Shorter Signatures Based on Tailor-Made Minimalist Symmetric-Key Crypto

Christoph Dobraunig\textsuperscript{1}, Daniel Kales\textsuperscript{2}, Christian Rechberger\textsuperscript{2}, Markus Schofnegger\textsuperscript{2}, and Greg Zaverucha\textsuperscript{3}

\textsuperscript{1}Lamarr Security Research, Graz, Austria
\textsuperscript{2}IAIK, Graz University of Technology, Graz, Austria
\textsuperscript{3}Microsoft Research, Redmond, WA, USA

April 25, 2022

Abstract

Signature schemes based on the MPC-in-the-head approach (MPCitH) have either been designed by taking a proof system and selecting a suitable symmetric-key primitive (Picnic, CCS16), or starting with an existing primitive such as AES and trying to find the most suitable proof system (BBQ, SAC19 or Banquet, PKC21). In this work we do both: we improve certain symmetric-key primitives to better fit existing signature schemes, and we also propose a new signature scheme that combines a new, minimalist one-way function with changes to a proof system to make their combination even more efficient. Our concrete results are as follows.

First, we show how to provably remove the need to include the key schedule of block ciphers. This simplifies schemes like Picnic and it also leads to the fastest and smallest AES-based signatures, where we achieve signature sizes of around 10.8 to 14.2 KB using AES-128, on average 10\% shorter than Banquet and 15\% faster.

Second, we investigate a variant of AES with larger S-boxes we call LSAES, for which we argue that it is likely to be at least as strong as AES, further reducing the size of AES-based signatures to 9.9 KB.

Finally, we present a new signature scheme, Rainier, combining a new one-way function called \textsc{Rain} with a Banquet-like proof system. To the best of our knowledge, it is the first MPCitH-based signature scheme which can produce signatures that are less than 5 KB in size; it also outperforms previous Picnic and Banquet instances in all performance metrics.

1 Introduction

Most currently used digital signature schemes deployed in today’s Internet infrastructure rely on the discrete logarithm assumption in elliptic curve groups, or the RSA assumption. While these hardness assumptions are believed to hold against classical attackers, the situation might change with the introduction of powerful quantum computers.
While no quantum computer threatening currently deployed cryptography exists today, the cryptographic community is already searching for public-key primitives which are secure against quantum attacks. To name a few examples, candidates for such primitives in the ongoing NIST post-quantum standardization project include schemes based on the hardness of lattice problems (e.g., [LDK+20, PFH+20]), problems from coding theory [ABC+20], the hardness of solving multivariate quadratic equation systems [DCP+20] or different assumptions related to finding isogenies between elliptic curves [JAC+20].

In addition to the above hardness assumptions, digital signatures can also be built solely using symmetric primitives, which has the advantage of relying on well-studied primitives instead of more “structured” hardness assumptions. Here, two examples are SPHINCS+ [BHK+19, HBD+20], relying only on hash functions, and Picnic [CDG+17, ZCD+20], relying on hash functions and a block cipher.

The constructions in the Picnic family follow an interesting design approach: The secret key of the signature scheme is a the input to a one-way function, with the public key being the corresponding output. A signature is a non-interactive zero-knowledge proof of knowledge (ZKPoK) of the secret input for the public output, with the message to be signed included in the challenge generation of the proof. Importantly, the ZKPoK also has to provide security against quantum attackers: a class of zero-knowledge proofs that fulfill this requirement are the ones based on the MPC-in-the-head (MPCitH) paradigm [IKOS07]. In these proofs, the prover simulates a multiparty computation protocol executing the function to be proven for a number of parties, commits to the internal state of all parties during the protocol, and is then later challenged to open a subset of the parties’ states to the verifier. If the internal state of the opened parties is consistent with a real execution, then the verifier gains some assurance that the execution was valid, with multiple parallel repetitions being executed to boost the soundness.

In Picnic, this one-way function is built using the LowMC block cipher [ARS+15] in the following construction.

**Construction 1.** A family of one-way functions \( \{f_x\} \) can be built from a block cipher \( E \) by defining \( f_x(k) := E_k(x) \), where \( E_k(x) \) denotes the encryption of the block \( x \) under the key \( k \). An instance of the one-way function is then sampled from the family by choosing a block \( x \) uniformly at random. The OWF output includes \( x \), defining the OWF relation \( ((x, y), k) \in R \Leftrightarrow E_k(x) = y \).

Clearly the security of the resulting one-way function is based directly on the security of \( E \) against key recovery attacks with a single known plaintext; we formalize this in Appendix A.4.

LowMC has the design goal of being efficiently computable in MPC protocols. This means that the internal state that needs to be committed to and revealed can be smaller and therefore, the overall size of the signature is reduced. This is accomplished by having a low multiplicative complexity, as multiplications (AND gates) are the most expensive part of MPC protocols that use linear secret sharing. This is a delicate balance, as the nonlinear operations of a block cipher are essential for security. LowMC is a relatively new design, and (like all ciphers) is not as well analyzed as AES [DR20]. While AES would be an obvious first choice as a block cipher to be used in a Picnic-style signature, its Boolean circuit representation (which is used in the ZKPoK protocols in Picnic,
ZKB++/ZKBoo \cite{CDG+17, GMO16}, and KKW \cite{KKW18}) is much larger than that of LowMC, leading to signatures of about 209 KB for ZKB++ \cite{GCZ16}, and 52 KB for KKW \cite{DOS19}.

Recently, a line of work has started to improve these AES-based signatures. In BBQ \cite{DOS19}, the authors modify the MPC protocol to execute a circuit over the AES-native field $\mathbb{F}_{2^8}$ (rather than a binary circuit) and show how to improve the computation of the inverse in the AES S-box. This reduces the size of AES-based signatures to about 30 KB, still larger than the 12 KB signatures that Picnic using LowMC can achieve. In a follow-up work called Banquet \cite{BdSGK+21}, the authors proposed a new MPCitH protocol that reduces signature sizes even further by allowing the prover to inject known values into the MPC computation, removing the need to perform the computation of the inverse in the MPC protocol. The verifier instead checks the validity of the injected values using a batched polynomial checking protocol. The final signature sizes for Banquet signatures range from 13.2 to 20 KB, which is close to the sizes of Picnic, however, Banquet instances with comparable sizes have significantly slower signing and verification times than Picnic.

When it comes to designing and selecting ciphers for MPC-in-the-head proofs, Banquet has shown that field inversion can be more efficient than expected, especially when measuring the amount of non-linearity per unit of proof size. Since field inversion is a choice non-linear operation in symmetric cipher design (e.g., it is the only non-linear operation in AES), a natural question is whether we can design a primitive to take full advantage of this efficiency. We further observe that the Banquet proof system is even more efficient when the inverse operations are in larger fields; which points to block cipher designs with large S-boxes. Inversion over large fields (we consider $\text{GF}(2^{32})$ to $\text{GF}(2^{256})$) would be a non-starter for traditional block ciphers, were performance is expected to be a handful of cycles per byte. However, in our MPCitH setting these inversions are comparatively cheap to prove.

We also note that it is possible to construct OWFs directly (see \cite{DGH+21} for a recent example), rather than starting from block cipher designs as we do in this work. However, our new design benefits from the large literature on secure block cipher design and cryptanalysis, and also gives a natural way to incorporate the field inversion operations that are friendly to MPCitH proof systems.

**Contributions.** In this work, we investigate three methods of reducing MPCitH signature sizes further, while simultaneously improving the performance of signing and verification. Our results cover a range of options from more conservative (but less performant), to more performant (but with stronger assumptions).

- We investigate the use of AES as a public permutation in a single-key Even–Mansour construction. The use of a public constant for the AES key removes the need to calculate the AES key schedule as part of the MPC protocol, reducing the number of S-boxes (and therefore inversions) from 200 to 160 for AES-128, which leads to smaller signatures using the Banquet proof system.
- In Banquet, the in- and outputs to the inverse functions are lifted from the AES field $\mathbb{F}_{2^8}$ to a larger field $\mathbb{F}_{2^{256}}$ to reduce the soundness error of the protocol. This step leads to an increase in signature size, since elements
of that larger field are included in the final signature. We investigate the security of a variant of AES that uses 32-bit S-boxes and show that we can then remove the lifting step, leading to smaller signatures.

- Finally, we present a one-way function, called RAIN (Random Affine Inverse Nyberg-inspired), that is ideally suited for the MPCitH protocol. RAIN follows a very simple design; having no key schedule and each round only consisting of a constant addition, a matrix multiplication and a field inversion. We then analyze its security as a one-way function, and finally build a signature algorithm called Rainier, using RAIN and a modified variant of the Banquet proof system. Rainier is, to the best of our knowledge, the first MPCitH-based signature scheme with signatures less than 5 KB in size and it also outperforms previous Picnic and Banquet instances in all performance metrics.

A side benefit of the EM and RAIN constructions is that signature private keys may be sampled uniformly whereas in BBQ and Banquet a negligible part of the key space must be excluded.

We have implemented all of these signature scheme variants, and provide detailed comparisons of running time, signature size and also briefly compare to other post-quantum signatures not based on the MPCitH paradigm. Our implementation is available at [https://github.com/IAIK/rainier-signatures](https://github.com/IAIK/rainier-signatures).

Cryptanalysis Scenario. We perform all dedicated cryptanalysis of our new primitives in the scenario given by Theorem 1. In detail, for our RAIN construction, we show that it provides security against key recovery attacks with a single known plaintext-ciphertext pair, as it is sufficient to be used as a one-way function in a Picnic-style signature. Security claims in this paper should be interpreted in this scenario, we do not give any guarantees for attacks with larger data complexities. This naturally prevents a large class of statistical attacks, since they usually require multiple pairs. For example, at least two pairs are necessary for a classical differential attack.

Related Work to Rain. Our proposal RAIN described in Section 4 shows several similarities to MiMC [AGR+16] and the Marvellous designs JARVIS [AD18] and (instances of) Vision [AAB+20]. These designs also aim to minimize a certain cost metric in arithmetization-oriented scenarios, and all three of them are built using a single large S-box covering the full permutation state.

The round function of RAIN is composed of two very different building blocks. First, we use the inversion in the field \( F_{2^n} \), arguably a very structured operation. The second part is then an unstructured random affine layer with operations in \( F_2 \). The latter building block together with the fact that our proof system can efficiently handle these two rather different types of operations is crucial for the concrete efficiency. For MiMC and its primary use cases, operations are preferred to all be in the same field and hence such an affine layer was not used at all, leading to a requirement of more than 80 rounds for security. For JARVIS/Vision this is reduced to 36 and without safety margin could be halved to 18 rounds. The main reason for this reduction is that a structured affine layer is introduced in every round and shown to allow for efficient implementations. With the unstructured affine layer in RAIN, our analysis suggests that 3 rounds
are already sufficient in the signature use-case, leading to substantial efficiency improvements despite the lack of structure. This is also supported by Nyberg’s analysis \[\text{[Nyb94]}\] showing that finite field inversion has a large distance from affine functions.

**Related Work to Large S-Boxes.** Using S-boxes or non-linear permutations working on more than eight bits is not new. In the early 1990s, we see for example the Knudsen–Nyberg cipher \[\text{[NK93]}\] using cubing in \(\mathbb{GF}(2^{37})\), or Subterranean \[\text{[CDGP93]}\] using the \(\chi\)-layer \[\text{[Dae95]}\] on 257 bits. Also in recent cipher designs, we see the use of fairly large S-boxes. For example, Rasta \[\text{[DEG+18]}\] and Subterranean 2.0 \[\text{[DMMR20]}\] use large versions of the \(\chi\)-layer, while Alzette \[\text{[BBDiS+20]}\] is an S-box design for 64-bit inputs/outputs based on modular additions, rotations and XORs. Other examples include MiMC \[\text{[AGR+16]}\], GMiMC \[\text{[AGP+19]}\], HadesMiMC \[\text{[GLR+20]}\], Poseidon \[\text{[GKR+21]}\], and Marvellous \[\text{[AAB+20]}\] instances with large S-boxes having a very simple algebraic description.

**Related Work to MPCitH Proof Systems for AES** Limbo \[\text{[dSGOT21]}\] is a proof system suitable for generic circuits, and can also be used to build signature schemes with runtimes and sizes very close to Banquet. We compare our variants to their proposed Limbo-Sign AES-128 signature in Table 4. A natural implication of our work on single-key Even-Mansour variants of AES is to investigate the performance of EM-AES signatures in Limbo. Based on the formulas given in \[\text{[dSGOT21]}\], and our comparison for AES-128, we expect similar performance to our Banquet variants, with Limbo having slightly larger signatures and slightly better runtimes.

## 2 Using Single-Key Even–Mansour

In this section we discuss using the single-key Even–Mansour (EM) construction as a one-way function. We start with background and security analysis, and give the construction we use for MPCitH-based signatures below in Section 2.1.

The single-key Even–Mansour scheme \[\text{[DKS12]}\], \[\text{[EM97]}\], \[\text{[Riv84]}\] is a way to construct a block cipher \(F\) from a cryptographic permutation \(\pi\) by adding a key \(k\) to the input \(x\) and to the output of the permutation, i.e.,

\[
F_k(x) = k + \pi(x + k).
\]  

In practice we can instantiate \(\pi\) with a block cipher such as AES, by fixing a random key \(p_0\), and making it a public constant. We thus avoid calculating the non-linear key schedule with a secret key. We write \(F : \{0,1\}^\kappa \times \{0,1\}^n \rightarrow \{0,1\}^n\), where \(n\) is the block size, and \(\kappa\) is the key size and \(\pi : \{0,1\}^n \rightarrow \{0,1\}^n\). We assume that \(\kappa = n\) and addition of \(n\)-bit strings is done with bitwise XOR.

**Security.** In the context of a signature scheme constructed from an MPC-in-the-head proof system, \(F\) must be a one-way function of the key: given some \((x,y)\) such that \(y = F_k(x)\), it should be difficult to find a \(k’\) such that \(F_{k’}(x) = y\). More formally, \(F\) is a secure OWF if the following probability

\[
\Pr[x \xleftarrow{\$} \{0,1\}^n, k \xleftarrow{\$} \{0,1\}^\kappa, k’ \xleftarrow{\$ \in A^\pi(x,F_k(x))} : F_{k'}(x) = F_k(x)]
\]  

5
is negligible in \( n \) for all polynomial-time adversaries \( A \), with oracle access to \( \pi \).

It is easy to see that removing either addition of \( k \) in the construction of \( F \) is insecure. The single-key EM scheme is a special case of the two-key scheme, that constructs a block cipher \( T \) from \( \pi \) using two keys as \( T(k_1, k_2, x) = k_2 + \pi(x + k_1) \). The function \( T \) is trivially not a secure one-way function (OWF) of the correct type: given \( (x, y) \) we can compute a \( (k_1, k_2) \) by first choosing \( k_1 \) at random and then setting \( k_2 = y + \pi(x + k_1) \). It is therefore important to formally show security of the single-key EM construction for signatures.

We model \( \pi \) as an ideal permutation, and consider attacks where the attacker is given oracle access to it. We would like to prove a lower bound on the number of queries to \( \pi \) for a successful attack, or equivalently, show that any attack making \( q \) queries succeeds with probability negligible in \( \kappa \). In [EM97], Even and Mansour prove a lower bound for the two-key construction, namely that an attack has \( DT = \Omega(2^n) \) where \( D \) is the data complexity (i.e., the number of encryption queries or decryption queries to \( F \)) and \( T \) is the time complexity (measured as the number of queries to \( \pi \)). Dunkelman, Keller and Shamir [DKS12] show that with small modifications, the proof also holds for the single-key EM construction. However, the analysis of [EM97] [DKS12] uses a reasonable but nonstandard CCA-like security definition. As shown above with the two-key construction, this definition does not imply security of EM in our setting. Dunkelman et al. comment that obtaining a nontrivial amount of information about the key is covered by the analysis, and our two-key attack above does not contradict this since the key produced by the attack is different from the original key used to create \( y \) with overwhelming probability. The main difference is that our OWF game may be won with any key for \( F \) while in the block cipher setting the attack must find the single correct key, in order to recover information about the plaintext. In addition, in our use-case as a one-way function the data complexity is limited to a single plaintext-ciphertext pair (i.e., \( D = 1 \)), meaning we can achieve \( n \)-bit security with a \( n \)-bit permutation, since the attacker must invest \( T = 2^n \) time. This is in contrast to traditional attack scenarios where attackers can trade off data and time complexity, i.e., by setting \( D = T = 2^{n/2} \).

**Theorem 2.** The single-key Even–Mansour construction (Eq. 1) is a secure one-way function, when the permutation \( \pi \) is an ideal random permutation.

**Proof.** The attacker \( A \) is initialized with \((x, y)\) and has oracle access to \( \pi \). We must show that the probability in Eq. 2 is negligible in \( n \) (the key size and block size in bits).

We say that a key \( K \) is consistent with the pair \((x, y)\) if \( y = F_K(x) \), i.e.,

\[
y = K + \pi(x + K).
\]

Just before producing an output, \( A \) has made \( q \) queries to \( \pi \), and has \( q \) pairs \((X_i, Y_i)\) where \( Y_i = \pi(X_i) \). W.l.o.g., we assume that inputs \( X_i \) to \( \pi \) are distinct.

Therefore, each query \( X_i \) has the form \( X_i = x + K_i \) for a distinct key \( K_i \). In one case, the query either gives \( A \) the correct key, and we have \( Y_i + K_i = y \). In the other case, the query does not give \( A \) the correct key, but reveals that \( K_i' = y - Y_i \), a second key distinct from \( K_i \), cannot be consistent with \((x, y)\).

