

# Efficient Lifting for Shorter Zero-Knowledge Proofs and Post-Quantum Signatures

Daniel Kales  
Graz University of Technology  
daniel.kales@iaik.tugraz.at

Greg Zaverucha  
Microsoft Research  
gregz@microsoft.com

May 12, 2022

## Abstract

MPC-in-the-head based zero-knowledge proofs allow one to prove knowledge of a preimage for a circuit defined over a finite field  $\mathbb{F}$ . In recent proofs the soundness depends on the size  $\mathbb{F}$ , and small fields require more parallel repetitions, and therefore produce larger proofs. In this paper we develop and systematically apply lifting strategies to such proof protocols in order to increase soundness and reduce proof size. The strategies are (i) lifting parts of the protocol to extension fields of  $\mathbb{F}$ , (ii) using reverse-multiplication friendly embeddings to pack elements of  $\mathbb{F}$  into a larger field and (iii) to use an alternative circuit representation. Using a combination of these strategies at different points in the protocol, we design two new proof systems well suited to small circuits defined over small fields.

As a case study we consider efficient constructions of post-quantum signatures, where a signature is a proof of knowledge of a one-way function preimage, and two commonly used one-way functions are defined over small fields (AES and LowMC). We find that carefully applying these lifting strategies gives shorter signatures than the state-of-the-art: our AES-based signatures are 1.3x shorter than Banquet (PKC 2021) and our LowMC-based signatures are almost 2x shorter than the NIST-candidate algorithm Picnic3. We implement our schemes and provide benchmarks. Finally, we also give other optimizations: some generally applicable to this class of proofs, and some specific to the circuits we focused on.

## 1 Introduction

An efficient class of zero-knowledge (ZK) proof systems can be constructed from multi-party computation (MPC) protocols, using the *MPC-in-the-head* (MPCitH) paradigm. The name refers to the key idea of the construction, having the prover simulate an MPC protocol “in their head” as part of the proof. As MPC protocols exist for arbitrary circuits, and can be customized for specific circuits, this paradigm is very flexible. The theoretical framework for MPCitH proofs was given in [IKOS07], and a first practical instantiation was presented in ZKBoo [GMO16] and its improved version ZKB++ [CDG<sup>+</sup>17]. Since then, the area of MPCitH proof systems has seen multiple new instantiations with different improvements and tradeoffs. For example, the KKW proof system [KKW18] uses

an MPC protocol with a preprocessing phase, allowing for a more communication-efficient online phase and a variable number of parties.

A major application of MPCitH proof systems are post-quantum signatures. The most well-known example is Picnic [CDG<sup>+</sup>17, ZCD<sup>+</sup>20] a family of signature schemes built using ZKB++ and KKW and an alternate candidate in the third round of the NIST Post-Quantum Standardization Project.<sup>1</sup> During key-generation, a one-way function (OWF) is used to generate a keypair: the input to the one-way function is the secret key and the output is the public key. A Picnic signature is a non-interactive ZK proof of knowledge of the secret input to the one-way function corresponding to the public output, with the message to be signed included in the generation of the challenge for the proof. The OWF in Picnic is based on the block cipher LowMC [ARS<sup>+</sup>15].

While ZKB++ and KKW focus on binary circuits, the MPCitH framework also allows for arithmetic circuits over larger fields. This was used, for example, in constructions focusing on signatures with (variants of) AES as the OWF, such as BBQ [dDOS19], Banquet [BdK<sup>+</sup>21] and Rainier [DKR<sup>+</sup>21]. The latter two protocols use a key idea from Baum and Nof [BN20]: instead of calculating nonlinear operations (e.g., the AES S-boxes) in the MPC protocol, shares of the result are injected by the prover as additional input. Then, a circuit-independent *checking protocol* is executed to verify that the injected values are indeed correct. In the BN proof system [BN20], which supports generic arithmetic circuits, but has very large proof sizes for circuits defined over small fields (such as the OWFs of interest for PQ signatures). The reason is that the soundness error of the checking protocol is  $1/|\mathbb{F}|$ , where  $\mathbb{F}$  is the field where the circuit is defined (mainly because a random field element is chosen as a challenge). Therefore the protocol requires a large number of parallel repetitions for small fields like  $\mathbb{F}_2$ . To get around this issue for AES, the Banquet checking protocol lifts elements from  $\mathbb{F}_{2^8}$  to a larger field  $\mathbb{F}_{2^{8\lambda}}$ , thereby increasing soundness. However, this also increases the size of the proof, since it now includes elements of the larger field. The Limbo proof system [DOT21] also uses lifting, and in [DKR<sup>+</sup>21] the authors explore alternative OWFs that are defined over large fields natively, so that lifting can be avoided. The latter approach is by far the most efficient to date (in the context of signature schemes), but does not apply to existing OWFs defined over small fields, such as AES and LowMC.

We dub this the *lifting problem*, and our goal is solve it in a more efficient way. If we apply the lifting map used in Banquet to binary circuits like LowMC we could for example lift each bit to a byte. But the lift is somewhat trivial, and the overhead, we will refer to as the *rate*, is high (eight in this example), and  $\mathbb{F}_{2^8}$  is still not as large as we'd like from a soundness perspective. For comparison, the rate in Banquet was 4 or 6 and the larger field was 32 or 48 bits. While this simple lifting approach has proven useful, a natural question is whether we can do better.

