendre PRF parameter $B = 9, \beta = 0.449$ with NIST security level I. The choice of these parameters result in best verification time. The number of multiplications is 26,631 and the number of hash operations is 10,152 in the signature verification algorithm.

Banquet [6] Choose the MPC-in-the-head parameter $n = 16$ and the number of parallel executions $\tau = 41$ for 128-bit security (result in best verification time), the verification algorithm has 4,147 hash evaluations and 11,348,390 algebraic computations.

Rainer [30] Choose the MPC-in-the-Head parameter $N = 16$ and repetition parameter $\tau = 33$ for Rainer3 with 128-bit security (result in best verification time), the verification algorithm has 3,337 hash evaluations and 25,763,233 algebraic computations.

SPHINCS+. In SPHINCS+ [14, Table 1], the authors provided an overview of the number of hash calls required for key generation, signing, and verification. We use the parameters of SPHINCS+(128s), where $n = 16, h = 63, d = 7, \log(t) = 12, k = 14, w = 16$, and $len = 35$ with security level 133, to obtain the total number of hash invocations in key generation (305,663), sign (2,397,129), and verification (4,172). We also consider SPHINCS+(128f), with parameters $n = 16, h = 66, d = 22, \log(t) = 6, k = 33, w = 16$, and $len = 35$ for security level 128. The resulting total number of hash invocations in key generation is 4,775, signing is 111,353, and verification is 12,639.