First we note that \( K_i' \) is distinct from \( K_i \) since \( Y_i + K_i \neq y \), then \( K_i \neq y - Y_i \). To see that \( K_i' \) cannot be consistent if \( K_i \) is not consistent, assume that \( K_i' \) was
consistent, which implies that

\[ Y_i + K'_i = \pi(x + K'_i) + K'_i \]
\[ \pi(x + K_i) + K'_i = \pi(x + K'_i) + K'_i \]
\[ \pi(x + K_i) = \pi(x + K'_i) \]

which is a contradiction since \( \pi \) is a permutation and \( K_i \neq K'_i \).

Now we argue that the remaining \( 2^n - 2q \) keys are equally likely to be consistent with \((x, y)\), given the information \( A \) has. Consider \( K^* \), one of the remaining keys, not equal to \( K_i \) or \( K'_i \). If \( A \) knew that \( K^* \) was not consistent with \((x, y)\), then \( A \) knows \( y - K^* \), and that

\[ y - K^* \neq \pi(x + K^*) \quad (3) \]

But since \( \pi \) is a random permutation, \( \pi(x + K^*) \) is uniformly selected from the \( 2^n - 2q \) remaining possible values, so Eq. (3) holds with probability \( 1 - 1/(2^n - 2q) \) and all remaining keys are equally likely.

Therefore, after \( q \) queries \( A \) succeeds with probability not more than \( 2q/2^n \). Alternatively, \( A \) succeeds after \( \Omega(2^n) \) queries.

When compared to the upper bound given by the complexity of a brute-force attack (where \( A \) succeeds with probability \( q/2^n \)), this lower bound is off by a factor of two. As described in the proof, each query to \( \pi \) can be used to rule out two keys, improving the brute force attack, to succeed with probability \( 2q/2^n \). Therefore the lower bound is tight, since it matches the complexity of the attack.

The assumption required for security of the EM-OWF is that \( \pi \) is an ideal random permutation. Assume that \( \pi \) is constructed by fixing the key of a block cipher \( E \) (since this is how we will construct \( \pi \) for signature schemes). How this assumption compares to directly assuming \( E \) is a OWF is not always clear. In general, the ideal permutation assumption is stronger than directly assuming that \( E \) is an OWF of the key (as is currently done in signature schemes such as Picnic and Banquet) since a function can still be one-way even with slightly biased outputs, and we can construct contrived examples of OWFs with biased output. But is the difference meaningful for any natural choices of \( E \)?

When \( E \) is 5-round AES, Grassi, Rechberger and Rønjom describe a distinguisher \[ \text{GRR17} \] that requires \( 2^{32} \) (plaintext, ciphertext) pairs and works for any key (for adaptively chosen ciphertexts improved to \( 2^{27.2} \) in \[ \text{BR19a} \] and to 6 rounds in \[ \text{BR19a} \]). In short, if the 2\(^{32} \) plaintexts are chosen carefully, then \[ \text{GRR17} \] shows that the set of ciphertexts possess a property (with probability 1) that would not be present for an ideal permutation (except with low probability). Therefore, Theorem \[ 2 \] says nothing about the security of the EM-OWF construction when \( \pi \) is built with 5-round AES.

By contrast, 5-round AES appears to be a secure OWF, as there are no known key-recovery attacks that work with a single (plaintext, ciphertext) pair. The best known attack in this class is given by Bouillaguet, Derbez and Fouque \[ \text{BDF11} \], applied to 4-round AES and is marginal, costing \( 2^{120} \) time and \( 2^{80} \) memory.

In one sense there is more positive evidence that 5-round AES is a secure OWF than there is for EM-OWF, however, neither problem is very well studied, so making conclusions with confidence is difficult. The issue of how the security
of the EM and non-EM one-way functions compare for practical choices of $E$ (such as 5-round AES) is an interesting open question.

**Multi-Target Security.** We take a closer look on the multi-target, or multi-user OWF security of the single-key EM construction in Appendix A.2.

**Post-Quantum Security of Single-Key EM** As with any one-way function, inputs can be recovered in time $O(2^{n/2})$ using quantum amplitude amplification [BHMT02] (a generalization of Grover’s algorithm [Gro96]), with costs similar to those for AES as described in [JNRV20]. Some recent work on the post-quantum security of EM [JST21, ABKM21] (that covers OWF security as a special case), shows that this is the best possible. The model used to analyze EM in [JST21, ABKM21] matches ours closely; the adversary may make classical queries to $F_k$, and quantum queries to $\pi$. In the signature/OWF scenario the adversary is given the output of a random classical query to $F_k$ (in the form of the public key), and $\pi$ is public so the adversary may implement it with classical or quantum hardware.

### 2.1 EM-OWF Constructions

Given a block cipher $E_k(x)$ with $n$-bit key size and block size, where $k$ is the secret key and $x$ is the input, we build the one-way function in $k$ as

$$F(x, k) = k + E_x(k).$$

As discussed above, this is an instance of the single-key EM construction when $\pi$ is random per user, and the plaintext input (denoted $x$ above) is fixed to zero. For example when $E$ is AES-128 we have a suitable OWF for use in Banquet key generation at security level L1, here $n = 128$ bits. A requirement for Banquet and BBQ is that the key be chosen so that there are no zero inputs to S-boxes, and key generation uses rejection sampling to find one (each sample is rejected w.p. $\approx 1/2$). Note that with the EM construction, we can instead sample $k$ uniformly at random, then vary $x$ until $E_x(k)$ has no zero S-box inputs.

There is no direct way to use the EM-OWF construction at security levels L3 and L5, because the block size of AES is limited to 128 bits. These higher security levels are best achieved with Rijndael [DR98], where $n$ can be 192 and 256 bits. Rijndael is not a standardized primitive (as AES-192 and AES-256 are), however it is a mature and well analyzed design.

For key generation in Picnic, since $n$ scales to 192 and 256 in LowMC, all three security levels may use the construction of Eq. (4) directly. As the key schedule in LowMC consists of only linear operations, using the EM construction will not reduce signature sizes, but can improve performance and simplify implementation, since deriving round keys is done with a matrix multiplication.

### 3 LSAES: AES with Larger S-boxes

In [dDOS19] and [BdSGK+21] it has been shown that MPC protocols using the fact that the S-box of AES-128 is based on field inversion can reduce the size of AES-based signatures. In this section, we present a mild tweak to AES-128.
We leave the description of AES-128 unchanged, except for the SubBytes part, where we replace the parallel byte-wise inversion in $F_{2^8}$ of 4 elements in a row with a single inversion in $F_{2^{32}}$. Essentially, this transforms AES-128 back to its predecessor having a 10-round SHARK-like cipher with 32-bit S-boxes, which we call LargeSboxAES-128 or LSAES-128 in short.

It is possible to consider LSAES with 16-, 32-, 64- or 128-bit S-boxes (with minor changes to the key schedule). For use in Banquet at security level L1, 32 bits are a natural choice, because the MPC protocol already lifts the 8-bit S-box (input, output) pairs to a 32-bit field to increase the soundness of the polynomial checking protocol. We will therefore focus on 32-bit S-boxes. We briefly discuss the signature sizes achievable by using LSAES with other S-box sizes in Appendix D.2.

### 3.1 Specification

Since LSAES-128 corresponds largely to AES-128, we only recall the structure of the cipher and the application of the S-boxes in this section. The details regarding the affine parts and the key schedule can be found in Appendix B. As AES-128, LSAES-128 mainly consists of two parts, the key schedule, where the round-keys $k^{(i)}$ are derived from the secret key $k$, and the data path, where an input $x$ is transformed by the round function $R_i$ and mixed with the round-keys $k^{(i)}$. There are ten rounds and eleven round keys. Thus, we have

$$F_k(x) = R_{10} \circ \cdots \circ R_2 \circ R_1(x),$$

where each round function $R_i \forall i < 10$ is defined as

$$R_i(x) = L \left( S \left( x + k^{(i)} \right) \right),$$

and the last round function $R_{10}$ is defined as

$$R_{10}(x) = S \left( x + k^{(10)} \right) + k^{(11)}.$$

#### 3.1.1 Structure of the State

For describing the cipher, we use the typical description of AES-128 as a rectangular $4 \times 4$ representation of the bytes $x_i$, shown in Fig. 13. A 128-bit input $x$ to the cipher is seen as a concatenation of byte elements $x = x_0 \parallel x_1 \parallel \cdots \parallel x_{15}$. In addition, we define the 32-bit elements $\chi_i = x_i \parallel x_{i+4} \parallel x_{i+8} \parallel x_{i+12} \ \forall i \in \mathbb{N} : 0 \leq i < 4$ leading to the representation shown in Fig. 14.

#### 3.1.2 Substitution Layer $S$

The substitution layer interprets the four 32-bit state values $\chi_i$ as elements of the field $GF(2^{32})$ defined by the irreducible polynomial $X^{32} + X^7 + X^3 + X^2 + 1$ (a different choice of irreducible polynomial is possible in case of existing software or hardware implementations). Then, the inversion operation is applied to each of the four $\chi_i$ with the additional convention that element 0 maps to 0. So we have

$$S(x) = S(\chi_0, \chi_1, \chi_2, \chi_3) = (\chi_0^{-1}, \chi_1^{-1}, \chi_2^{-1}, \chi_3^{-1}).$$
3.2 Cryptanalysis

One major feature of AES-128 is its resistance against differential [BS91] and linear [Mat94] cryptanalysis. Hence, those are not the dominant attack vectors covering the highest number of rounds. After years of cryptanalysis, attack vectors that can turn the strong alignment of AES’s round function to their advantage, like impossible differential attacks [MDRMH10] and meet-in-the-middle [DFJ13] attacks, have turned out to cover the highest number of rounds. This section shows that LSAES retains excellent resistance against differential and linear cryptanalysis while also reaching at least the same security level against attacks exploiting strong alignment. Note that we consider the strongest attacks on AES without restricting the data complexity.

3.2.1 Differential and Linear Cryptanalysis

The use of 32-bit S-boxes in the columns of the AES state while leaving all linear transformation intact essentially leads to LSAES being a SHARK-like [RDP+96] design. The reasons for this are as follows (referring to the notation in Appendix B).

- The linear transformation \( AB_i \) is performed four times in parallel on 8-bit chunks of our 32-bit state variable values \( \chi_i \), while \( SR \) is just a re-ordering of bytes within each state value \( \chi_i \). Hence, we can see those two linear layers as part of 32-bit S-boxes, which are now just affine equivalents to the inversion in \( \text{GF}(2^{32}) \).

- Applying four \( 4 \times 4 \) MDS matrices with coefficients in \( \mathbb{F}_2^{8 \times 8} \) in parallel on 8 bit parts of our state words \( \chi_i \) corresponds to the application of a single \( 4 \times 4 \) MDS matrix with coefficients in \( \mathbb{F}_2^{32 \times 32} \) [KLSW17].

Hence, we now know that a linear or differential trail has at least five active S-boxes per 2 rounds. So we have at least 25 active S-boxes in total. Our S-boxes have a maximum differential probability of \( \delta = 2^{-30} \) and a maximum correlation of \( \lambda = 2^{-15} \) [Nyb94]. Hence, the theoretical values for 10 round differential trails of \( \delta = 2^{-750} \) and for 10 round linear trails of \( \lambda = 2^{-375} \) let us conjecture that those attack vectors do not pose a threat to LSAES.

3.2.2 Attacks Exploiting Strong Alignment

AES has a strongly aligned structure that leads to nice bounds on the differential probability and correlation of trails, used to give convincing arguments that
AES resists differential and linear cryptanalysis. If we take a closer look at one round of AES, we see that the only function directly providing diffusion between byte-borders is MixColumns. In particular, MixColumns is a linear function mixing the 4 bytes within each of the four columns of the AES state separately.

The attacks that reach the largest number of rounds for AES-128 exploit these characteristics of AES. Attacks that exploit the strong alignment of AES-128 are (amongst others) the Integral/Square attacks [DKR97, FKL’01], impossible differential attacks [MDRMH10], or meet-in-the-middle attacks [DS08], where the last one gives the most efficient attacks on seven rounds of AES-128 [DFJ13].

For LSAES-128, the diffusion of the linear layer stays the same. The only thing we change is the S-box. Since the S-boxes of LSAES-128 are applied on all 4 bytes belonging to one row, we now have one additional component directly providing diffusion between byte-borders. In contrast to MixColumns, those S-boxes used in LSAES-128 are non-linear functions. Hence, we expect that all attacks exploiting the strong alignment and the direct linear mixture of the bytes in AES-128 do not perform better for LSAES-128. On the contrary, we expect that most of them will actually perform worse. For instance, we have implemented LSAES-128 in a tool capable of finding meet-in-the-middle and impossible differential attacks [DF16]. In contrast to AES-128, this tool does not give any indications for the existence of meet-in-the-middle and impossible differential attacks for seven rounds of LSAES-128.

3.2.3 Algebraic Attacks
Attempts to use the relatively simple algebraic structure for attacks on AES [CP02, MR02] turned out to be not fruitful [CL05, CMR06]. We conjecture that this also applies to LSAES.

4 Rain
Compared to LSAES, we go further with RAIN by increasing the S-box size and also modifying the linear layer. We summarize our design rationale in the following. We use inversion in $F_{2^n}$, where $n$ is the state size, since large inversions are efficient to prove in Banquet-like protocols [BrSGK’21]. Compared to many other settings, the actual structure of the linear layer does not impact the performance much for our MPCitH use-cases. Hence, we use multiplication by a randomly chosen matrix $M \in F_{2^n}^{n \times n}$ (we ensure they also have a dense linearized polynomial that is of maximum degree when expressed over the same field as the inversion) as our linear layer in order to improve the diffusion compared to more structured linear layers. Recent designs also using large finite field inversions (e.g., Vision [AAB’20] and Jarvis [AD18]) use an affine layer that can be efficiently described in the same field. This is the main difference between these designs and RAIN.

Another way of seeing RAIN is as an iteration of a larger variant of the AES S-box [DR20] using a more unstructured affine mapping. Since, to the best of our knowledge, Nyberg [Nyb94] proposed the use of an inverse augmented affine permutation, we call our proposal RAIN (Random Affine Inverse Nyberg-inspired).

Since we use RAIN with only a small amount of rounds, we have decided to
use different matrices and round constants per round to improve the diffusion as much as possible and prohibit symmetry properties. This choice has a negligible effect on the proof system performance compared to always using the same matrix in each round and no round constants. We are also using a trivial key schedule \((k_i = k)\) to further simplify the design.

We emphasize again that \textsc{Rain} is not a classical block cipher, and is only intended to be used in Theorem 1 for a MPCitH proof system. This implies that the data complexity available to an attacker is restricted to a single known (input, output) pair.

### 4.1 Specification

We define a keyed permutation \(F_k(x)\) consisting of a nonlinear operation \(S\) and a constant addition over a large field \(\mathbb{F}_{2^n}\), together with a linear layer. We choose \(S: \mathbb{F}_{2^n} \to \mathbb{F}_{2^n}\) such that

\[
S(x) = x^{2^n-2} = \begin{cases} x^{-1} & \text{if } x \neq 0, \\ 0 & \text{otherwise.} \end{cases}
\]

The entire permutation is described as a concatenation of round functions \(R_i\), where each round function \(R_i \forall i < r\) is defined as

\[
R_i(x) = M_i \left( S \left( x + k + c(i) \right) \right),
\]

and the last round function \(R_r\) is defined as

\[
R_r(x) = k + S \left( x + k + c(r) \right).
\]

We denote the round constant in round \(i\) by \(c(i)\), and \(M_i \in (\mathbb{F}_2)^{n \times n}\) is the linear layer matrix over \(\mathbb{F}_2\) used in round \(i\). Every such matrix \(M_i\) can also be represented as a linearized polynomial \(L_i \in \mathbb{F}_2[x]\), namely

\[
L_i(X) = \sum_{j=0}^{n-1} a_j^{(i)} X^{2^j}
\]

for some coefficients \((a_j)^{n-1}_{j=0}\). In our design, we will make sure that \(a_j \neq 0\) for \(j \in \{0, \ldots, n-1\}\). This implies that the linearized polynomial is of maximum degree and as dense as possible.

A graphical overview of the construction is shown in Fig. 2. The details for the generation of the round constants and matrices are given in Appendix C.

![Figure 2: The \textsc{Rain} permutation with \(r = 3\) rounds. \(M_i\) denotes the multiplication with an unstructured invertible matrix over \(\mathbb{F}_2\) in the \(i\)-th round.](image)
4.2 Concrete Instances

The concrete instances we focus on are defined in the following.

- **128-Bit Security.** We use $\mathbb{F}_{2^{128}}$ and the reduction polynomial $X^{128} + X^7 + X^2 + X + 1$. This polynomial is also used in AES-GCM [MV04].

- **192-Bit Security.** We use $\mathbb{F}_{2^{192}}$ and the reduction polynomial $X^{192} + X^7 + X^2 + X + 1$.

- **256-Bit Security.** We use $\mathbb{F}_{2^{256}}$ and the reduction polynomial $X^{256} + X^{10} + X^5 + X^2 + 1$.

4.3 Cryptanalysis

In this section, we will show that 3 rounds of the construction are sufficient in order to provide security. Indeed, we do not consider fewer than 3 rounds, mainly to avoid the possibility of splitting the construction into two single-round parts.