One of the strategies we will explore in this paper is to use a *reverse multiplication friendly embedding* (RMFE). An RMFE allows us to encode multiple field elements into an element in a field extension with better rate. RMFEs were introduced by Cascudo et al. [CCXY18] in their work on the amortized communication complexity of MPC protocols for binary circuits. The power of the embedding is that coordinate-wise products in the base field are mapped

---

<sup>1</sup>See <https://csrc.nist.gov/projects/post-quantum-cryptography>.

to multiplications in the extension field. For example, the  $(3, 5)_2$ -RMFE allows us to lift a batch of 3 elements of  $\mathbb{F}_2$  into  $\mathbb{F}_{2^5}$ , with rate  $5/3 = 1.6$ . Once we have encoded groups of bits  $(x_1, x_2, x_3)$  and  $(y_1, y_2, y_3)$  we can multiply the encoded values in  $\mathbb{F}_{2^5}$ , then decode to get the three ANDs  $(x_1 \cdot y_1, x_2 \cdot y_2, x_3 \cdot y_3)$ . Importantly, the encoding and decoding operations are linear, meaning parties in the MPC protocol can compute the maps on their shares locally to obtain shares of the encoded value. There are somewhat strict limitations on the arithmetic operations one can perform on encoded values such that the decoded values are correct, but we show that an efficient checking protocol is possible.

## 1.1 Contributions

In this paper we make the following contributions.

**Lifting strategies** After reviewing some known strategies for lifting, such as the simple lifting technique described above, we show how to lift with RMFEs. This is not straightforward due to the limitations of RMFEs, and we find that they cannot be directly applied to existing protocols without modification. We then discuss alternative circuit representations, and how representation can lead to shorter proofs. We give two concrete examples for our case study circuits AES and LowMC. The AES example is well-known but illustrative in our context, and our alternate representation of LowMC is a novel way to evaluate the three-bit S-box with one  $\mathbb{F}_{2^3}$  multiplication, as opposed to three  $\mathbb{F}_2$  multiplications.

**New proof systems** We present two new proof systems in this paper. The first is called **BN++**, and can be seen as an optimized version of the checking protocol from [BN20] (where it is called the sacrificing-based proof protocol). **BN++** improves on the proof sizes of [BN20] by roughly 2.5x depending on the circuit and field size, and is also the best choice for signature schemes (without lifting) for certain OWFs. (As secondary case study explained below, we use **BN++** to reduce the size and CPU costs of the Rainier [DKR<sup>+</sup>21] signature scheme.) **BN++** also supports RMFE-based checking, and is less sensitive to the lifting problem, since it can efficiently repeat the checking protocol many times when the field size is small.

We then present **Helium**, a proof system optimized for small fields based on the proof systems Banquet [BdK<sup>+</sup>21] and Rainier [DKR<sup>+</sup>21]. The communication efficiency of **Helium** comes from carefully selecting the point in the protocol where elements are lifted to larger fields, using simple lifting, RMFEs or a combination of both (the best combination of strategies depends on the circuit). By lifting later in the protocol, more of the communication is in the base field as opposed to the extension field, significantly reducing proof sizes.

We also provide a soundness analysis and concrete parameters for both **BN++** and **Helium**.

**Shorter OWF-based signatures** Our main motivation for designing **BN++** and **Helium** is for use in Picnic-like signature schemes with post-quantum security. The designs are flexible enough to use many OWFs from the literature and we investigate instantiations of our signatures with three OWFs: LowMC [ARS<sup>+</sup>15]

the OWF used in Picnic, AES-128 as the most conservative choice, and Rain [DKR<sup>+</sup>21], a new cipher optimized for MPCitH-based signatures.<sup>2</sup>

For LowMC-based signatures, we select suitable RMFEs and give concrete parameters for BN++ and Helium-based signatures. Our new designs offer better size-speed tradeoffs than the current state-of-the-art, i.e., for a fixed running time or size budget, one of our schemes will be shorter, or faster. We can obtain signatures of 6 582 bytes (at the 128-bit security level); these are 1.9x shorter than the 12.5 KB signatures of Picnic3 (the most recent version of Picnic [KZ20b]), while maintaining practical running times (below 20ms @ 3.6GHz) with our initial implementation of Helium+LowMC. When matching Picnic3 on running time, our current BN++LowMC implementation has 11 KB signatures, still about 14% shorter.

Our AES-based signatures are below 10 KB, and are the shortest in the literature being 1.36x shorter than Banquet [BdK<sup>+</sup>21] (the previous shortest, across a range of parameters), while simultaneously having more than 2x faster signing and verification times. With the Rain OWF we improve on the size and performance numbers in [DKR<sup>+</sup>21] by about 15–20%, giving the shortest MPCitH-based signatures (for any OWF) to date.

We also provide a comparison to other PQ signature schemes that are candidates in the third round of the NIST PQC process. The schemes making structured hardness assumptions (e.g., the lattice or multivariate-based schemes) have lower running times, though our new schemes have (sometimes much) shorter keys and more conservative assumptions. SPHINCS+ is the other candidate besides Picnic making limited assumptions, and comparison here generally favors Helium, except for verify times. For example, signatures with Helium+LowMC can be 1.2x shorter, 6x faster to create (compared to the `small` SPHINCS+ parameters). Alternatively, Helium+AES signatures are 1.2x larger, with a similar improvement in signing time.

## 1.2 Related Work

Originally, reverse multiplication friendly embeddings were introduced in the MPC literature in [CCXY18] and independently in [BMN18]. RMFEs were then used in later works ([DLN19, DLS20, CG20, ACE<sup>+</sup>21], amongst others), mostly to reduce the soundness errors of various checks. One difficulty in the application of RMFEs in MPC protocols is that the parties have to compute the input values for multiplication gates on the fly. For zero-knowledge proofs, the prover knows the value for each wire in the circuit beforehand and can use this knowledge to more efficiently batch values using RMFEs, which is not always possible in MPC protocols, where RMFEs are mainly used to amortize the computation of the same circuit with different input values.