For the sake of simplicity, we omit using the round-specific indices of the linearized permutation polynomials $L_i$, i.e., we write $L$ instead of $L_i$. Again, we highlight that due to the intended use of RAIN in Theorem 1, we limit the attacker to be able to use only a single (input, output) pair ($v_{\text{in}}, v_{\text{out}}$), and the goal is to recover a full key $k$ which maps $v_{\text{in}}$ to $v_{\text{out}}$.

4.3.1 Gröbner Basis Attacks

We try to represent the keyed permutation as a system of equations, which we then want to solve by converting it into a Gröbner basis and solving the univariate equations (after order transformation) for the remaining variables.

**Equation System over $\mathbb{F}_2$.** A straightforward way to build such a system is by working with variables in $\mathbb{F}_2$. We would then have $n$ variables and $n$ equations for a key size of $n$ bits. Each equation describes a single output bit in the (known) input bits and the key variables. Working purely with Gröbner bases, we assume this system of equations to be hard to solve, as each round has an algebraic (bit-level) degree of $n - 1$, which is the maximum for a permutation.\footnote{Indeed, current results in the literature suggest that even low-degree equation systems over $\mathbb{F}_2$ are hard to solve (see e.g. [JVT17, NNY18]).} Note that this also holds for the decryption direction, since we use the inverse function. Besides the approach discussed here, this property also makes recent low-data attacks on schemes with low algebraic degrees (e.g., [Din21]) infeasible. Hence, we conjecture that 3 rounds are sufficient in order to provide security w.r.t. Gröbner basis attacks using equation systems over $\mathbb{F}_2$.

**Equation System over $\mathbb{F}_{2^n}$ (Single Variable).** A more efficient approach is to instead focus on equation systems over $\mathbb{F}_{2^n}$. As we only have a single input, we can represent the whole equation by a single equation in a single key variable (assuming a linear key schedule). Note that the resulting system of equations with a single equation is automatically a Gröbner basis, hence this step is free. However, we still need to account for the complexity of solving this equation for the single key variable. Indeed, with overwhelming probability, the resulting
polynomial has a maximum degree of $2^n - 2 = d$ over $\mathbb{F}_{2^n}$ and is also dense. The cost of finding the root of such a polynomial is an element in $\mathcal{O}(d^3 n^2 + dn^3)$ \cite{Gen07}. Since the maximum degree is already reached after a single round, we conjecture that 3 rounds provide ample security w.r.t. to this approach.

**Equation System over $\mathbb{F}_{2^n}$ (Intermediate Variables).** Another strategy is to add intermediate variables in order to decrease the degree of the inverse operation. This can be done by using the fact that

$$x^{-1} = y \implies xy = 1$$

for every nonzero $x \in \mathbb{F}_{2^n}$. Hence, instead of using a single equation for the whole permutation, we can introduce an intermediate variable in each round and use it to translate every nonlinear operation into a degree-2 relation. This fact holds with a probability of $(1 - 2^{-n})^r$ when using $r$ rounds. Then, the equation system for $r$ rounds consists of $r$ variables and $r$ equations in $\mathbb{F}_{2^n}[x_1, x_2, \ldots, x_{r-1}, k]$, namely

$$
\begin{align*}
(v_{\text{in}} + k + c(1)) \cdot L^{-1}(x_1) - 1 &= 0, \\
(x_{j-1} + k + c(j)) \cdot L^{-1}(x_j) - 1 &= 0 \quad \forall j \in \{2, \ldots, r-1\}, \\
(x_{r-1} + k + c(r)) \cdot (v_{\text{out}} - k) - 1 &= 0.
\end{align*}
$$

(5)

Note that the last equation has a degree of 2, while the other $r-1$ equations have a high degree, since we assume $L^{-1}$ to be a linearized polynomial of high degree.\footnote{Indeed, we generate $L^{-1}$ such that it is guaranteed to have a high degree.} Concretely, using this system of equations and generic estimates, we would assume the degree of regularity of this system to be

$$D_{\text{reg}} = 1 + \sum_{i=1}^{r} \deg(f_i) - 1 = 3 + \sum_{i=1}^{r-1} 2^{n-1} - 1 = 3 + (r-1)(2^{n-1} - 1) \geq 2^{n-1}$$

(6)

for $r \geq 2$ and practical $n$. We immediately see that

$$
\left(\frac{r + 1 + 2^{n-1}}{2^{n-1}}\right)^\omega > \left(\frac{1 + 2^{n-1}}{2^{n-1}}\right)^\omega = (1 + 2^{n-1})^\omega > 2^n
$$

and hence

$$\log_2 \left(\frac{r + 1 + 2^{n-1}}{2^{n-1}}\right)^\omega > n$$

for $\omega \geq 2$ and $r \geq 1$. However, note that this analysis assumes that the equations result from a semi-regular system of equations, which is clearly not the case due to the linearized polynomial $L$. Therefore, we evaluated the actual degree reached in practical experiments.

**Degrees Reached During the Computation.** An overview of some of our results is given in Fig. 3. In this illustration, $D_{\text{reg}}$ denotes the estimated degree of regularity assuming a semi-regular system of equations. On the other hand, $D$ denotes the highest degree reached during a Gröbner basis computation.

In our tests, we observed that the minimum degree resulting from one possible reduction between two high-degree equations (i.e., a degree of $2 \cdot 2^{n-1} = 2^n$)
Table 1: Gröbner basis attack strategies on RAIN.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>#vars = #eqs</th>
<th>Degrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>System over $F_2^n$</td>
<td>$n$</td>
<td>$n - 1$</td>
</tr>
<tr>
<td>System over $F_2^n (1)$</td>
<td>$r$</td>
<td>$2^{n-1} + 1, 2$</td>
</tr>
<tr>
<td>System over $F_2^n (2)$</td>
<td>$r - 1$</td>
<td>$2^{n-1} + 1, 2^{n-1} + 2$</td>
</tr>
</tbody>
</table>

Figure 3: Comparison of the estimated degree of regularity $D_{\text{reg}}$ (computed using Eq. (6)) and the highest degree $D$ encountered in practical Gröbner basis computations for $r \in \{2, 3\}$ rounds. We note that some of the lines are overlapping, which indicates that the estimated degree matches the practical one.

is always reached for $r \geq 3$. Indeed, the actual degree reached in experiments matched the estimated degree for $r = 3$ and came very close for $r = 2$. In many cases, $D$ was also equal to the first-fall degree.

Further, the complexity of the Gröbner basis conversion using e.g. the FGLM algorithm [FGLM93] depends on the degree of the final recovered univariate polynomial in the lexicographically ordered basis. In our tests, this degree reached its maximum for $r \geq 3$, and the resulting polynomial was also dense. Further, we could only observe a small number of solutions for this polynomial, essentially matching the expected number of keys mapping a given plaintext to a given ciphertext. We note that any spurious solutions arising through the factorization or root-finding process are not valid solutions to the original system, and thus not of interest to an attacker. Following these results, we conjecture that a similar behavior can also be observed for larger block sizes and that the construction provides the advertised level of security if $r \geq 3$.

**Toy Versions and Comparison with Exhaustive Search.** The Gröbner basis computations for block sizes of $n > 10$ bits start to get increasingly expensive, mainly due to the high degrees in the computation. For versions with reduced state sizes, we can confirm that the attacks perform significantly worse than a simple exhaustive search on a small number of bits. We expect that the same is also true for practical state sizes.

Since we have $r - 1$ linear layers, we only have a single linear layer for $r = 2$ rounds, and hence only one high-degree equation in the resulting equation system.
A Different Representation. The two representations given above (i.e., one variable for the entire permutation or one intermediate variable for each round) are not the only possible descriptions of the function. Indeed, given Eq. (5), for example it is possible to skip some variables and equations, since

\[ L^{-1}(x_i) = \frac{1}{x_i - 1 + k + c(i)} \implies x_i = L \left( \frac{1}{x_i - 1 + k + c(i)} \right), \]

and

\[ x_i + k + c(i+1) = \frac{1}{v_{out} - k} \implies x_i = \frac{1}{v_{out} - k} - k - c(i+1), \]

and hence

\[ L \left( \frac{1}{x_i - 1 + k + c(i)} \right) = \frac{1}{v_{out} - k} - k - c(i+1), \]

which implies

\[ \left( a_{n-1} + \sum_{j=0}^{n-2} a_j \left( x_{i-1} + k + c(i) \right)^{2^{n-1-2j}} (v_{out} - k) \right) - \left( x_{i-1} + k + c(i) \right)^{2^{n-1}} \left( (k + c(i+1)) (v_{out} - k) - 1 \right) = 0 \]

for the last round, where \((a_j)_{j=0}^{n-1}\) are the coefficients of \(L\) (we omit the distinction between the different linear layers for simplicity). A similar technique can also be used to skip every second variable in a construction using more than 3 rounds. However, even though this approach may speed up the Gröbner basis computations for a low number of equations and variables (although the degree of regularity was the same in practical experiments), this does not necessarily mean that the final solving step will also be faster. In particular, in our practical tests we observed that the recovered univariate polynomials were mostly dense and of maximum degree, especially with larger \(n\). Exploiting this approach further, we would eventually end up at the single-variable equation system given at the beginning of this section, where the prohibitively expensive step is not the Gröbner basis computation, but the factorization of the final polynomial.

Concrete Runtime Results. In Fig. [4] we compare the runtime of the entire solving step for a key-recovery attack and different state sizes \(n\). Our experiments suggest that the expected runtime more than doubles when increasing \(n\) by one for \(n \geq 6\). We therefore reasonably assume that the same behavior continues for larger block sizes, and in particular for \(n \geq 128\). All tests were done with Sage.

Comparison with Similar Constructions. Note that our design is similar to Jarvis [AD18], since it uses the same S-box and a similar round function. Moreover, there have been attacks on Jarvis [ACG+19] which use a similar approach as the one taken here. However, these attacks mainly exploit the low degree of the linear layer (which is required by the design strategy). This is not possible for Rain, since we ensure that each linear layer has maximum degree. Indeed, in [ACG+19 Section 7], the authors compare the affine part of the Jarvis S-box with the affine part of the AES S-box. The latter also has a maximum degree and cannot be split into two low-degree components, which is possible in Jarvis.
Moreover, our construction shows similarities with MiMC [AGR+16], whose S-box also covers the full state and can also be described by low-degree polynomials. Although MiMC has recently been attacked [EGL+20], the proposed method is not applicable in our setting due to the limited amount of data available.

### 4.3.2 Other Attacks

Since we only consider an attacker who has access to a single (input, output) pair, we mainly analyzed attack vectors which are based on solving equation systems over some finite field. However, for completeness, we briefly mention other attack vectors here.

**Differential and Linear Attacks.** Note that these statistical attacks in their basic form need multiple (plaintext, ciphertext) pairs and are thus not directly applicable in our scenario. Still, even when considering multiple pairs, we would be counting the number of active S-boxes in each round. Since our design consists of a single large S-box, for a nonzero input difference in each round clearly one S-box is active. Further, the inversion mapping over \( \mathbb{F}_{2^n} \) provides sufficiently strong linear and differential properties [Nyb94] such that the maximal differential probability is less than \( 2^{-2n} \) and the maximum linear correlation is less than \( 2^{-n} \) after 3 rounds.

**Interpolation.** We do not consider interpolation attacks in our setting, since multiple data pairs are needed in order to interpolate. However, we still assume the polynomial representation of the function to be dense after 3 rounds, mainly due to the uses of the inversion mapping.

**Linearization.** In a linearization attack, we essentially replace every monomial of degree greater than 1 in the equation system by a new variable, until eventually arriving at a linear equation systems. After that, a simple equation solving algorithm can be used to attack the primitive.

In our scenario, this attack is not applicable since increasing the number of variables (and equations) with the linearization requires an increased number of data, which is not accessible to the attacker. Secondly, the number of monomials is large thanks to the density of the inverse function. Finally, note that introducing intermediate variables in a linearization step does not seem to be feasible, because their values would change with each requested data pair.
Table 2: Attack strategies and number of rounds to prevent each of them. Attacks not applicable are denoted by “N/A”.

<table>
<thead>
<tr>
<th>Type</th>
<th>Nr. of rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Differential cryptanalysis</td>
<td>N/A</td>
</tr>
<tr>
<td>Linear cryptanalysis</td>
<td>N/A</td>
</tr>
<tr>
<td>Integral/saturation attacks</td>
<td>N/A</td>
</tr>
<tr>
<td>Higher-order differentials, cube attacks</td>
<td>N/A</td>
</tr>
<tr>
<td>Gröbner basis (full-round)</td>
<td>2</td>
</tr>
<tr>
<td>Gröbner basis (intermediate variables)</td>
<td>3</td>
</tr>
<tr>
<td>Interpolation</td>
<td>N/A</td>
</tr>
<tr>
<td>Linearization</td>
<td>N/A</td>
</tr>
<tr>
<td>Guess and determine</td>
<td>2</td>
</tr>
</tbody>
</table>

Higher-Order Differential Attacks. In higher-order differential attacks [Lai94], a low degree of the construction over $\mathbb{F}_2^n$ (i.e., the algebraic degree) is exploited. However, this attack vector also needs more than a single pair, and is hence not directly applicable in our setting. Nevertheless, variations of the attack might work for very low degrees [DMRS20]. Therefore, we mention that the algebraic degree of the inversion mapping in $\mathbb{F}_2^n$ is $n - 1$ (i.e., the maximum for a permutation), and hence a higher-order differential distinguisher is infeasible since it would use the full space. The same is true also for the inverse direction.

Guess-and-Determine Attacks. Guess-and-determine attacks can often be used together with other algebraic techniques in order to reduce the complexity of the main attack. For example, a couple of variables may be guessed in order to simplify the resulting equation systems. For RAIN, we can essentially guess over $\mathbb{F}_2^n$ or directly over $\mathbb{F}_2$. The former approach matches the complexity of exhaustive search after guessing a single element. In the second approach, we are allowed to guess $b < n$ bits. Note, however, that the mapping $x \mapsto x^{-1}$ provides full diffusion on the bit level. While it may be possible to linearize specific component functions after one nonlinear operation by guessing $b < n$ bit variables, we expect this property to vanish after at most 2 full rounds. Practical experiments support this conclusion.

4.3.3 Recommended Number of Rounds

We summarize the results of our security analysis and give the number of rounds to prevent the attacks in Table 2. We conclude that 3 rounds provide security against attackers who are only allowed to use a single (plaintext, ciphertext) pair. In the following, we primarily use 3 or 4 rounds and call the resulting instances $\text{RAIN}_3$ and $\text{RAIN}_4$, respectively. While our analysis shows that 3 rounds are sufficient, we propose to primarily use $\text{RAIN}_4$, with an additional round of security margin. In our evaluation in Section 6, we give numbers for both the 3-round and 4-round variants for comparison. These round numbers are valid independent of the block size $n$, when considering $n$-bit security.
Table 3: Parameters for the Banquet proof system \cite{BdSGK+21} for $\kappa$-bit security, a total of $m = (m_1 \cdot m_2)$ inversions over a native field of $\mathbb{F}_{2^n}$, lifted to an extension field $\mathbb{F}_{2^{n\lambda}}$ in the proof.

<table>
<thead>
<tr>
<th>OWF</th>
<th>$\kappa$</th>
<th>$m$</th>
<th>$m_1$</th>
<th>$m_2$</th>
<th>$n$</th>
<th>$\lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>EM-AES-128</td>
<td>128</td>
<td>160</td>
<td>10</td>
<td>16</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>LSAES-128</td>
<td>128</td>
<td>50</td>
<td>5</td>
<td>10</td>
<td>32</td>
<td>1</td>
</tr>
<tr>
<td>EM-LSAES-128</td>
<td>128</td>
<td>40</td>
<td>5</td>
<td>8</td>
<td>32</td>
<td>1</td>
</tr>
</tbody>
</table>

5 Constructing Signatures from our Designs

We now discuss how one can use the designs presented in the previous sections to build post-quantum signature schemes.

5.1 EM-AES, LSAES, EM-LSAES

For the constructions described in Section 2.1 and Section 3 we use the Banquet proof system \cite{BdSGK+21} directly. Using the Limbo proof system \cite{dSGOT21} would also be possible, but from the formulas given by the authors the resulting signatures would be slightly larger. We give the parameters (in the notation of \cite[Section 4]{BdSGK+21}) for the different instantiations in Table 3. The number of parties $N$ and parallel repetitions $\tau$ can be chosen as a tradeoff between size and speed, following the soundness analysis of \cite[Section 6.1]{BdSGK+21}.

Key generation also follows Banquet and BBQ: secret keys are randomly sampled until one is found such that none of the S-box inputs are zero. As explained in Section 2.1 for the EM options we can instead sample the key uniformly at random, and choose the per-user random value until no S-box inputs are zero.

In the case of LSAES, the probability of a zero input per S-box is decreased from $1/256$ to $1/2^{32}$, and the total number of S-boxes is nearly four times lower, so the probability of a zero input per key, given by $(1 - 2^{-32})^\# S-boxes$, is about $2^{-26}$. As argued in \cite{dDOS19, BdSGK+21}, this excludes only a small portion of the key space, and does not reduce security significantly.

5.2 Rainier – A Signature Scheme based on Rain

For building a signature scheme from our RAIN design, we do not use Banquet directly. Instead, we use a simplified variant of Banquet that is better suited for the low number of inversions in RAIN. We give a short summary of Banquet and the modifications we make to the protocol in the following.