The BN++ protocol bears some similarity to Limbo [DOT21], which also uses a sacrificing-based protocol to check multiplications. Our protocol is somewhat simpler, and admits a lower soundness error – we comment more on why in Section 2.6. Limbo first uses randomization to compress many multiplications into a dot-product. This dot-product is then checked using a recursive proof protocol. While this strategy leads to efficient proofs for large circuits, for the

---

<sup>2</sup>Our implementations are available at [https://github.com/IAIK/bnpp\\_helium\\_signatures](https://github.com/IAIK/bnpp_helium_signatures).

circuits we are focusing on in this work in the context of signature schemes, our protocol can produce much smaller signatures. The Limbo authors also discuss using RMFEs in a different way to ours [dOT21, Appendix A]. They propose batching proofs in the “multi-instance case” when the prover is proving  $\mathcal{C}(w_1), \dots, \mathcal{C}(w_h)$  (same circuit with multiple witnesses). This is more in line with the amortization in traditional MPC and does not seem applicable to Picnic-like signatures.

The TurboIKOS proof system [GHS<sup>+</sup>21] presents a series of optimizations to the BN protocol, applying ideas originating in the Turbospeedz MPC protocol [BNO19]. The base TurboIKOS protocol has communication cost slightly higher than BN++ and Helium, at 3 vs. about 2 field elements per gate. The first two optimizations made by BN++ are also used in a similar fashion in TurboIKOS (where they appeared first), and TurboIKOS has an optimization that is equivalent to the third in BN++ in terms of performance (i.e., both optimizations reduce the number of field elements broadcast for the checking protocol by one). For completeness we present BN++ as a series of optimizations starting from BN in Section 2.7. A further optimization of TurboIKOS can reduce size when the number of parties is small relative to the circuit size. We describe this technique in more detail in Appendix B.3, as it can be applied to BN++. However, we find that since the number of parties is the limiting factor for soundness, it is almost always better to increase the number of parties and forgo this optimization.

Outside the category of MPCitH-based proofs, multiple works also design circuit-based ZK proof systems with PQ security, like Ligerio [AHIV17, BFH<sup>+</sup>20] and Aurora [BCR<sup>+</sup>19]. Recently, Cascudo and Giunta [CG21] introduced a way to utilize reverse multiplication friendly embeddings in Aurora and Ligerio, improving their argument sizes by a factor of up to 1.65x and 3.71x, respectively. As the size of the circuit increases, these systems will quickly outperform our proof systems, since their proof size is sublinear in the circuit size. However, for the OWFs we are interested in, our proofs are much shorter.

### 1.3 Notation

In Table 1 we define some of the notation we use frequently. Additionally, in the MPC protocols we discuss, the prover will create secret shares of a value  $x$  by having each party sample their share of  $x$  from their random tape. If  $x$  must be a uniform random value, this is sufficient, but to create a sharing of a given value, the prover additionally computes a “delta value” or “offset value” to correct the sharing:

$$\Delta x = x - \sum_{i=1}^N x^{(i)} .$$

The value  $\Delta x$  is public, and the first party updates their share with it:  $x^{(1)} = x^{(1)} + \Delta x$ .

## 2 The BN++ Zero-Knowledge Proof System

In this section, we first review the Baum-Nof zero-knowledge protocol, as presented in [BN20], then give a series of four optimizations, all aimed at reducing the proof size that make up the BN++ proof system.

$\kappa$	Security parameter
$[x]$	The set $\{1, \dots, x\}$
$N$	Number of parties
$\tau$	Number of parallel repetitions
$e$	Index of repetition $e = 1, \dots, \tau$
$i$	Index of party $P_i$ , $i = 1, \dots, N$
$\bar{i}, \bar{i}_e$	Index of unopened party, in repetition $e$
$a^{(i)}$	Party $i$ 's share of $a$ ; sharing is additive $a = \sum_{i=1}^N a^{(i)}$
$\mathcal{C}, C$	An arithmetic circuit $\mathcal{C}$ with $C$ multiplication gates
$M$	Number of executed multiplication checks
$\mathbb{F}$	Field where $\mathcal{C}$ is defined
$\mathbb{K}$	Extension field of $\mathbb{F}$ , output of RMFE encoding
$k$	Number of $\mathbb{F}$ -elements encoded to one $\mathbb{K}$ -element
$\phi$	RMFE encoding function $\phi : \mathbb{F}^k \rightarrow \mathbb{K}$
$\psi$	RMFE decoding function $\psi : \mathbb{K} \rightarrow \mathbb{F}^k$

Table 1: Frequently used notation.

## 2.1 The Baum-Nof Zero-Knowledge Proof

In [BN20], Baum and Nof presented a zero-knowledge argument of knowledge based on the MPC-in-the-head approach by Ishai et al. [IKOS07], following Katz, Kolesnikov and Wang [KKW18] by using an MPC protocol with pre-processing. The protocol in [BN20] uses standard multiplication triples (or Beaver triples [Bea92]), but instead of revealing the pre-processing phase of some iterations in a cut-and-choose fashion, they instead use a common technique from traditional MPC and show that a triple is correct by “sacrificing” another one. A random challenge is jointly generated, or provided by a special verifier party, which is natural in the context of MPCitH proofs.

This procedure is repeated below, checking a triple  $(x, y, z)$  using a second triple  $(a, b, c)$ .

1. The verifier provides a random  $\epsilon \in \mathbb{F}$ .
2. The parties locally set  $\alpha^{(i)} = \epsilon \cdot x^{(i)} + a^{(i)}, \beta^{(i)} = y^{(i)} + b^{(i)}$ .
3. The parties open  $\alpha$  and  $\beta$  by broadcasting their shares.
4. The parties locally set  $v^{(i)} = \epsilon \cdot z^{(i)} - c^{(i)} + \alpha \cdot b^{(i)} + \beta \cdot a^{(i)} - \alpha \cdot \beta$ .
5. The parties open  $v$  by broadcasting their shares and check that  $v = 0$ .

In the context of an MPCitH proof, the first triple  $(x, y, z)$  comes from the circuit evaluation (with shares of  $z$  being injected by the prover), and the second triple  $(a, b, c)$  is a random triple, whose main job is to hide the first triple in the broadcast values  $\alpha$  and  $\beta$ .

A signature scheme based on the BN proof protocol [BN20] is depicted in Figure 1. The circuit is assumed to be a one-way function, with input  $\text{sk}$  and output  $\text{pk}$  ( $\text{sk}$  is the signing key and  $\text{pk}$  is the public key, consisting of the circuit output  $\text{ct}$  and optional public parameters). For soundness, the base protocol is repeated  $\tau$  times in parallel. Several hash functions: `Commit`,  $H_1$  and  $H_2$  are required; as well as the two pseudorandom generators: `Expand` and `ExpandTape`. All of these functions can be instantiated with the SHAKE128 extendable output

function (or SHAKE256 for larger security levels), with different constants added for domain separation. The helper function `Sample( $t$ )` samples elements from a random tape  $t$  that was output by `ExpandTape`, keeping track of the current position on the tape. The way the per-party seeds in each parallel repetition are derived from a root seed in a binary tree is the same technique (originating in [KKW18]) used in Picnic, Banquet, Rainier and other MPCitH-based proofs.

## 2.2 Optimized Proof Size Overview

Here we quantify the proof size of the BN protocol without optimizations, then summarize how each of the optimizations that make up BN++, described in this section, reduce the proof size.

The total proof size of the protocol in Figure 1 is

$$3\kappa + \tau \cdot (4\kappa + \kappa \cdot \lceil \log_2(N) \rceil + \mathcal{M}(C)),$$

where  $\mathcal{M}(C) = 5C \log_2(|\mathbb{F}|)$  is the size of the checking protocol to ensure the multiplications are correct. Referring to Figure 1, the five field elements per multiplication are  $(\Delta c_{e,\ell}, \Delta z_{e,\ell}, \alpha_{e,\ell}^{(i_e)}, \beta_{e,\ell}^{(i_e)}, v_{e,\ell}^{(i_e)})$ . Also note that this assumes that the output of  $\mathcal{C}$  is  $\kappa$  bits (as will be the case for the signature schemes we consider). After Optimization 1, the proof size will be

$$3\kappa + \tau \cdot (3\kappa + \kappa \cdot \lceil \log_2(N) \rceil + \mathcal{M}(C)). \quad (1)$$

Optimizations 2,3,4 will each reduce  $\mathcal{M}(C)$ , as summarized in Table 2. In Table 2 we also include  $\mathcal{M}(C)$  for BN++RMFE, the protocol where RMFEs are used with BN++ to improve performance in small fields. The sizes of the additional parameters for BN++ and BN++RMFE are given in Section 3.3. As mentioned above, optimizations 1 and 2 were first used in [GHS<sup>+</sup>21].

Proof protocol	$\mathcal{M}(C)$
BN	$5C \log_2( \mathbb{F} )$
BN + Opt. 2	$4C \log_2( \mathbb{F} )$
BN + Opt. 2,3	$3C \log_2( \mathbb{F} )$
BN++ (BN + Opt. 2,3,4)	$(2C + 1) \log_2( \mathbb{F} )$
BN++-Simple Lifting	$C \log_2( \mathbb{F} ) + (C + 1) \log_2( \mathbb{K} )$
BN++RMFE	$(2 \lceil C/k \rceil + 1) \cdot (\log_2( \mathbb{K} ))$

Table 2: Summary of proof sizes after successive optimizations building up to the BN++ zero-knowledge proofs. Here we give the size of the checking protocol,  $\mathcal{M}(C)$ , only, for the full proof size see Equation (1).

## 2.3 Optimization 1: Removing the Output Broadcast

In [BN20], the authors describe an optimization that uses a random linear combination of all output shares  $\text{ct}_e^{(i)}$  to reduce communication to a single field element. We now give an optimization that saves all communication with regards to the output shares.

Sign(sk, msg):