5.2.1 High-Level Roadmap

We rely on the ideas of Banquet \cite{BdSGK+21}, which introduced a new MPCitH proof protocol suited for block ciphers using field inverses. Instead of computing the inverse operation using a square-and-multiply approach or relying on other techniques from the MPC literature like the masked inversion of BBQ \cite{dDOS19}, the Banquet proof protocol instead injects the output of the field inverse $t_i = s_i^{-1}$.
as an additional input to the protocol. While this removes the need to compute it during the MPC evaluation, the parties must now validate the injected values. A simple approach is to multiply $s_i$ and $t_i$ in the MPC protocol and check if the result is 1 (inputs $s_i = 0$ are filtered out during key generation). However, this again introduces additional overhead to perform the multiplications. An optimization proposed in Banquet is to instead interpolate two degree-$(m - 1)$ polynomials $S$ and $T$ using all of the inputs and outputs of the $m$ inverse operations, respectively, and show that their product is equal to a degree-$(2m - 2)$ polynomial $P$ which evaluates to 1 at the points $[1,m]$. This check is done by evaluating the polynomials at a randomly chosen point $R$ and checking that $S(R) \cdot T(R) = P(R)$. To make this polynomial proof zero-knowledge, an additional random point is included when interpolating $S$ and $T$ in the MPC protocol. This ensures that the evaluation of the polynomials at the point $R$ does not leak any information about the interpolated values.

To further improve efficiency, Banquet first splits the $m$ values into a square of size $\sqrt{m} \times \sqrt{m}$ and then interpolates the rows to get $\sqrt{m}$ polynomials of degree $\sqrt{m}$. Then, a random linear combination of these smaller polynomials is checked in a similar fashion, reducing the number of elements that need to be communicated from $2m$ to $m + O(\sqrt{m})$. However, in our investigation we observed that while this step is beneficial for Banquet (where $m$ is in the hundreds), it is not for our Rain design (with $m \in \{3, 4\}$). With such a small $m$ the additional overhead of this approach results in larger signatures. We therefore take a step back and use the simple polynomial checking protocol outlined above for Rain.

In general, the crossover point seems to be for $m$ around 40–50, depending also on $N$ and the field size. For instance, with EM-LSAES-128 (which uses a 32-bit field), we have $m = 40$, and we estimate that when $N \geq 256$ the simplified protocol produces shorter proofs.

In addition to producing smaller signatures (at least for Rain) and having a simpler protocol description, the simple polynomial checking protocol, which we call Rainier in the following, also transforms the proof from a 7-pass protocol to a 5-pass protocol. This means that when we make it non-interactive using the Fiat-Shamir transformation, we can rely on the previous analysis of 5-pass protocols by Kales and Zaverucha [KZ20a] for the soundness calculation. The new soundness analysis then results in a lower number of parallel repetitions, further reducing signature size.

Further Optimizations. Notice that we do not need to include all 3 points $S(R)$, $T(R)$ and $P(R)$ in the final signature, since from any two of them we can calculate the third one. Therefore, we only include $S(R)$ and $T(R)$ in the final signature and save one field element per repetition. (Note that this can also be applied to the original Banquet protocol for minor size reductions.) Additionally, in a similar fashion to Picnic, we can omit the inclusion of $\Delta s$, $\Delta t$ and $\Delta P$ in the signature in case that the challenged party is the first one ($j = 1$). This would save, on average, $1/N \cdot \tau \cdot (\kappa + L \cdot \kappa + (L + 1) \cdot \kappa)$ bits in the signature. However, for a larger number of parties this does not result in significant savings (e.g., for 128-bit security and 64 parties we save 44 bytes on average), and we therefore choose not to use this optimization, which has the drawback of making signatures variable-length. However, if future work were to
use $N = 2$, this would be a significant optimization.

**Remark 3.** As mentioned, one alternative to the Banquet-style protocol above is to directly calculate $s_i \cdot t_i$ in the MPC protocol and check that the result is equal to one. This can be done using a variant of [KKW18] working over the native field. However, a quick analysis of the resulting signature size shows that this approach is inferior: A signature using this approach with RAIN at the 128-bit security level has 512 bits of MPC input per repetition, 384 bits of preprocessing information and 384 bits of online communication, totaling 1280 bits per repetition. Together with the commitments and seeds, a KKW-style signature would be 12.9 KB in size, which is just a little larger than Picnic3 for the same parameter choices. As we will show in Section 6, Rainier can produce much smaller signatures, highlighting the interplay between the cipher and the MPCitH proof system.

### 5.2.2 The Rainier signature scheme

The key generation function of Rainier, $\text{KeyGen}(1^\kappa)$ uses RAIN in Theorem 1, i.e., it samples $k, p \leftarrow \{0, 1\}^\kappa$, computes $c \leftarrow \text{RAIN}_k(p)$. If any of the inverse operations in the computation have a zero input, repeat with a new random $p$ until there are no zero inputs. Output the secret key $sk = k$ and the public key $pk = (p, c)$. Due to the large size of the field, the probability of the zero case occurring is negligible.

We give the full signing algorithm of Rainier in Fig. 5 and Fig. 6, and show the verification algorithm in Fig. 7. These algorithms make use of several hash functions: $\text{Commit}$, $H_1$ and $H_2$; as well as two pseudorandom generators: $\text{Expand}$ and $\text{ExpandTape}$. All of these are instantiated using SHAKE128 (or SHAKE256 for larger security levels), with different constants added for domain separation. $\text{Sample}(t)$ is a helper function that samples elements from a random tape $t$ that was output by $\text{ExpandTape}$, keeping track of the current position on the tape.

### 5.2.3 Soundness Analysis and Parameter Selection

Rainier is parameterizable on the security level $\kappa$ (which is also equal to the key and block size of RAIN, and inverse operations are done in $\text{GF}(2^\kappa)$), the number of parties $N$ and the number of parallel repetitions $\tau$.

In the following we give an analysis of the soundness of the protocol, based on the previous analysis done in [BdSGK+21], taking into account our modifications to the protocol. In Phase 2 of Fig. 6, a random value $R_e$ is produced for each repetition which is the point at which the polynomials $S_e$, $T_e$ and $P_e$ are evaluated. Remember that for all $l \in [L]$ the values of $S_e(l)$ and $T_e(l)$ correspond to the input and output of the inverse operation respectively and that we set up $P_e$ in such a way that $P_e(l) = 1$. If $S_e \cdot T_e \neq P_e$, we can bound the probability of a random challenge $R_e$ not detecting this inequality by the number of zeroes of the polynomial $Q_e = P_e - S_e \cdot T_e$. In the case of a cheating prover, $Q_e(R_e)$ may be zero by chance, but the number of zeroes of $Q_e$ is bounded by Theorem 4.

---

4We note that hashing is the dominant part of the overall runtime (about 55% from our benchmarks) and the usage of a faster hash function (e.g., SHA-2 or Haraka [KLMR16]) can substantially improve performance.
Lemma 4 (Schwartz–Zippel Lemma). Let $Q(x) \in \mathbb{F}[x]$ be a non-zero polynomial
Figure 6: Rainier Signature Scheme - Phases 2-5. Computation of the checking protocol, challenging and opening of the views of the checking protocol.

Phase 2: Challenging the checking polynomials.
1: Compute challenge hash: \( h_1 \leftarrow H_1(\text{msg}, \text{pk}, \sigma_1) \).
2: Expand \( h_1 \): \((R_e)_{c \in [\tau]} \leftarrow \text{Expand}(h_1)\) with \( R_e \in \mathbb{F}_{2^\kappa} \setminus [0, L - 1] \).

Phase 3: Committing to the views of the checking protocol.
1: for each execution \( e \) do
2: for each party \( i \) do
3: Compute and open: \( a_e^{(i)} \leftarrow S_e^{(i)}(R_e) \), \( b_e^{(i)} \leftarrow T_e^{(i)}(R_e) \) and \( c_e^{(i)} \leftarrow P_e^{(i)}(R_e) \).
4: Set \( \sigma_2 \leftarrow (S_e(R_e), T_e(R_e), P_e(R_e), (a_e^{(i)}, b_e^{(i)}, c_e^{(i)})_{i \in [N]} e \in [\tau]) \).

Phase 4: Challenging the views of the checking protocol.
1: Compute challenge hash: \( h_2 \leftarrow H_2(\text{salt}, h_1, \sigma_2) \).
2: Expand hash: \((\bar{i}_e)_{e \in [\tau]} \leftarrow \text{Expand}(h_2)\) where \( \bar{i}_e \in [N] \).

Phase 5: Opening the views of the checking protocol.
1: for each execution \( e \) do
2: \( \text{seeds}_e \leftarrow \{ \log_2(N) \text{ nodes needed to compute } \text{seed}_{e,i} \text{ for } i \in [N] \setminus \{\bar{i}_e\} \} \).
3: Output \( \sigma \leftarrow (\text{salt}, h_1, h_2, (\text{seeds}_e, \text{com}_{\bar{i}_e}^{(e)}), \Delta \text{sk}_e, (\Delta t_{e,1})_{1 \in [L]}, \Delta P_e(k)_{k \in [L, 2L]}, S_e(R_e), T_e(R_e))_{e \in [\tau]} \).

of degree \( d \geq 0 \). For any finite subset \( S \) of \( \mathbb{F} \),

\[
\Pr \left[ r \overset{\$}{\leftarrow} S : Q(r) = 0 \right] \leq \frac{d}{|S|}.
\]

Since the polynomial \( Q_e \) is of degree \( 2L \), and we sample the challenge \( R_e \) from a set of size \( 2^\kappa - L \), we arrive at a maximum cheating probability of \( \frac{2^\kappa - L}{2L} \).

The second challenge in Phase 4 of Fig. 6 chooses the player whose state does not get revealed. A cheating prover has a \( 1/N \) chance of guessing this challenge and therefore cheating.

Following the analysis of 5-pass protocols without early abort in [KZ20a], we provide an attack strategy with the minimum work, where we cheat \( \tau_1 \) times for the first challenge (i.e., produce a polynomial \( P_e \neq S_e \cdot T_e \) and hope that the random challenge hits a random zero in \( Q_e \)) and cheat in the remaining \( \tau_2 = \tau - \tau_1 \) instances by guessing which player does not get revealed (the second challenge) and cheat in the MPC computation of that player.

The minimum cost of the attack is then given by

\[
\text{Cost}(\kappa, N, \tau) = \frac{1}{\text{SPMF}(\tau, \tau_1, 2L/(2^\kappa - L))} + N^{\tau - \tau_1}.
\]

where \( \text{SPMF} \) is the summed probability mass function,

\[
\text{SPMF}(n, k, p) = \sum_{k' = k}^{n} \binom{n}{k'} p^{k'} (1 - p)^{n-k'},
\]

where each term gives the probability of guessing correctly in \( k' \) of \( \tau \) independent trials, each with success probability \( p \). The choice of \( \tau_1 \) that minimizes the attack...
Verify(pk, msg, σ) :
1: Parse σ ← (salt, h₁, h₂, (seeds_e, com_{i_e}, Δsk_e, (Δl_e,ℓ)_{ℓ∈[L]}),
   (ΔP_e(k))_{k∈[L,2L]}, S_e(R_e), T_e(R_e))_{e∈[e]},
2: Expand hashes as (R_e)_{e∈[e]} ← Expand(h₁) and (i_e)_{e∈[e]} ← Expand(h₂).
3: for each execution e do
4:   Use seeds_e to recompute seed_{i_e}^{(i)} for i ∈ [N] \ i_e. 
5:   for each party i ∈ [N] \ i_e do
6:     Recomputer com_{i_e}^{(i)} ← Commit(salt, e, i, seed_{i_e}^{(i)}),
7:     ExpandTape(salt, e, i, seed_{i_e}^{(i)} and sk_{e}^{(i)} ← Sample(tape_{i_e}^{(i)}).
8:     if i = 1 then
9:       Adjust first share: sk_{e}^{(i)} ← sk_{e}^{(i)} + Δsk_e.
9:   for each S-box ℓ do
10:     Compute linear operations in RAIN to obtain s_{e,ℓ}. 
11:     Sample output share: t_{e,ℓ}^{(i)} ← Sample(tape_{i_e}^{(i)}).
12:     if i = 1 then
13:       Adjust first share: t_{e,ℓ}^{(i)} ← t_{e,ℓ}^{(i)} + Δl_e,ℓ.
14:     Recompute output broadcast ct_{i_e}^{(i)}.
15:     Do as in Phase 1, Lines 19–21 to interpolate S_{e}^{(i)}, T_{e}^{(i)}.
16:   for k from 0 to L − 1 do
17:     if i = 1, set P_{e}^{(i)}(k) = 1; otherwise set P_{e}^{(i)}(k) = 0.
18:   for k from L to 2L do
19:     Sample share: P_{e}^{(i)}(k) ← Sample(tape_{i_e}^{(i)}).
20:   if i = 1 then
21:     Adjust first share: P_{e}^{(i)}(k) ← P_{e}^{(i)}(k) + ΔP_e(k).
22:     Interpolate P_{e}^{(i)} and compute c_{e}^{(i)} ← P_{e}^{(i)}(R_e).
23:     Compute a_{e}^{(i)} ← S_{e}(R_e) and b_{e}^{(i)} ← T_{e}^{(i)}(R_e).
24:     Compute P_e(R_e) ← S_e(R_e) · T_e(R_e).
25:     Compute missing output broadcast ct_{i_e}^{(i)} = ct − ∑_{i′≠i} c_{i′}^{(i)}.
26:     Compute missing shares a_{e}^{(i)} ← S_e(R_e) − ∑_{i′≠i} a_{e}^{(i′)}, b_{e}^{(i)} ← T_e(R_e) − ∑_{i′≠i} b_{e}^{(i′)}
27:     Set h'_1 ← H_1 \left(\frac{msg, pk, salt, ((com_{i_e}^{(i)}))_{i∈[N]}(ct_{i_e}^{(i)}))_{i∈[N]} \right),
28:     Set h'_2 ← H_2 \left((a_{e}^{(i)}, b_{e}^{(i)}, c_{i_e}^{(i)}))_{i∈[N]}e∈[e] \right).
29:     Output accept iff \(h'_1 = h_1 \) and \(h'_2 = h_2\).

Figure 7: Rainier Verification algorithm.

cost gives the optimal attack

\[ τ_1 = \arg \min_{0 ≤ τ' ≤ τ} \frac{1}{SPMF(τ, τ', 2L/(2^κ - L))} + N^{τ-τ'}. \]

To select secure parameters, we fix κ and N and increase the value of τ until
the best attack strategy has an average cost of $2^\kappa$ or more. A script to select secure parameters is included in the source code and was used to generate all parameter sets in this work.

### 5.2.4 Security Proof

Although our protocol is similar in nature to the proof protocol of Banquet [BdSGK+21], the modifications to the protocol (reducing it from 7 to 5 internal rounds) mean that the existing proof does not apply directly. Like Banquet, we conjecture that Rainier is also secure in the quantum random oracle model (QROM), as there has been much recent progress in QROM analysis for signature schemes constructed from $\Sigma$-protocols (see, e.g., [DFMS19, DFM20, GHHM20]), but we leave a formal proof to future work. A good starting point is the work of Don et al. [DFM20], who give a generic QROM security analysis of signatures schemes constructed from 5-round $\Sigma$-protocols using the Fiat-Shamir transform.

**Theorem 5.** The Rainier signature scheme is EUF-CMA-secure, assuming that $\text{Commit}$, $H_1$, $H_2$ and $\text{Expand}$ are modelled as random oracles, $\text{ExpandTape}$ is a secure PRG, the seed tree construction is computationally hiding, the $(N, \tau, L)$ parameters are appropriately chosen, and that $\text{KeyGen}$ is a secure one-way function.

**Proof.** See Appendix A.3 for the full proof.

**Alternative Proof systems for Rain** Recently and in concurrent work, Kales and Zaverucha [KZ21] presented a new proof system which could be used together with Rain to build a signature scheme. Since no implementation of their proof system exists at this time, we leave a full integration and implementation for future work. Based on their theoretical estimates, we expect their proof system to reduce the proof size by 2 elements of $\mathbb{F}$ per repetition. For concrete Rainier parameters, this results in a size reduction of about 8 – 10%.

### 6 Performance Evaluation & Comparison with Other Designs

We now compare the performance in terms of signature sizes and signing/verification run-times of several post-quantum signature schemes. To ensure a fair comparison, all candidates are benchmarked on the same machine, a standard Intel desktop i7-4790 CPU @ 3.60GHz. For Banquet [BdSGK+21], we used the public implementation [Ban21], which we also used as a starting point for our own C++ implementations of Banquet using the single-key Even–Mansour variants, LSAES and Rainier$^5$. For other signature schemes, we use their implementation from SUPERCOP$^6$.

---

$^5$Our implementations are available at [https://github.com/IAIK/rainier-signatures](https://github.com/IAIK/rainier-signatures)

$^6$[https://bench.cr.yp.to/supercop.html](https://bench.cr.yp/to/supercop.html), version 20210125
6.1 Performance Evaluation

We compare the designs explored in this work to several other post-quantum signature schemes, namely the ones in the third round of the NIST PQC standardization project; of these, the ones with the most similarity are Picnic [ZCD+20], which is also an MPCitH-based scheme, and SPHINCS+ [HBD+20] which also only relies on the security of symmetric-key primitives.

We focus our comparisons on the 128-bit security level (NIST level L1), as this is expected to be sufficiently secure in the near term, and due to the fact that some of the constructions like EM-AES are not able to instantiate higher security levels in a straightforward way. We give additional data for the 192-bit and 256-bit security levels in Appendix D.1.