**Phase 1: Committing to the seeds and views of the parties.**

- 1: Sample a random salt  $\text{salt} \xleftarrow{\$} \{0, 1\}^{2\kappa}$ .
- 2: **for** each parallel repetition  $e$  **do**
- 3:   Sample a root seed:  $\text{seed}_e \xleftarrow{\$} \{0, 1\}^\kappa$ .
- 4:   Derive  $\text{seed}_e^{(1)}, \dots, \text{seed}_e^{(N)}$  as leaves of binary tree from  $\text{seed}_e$ .
- 5:   **for** each party  $i$  **do**
- 6:     Commit to seed:  $\text{com}_e^{(i)} \leftarrow \text{Commit}(\text{salt}, e, i, \text{seed}_e^{(i)})$ .
- 7:     Expand random tape:  $\text{tape}_e^{(i)} \leftarrow \text{ExpandTape}(\text{salt}, e, i, \text{seed}_e^{(i)})$ .
- 8:     Sample witness share:  $\text{sk}_e^{(i)} \leftarrow \text{Sample}(\text{tape}_e^{(i)})$ .
- 9:     Compute witness offset:  $\Delta\text{sk}_e \leftarrow \text{sk} - \sum_i \text{sk}_e^{(i)}$ .
- 10:    Adjust first share:  $\text{sk}_e^{(1)} \leftarrow \text{sk}_e^{(1)} + \Delta\text{sk}_e$ .
- 11:    **for** each multiplication gate with index  $\ell \in [C]$  **do**
- 12:     For each party  $i$ , set  $(a_{e,\ell}^{(i)}, b_{e,\ell}^{(i)}, c_{e,\ell}^{(i)}) \leftarrow \text{Sample}(\text{tape}_e^{(i)})$ .
- 13:     Compute  $a_{e,\ell} = \sum_{i=1}^N a_{e,\ell}^{(i)}$ ,  $b_{e,\ell} = \sum_{i=1}^N b_{e,\ell}^{(i)}$ ,  $c_{e,\ell} = \sum_{i=1}^N c_{e,\ell}^{(i)}$ .
- 14:     Compute offset  $\Delta c_{e,\ell} = a_{e,\ell} \cdot b_{e,\ell} - c_{e,\ell}$ .
- 15:     Adjust first share:  $c_{e,\ell}^{(1)} \leftarrow c_{e,\ell}^{(1)} + \Delta c_{e,\ell}$
- 16:    **for** each gate  $g$  in  $\mathcal{C}$  with index  $\ell$  **do**
- 17:     **if**  $g$  is an addition gate with inputs  $(x, y)$  **then**
- 18:       The parties locally compute the output share:
- 19:        $z^{(i)} = x^{(i)} + y^{(i)}$
- 20:     **if**  $g$  is a multiplication gate with inputs  $(x_{e,\ell}, y_{e,\ell})$  **then**
- 21:       Compute output shares  $z_{e,\ell}^{(i)} = \text{Sample}(\text{tape}_e^{(i)})$ .
- 22:       Compute offset  $\Delta z_{e,\ell} = x_{e,\ell} \cdot y_{e,\ell} - \sum_{i=1}^N z_{e,\ell}^{(i)}$ .
- 23:       Adjust first share  $z_{e,\ell}^{(1)} \leftarrow z_{e,\ell}^{(1)} + \Delta z_{e,\ell}$ .
- 24:    Let  $\text{ct}_e^{(i)}$  be the output shares of online simulation.
- 25: Set  $\sigma_1$  to:
- 26:  $(\text{salt}, ((\text{com}_e^{(i)})_{i \in [N]}, (\text{ct}_e^{(i)})_{i \in [N]}, \Delta\text{sk}_e, (\Delta c_{e,\ell}, \Delta z_{e,\ell})_{\ell \in [C]})_{e \in [\tau]}$ .

**Phase 2: Challenging the checking protocol.**

- 1: Compute challenge hash:  $h_1 \leftarrow H_1(\text{salt}, \text{msg}, \sigma_1)$ .
- 2: Expand hash:  $(\epsilon_{e,\ell})_{\ell \in [C], e \in [\tau]} \leftarrow \text{Expand}(h_1)$  where  $\epsilon_{e,\ell} \in \mathbb{F}$ .

**Phase 3: Commit to simulation of checking protocol.**

- 1: **for** each multiplication gate with index  $\ell \in [C]$  **do**
- Simulate the triple checking protocol as defined above. Let  $\alpha_{e,\ell}^{(i)}$  and  $\beta_{e,\ell}^{(i)}$
- 2:   be the two broadcast values and let  $v_{e,\ell}^{(i)}$  be the output of the checking protocol, for all parties  $i \in [N]$ .
- 3: Set  $\sigma_2 \leftarrow (\text{salt}, (((\alpha_{e,\ell}^{(i)}, \beta_{e,\ell}^{(i)}, v_{e,\ell}^{(i)})_{i \in [N]})_{\ell \in [C]})_{e \in [\tau]}$ .

**Phase 4: Challenging the views of the MPC protocol.**

- 1: Compute challenge hash:  $h_2 \leftarrow H_2(h_1, \sigma_2)$ .
- 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 repetition  $e$  **do**
- 2:    $\text{seeds}_e \leftarrow \{\log_2(N) \text{ nodes to compute } \text{seed}_{e,i} \text{ for } i \in [N] \setminus \{\bar{i}_e\}\}$ .
- 3:   Output  $\sigma \leftarrow (\text{salt}, h_1, h_2, (\text{seeds}_e, \text{com}_e^{(\bar{i}_e)}, \Delta\text{sk}_e, \text{ct}_e^{(\bar{i}_e)}, (\Delta c_{e,\ell}, \Delta z_{e,\ell}, \alpha_{e,\ell}^{(\bar{i}_e)}, \beta_{e,\ell}^{(\bar{i}_e)}, v_{e,\ell}^{(\bar{i}_e)})_{\ell \in [C]})_{e \in [\tau]}$ .

Figure 1: Signature scheme based on the Baum-Nof MPCitH proof protocol [BN20], signing algorithm.