In Fig. 8, we give a graphical overview of the signing performance of the schemes explored in this work and compare them to Picnic and SPHINCS+. For almost all of the schemes in Fig. 8, verification has similar performance to signing, except for SPHINCS+, where the verification is much faster, in the range of 0.2-2.6ms, depending on the instance and used hash function. We give the numerical data and detailed parameter sets in Appendix D, Table 5.

Fig. 8 shows that Rainier₃ and the more conservative Rainier₄ (based on Rain₃ and Rain₄, respectively) outperform the other schemes considerably. The flexible parametrization of MPCitH-based signatures results in a large array of possible instances that have varying tradeoffs between signature size and performance. When compared to the small SPHINCS+ variants, Rainier can offer signatures with similar size but two orders of magnitude faster signing or can offer similar signing performance while reducing signature sizes from 8 to 5 KB. Comparing to Picnic3, we can also have signatures that are less than half the size with the same performance. In Fig. 8 we also see the expected improvements.
from single-key Even-Mansour (EM) and LSAES. We see approximately 10% improvement in signature sizes (and running times) when using the EM variants, since the AES key schedule can be evaluated publicly in the MPCitH protocol. The instances using LSAES-128 also show a similar improvement compared to using AES-128, and the EM-LSAES instances can have signatures below 10 KB.

We compare Banquet and Rainier to the third-round candidates in the NIST PQC project in Table 3, as well as a signature built using the Limbo proof system [dSGOT21], which offers a tradeoff of size and speed compared to Banquet. Since the public implementation of Limbo is intended for circuits over \( \mathbb{F}_2 \) and their Limbo-Sign AES-128 variant does not have a public implementation, we compare directly to the benchmarks from [dSGOT21]. We also remark that the recursive nature of the Limbo proof system is not that well suited to the small number of multiplications in Rainier. Even when executing a single recursion in Limbo, based on the formulas given in [dSGOT21], proof sizes are already larger than Rainier.

In Table 4, we also highlight the public key sizes since in a PKI environment, an X509 certificate includes both a signature and a public key. In this scenario the combined size of public key + signature is relevant, benefitting the signature schemes based on symmetric-key primitives having very small public keys, allowing them to somewhat offset their larger signatures.

Table 4: Comparison of public-key and signature sizes at the 128-bit security level for the third-round candidates of the NIST PQC standardization project and the designs explored in this work. Size in bytes, time in ms. Limbo numbers are taken directly from [dSGOT21].

<table>
<thead>
<tr>
<th>Scheme</th>
<th>pk</th>
<th>sig</th>
<th>Sign Verify</th>
</tr>
</thead>
<tbody>
<tr>
<td>Picnic1-L1-FS [ZCD+20]</td>
<td>32</td>
<td>32860</td>
<td>1.60 1.31</td>
</tr>
<tr>
<td>Picnic3-L1 [ZCD+20]</td>
<td>32</td>
<td>12468</td>
<td>5.27 3.99</td>
</tr>
<tr>
<td>sphincs128sha256simple [HBD+20]</td>
<td>32</td>
<td>8080</td>
<td>248.37 0.75</td>
</tr>
<tr>
<td>sphincsf128sha256simple [HBD+20]</td>
<td>32</td>
<td>16976</td>
<td>14.73 1.79</td>
</tr>
<tr>
<td>Dilithium2 [LDK+20]</td>
<td>1312</td>
<td>2420</td>
<td>0.07 0.03</td>
</tr>
<tr>
<td>Falcon-512 [PFH+20]</td>
<td>897</td>
<td>666</td>
<td>0.11 0.02</td>
</tr>
<tr>
<td>Rainbow Ia-Classic [DCP+20]</td>
<td>161600</td>
<td>66</td>
<td>0.02 0.01</td>
</tr>
<tr>
<td>GeMSS128v2 [CFM+20]</td>
<td>352188</td>
<td>33</td>
<td>320.99 0.08</td>
</tr>
<tr>
<td>Banquet-AES-128 [BdSGK+21]</td>
<td>32</td>
<td>13284</td>
<td>47.31 43.03</td>
</tr>
<tr>
<td>Limbo-Sign AES-128 [dSGOT21]</td>
<td>32</td>
<td>14512</td>
<td>29.00 27.00</td>
</tr>
<tr>
<td>Banquet-EM-AES-128</td>
<td>32</td>
<td>11940</td>
<td>41.05 36.88</td>
</tr>
<tr>
<td>Banquet-EM-LSAES-128</td>
<td>32</td>
<td>10496</td>
<td>20.99 18.91</td>
</tr>
<tr>
<td>Rainier3-128 ((N = 16, \tau = 33))</td>
<td>32</td>
<td>8544</td>
<td>0.87 0.81</td>
</tr>
<tr>
<td>Rainier3-128 ((N = 107, \tau = 20))</td>
<td>32</td>
<td>6176</td>
<td>2.96 2.92</td>
</tr>
<tr>
<td>Rainier3-128 ((N = 1624, \tau = 13))</td>
<td>32</td>
<td>4880</td>
<td>28.28 28.16</td>
</tr>
<tr>
<td>Rainier3-128 ((N = 16, \tau = 33))</td>
<td>32</td>
<td>9600</td>
<td>1.03 0.96</td>
</tr>
<tr>
<td>Rainier4-128 ((N = 107, \tau = 20))</td>
<td>32</td>
<td>6816</td>
<td>3.47 3.42</td>
</tr>
<tr>
<td>Rainier4-128 ((N = 1625, \tau = 13))</td>
<td>32</td>
<td>5260</td>
<td>33.41 33.11</td>
</tr>
</tbody>
</table>

As Table 4 shows, the lattice based schemes Dilithium and Falcon provide the best combined sizes for public key and signature, while also having very fast signing and verification operations. The multivariate schemes Rainbow and
GeMSS suffer in this scenario due to their large public keys, however Rainbow has the fastest signing and verification times. Rainier provides smaller certificate sizes than Picnic and SPHINCS\textsuperscript{+}, and can also improve upon their signing times, while SPHINCS\textsuperscript{+} providing faster verification times for their small parameter set than similar sized Rainier instances.

**Other One-Way Function Designs.** We discuss using alternative one-way functions such as Vision in MPCitH-based signatures in Appendix D.2.

### 7 Conclusions and Future Work

In this work, we present MPCitH-based signature schemes that produce – to the best of our knowledge – the smallest signature sizes when compared to existing schemes in this category. In addition, we open up or reinforce several exciting directions for future research.

When implementing Rainier, an implementation of the multiplication in $\text{GF}(2^n)$ for relatively large $n$ is needed. The same is true for signature schemes based on elliptic curves over binary fields. Hence, basing Rainier on the same field could utilize shared resources, e.g., use the same multiplier in hardware. Therefore, it seems worthwhile to explore the overhead of a hybrid signature scheme of Rainier and a signature scheme using elliptic curves over binary fields as given by NIST [KG13] compared to implementations that just perform either of them.

Since we profit from using AES in Even–Mansour mode, it is of interest to evaluate which distinguisher on round-reduced AES can actually speed up key recovery attacks on round-reduced AES in Even–Mansour mode only having a single (plaintext, ciphertext) pair. In this OWF security setting, is it easier to recover the key of round-reduced AES or round-reduced AES in Even–Mansour mode? What is the difference in the number of rounds that can be attacked? We also note that when AES is used in EM mode for signature key pair generation, since the key used to define the permutation is public, the key schedule used to derive round keys may be far more complex (e.g., it could be SHAKE), without increasing signature size since it is no longer evaluated in the MPC computation. Does this improve the security of the EM construction?

Since MPCitH-based signature schemes are built from interactive proof protocols, it is possible to use them interactively in certain authentication scenarios (e.g., smart cards). This has been investigated for ZKBoo, Picnic, and Banquet in [GMO16, KZ20b, BdSGK21]. Performance in this scenario is roughly an order of magnitude better than the non-interactive case, is the same true of Rainier?

The number of rounds we use for LSAES in Table 5 is the same as AES (e.g., 10 at the 128-bit security level). For OWF security, this seems to provide ample margin, and reduced-round variants would offer better performance. So this raises the question of how many rounds are sufficient for OWF security of LSAES and EM-LSAES with 32-bit and larger S-boxes? Another interesting topic is the security of Rain. While our current analysis covers its usage in the signature scheme Rainier, it would be interesting to conduct more research on its security when allowing more than the single data pair we consider in our scenario.
References


Andreas Hulsing, Daniel J. Bernstein, Christoph Doobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos


A Security Proofs

A.1 Building a OWF from a Block Cipher

Chase et al. [CDG+17, App. D] show that any block cipher \( E \) (with key size equal to the block size) together with Theorem 1 yields a family of secure one-way functions assuming \( E \) is a pseudorandom function (PRF). However, their proof technique boils down to building a distinguisher for the PRF with two oracle queries and while this is a nice generic result, in the scenario of the signature use-case, we actually do not need security against two queries, but only ever reveal a single plaintext and its corresponding ciphertext. We therefore show that we can also base the security of the resulting one-way function on the security of the block cipher against key recovery attacks with a single known plaintext.

First, we recall the general definition of a security against key-recovery attacks from [GB01], with slight changes to fit our notation.

**Definition 6.** Let \( E : \mathcal{K} \times \mathcal{M} \mapsto \mathcal{M} \) be a family of functions and let \( \mathcal{A} \) be an algorithm with access to an oracle \( O \) and outputs a string. We consider the experiment:

\[
\text{Experiment } \text{Exp}_{kr}^E(\mathcal{A}) :
\]

\[
K \overset{\$}{\leftarrow} \text{Keys}(E) \\
K' \leftarrow O^{K} \\
\text{if } K = K' \text{ then return } 1, \text{ else return } 0
\]

The kr-advantage of \( \mathcal{A} \) is defined as

\[
\text{Adv}_{kr}^E(\mathcal{A}) = \Pr[\text{Exp}_{kr}^E(\mathcal{A}) = 1].
\]

We say that \( E \) is kr-secure if \( \text{Adv}_{kr}^E(\mathcal{A}) \) is negligible.

Based on the exact definition of the oracle \( O \), this allows modeling all kinds of key-recovery attacks, e.g., known-plaintext (KP) or chosen-plaintext (CP) attacks. Below we give the definition of the oracle \( O \) for a \( q \)-KP attack, where an attacker gets access to \( q \) known plaintexts.

**Algorithm 1** \( q \)-KP oracle \( O_K() \):

1. if \( |\mathcal{Q}| \geq q \) then abort. \( \triangleright \) We already answered \( q \) queries.
2. \( x \overset{\$}{\leftarrow} \mathcal{M} \).
3. if \( x \in \mathcal{Q} \) then goto step 2. \( \triangleright \) Check if \( x \) is fresh.
4. \( x \rightarrow \mathcal{Q} \).
5. Return \( x, E_K(x) \).

In our scenario, we are considering the key recovery security with 1 known plaintext, i.e., we have a 1-KP oracle. We additionally weaken the winning condition in \( \text{Exp}_{kr}^E \) in that we do not require \( K = K' \) to hold, but rather say that \( K' \) has to be a consistent key for the plaintexts in \( \mathcal{Q} \), which just means that \( E_K(x) = E_{K'}(x) \) for all \( x \in \mathcal{Q} \). This captures the fact that multiple keys might exist that map a single plaintext to a ciphertext, however, on average we expect only a single key since we always require \( |\mathcal{M}| \geq |\mathcal{K}| \) in practice.
Definition 7 (1-KP-kr-security). Let $E : K \times M \mapsto M$ be a family of functions and let $A$ be an algorithm with access to the 1-KP oracle $O_K$ and outputs a string. We consider the experiment:

Experiment $\text{Exp}_{E}^{1\text{-KP-kr}}(A)$:

1. $K \overset{\$}{\leftarrow} \text{Keys}(E)$
2. $K' \leftarrow A^{O_K}$
3. If $E_K(x) = E_{K'}(x)$ then return 1, else return 0

The 1-KP-kr-advantage of $A$, denoted $\text{Adv}_{E}^{1\text{-KP-kr}}(A)$, is the probability that $A$ outputs 1 in the experiment. We say $E$ is 1-KP-kr-secure if $\text{Adv}_{E}^{1\text{-KP-kr}}(A)$ is negligible.

We also recall the definition of a one-way function from [GB01].

Definition 8. An algorithm $f : \{0,1\}^* \mapsto \{0,1\}^*$ is a one-way function if:

1. there exists a PPT that given $x$ outputs $f(x)$;
2. and for every PPT algorithm $A$ there is a negligible function $\epsilon(\cdot)$ so that

$$\Pr[x \overset{\$}{\leftarrow} \{0,1\}^\kappa, y \leftarrow f(x), x^* \leftarrow A(1^\kappa, y) : f(x^*) = y] \leq \epsilon(\kappa).$$

Theorem 9. Using a block cipher $E$ that is 1-KP-kr-secure in Theorem 7 results in a one-way function $f_\kappa(k) := (x, E_k(x))$, where $x$ selects the concrete instance from the family of one-way functions according to Theorem 7.

Proof. It is trivial to see that condition (1) of Theorem 8 holds if the original $E$ is efficiently computable, since for any key $k \in K$ and any plaintext block $x \in M$, we can always compute the ciphertext $y = E_k(x)$ with a single call to $E$. For condition (2), we build a basic reduction by using the adversary against the OWF property to build an adversary $B$ against the 1-KP-kr game. $B$ calls the 1-KP oracle $O$ and receives $(x, y)$ and passes it as an input to the OWF adversary $A$. $B$ then returns the output of $A$ as its own output. It is easy to see that our reduction $B$ has the same success probability as $A$, namely $\text{Adv}_{E}^{1\text{-KP-kr}}(A)$.

A.2 Multi-Target Security of EM-OWF

In the context of a signature scheme, each of $\ell$ users has a public key $(x_i, y_i)$ and an attacker must find a key $K$ such that $y_i = F_K(x_i)$ for any $i \in [\ell]$. Ideally the cost of attacking any one of $\ell$ users is at least as expensive as attacking one user.

There are some multi-target attacks known for single-key EM in the context of encryption. Fouque, Joux and Mavromati [FJM14] give a multi-target key recovery attack (that requires multiple (adaptive) queries to $E$, which are not possible when $E$ is used as a key generation function for signatures). Mouha and Luykx [ML15] prove that the advantage of distinguishing multiple EM instances from multiple permutations depends on the number of instances.

To mitigate multi-user attacks we consider two possible countermeasures. The first is to choose a random $x_i$ per user, and the second is to choose a random permutation $\pi_i$ per user. Without mitigation, a simple multi-target attack in our scenario is possible: simply guess $K^*$, compute $y^* = F_{K^*}(x)$ for the fixed $x$, and compare $y^*$ to all $y_i$; if a match is found, then $K^*$ is consistent with
the matching \((x, y_i)\). Another attack remains when \(x_i\) is random, but \(\pi\) is fixed. Note that a query \(X\) to \(\pi\) corresponds to some key for each user, namely \(K_i\) such that \(X = K_i + x_i\). Then one key for each user can be tested by checking whether \(K_i - X = y_i\), and so the queries to \(\pi\) are amortized across all users, and security loses a factor \(\ell\).

To create a random \(\pi_i\) each user also generates a random key as the public constant for the the block cipher used to construct \(\pi\), and outputs the public key \((x_i, y_i, \pi_i)\). Once \(\pi\) is random per user, it does not appear necessary to also choose random \(x_i\), so we fix it to zero.

### A.3 Security Proof of Rainier

Since the Rainier signature scheme is similar to Banquet the security analysis is also similar. The schemes are just different enough that the results do not directly apply, necessitating a separate analysis for Rainier. That said, this section is a simplified version of the corresponding section in [BdSGK+21] where the scheme and analysis are modified for a 5-round protocol, rather than the 7-round one as in [BdSGK+21]. Many parts of the analysis are identical, and we give full credit to [BdSGK+21].

In Theorem 5, we prove that Rainier is an EUF-CMA secure signature scheme. Our definition of unforgeability under chosen message attacks is the standard one, as defined in [Kat10, Definition 1.6]. We first prove in Theorem 10 that Rainier is EUF-KO secure, i.e., secure against forgery attacks where the attacker is only given the public key, and no signature queries. A formal definition is obtained from the EUF-CMA definition by removing the adversary’s access to the signing oracle. The idea is that because the protocol is sound, if an attacker successfully creates a forgery, then by reading the random oracle query history, we can extract the secret key, inverting the one-way function used in key generation.

Then to show that the scheme is EUF-CMA secure in Theorem 5, we additionally show that signatures may be simulated without knowledge of the private key, by programming the random oracles.

All adversaries in this section are assumed to be probabilistic polynomial time (in \(\kappa\)) algorithms.

**Lemma 10.** Let \(\text{Commit, } H_1, H_2\) be modeled as random oracles, \(\text{Expand}(\cdot)\) be modeled as a random function, and let \((N, \tau, \kappa, L)\) be parameters of the Rainier signature scheme. Let \(\mathcal{A}\) be an adversary against the EUF-KO security of Rainier that makes a total of \(Q\) random oracle queries. Assuming that KeyGen is an \(\varepsilon_{\text{owf}}\)-hard one-way function, then \(\mathcal{A}\)'s advantage in the EUF-KO game is

\[
\varepsilon_{\text{ko}} \leq \varepsilon_{\text{owf}} + \frac{\tau N + 1)Q^2}{2\kappa} + \Pr[X + Y = \tau],
\]

where \(\Pr[X + Y = \tau]\) is as described in the proof.

**Remark 11.** We do not express \(\Pr[X + Y = \tau]\) as a closed function; we must choose parameters \((N, \tau, L)\) for Rainier such that it is negligible in \(\kappa\).