Note that in the setting of the proof, the output of the circuit  $\text{ct}$  is public, so it is known to the verifier. Furthermore, from the  $N - 1$  seeds revealed to the verifier, the verifier can recompute  $\text{ct}_e^{(i)}$  for all opened parties of a repetition and then recalculate the missing share

$$\text{ct}_e^{(\bar{i}_e)} = \text{ct} - \sum_{\substack{i=1 \\ i \neq \bar{i}_e}}^N \text{ct}_e^{(i)} .$$

Intuitively, since all  $\text{ct}_e^{(i)}$  are input to the hash function in Phase 2, this ensures that the verifier is using the same shares of  $\text{ct}$  as the prover. This optimization saves including the unopened party's output broadcast message  $\text{ct}_e^{(\bar{i}_e)}$  in the proof, saving one output value per repetition. The proof size formula is given in Equation (1). The concrete size of the savings depends on the output size of  $\mathcal{C}$ . For LowMC at security level L1, this amounts to 129 bits, and once our other parameters are chosen we use 18 repetitions so the total savings is about 290 bytes.

## 2.4 Optimization 2: Removing the Final Checking Protocol Broadcast

In [BN20], the authors describe another optimization that again uses a random linear combination of all  $C$  output shares of the multiplications check  $v_{e,\ell}^{(i)}$  to reduce communication to a single field element. We now show how to reduce this communication entirely.

As before, the verifier knows the plain output of the multiplications check, as an accepting check should output  $v_{e,\ell} = 0$ . Again,  $N - 1$  seeds are revealed to the verifier, meaning he can recompute  $v_{e,\ell}^{(i)}$  for all opened parties of a repetition and then re-calculate the missing share

$$v_{e,\ell}^{(\bar{i}_e)} = 0 - \sum_{\substack{i=1 \\ i \neq \bar{i}_e}}^N v_{e,\ell}^{(i)} .$$

This optimization saves including  $v_{e,\ell}^{(\bar{i}_e)}$  in the signature, saving another field element per multiplication gate compared to the base protocol (or one field element per repetition compared to the optimization in [BN20]).

After Optimization 2,  $\mathcal{M}(C) = 4C \log_2(|\mathbb{F}|)$ , and together with Equation (1) we get the total proof size.

## 2.5 Optimization 3: Remove the Broadcast of $\beta$

In the standard triple verification procedure from Section 2.1 the parties need to broadcast both  $\alpha$  and  $\beta$ . This is needed in general, so that one can verify arbitrary triple pairs using this procedure. But consider two triples that are related as follows:  $(x, y, z)$  and  $(a, -y, c)$ . Due to the structure of the proof, we can easily create the second multiplication triple by the parties locally computing  $-y^{(i)} = -(y^{(i)})$ , randomly sampling  $a^{(i)}$  and  $c^{(i)}$  locally and the prover then injects  $\Delta c$  as before to fix the shares of  $c$ . If we now execute the same checking

protocol, we have  $\beta = y + (-y)$ , so  $\beta = 0$ , removing the need for a broadcast. One can also think of this optimization as performing *circuit-dependent* preprocessing. For simplicity we'll compute  $\beta = y - b$ , so that  $b = y$  (rather than  $-y$ ) and modify the computation of  $v$  accordingly.

1. The verifier provides a random challenge  $\epsilon \in \mathbb{F}$ .
2. The parties locally set  $\alpha^{(i)} = \epsilon x^{(i)} + a^{(i)}$ .
3. The parties open  $\alpha$  by broadcasting their shares.
4. The parties locally set  $v^{(i)} = \alpha y^{(i)} - \epsilon z^{(i)} - c^{(i)}$ .
5. The parties open  $v$  by broadcasting their shares and output ACC iff  $v = 0$ .

The security of this protocol can be analyzed in a similar fashion to [BN20, Lemma 2], however, we will add an additional optimization step in the following section, then analyze the security of the resulting protocol.

After combining optimizations 1, 2 and 3, we must communicate the values  $(\alpha_{e,\ell}^{(i)}, \Delta c_{e,\ell}, \Delta z_{e,\ell})$  for each of the  $C$  multiplication gates, so the size of the checking protocol is  $\mathcal{M}(C) = 3C \log_2(|\mathbb{F}|)$  which we can plug into the proof size formula given in Equation (1).

## 2.6 Optimization 4: Dot-Product Checking

In this optimization we again modify the checking protocol. We observe that the protocol is proving that *both*  $(x, y, z)$  and  $(a, b, c)$  are valid multiplication triples. However, for correctness of the circuit, we only need to prove that  $(x, y, z)$  is a valid multiplication triple, and for  $(a, b, c)$  we only require that  $a$  and  $b$  are random, so that  $x$  and  $y$  are hidden in the computation of  $\alpha$  and  $\beta$ , and that  $c$  is correlated to  $(a, b)$  in a way that allows us to create a checking protocol. We can batch verification of all  $C$  triples,  $(x_\ell, y_\ell, z_\ell)_{\ell=1}^C$  given a random dot product,  $((a_\ell, b_\ell)_{\ell=1}^C, c)$  where  $c = \sum_{\ell=1}^C a_\ell b_\ell$ , as follows.

For simplicity we start our description of the protocol from Optimization 3 (when  $b = y$ ), but this optimization can be applied independently (i.e., when  $b \neq y$ ). Here all  $\ell = 1, \dots, C$  multiplication gates are checked at once, the input is  $(x_\ell, y_\ell, z_\ell)_{\ell=1}^C$  and  $((a_\ell, b_\ell)_{\ell=1}^C, c)$ , values that are secret-shared amongst the parties.