**Proof.** We give an algorithm \(B\) which uses the EUF-KO adversary \(\mathcal{A}\) to compute a pre-image for the key generation OWF.

Algorithm \(B\) simulates the EUF-KO game using the random oracles \(H_c\) (shorthand for \(\text{Commit}\)), \(H_1\) and \(H_2\) and query lists \(Q_c, Q_1\) and \(Q_2\). In addition,
Algorithm 2 $H_c(q_c = (\text{salt}, e, i, \text{seed}))$:

1: $x \leftarrow \{0, 1\}^{2\kappa}$. \\
2: if $x \in \text{Bad}$ then abort. \>
3: $x \rightarrow \text{Bad}$. \>
4: $(q_c, x) \rightarrow \mathcal{Q}_c$. \>
5: Return $x$.

$B$ maintains three tables $T_{sh}$, $T_{in}$ and $T_{op}$ to store shares of the parties, inputs to the MPC protocol and openings of the polynomial checking protocol that it recovers from $A$’s RO queries. $B$ also maintains a set $\text{Bad}$ to keep track of the outputs of all three random oracles. We also ignore calls to $\text{Expand}()$ in our analysis, since they are used to expand outputs from $H_1$ and $H_2$ when $\text{Expand}()$ is a random function this is equivalent to increasing the output lengths of $H_1$ and $H_2$.

Behavior of $B$. On input $pk$, a OWF challenge, algorithm $B$ forwards it to $A$ as a Rainier public key for the EUF-KO game. It lets $A$ run and answers its random oracle queries in the following way. We assume (w.l.o.g.) that Algorithm 2, Algorithm 3 and Algorithm 4 only consider queries that are correctly formed, and ignore duplicate queries.

- $H_c$: When $A$ queries the commitment random oracle, $B$ records the query to learn which commitment corresponds to which seed. See Algorithm 2.

- $H_1$: When $A$ commits to seeds and sends the offsets for the secret key and the inverse values, $B$ checks whether the commitments were output by its simulation of $H_c$. If any were for some $e$ and $i$, then $B$ is able to reconstruct the shares for party $i$ in execution $e$. If $B$ was able to reconstruct every party’s share for any $e$, then it can use the offsets included in $\sigma_1$ to extract the values used by $A$ in that execution. For the checking polynomials, $B$ uses the newly sampled response to expand the challenges and extract the checking polynomials. See Algorithm 3.

- $H_2$: No extraction takes place during this random oracle simulation. See Algorithm 4.

When $A$ terminates, $B$ checks the $T_{in}$ table for any entry where the extracted $sk_e$ is consistent with $pk$. If a match is found, $B$ outputs $sk_e$ as a pre-image for the OWF, otherwise $B$ outputs $\perp$.

Advantage of the reduction. Given the behavior presented above, we have the following by the law of total probability:

$$
\Pr[A \text{ wins}] = \Pr[A \text{ wins } \land B \text{ aborts}] + \Pr[A \text{ wins } \land B \text{ outputs } \perp] \\
+ \Pr[A \text{ wins } \land B \text{ outputs } sk] \\
\leq \Pr[B \text{ aborts}] + \Pr[A \text{ wins } \mid B \text{ outputs } \perp] \\
+ \Pr[B \text{ outputs } sk].
$$

(7)

Let $Q_{\text{com}}$, $Q_1$ and $Q_2$ denote the number of queries made by $A$ to each respective random oracle. Given the way in which values are added to $\text{Bad}$, we
Algorithm 3 $H_1(q_1 = \sigma_1)$:

1. for $e \in [\tau], i \in [N]$ do $\text{com}_e^{(i)} \rightarrow \text{Bad}$.

If the committed seed is known for a certain $e, i$, then $B$ records the shares of the secret key and of the inverse values from that party that derive from that seed and the offsets committed to in $\sigma_1$:

2. for $(e, i) \in [\tau] \times [N] : \exists \text{seed}_e^{(i)}: ((\text{salt}, e, i, \text{seed}_e^{(i)}), \text{com}_e^{(i)}) \in Q_c$ do

3. $\text{sk}_e^{(i)}, (t_e^{(i)})_e \leftarrow \text{ExpandTape}(\text{salt}, e, i, \text{seed}_e^{(i)})$.

4. if $i = 1$ then $\text{sk}_e^{(i)} \leftarrow \text{sk}_e^{(i)} + \Delta \text{sk}_e$ and $(t_e^{(i)})_e \leftarrow (t_e^{(i)})_e + \Delta t_e$.

5. $(\text{sk}_e^{(i)}, (t_e^{(i)})_e) \rightarrow T_{\text{in}}[q_1, e, i]$.

6. for each $e : \forall i, T_{\text{in}}[q_1, e, i] \neq \emptyset$ do

7. $\text{sk}_e \leftarrow \sum \text{sk}_e^{(i)}$ and $(t_e)_e \leftarrow (\sum, t_e^{(i)})_e$.

8. $(\text{sk}_e, (t_e)_e) \rightarrow T_{\text{in}}[q_1, e]$.

9. $x \leftarrow \{0, 1\}^{2\kappa}$.

10. if $x \in \text{Bad}$ then abort.

11. $x \rightarrow \text{Bad}$.

12. $(q_1, x) \rightarrow Q_1$.

Store the multiplication checking values.

13. $(R_e)_e \leftarrow \text{Expand}(x)$.

14. for each $e : T_{\text{in}}[q_1, e] \neq \emptyset$ do

15. $(P_e(R_e), S_e(R_e), T_e(R_e))_e \rightarrow T_{\text{op}}[q_2, e]$.

16. Return $x$.

have:

$$\Pr[B \text{ aborts}] = \frac{\#\text{times an } x \text{ is sampled} \cdot \Pr[B \text{ aborts at that sample}]}{(Q_{\text{com}} + Q_1 + Q_2)} \cdot \max |\text{Bad}| \leq (Q_{\text{com}} + Q_1 + Q_2) \cdot \frac{Q_{\text{com}} + (\tau N + 1)Q_1 + 2Q_2}{2^{2\kappa}} \leq \frac{(\tau N + 1)(Q_{\text{com}} + Q_1 + Q_2)^2}{2^{4\kappa}}. \tag{8}$$

We now analyze the probability of $A$ winning the EUF-KO experiment conditioned on the event that $B$ outputs $\bot$, i.e., no pre-image to $pk$ was found on the query lists.

Cheating in the first round. For any query $q_1 \in Q_1$, and its corresponding answer $h_1 = (R_e)_e \in [\tau]$, let $G_1(q_1, h_1)$ be the set of indices $e \in [\tau]$ of “good executions” where both $T_{\text{in}}[q_1, e] = (\text{sk}_e, (t_e)_e \in [\ell])$ is non-empty and and it holds that

$$P_e(R_e) = S_e(R_e) \cdot T_e(R_e). \tag{9}$$
Cheating in the second round.

Algorithm 4 $H_2(q_2 = (h_1, \sigma_2))$:

1: $h_1 \rightarrow \text{Bad}$.
2: $x \leftarrow \{0, 1\}^{2^n}$.
3: if $x \in \text{Bad}$ then abort.
4: $x \rightarrow \text{Bad}$.
5: $(q_2, x) \rightarrow Q_2$.
6: Return $x$.

If there does not exist such a $q_1$, let $G_1(q_1, h_1) = \emptyset$.

For any such good execution $e \in G_2(q_2, h_2)$, since $B$ outputs $\bot$ but $A$ wins, this implies that either the challenges in the first round were such that Eq. (9) held (in which case any value of $R_e$ passes the check), or the challenge $R_e$ was sampled such that Eq. (10) held. Conditioning on the first event not happening, Theorem 4 gives us that the second happens with probability at most $p_1 := 2L / (2^n - 2L)$, given that $h_1$ is distributed uniformly at random (which holds assuming $H_1$ and $\text{Expand}()$ are random functions).

As the response $h_1$ is uniform, each $e \in [\tau]$ has the same independent probability of being in $G_1(q_1, h_1)$, given that $B$ outputs $\bot$. We therefore have that $\#G_1(q_1, h_1) \mid_\bot \sim X_{q_1}$, where $X_{q_1} = \mathcal{B}(\tau, p_1)$, where $\mathcal{B}(\tau, p_1)$ is the binomial distribution with $\tau$ events, each with success probability $p_1$. Letting $(q_{\text{best}1}, h_{\text{best}1})$ denote the query-response pair which maximizes $\#G_1(q_1, h_1)$, we then have that

$$\#G_1(q_{\text{best}1}, h_{\text{best}1}) \mid_\bot \sim X = \max_{q_1 \in \mathcal{Q}_1} \{X_{q_1}\}.$$ 

Cheating in the second round. Each second round query $q_2 = (h_1, \sigma_2)$ that $A$ makes to $H_2$ can only be used in a valid signature if there exists a corresponding query $(q_1, h_1) \in \mathcal{Q}_1$. Then for each “bad” first-round execution $e \in [\tau] \backslash G_1(q_1, h_1)$, either verification failed, in which case $A$ couldn’t have won, or the verification passed, despite Eq. (9) not being satisfied. This implies that exactly one of the parties must have cheated. At least one cheater is required for verification to pass, but as $N - 1$ parties are opened, verification would fail if more than one party cheated.

Since $h_2 \in [N]^\tau$ is distributed uniformly at random, the probability that this happens for all such “bad” first-round executions $e$ is

$$\left(\frac{1}{N}\right)^{\tau - \#G_1(q_1, h_1)} \leq \left(\frac{1}{N}\right)^{\tau - \#G_1(q_{\text{best}1}, h_{\text{best}1})}.$$ 

The probability that this happens for at least one of the $Q_2$ queries made to $H_2$ is

$$\Pr[A \text{ wins } | \#G_1(q_{\text{best}1}, h_{\text{best}1}) = \tau_1] \leq 1 - \left(\frac{1}{N}\right)^{\tau - \tau_1} Q_2.$$ 

Finally conditioning on $B$ outputting $\bot$ and summing over all values of $\tau_1$, we have that

$$\Pr[A \text{ wins } | \bot] \leq \Pr[X + Y = \tau]$$

(10)

where $X$ is as before, and $Y = \max_{q_2 \in \mathcal{Q}_2} \{Y_{q_2}\}$ where the $Y_{q_2}$ variables are independently and identically distributed as $\mathcal{B}(\tau - X, 1/N)$. 

43
Conclusion. Bringing Eq. (7), Eq. (8) and Eq. (10) together, we obtain the following.

\[
\Pr[A \text{ wins}] \leq \frac{(\tau N + 1)(Q_{\text{com}} + Q_1 + Q_2)^2}{2^{2n}} + \Pr[X + Y = \tau] + \Pr[B \text{ outputs } sk]
\]

Assuming KeyGen is an \(\varepsilon_{\text{OWF}}\)-secure OWF and setting \(Q = Q_{\text{com}} + Q_1 + Q_2\) gives the required bound and concludes the proof.

We assume that \(\text{ExpandTape}()\) is a secure pseudorandom generator (PRG), again using the standard definition, see for example [BS20, Definition 3.1]. In our implementation \(\text{ExpandTape}\) is implemented with the SHA-3 based extendable output function SHAKE [NIS15]. The assumption related to the tree derivation construction for random seeds is that it must be hiding. Informally, this means that after revealing a subset of the seeds (e.g., \(N - 1\) of \(N\) seeds), the remaining seeds remain hidden to a computationally bounded adversary. In [CDG+20, Section 6.3] it is shown that this holds when the hash function used to derived seeds is modelled as a random oracle. Secure one-way functions are defined in [Gol07, Section 2.2].

**Theorem** 5. The Rainier signature scheme is EUF-CMA-secure, assuming that Commit, \(H_1\), \(H_2\) and Expand are modelled as random oracles, \(\text{ExpandTape}\) is a secure PRG, the seed tree construction is computationally hiding, the \((N, \tau, L)\) parameters are appropriately chosen, and that KeyGen is a secure one-way function.

**Proof.** Fix an attacker \(A\). We define a sequence of games where the first corresponds to \(A\) interacting with the real signature scheme in the EUF-CMA game. Through a series of hybrid arguments we show that this is indistinguishable from a simulated game, under the assumptions above. Let \(G_0\) be the unmodified EUF-CMA game and let \(B\) denote an adversary against the EUF-KO game that acts as a simulator of the EUF-CMA game to \(A\). As we’re in the random oracle model: when \(A\) queries one of its random oracles, \(B\) first checks if that query has been recorded before; if so, then it responds with the recorded answer; if not, \(B\) forwards the query to its corresponding random oracle, records the query and the answer it receives and forwards the answer to \(A\). Let \(G_i\) denote the probability that \(A\) succeeds in game \(G_i\). At a high level, the sequence of games is as follows:

- **\(G_0\)**: \(B\) knows a real secret key \(sk\) and can compute signatures honestly;
- **\(G_1\)**: \(B\) replaces real signatures with simulated ones which no longer use \(sk\);

\(B\) then uses the EUF-KO challenge \(pk^*\) in its simulation with \(A\).

We note that \(A\)’s advantage in the EUF-CMA game is \(\varepsilon_{\text{CMA}} = G_0 - (G_0 - G_1) + G_1\) and we obtain a bound on \(G_0\) by first bounding \(G_0 - G_1\) and then \(G_1\).

**Hopping to Game \(G_1\).** When \(A\) queries the signing oracle, \(B\) simulates a signature by sampling a random secret key \(sk^*\), choosing a party \(P_i^*\) at random and cheating in the verification phase and in the broadcast of the output shares \(ct_e^{(i)}\) such that the circuit still outputs the correct ciphertext, and finally ensuring
that the values observed by $A$ are sampled independently of $sk^*$ and with a
distribution that is computationally indistinguishable from a real signature. $B$
programs $H_1$ to return the $R_e$ values that it sampled, and $H_2$ to hide the cheating
party $P_{i^*}$ in Phase 5.

We now argue that the simulated signatures in $G_1$ are computationally
indistinguishable from real signatures in $G_0$. We list a series of (sub) game hops
which begins with $G_0$, where $sk$ is known and signatures are created honestly,
and ends with $G_1$, where signatures are simulated without using $sk$. With each change to $B$’s behavior, we give an argument as to why the simulation remains
indistinguishable, and quantify these below.

1. The initial $B$ knows the real $sk$ and can compute honest signatures as
in the protocol. It only aborts if the salt that it samples in Phase 1 has
already been queried. As its simulation is perfect, $B$ is indistinguishable
from the real EUF-CMA game as long as it does not abort.

2. Before beginning, the next $B$ samples $h_2$ at random and expands it to obtain
$(i^*_e)_{e \in \tau}$; these are the unopened parties, which $B$ will use for cheating. It
proceeds as before and programs the random oracle $H_2$ so that it outputs
$h_2$ when queried in Phase 6. If that query has already been made, $B$ aborts
the simulation.

3. In Phase 1, the next $B$ replaces $seed_e^{(i^*)}$ in the binary tree, for each $e \in \tau$,
by a randomly sampled one. This is indistinguishable from the previous
version of $B$ assuming that the tree structure is hiding.

4. The next $B$ replaces the random tapes for party $i^*$, i.e., the outputs of
$\text{ExpandTape}(\text{salt}, e, i^*, seed_e^{(i^*)})$, by random outputs (independent of the
seed). This is indistinguishable from the previous reduction assuming that
$\text{ExpandTape}()$ is a secure PRG.

5. The next $B$ replaces the commitments of the unopened parties $\text{com}_{e}^{(i^*)}$
with random values (i.e., without querying $\text{Commit}$). $B$ aborts if $A$ queries
$x$ such that $\text{Commit}(x)$ was output by $B$.

6. Before starting Phase 2, the next $B$ samples $h_1$ at random and expands
it to obtain $(R_e)_{e \in \tau}$; this will enable it to sample the checking values at
random. It then proceeds as before and programs the random oracle $H_1$
to output $h_1$ in Phase 2. If that query has already been made, $B$ aborts the
simulation.

7. In Phase 1, the next $B$ interpolates $S_e^{(i)}$ for $i \in [N] \setminus \{i^*\}$, samples the
values $S_e(R_e)$ at random, computes $S_e^{(i^*)}(R_e) = S_e(R_e) - \sum_{i \neq i^*} S_e^{(i)}$ and
interpolates $S_e^{(i^*)}$ using $k \in \{0, \ldots, L - 1\} \cup \{R_e\}$. It does the same for the $T$
polynomials and computes $P_e$ and the offsets according to the protocol. As
the uniform distribution of honestly generated $S_e(R_e)$ and $T_e(R_e)$ (opened
in Phase 3) comes from the uniform distribution of $\bar{s}_e$ and $\bar{t}_e$ read from the
random tape (recall that $seed_e^{(i^*)}$ is no longer used), this is indistinguishable
from the previous hop. The same holds for the shares of party $P_{i^*}$
that are opened in Phase 5. The distribution of the $\Delta P_e$ offsets is therefore
also indistinguishable from a real signature as they are computed honestly.
from indistinguishable elements. (At this stage the \( P_i \) polynomials always satisfy the check since \( B \) is still using a real \( sk \).)

8. In Phase 5, the next \( B \) replaces \( c^{(r)}_i \) with \( P^{(r)}_i(R_e) = \sum_{j \neq i^*} P^{(1)}_i(R_e) \). This is indistinguishable because the \( P^{(1)}_i(R_e) \) values, for \( i \neq i^* \), are computed honestly, and the \( P_i \) value is distributed identically to an honest signature (because \( S_{e,j} \) and \( T_{e,j} \) are). From now on, the Schwartz-Zippel check always passes, even if the product relation doesn’t hold, and the distribution of everything that \( A \) can observe is indistinguishable from an honest signature and independent of hidden values.