1. The verifier provides a random challenge  $(\epsilon_1, \dots, \epsilon_C) \in \mathbb{F}^C$ .
2. The parties locally set  $\alpha_\ell^{(i)} = \epsilon x_\ell^{(i)} + a_\ell^{(i)}$ .
3. The parties open  $(\alpha_1, \dots, \alpha_C)$  by broadcasting their shares.
4. The parties locally set

$$\begin{aligned} v^{(i)} &= \epsilon_1 z_1^{(i)} - \alpha_1 b_1^{(i)} \\ &+ \dots \\ &+ \epsilon_C \cdot z_C^{(i)} - \alpha_C \cdot b_C^{(i)} \\ &- c^{(i)} \end{aligned}$$

Note that each of the  $C$  lines above is basically one instance of the non-batched multiplication check, except that the  $C$  values of  $c_\ell = a_\ell b_\ell$  are summed together on the last line.

5. The parties open  $v$  by broadcasting  $v^{(i)}$  and output ACC iff  $v = 0$ .

The security of this protocol can be analyzed with a combination of related ideas from [BN20, Lemma 2], and [dOT21, Lemma 4.1]. When compared to the multiplication check protocol in [dOT21], our protocol uses independent random challenges  $\epsilon_i$ , rather than  $(R, R^2, \dots, R^{C-1})$  for a single random  $R \in \mathbb{F}$ . Therefore, we can apply the Schwartz-Zippel lemma to a degree 1, multivariate polynomial, rather than a degree  $C - 1$  univariate polynomial. This decreases the soundness error by a factor of  $C - 1$ , which is especially significant when the field size is small.<sup>3</sup>

First we recall the multivariate version of the Schwartz-Zippel lemma [DL77, Zip79, Sch80].

**Lemma 1** (General Schwartz-Zippel lemma). *Let  $P(x_1, \dots, x_n)$  be a non-zero polynomial of  $n$  variables with total degree  $d$  over  $\mathbb{F}$ . For any finite subset  $S$  of  $\mathbb{F}$ , with at least  $d$  elements in it,*

$$\Pr[(r_1, \dots, r_n) \xleftarrow{\$} S^n : P(r_1, \dots, r_n) = 0] \leq \frac{d}{|S|}.$$

The total degree of  $P$  is the largest sum of the exponents in a term of  $P$ . For example,  $P(x_1, x_2) = 4x_1^2x_2^3 + x_1x_2^2 + 1$  has degree 5, and  $P(x_1, x_2) = 7x_1 + 3x_2$  has degree 1.

Now we prove security of our dot-product based checking protocol, which also covers Optimization 3 a special case when  $C = 1$ .

**Lemma 2.** *If the secret-shared input  $(x_\ell, y_\ell, z_\ell)_{\ell=1}^C$  contains an incorrect multiplication triple, or if the shares of  $((a_\ell, y_\ell)_{\ell=1}^C, c)$  form an incorrect dot product, then the parties output ACC in the sub-protocol with probability at most  $1/|\mathbb{F}|$ .*

*Proof.* Let  $\Delta_{z_\ell} = z_\ell - x_\ell \cdot y_\ell$  and  $\Delta_c = c - \sum a_\ell \cdot y_\ell$ . If the parties output ACC, then  $v = 0$ , leading to:

$$\begin{aligned} v &= -c + \sum \epsilon_\ell \cdot z_\ell - \alpha_\ell \cdot y_\ell \\ &= -c + \sum \epsilon_\ell \cdot (\Delta_{z_\ell} + x_\ell \cdot y_\ell) - (\epsilon_\ell \cdot x_\ell - a_\ell) \cdot y_\ell \\ &= -c + \sum \epsilon_\ell \Delta_{z_\ell} + \epsilon_\ell \cdot z_\ell - \epsilon_\ell \cdot z_\ell + a_\ell \cdot y_\ell \\ &= -c + \sum a_\ell \cdot y_\ell + \sum \epsilon_\ell \Delta_{z_\ell} \\ &= \epsilon_1 \Delta_{z_1} + \epsilon_2 \Delta_{z_2} + \dots + \epsilon_C \Delta_{z_C} - \Delta_c = 0 \end{aligned}$$

Define a multivariate polynomial

$$Q(X_1, \dots, X_C) = X_1 \cdot \Delta_{z_1} + \dots + X_C \cdot \Delta_{z_C} - \Delta_c$$

in  $\mathbb{F}[X_1, \dots, X_C]$  and note that  $v = 0$  iff  $Q(\epsilon_1, \dots, \epsilon_C) = 0$ . When  $Q$  is the zero polynomial, then all  $\Delta_{z_\ell} = 0$  and  $\Delta_c = 0$  are zero, implying  $z_\ell = x_\ell \cdot y_\ell$  and  $c = \sum a_\ell \cdot b_\ell$ , so  $v = 0$  is the correct result.

<sup>3</sup>Whether Limbo [dOT21] can use independent challenges and also have improved analysis along the lines of our Lemma 2 is an interesting question. For the simpler MultCheck protocol in [dOT21], it appears possible, but the final Limbo ZK proof uses the CompressedMC checking protocol, which seems to rely on univariate polynomials and a single challenge.