9. The final \( B \) replaces the real \( sk \) by a random key \( sk^* \) and cheats on the broadcast of party \( P_{i^*} \)'s output share \( ct_e^{(r^*)} \) such that it matches what is expected, given the \( N - 1 \) other shares. As \( sk^{(r^*)} \) is independent from the seeds \( A \) observes, the distribution of \( \Delta sk^*_e \) is identical and \( A \) has no information about \( sk^* \). As \( P_{i^*} \) is never opened, \( B \)'s cheating on \( ct_e^{(r^*)} \) can’t be detected.

We can conclude that \( B \)'s simulation of the signing oracle is indistinguishable and that \( A \) behaves exactly as in the real EUF-CMA game unless an abort happens.

There are four points at which \( B \) could abort: if the salt it sampled has been used before, if the committed value it replaces is queried, or if its queries to \( H_1 \) and \( H_2 \) have been made previously. Let \( Q_{\text{salt}} \) denote the number of different salts queried during the game (by both \( A \) and \( B \)); each time \( B \) simulates a signature, it has a maximum probability of \( Q_{\text{salt}}/2^{2\kappa} \) of selecting an existing salt and aborting.

Let \( Q_e \) denote the number of queries made to \( \text{Commit} \) by \( A \), including those made during signature queries. Since \( \text{Commit} \) is a random oracle, and \( \text{seed}_e^{(r^*)} \) is a uniformly random \( \kappa \)-bit value not used by \( B \) elsewhere, each time \( B \) attempts a new signature, it has a maximum probability of \( Q_e/2^\kappa \) of replacing an existing commitment and aborting.

Similarly for \( H_1 \), resp. \( H_2 \), \( B \) has a maximum probability of \( Q_1/2^{2\kappa} \), resp. \( Q_2/2^{2\kappa} \) of aborting, where \( Q_1 \) and \( Q_2 \) denote the number of queries made to each random oracle during the game. Note that \( B \) samples one salt, replaces \( \tau \) commitments and makes one query to both \( H_1 \) and \( H_2 \) for each signature query.

Let \( Q_s \) be the total number of signature queries, therefore

\[
G_0 - G_1 \leq Q_s \cdot (\tau \cdot \epsilon_{\text{PRG}} + \epsilon_{\text{TREE}} + \Pr[B \text{ aborts}])
\]

where

\[
\Pr[B \text{ aborts}] \leq Q_{\text{salt}}/2^{2\kappa} + Q_e/2^\kappa + Q_1/2^{2\kappa} + Q_2/2^{2\kappa} \\
= (Q_{\text{salt}} + Q_1 + Q_2)/2^{2\kappa} + Q_e/2^\kappa \\
\leq (Q_1 + Q_2)/2^{2\kappa - 1} + Q_e/2^\kappa \quad \text{(Since } Q_{\text{salt}} \leq Q_1 + Q_2 \text{)}
\]

(\( \text{where } Q = Q_1 + Q_2 + Q_e \)).

**Bounding \( G_1 \).** In \( G_1 \), \( B \) is no longer using the witness and is instead simulating signatures only by programming the random oracles; it therefore replaces the honestly computed \( pk \) with and instance \( pk^* \) of the EUF-KO game. We see that
if $A$ wins $G_1$, i.e. outputs a valid signature, then $B$ outputs a valid signature in the EUF-KO game, and so we have

$$G_1 \leq \varepsilon_{\text{ko}} \leq \varepsilon_{\text{owf}} + \frac{(\tau N + 1)Q^2}{2k} + \Pr[X + Y = \tau],$$

where the bound on the advantage $\varepsilon_{\text{ko}}$ of a EUF-KO attacker follows from Theorem 10. By a union bound, we have that

$$\varepsilon_{\text{cma}} \leq \varepsilon_{\text{owf}} + \frac{(\tau N + 1)Q^2}{2k} + \Pr[X + Y = \tau] + Q_s \cdot (\tau \cdot \varepsilon_{\text{prg}} + \varepsilon_{\text{tree}} + Q/2^k).$$

Assuming that $\text{ExpandTape()}$ is a secure PRG that is $\varepsilon_{\text{prg}}$-close to uniform, that the seed tree construction is hiding (so that $\varepsilon_{\text{tree}}$ is negligible), that key generation is a one-way function and that parameters $(N, \tau, L)$ are appropriately chosen implies that $\varepsilon_{\text{cma}}$ is negligible in $\kappa$.

B Linear Layer and Key Schedule of LSAES

B.1 Linear layer $L$

The linear layer $L$ consists the serial application of the linear function $AB$, $SR$, and $MC$, where as usual in descriptions of AES-128, the linear layer $AB$ is seen as part of the description of the S-box. So we have $L(x) = MC \circ SR \circ AB(x)$.

The layer $AB$ interprets all bytes $x_i$ of the state as a series of bits $x_i = b_7 \parallel b_6 \parallel b_5 \parallel b_4 \parallel b_3 \parallel b_2 \parallel b_1 \parallel b_0$. On each of the resulting bit vectors, the following affine transformation is performed:

$$AB_i(x_i) = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}. $$

The layer shift rows $SR$ performs a row-wise rotation of the square state by the row index $j$ to the left. So we get

$$SR(x) = x_0 \parallel x_3 \parallel x_{10} \parallel x_{15} \parallel x_4 \parallel x_5 \parallel x_{14} \parallel x_9 \parallel x_8 \parallel x_{13} \parallel x_2 \parallel x_7 \parallel x_1 \parallel x_6 \parallel x_{11}. $$

For the MixColumns layer, each byte of the state is interpreted as an element GF($2^8$) using the irreducible polynomial $X^8 + X^4 + X^3 + X + 1$. On each of the rows $h$ ($0 \leq h < 4$), the following matrix multiplication is applied

$$MC_h(x_{0+h}, x_{1+h}, x_{2+h}, x_{3+h}) = \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_{0+h} \\ x_{1+h} \\ x_{2+h} \\ x_{3+h} \end{pmatrix}. $$

47
B.2 Key Schedule

Given our secret key \( k \) we interpret it as as bytes and immediately get our first round key \( k^{(1)} = k = k_0^{(1)} || k_1^{(1)} || \ldots || k_{15}^{(1)} \). To compute round key \( k^{(i+1)} \) from \( k^{(i)} \), we do the following

\[
\begin{align*}
    k^{(i+1)}_0 &= k^{(i)}_0 + AB_i(t^{(i)}_0) + 2^{i-1} \\
    k^{(i+1)}_j &= k^{(i)}_j + AB_i(t^{(i)}_j), \quad (1 \leq j < 4) \\
    k^{(i+1)}_j &= k^{(i)}_j + k^{(i+1)}_{j-4}, \quad (4 \leq j < 16)
\end{align*}
\]

C Matrix and Round Constant Generation for Rain

For the sake of simplicity, in this section we omit using the round-specific indices of the linear layer matrices \( M_i \) and the linearized permutation polynomials \( L_i \), i.e., we write \( M \) and \( L \) instead of \( M_i \) and \( L_i \), respectively.

C.1 Building the Linear Layer \( M \) from \( L(X) \)

In order to build our linear layer \( M \), we first pseudo-randomly generate a dense and linearized polynomial \( L(X) \in \mathbb{F}_{2^n}[X] \) of maximum degree, i.e., a polynomial with \( n \) nonzero coefficients and a degree of \( 2^n - 1 \). Note that we need this polynomial to be a permutation polynomial, and we also want the compositional inverse of this polynomial to fulfill the same properties.

For this purpose, we first define the Dickson matrix

\[
D_L = \begin{pmatrix}
a_0 & a_1 & \cdots & a_{n-1} \\
a_2^{n-1} & a_0 & \cdots & a_2^{n-2} \\
& & \vdots & \vdots \\
a_2^{n-1} & a_2^{n-1} & \cdots & a_0
\end{pmatrix}
\]

of the linearized polynomial \( L(X) = \sum_{i=0}^{n-1} a_i X^{2^i} \). It is known \[LN96\] that \( L \) is a permutation polynomial if and only if \( D_L \) is invertible.

In order to also ensure that the inverse of \( L \) fulfills the same properties (i.e., nonzero coefficients and maximum degree), we simply compute it. Indeed, the inverse \( L^{-1}(X) \) of \( L(X) \) is defined as

\[
L^{-1}(X) = \frac{1}{\det(D_L)} \sum_{i=0}^{n-1} \pi_i X^{2^i},
\]

where \( \pi_i \) is the \((i, 0)\)-th cofactor of \( D_L \). Hence, in order to generate a “good” polynomial \( L(X) \), we pseudo-randomly generate coefficients \( \{a_i\}_{i=0}^{n-1} \) until both \( L(X) \) and \( L^{-1}(X) \) have \( n \) nonzero coefficients and a degree of \( 2^n - 1 \).

The second step is to generate the matrix \( M \in \mathbb{F}_{2^n}^{n \times n} \) from \( L \). This transformation has explicitly been shown in \[Car63\] and has been revisited in \[WL13\]. We quickly recall it here.
Assume $L(X)$ is given. Then, we first compute the ordered power basis
\[
\{\beta_1, \beta_2, \ldots, \beta_n\} = \{\beta, \beta^2, \ldots, \beta^{n-1}\}
\]
and its dual basis
\[
\{\beta'_1, \beta'_2, \ldots, \beta'_n\},
\]
i.e., a basis such that
\[
\text{tr}(\beta_i \beta'_j) = \delta_{i,j}
\]
for $i \in [0, n]$ and $j \in [0, n]$, where $\text{tr} : \mathbb{F}_{2^n} \to \mathbb{F}_2$ is the trace function and $\delta_{i,j}$ is the Kronecker delta. Now, the element $M_{i,j}$ at the $j$-th column of the $i$-th row in the matrix $M$ is defined as
\[
M_{i,j} = \text{tr}(\beta'_i L(\beta_j)).
\]

Hence, to summarize, we first generate a suitable linearized permutation polynomial $L(X)$, and we then compute the corresponding matrix. Ignoring the invertibility evaluation (with high probability, an invertible polynomial will be found after only a few trials), the cost of this approach is approximately $n^2 + n^3 \in O(n^3)$ field operations. For example, using Sage, we are able to construct such a matrix for $n = 128$ in only a couple of seconds on an ordinary computer.

### C.1.1 Pseudo-Randomly Sampling $M$

For completeness, we mention that it is also possible to construct the linear layer matrices by pseudo-randomly sampling them (i.e., without first choosing $L$ and then computing the corresponding $M$). Here we argue that this approach will also lead to secure linear layers with high probability.

It is well-known that there exists a one-to-one relation between the general linear group $\text{GL}(n, \mathbb{F}_2)$ and the set of linearized permutation polynomials with coefficients in $\mathbb{F}_{2^n}$, which is also called the Betti–Mathieu group \cite{Dic01, Car63}. It is also known that the number of invertible $n \times n$ matrices over $\text{GF}(2)$ is
\[
\prod_{i=0}^{n-1} 2^n - 2^i > (2^n - 2^{n-1})^n = (2^{n-1})^n.
\]

Since $n \log_2(2^{n-1}) = \log_2(2^{2n^2}) - n$, it follows that at most $n$ bits of entropy are lost when considering only invertible matrices. This means that, in the worst case, no maximum-degree monomial appears in the set of all linearized permutation polynomials of degree at most $2^{n-1}$ over a fixed field, and the appearances of all other degrees are uniformly distributed. Hence, the term with the second-highest degree will appear with an overwhelming probability of at least $1 - 2^{-n}$. However, in practical tests we observed that the probability of a maximum-degree term appearing is also around $1 - 2^{-n}$. Further, the probability of the polynomial to contain all powers of 2 less than or equal to $2^{n-1}$ is approximately $(1 - 2^{-n})^n$, as expected. We therefore conjecture that any pseudo-randomly chosen $n \times n$ matrix over $\mathbb{F}_2$ will result in a high-degree and dense (as far as possible) permutation over $\mathbb{F}_{2^n}$.
C.2 Pseudo-Random Number Generation

We generate the round constants and the linear layers using the SHAKE256 extendable output function \cite{NIS15} with the input Rain-n-R (encoded as UTF-8 text), where \( N \) is replaced with the state size in bits and \( R \) is replaced with the number of rounds, e.g., Rain-128-3 for Rain3-128. We first generate the \( n \)-bit round constants \( c^{(i)} \) by squeezing \( n \) bits from the SHAKE256 instance for each constant. Then, we use the same method to generate the coefficients for our linear layers.

C.3 Concrete Instances

The round constants and matrices used for Rain-n, where \( n \in \{128, 192, 256\} \), are publicly available.\footnote{https://github.com/IAIK/rainier-signatures} These files also contain the coefficients of the corresponding linearized permutation polynomials.

D Additional Performance Data and Evaluation

We explore the performance data of a huge range of instances for the variants presented in this paper at the 128-bit security level in Table 5. Most of these instances are included in Fig. 8.

D.1 Instances for larger Security Levels

In Table 6, we give some numbers for Rainier for the 192-bit and 256-bit security levels.
Table 5: Comparison of signatures discussed in this work at the 128-bit security level. Times are in ms, sizes are in bytes.

<table>
<thead>
<tr>
<th>Design</th>
<th>$N$</th>
<th>$\tau$</th>
<th>$m_1$</th>
<th>$m_2$</th>
<th>$\lambda$</th>
<th>Sign</th>
<th>Verify</th>
<th>Sig. size</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Banquet</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES-128</td>
<td>16</td>
<td>41</td>
<td>10</td>
<td>20</td>
<td>4</td>
<td>7.03</td>
<td>5.32</td>
<td>9760</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>35</td>
<td>10</td>
<td>20</td>
<td>4</td>
<td>10.01</td>
<td>8.27</td>
<td>17456</td>
</tr>
<tr>
<td></td>
<td>57</td>
<td>31</td>
<td>10</td>
<td>20</td>
<td>4</td>
<td>15.56</td>
<td>13.37</td>
<td>15984</td>
</tr>
<tr>
<td></td>
<td>107</td>
<td>24</td>
<td>10</td>
<td>20</td>
<td>6</td>
<td>23.03</td>
<td>20.45</td>
<td>14784</td>
</tr>
<tr>
<td></td>
<td>255</td>
<td>21</td>
<td>10</td>
<td>20</td>
<td>6</td>
<td>47.31</td>
<td>43.03</td>
<td>13284</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>20</td>
<td>10</td>
<td>20</td>
<td>6</td>
<td>90.30</td>
<td>82.67</td>
<td>12976</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>18</td>
<td>10</td>
<td>20</td>
<td>6</td>
<td>161.51</td>
<td>148.55</td>
<td>11976</td>
</tr>
<tr>
<td><strong>Banquet</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EM-AES-128</td>
<td>16</td>
<td>41</td>
<td>16</td>
<td>10</td>
<td>4</td>
<td>5.57</td>
<td>4.28</td>
<td>17480</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>35</td>
<td>16</td>
<td>10</td>
<td>4</td>
<td>8.04</td>
<td>6.72</td>
<td>15496</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>31</td>
<td>16</td>
<td>10</td>
<td>1</td>
<td>7.49</td>
<td>6.54</td>
<td>13488</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>28</td>
<td>5</td>
<td>10</td>
<td>1</td>
<td>13.27</td>
<td>11.82</td>
<td>12640</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>25</td>
<td>5</td>
<td>10</td>
<td>1</td>
<td>25.28</td>
<td>22.92</td>
<td>11696</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>24</td>
<td>5</td>
<td>10</td>
<td>1</td>
<td>51.52</td>
<td>47.10</td>
<td>11616</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>22</td>
<td>5</td>
<td>10</td>
<td>1</td>
<td>98.97</td>
<td>91.77</td>
<td>11008</td>
</tr>
<tr>
<td><strong>Banquet</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES-128</td>
<td>16</td>
<td>41</td>
<td>5</td>
<td>10</td>
<td>1</td>
<td>2.93</td>
<td>2.37</td>
<td>16496</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>35</td>
<td>5</td>
<td>10</td>
<td>1</td>
<td>4.35</td>
<td>3.74</td>
<td>14656</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>31</td>
<td>5</td>
<td>10</td>
<td>1</td>
<td>7.49</td>
<td>6.54</td>
<td>13488</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>28</td>
<td>5</td>
<td>10</td>
<td>1</td>
<td>13.27</td>
<td>11.82</td>
<td>12640</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>25</td>
<td>5</td>
<td>8</td>
<td>1</td>
<td>25.28</td>
<td>22.92</td>
<td>11696</td>
</tr>
<tr>
<td></td>
<td>512</td>
<td>24</td>
<td>5</td>
<td>8</td>
<td>1</td>
<td>51.52</td>
<td>47.10</td>
<td>11616</td>
</tr>
<tr>
<td></td>
<td>1024</td>
<td>22</td>
<td>5</td>
<td>8</td>
<td>1</td>
<td>98.97</td>
<td>91.77</td>
<td>11008</td>
</tr>
<tr>
<td><strong>Rainier</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rain-128</td>
<td>8</td>
<td>44</td>
<td></td>
<td></td>
<td></td>
<td>0.70</td>
<td>0.60</td>
<td>10656</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td>0.87</td>
<td>0.81</td>
<td>8544</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td>1.29</td>
<td>1.23</td>
<td>7440</td>
</tr>
<tr>
<td></td>
<td>57</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td>1.87</td>
<td>1.82</td>
<td>6720</td>
</tr>
<tr>
<td></td>
<td>107</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td>2.96</td>
<td>2.92</td>
<td>6176</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td>5.65</td>
<td>5.63</td>
<td>5336</td>
</tr>
<tr>
<td></td>
<td>920</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td>16.82</td>
<td>16.70</td>
<td>5024</td>
</tr>
<tr>
<td></td>
<td>1624</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td>28.28</td>
<td>28.16</td>
<td>4880</td>
</tr>
<tr>
<td></td>
<td>3180</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>51.90</td>
<td>51.49</td>
<td>4704</td>
</tr>
<tr>
<td></td>
<td>7121</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td>105.98</td>
<td>105.15</td>
<td>4496</td>
</tr>
<tr>
<td></td>
<td>65384</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td>801.98</td>
<td>794.35</td>
<td>4128</td>
</tr>
<tr>
<td><strong>Rainier</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rain-128</td>
<td>8</td>
<td>44</td>
<td></td>
<td></td>
<td></td>
<td>0.81</td>
<td>0.71</td>
<td>12064</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td>1.03</td>
<td>0.96</td>
<td>9600</td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td>1.50</td>
<td>1.44</td>
<td>8304</td>
</tr>
<tr>
<td></td>
<td>57</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td>2.19</td>
<td>2.14</td>
<td>7456</td>
</tr>
<tr>
<td></td>
<td>107</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td>3.47</td>
<td>3.42</td>
<td>6816</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td>6.65</td>
<td>6.60</td>
<td>6080</td>
</tr>
<tr>
<td></td>
<td>920</td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td>20.01</td>
<td>19.76</td>
<td>5472</td>
</tr>
<tr>
<td></td>
<td>1625</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td>33.41</td>
<td>33.11</td>
<td>5296</td>
</tr>
<tr>
<td></td>
<td>3181</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>60.54</td>
<td>60.49</td>
<td>5088</td>
</tr>
<tr>
<td></td>
<td>7124</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td>124.88</td>
<td>121.79</td>
<td>4848</td>
</tr>
<tr>
<td></td>
<td>65422</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td>952.73</td>
<td>941.02</td>
<td>4416</td>
</tr>
</tbody>
</table>
Table 6: Performance comparison at the 192-bit and 256-bit security levels for $N \in \{16, 32, 64, 128, 256\}$. Times are in ms, sizes are in bytes.

<table>
<thead>
<tr>
<th>Design</th>
<th>$N$</th>
<th>$\tau$</th>
<th>$m_1$</th>
<th>$m_2$</th>
<th>$\lambda$</th>
<th>Sign</th>
<th>Verify</th>
<th>Sig. size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banquet</td>
<td>16</td>
<td>62</td>
<td>16</td>
<td>26</td>
<td>4</td>
<td>18.86</td>
<td>14.24</td>
<td>51216</td>
</tr>
<tr>
<td>AES-192x2</td>
<td>32</td>
<td>53</td>
<td>16</td>
<td>26</td>
<td>4</td>
<td>28.77</td>
<td>23.95</td>
<td>45072</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>40</td>
<td>16</td>
<td>26</td>
<td>6</td>
<td>42.72</td>
<td>36.98</td>
<td>39808</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>36</td>
<td>16</td>
<td>26</td>
<td>6</td>
<td>74.43</td>
<td>66.18</td>
<td>36704</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>32</td>
<td>16</td>
<td>26</td>
<td>6</td>
<td>128.80</td>
<td>116.41</td>
<td>34080</td>
</tr>
<tr>
<td>Banquet</td>
<td>16</td>
<td>84</td>
<td>20</td>
<td>25</td>
<td>4</td>
<td>29.99</td>
<td>23.23</td>
<td>83488</td>
</tr>
<tr>
<td>AES-256x2</td>
<td>32</td>
<td>63</td>
<td>20</td>
<td>25</td>
<td>6</td>
<td>42.30</td>
<td>35.23</td>
<td>73114</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>54</td>
<td>20</td>
<td>25</td>
<td>6</td>
<td>67.96</td>
<td>59.51</td>
<td>64420</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>48</td>
<td>20</td>
<td>25</td>
<td>6</td>
<td>116.85</td>
<td>104.65</td>
<td>58816</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>43</td>
<td>20</td>
<td>25</td>
<td>6</td>
<td>205.26</td>
<td>185.95</td>
<td>54082</td>
</tr>
<tr>
<td>Banquet</td>
<td>16</td>
<td>62</td>
<td>8</td>
<td>13</td>
<td>1</td>
<td>7.23</td>
<td>5.81</td>
<td>44024</td>
</tr>
<tr>
<td>LSAES-192x2</td>
<td>32</td>
<td>53</td>
<td>8</td>
<td>13</td>
<td>1</td>
<td>11.40</td>
<td>9.65</td>
<td>38924</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>45</td>
<td>8</td>
<td>13</td>
<td>1</td>
<td>19.17</td>
<td>16.68</td>
<td>34148</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>41</td>
<td>8</td>
<td>13</td>
<td>1</td>
<td>35.85</td>
<td>32.04</td>
<td>32108</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>37</td>
<td>8</td>
<td>13</td>
<td>1</td>
<td>65.92</td>
<td>59.43</td>
<td>29876</td>
</tr>
<tr>
<td>Banquet</td>
<td>16</td>
<td>84</td>
<td>5</td>
<td>25</td>
<td>1</td>
<td>12.69</td>
<td>10.00</td>
<td>73408</td>
</tr>
<tr>
<td>LSAES-256x2</td>
<td>32</td>
<td>72</td>
<td>5</td>
<td>25</td>
<td>1</td>
<td>19.25</td>
<td>15.88</td>
<td>65248</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>63</td>
<td>5</td>
<td>25</td>
<td>1</td>
<td>31.69</td>
<td>26.91</td>
<td>59128</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>56</td>
<td>5</td>
<td>25</td>
<td>1</td>
<td>55.57</td>
<td>48.15</td>
<td>54368</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>50</td>
<td>5</td>
<td>25</td>
<td>1</td>
<td>99.71</td>
<td>88.17</td>
<td>50160</td>
</tr>
<tr>
<td>Rainier$_3$</td>
<td>16</td>
<td>49</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.97</td>
<td>1.77</td>
<td>18944</td>
</tr>
<tr>
<td>RAIN$_3$-192</td>
<td>32</td>
<td>40</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2.90</td>
<td>2.76</td>
<td>16448</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>33</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4.38</td>
<td>4.23</td>
<td>14384</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>29</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>7.36</td>
<td>7.22</td>
<td>13352</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>25</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>12.85</td>
<td>12.60</td>
<td>12128</td>
</tr>
<tr>
<td>Rainier$_4$</td>
<td>16</td>
<td>49</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2.30</td>
<td>2.07</td>
<td>21296</td>
</tr>
<tr>
<td>RAIN$_4$-192</td>
<td>32</td>
<td>40</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>3.34</td>
<td>3.15</td>
<td>18368</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>33</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>5.06</td>
<td>4.90</td>
<td>15968</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>29</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8.55</td>
<td>8.38</td>
<td>14744</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>25</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>15.09</td>
<td>14.81</td>
<td>13328</td>
</tr>
<tr>
<td>Rainier$_3$</td>
<td>16</td>
<td>65</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>3.05</td>
<td>2.83</td>
<td>33440</td>
</tr>
<tr>
<td>RAIN$_3$-256</td>
<td>32</td>
<td>53</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4.48</td>
<td>4.27</td>
<td>28992</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>44</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6.97</td>
<td>6.80</td>
<td>25504</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>38</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>11.74</td>
<td>11.57</td>
<td>23264</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>33</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>20.40</td>
<td>20.23</td>
<td>21280</td>
</tr>
<tr>
<td>Rainier$_4$</td>
<td>16</td>
<td>65</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>3.46</td>
<td>3.18</td>
<td>37600</td>
</tr>
<tr>
<td>RAIN$_4$-256</td>
<td>32</td>
<td>53</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4.99</td>
<td>4.75</td>
<td>32384</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>44</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>7.71</td>
<td>7.47</td>
<td>28320</td>
</tr>
<tr>
<td></td>
<td>128</td>
<td>38</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>13.08</td>
<td>12.80</td>
<td>25696</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td>33</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>23.16</td>
<td>22.73</td>
<td>23392</td>
</tr>
</tbody>
</table>
We can see the same trends as for the 128-bit security level, where the LSAES-variants provide a 10-15% improvement in signature size over standard Banquet, while usually being about twice as fast. Recall that the EM construction is not directly instantiable with AES-192 and 256 since it requires an $s$-bit permutation for $s$-bit security. Rainier produces much smaller signatures than Banquet and also compares favorably to Picnic3 (with signatures of 27.4 KB at the 192-bit and 48.4 KB at the 256-bit security level) and SPHINCS+ (with its “small” parameter sets having 17 KB at the 192-bit and 29.8 KB at the 256-bit security level, and its “fast” parameter sets being comparable in size to Picnic3). Furthermore, the signing and verification speeds of Rainier are again much faster than the other designs (with the exception of the verification times in SPHINCS+, which are in the order of 0.5-5ms for all parameter sets). We give a visual representation of the signing times and signatures sizes at the 192-bit and 256-bit security level for various schemes in Fig. 9a and Fig. 9b.

D.2 Alternative OWF Designs for MPCitH Signatures

Most existing proposals and designs of block ciphers and one-way-functions are, like Rijndael/AES, designed for very different use-cases and hence have no chance of being competitive with Rain. However we discuss a few where this can not be ruled out a priori and take a quick look how they could be used in the proof protocol of Banquet [BdSGK+21] or the modified variant presented in Section 5.2.2.

Vision [AAB+20]. Vision is a block cipher design intended for use-cases in multi-party computation and zero-knowledge proofs with the goal of reducing the arithmetic complexity of the cipher. Vision is a family of block ciphers operating over binary fields, where the only non-linear operation is the field inverse. This makes Vision an easy fit for the Banquet proof protocol. One point of remark is Vision’s heavy key schedule, which essentially corresponds to a second evaluation of the cipher itself; while in many scenarios this can be amortized over many encryption calls, for the signature use-case we only prove a single encryption, resulting in a large overhead. However, we can also use Vision in a single-key Even–Mansour construction as discussed in Section 2.1. We also give the sizes for these variants. While we have not implemented the full signature for the following variants, we give the calculated signature sizes for three different Vision instances at the 128-bit security level in the Banquet proof protocol in Table 7. The first one is Vision Mark I, an instance recommended in [AAB+20] intended to be similar to AES. The second and third ones are generated with the provided parameter generation script [https://github.com/KULeuven-COSIC/Marvellous] intended to resemble the design choices of LSAES and Rain with field sizes of 32 and 128 bits respectively. However, we remark that the resulting parameters are for general use and could potentially be reduced for the attack scenario presented by the use in the signature construction, where only a single (plaintext, ciphertext) pair is published.

Table 7 highlights our remarks about the heavy key schedule of Vision as the signatures produced by using Vision instances are larger than all other alternatives we investigated and even unmodified Banquet. When considering the EM variants, we see that the signature size of EM-Vision Mark I and EM-Vision
Figure 9: Comparison of signing time and signature size of various schemes at 192-bit and 256-bit security levels. Picnic instances include all proposed third round parameter sets. SPHINCS$^+$ instances include all simple parameter sets (haraka, sha256, shake256, in order of decreasing performance).

Variant 2 are equal to EM-AES-128 and EM-LAES-128, respectively. This is because of the similarities of the internal structure of the ciphers, resulting in the same number of inversions for the same field sizes. For Variant 3, the number of internal rounds is much larger than RAIN, since the parameter selection script takes into account all types of attacks, even those not applicable in the signature scenario. A future dedicated analysis could reduce the number of rounds needed for security, but due to the structure in the affine layer of Vision the number of rounds will very likely need to be larger than for RAIN.

LegRoast and PorcRoast [Bd20]. Leg- and PorcRoast are two recent proposals of MPCitH-based signature schemes that use the Legendre PRF instead of a block cipher as a one-way function. Security is based on a more structured number-theoretic assumption, which arguably puts these in a different
Table 7: Signature sizes in bytes for different instances of Vision with a security level of 128 bits using a state of \( m \) elements over the field \( \mathbb{F}_{2^m} \) and \( r \) rounds of the cipher (resulting in a total of \( \#\text{inv.} \) inverses). The proof system parameters are set to \( N = 64 \) with the minimum number of rounds \( \tau \) chosen so that the signature scheme provides 128 bits of security and the remaining parameters chosen as recommended by [BdSGK+21].

| Instance          | \( n \) | \( m \) | \( r \) | \( \#\text{inv.} \) | \( |\text{sig}| \) |
|-------------------|--------|--------|--------|----------------|-------------|
| Vision Mark I     | 8      | 16     | 10     | 320            | 21 176      |
| Vision Variant 2  | 32     | 4      | 10     | 80             | 17 952      |
| Vision Variant 3  | 128    | 1      | 36     | 72             | 41 184      |
| EM-Vision Mark I  | 8      | 16     | 10     | 160            | 14 232      |
| EM-Vision Variant 2 | 32 | 4      | 10     | 40             | 12 000      |
| EM-Vision Variant 3 | 128 | 1      | 36     | 36             | 25 056      |
| Banquet (AES-128) | 8      | 16     | 10     | 200            | 15 968      |
| Banquet (EM-AES-128) | 8 | 16     | 10     | 160            | 14 232      |
| Banquet (LSAES-128) | 32 | 4      | 10     | 50             | 13 488      |
| Banquet (EM-LSAES-128) | 32 | 4      | 10     | 40             | 12 000      |
| Rainier\(_3\)     | 128    | 1      | 3      | 3              | 6 720       |
| Rainier\(_4\)     | 128    | 1      | 4      | 4              | 7 456       |

category than the schemes in this paper, but we provide a brief comparison. On our machine, the LegRoast parameter sets have signature sizes of 12.5 to 16.48 KB with a signing time of 3.19 to 17.95 ms, while the PorcRoast parameter sets have signature sizes of 6.4 to 8.8 KB with signing times of 1.43 to 8.85 ms (with larger signatures leading to faster signing times). Public keys are 4 KB, compared to 32 bytes in Rainier.

**LSAES with Larger S-Boxes.** As mentioned above, the choice of 32-bit S-boxes in LSAES was made to match the field \( \mathbb{F}_{2^8} \) used in Banquet (where \( \lambda = 4 \) for L1). However, we can further reduce the number of S-boxes by using 64-bit or even 128-bit S-boxes. Since the (LS)AES key schedule operates on 32-bits at a time, this is the largest size for the key schedule, but rather than mixing S-box sizes, here we focus on EM-LSAES and for simplicity fix \( N = 256 \).

With 64-bit S-boxes, EM-LSAES has only 20 S-boxes, and the signature size is 9 168 bytes using the Rainier protocol and 9 216 bytes using the Banquet protocol, compared to 10 496 bytes for 32-bit S-boxes (Table 7). With 128-bit S-boxes, EM-LSASES has only 10 S-boxes and signatures are 9 072 or 9 312 bytes using the simplified or full Banquet protocol. Unfortunately sizes do not really decrease further with larger S-boxes, and RAIN gives much better performance. This can be explained by having 10 rounds in the LSAES variants vs. 3 rounds in RAIN. Indeed, note that signature sizes of \( r \)-round EM-LSAES with 128-bit S-boxes is the same as \( r \)-round RAIN. This motivates study of reduced-round versions of EM-LSAES, which has a weaker linear layer than RAIN, however, we expect that 7 and perhaps even 5 rounds to be sufficient.
Other AES-Based Permutations. In order to use the EM OWF construction at higher security levels, we require 192 and 256-bit permutations. Using Rijndael is one option, but there are also constructions of permutations from AES. The motivation for these constructions is that they make use of hardware acceleration for AES (such as the AESNI instructions), while no such hardware is available for Rijndael.

The Simpira v2 permutation (with $b \geq 2$) together with the EM construction gives a candidate OWF for Banquet-like signatures at security level L5 (256-bit security) [GM16]. However, the number of AES rounds required is 30 rounds, compared to AES-256x2 (two calls to AES-256), which uses 28 AES rounds + the key schedule. So it seems Simpira v2 is not a good choice for our application.

Haraka v2 [KLMR16] defines the permutation $\pi_{256}$. This permutation has 5 rounds, where each $\pi_{256}$ round uses four AES rounds, for a total of 20 AES rounds. Aside from the AES rounds, the state is permuted, so counting the AES rounds is sufficient when discussing signature signature sizes. The parameters in [KLMR16] were chosen and analyzed to provide 256 bits of second preimage resistance, meaning Haraka-v2-256 is a OWF suitable for use in a Banquet-like scheme, where key generation is $pk = \text{Haraka-v2}(sk) = sk \oplus \pi_{256}(sk)$ for random 256-bit secret key $sk$ (ignoring that a salt is required to prevent multi-target attacks; we assume some of the round constants can be selected per-party, to give each user a distinct instance of $\pi_{256}$).

Then Banquet-Haraka-v2-256 provides 256-bit security, with 320 eight-bit S-boxes. Signatures range from 66.7–44.4 KB (with $N = 16$–256) better than any of the Banquet-based options in Table 4 but worse than the Rainier options.

One can also consider the Haraka v2 analogues constructed with LSAES rounds, instead of AES rounds (with the caveat that the security analysis must be revisited). Then instead of 320 8-bit S-boxes, we have 80 32-bit S-boxes. Here the signature sizes range from 54.6–39.4 KB (with $N = 16$–256).