In the case of a cheating prover,  $Q$  is nonzero, and by the multivariate version of the Schwartz-Zippel lemma (see Lemma 1), the probability that  $Q(\epsilon_1, \dots, \epsilon_C) = 0$  is at most  $1/|\mathbb{F}|$ , since  $Q$  has total degree 1 and  $(\epsilon_1, \dots, \epsilon_C)$  is chosen uniformly at random.  $\square$

**Privacy** We show that the above checking protocol is  $(N - 1)$ -private, by defining a simulator  $\mathcal{S}$  that obtains shares  $\{z_\ell^{(i)}, b_\ell^{(i)}, c_\ell^{(i)}\}_{\ell \in [C]}$  for all parties  $i \in [N]$  except for one, denoted  $\bar{i}$ . Simulator  $\mathcal{S}$  chooses  $\alpha_\ell^{(i)}$  at random for all parties. Then using the shares  $(z_\ell^{(i)}, b_\ell^{(i)}, c_\ell^{(i)})$ ,  $\mathcal{S}$  computes  $v^{(i)}$  honestly for the  $N - 1$  parties excluding party  $i$ . For party  $\bar{i}$ 's share, since  $\mathcal{S}$  knows that  $v = 0$  in an accepting run of the protocol, it can solve for  $v^{(\bar{i})} = 0 - \sum_{i \neq \bar{i}} v^{(i)}$  exactly as in Optimization 2. Now we argue that  $\mathcal{S}$ 's output is correctly distributed. First we note that in a real execution  $\alpha_\ell = \epsilon_\ell x_\ell + a_\ell$  is uniformly distributed in  $\mathbb{F}$  since  $a_\ell$  is a uniform random value, and the simulated value of  $\alpha_\ell$  is also uniformly random in  $\mathbb{F}$ . Next, the  $N - 1$  shares of  $v$  computed honestly are correctly distributed, and there is only one choice for party  $\bar{i}$ 's share that makes the parties accept, making it the same in both simulated and real transcripts.

**BN++ Signature size** With optimizations 1,2,3 and 4, we must communicate  $(\Delta z_{e,\ell}, \alpha_{e,\ell}^{(i)})$  once for each of the  $C$  multiplication gates, and one  $\Delta c_e$  per repetition. The size is

$$\mathcal{M}(C) = (2C + 1) \cdot \log_2(|\mathbb{F}|),$$

for the checking protocol, which together with Equation (1) gives the total proof size.

## 2.7 BN++: New Protocol with all Optimizations

In Figure 2 we describe the new signing algorithm using optimizations 1, 2, 3, and 4. In Figure 3 we describe the corresponding verification algorithm. The setup, keypair, and hash functions are the same as in our description of the original BN protocol in Section 2.1.

### 2.7.1 Soundness

Now we analyze the soundness error of the protocol, informally, in order to select the parameters  $\tau$  (number of parallel repetitions) and  $N$  (number of parties per MPC execution). A more formal analysis is given in Lemma 5, where we give an extractor and analyze its success probability to prove unforgeability of the BN++ signature scheme. Since the non-interactive BN++ proof is a canonical 5-round protocol constructed with the Fiat-Shamir transform, we can apply the existing analysis in [KZ20a, §4.1], similar to Banquet [BdK<sup>+</sup>21] (though it is seven rounds) and Rainier [DKR<sup>+</sup>21].

We provide an attack strategy that minimizes the attack cost, where we cheat  $\tau_1$  times for the first challenge and  $\tau_2 = \tau - \tau_1$  times for the second challenge. To cheat in the first challenge the attacker must pass the multiplication check, and this happens with probability  $1/|\mathbb{F}|$ , by Lemma 2. For the remaining  $\tau_2$  instances he must cheat in the MPC computation of one party, and hope that the selected party remains unopened.

Sign(sk, msg):

**Phase 1: Committing to the seeds and views of the parties.**

- 1: Sample a random salt:  $\text{salt} \xleftarrow{\$} \{0, 1\}^{2\kappa}$ .
- 2: **for** each parallel repetition  $e$  **do**
- 3:   Sample a root seed:  $\text{seed}_e \xleftarrow{\$} \{0, 1\}^\kappa$ .
- 4:   Derive  $\text{seed}_e^{(1)}, \dots, \text{seed}_e^{(N)}$  as leaves of a binary tree from  $\text{seed}_e$ .
- 5:   **for** each party  $i$  **do**
- 6:     Commit to seed:  $\text{com}_e^{(i)} \leftarrow \text{Commit}(\text{salt}, e, i, \text{seed}_e^{(i)})$ .
- 7:     Expand random tape:  $\text{tape}_e^{(i)} \leftarrow \text{ExpandTape}(\text{salt}, e, i, \text{seed}_e^{(i)})$
- 8:     Sample witness share:  $\text{sk}_e^{(i)} \leftarrow \text{Sample}(\text{tape}_e^{(i)})$ .
- 9:     Compute witness offset:  $\Delta \text{sk}_e \leftarrow \text{sk} - \sum_i \text{sk}_e^{(i)}$ .
- 10:    Adjust first share:  $\text{sk}_e^{(1)} \leftarrow \text{sk}_e^{(1)} + \Delta \text{sk}_e$ .
- 11:    **for** each gate  $g$  in  $\mathcal{C}$  with index  $\ell$  **do**
- 12:     **if**  $g$  is an addition gate with inputs  $(x, y)$  **then**
- 13:       Party  $i$  locally computes the output share  $z^{(i)} = x^{(i)} + y^{(i)}$ .
- 14:     **if**  $g$  is a multiplication gate with inputs  $(x_{e,\ell}, y_{e,\ell})$  **then**
- 15:       Compute output shares  $z_{e,\ell}^{(i)} = \text{Sample}(\text{tape}_e^{(i)})$ .
- 16:       Compute offset  $\Delta z_{e,\ell} = x_{e,\ell} \cdot y_{e,\ell} - \sum_{i=1}^N z_{e,\ell}^{(i)}$ .
- 17:       Adjust first share  $z_{e,\ell}^{(1)} \leftarrow z_{e,\ell}^{(1)} + \Delta z_{e,\ell}$ .
- 18:       For each party  $i$ , set  $a_{e,\ell}^{(i)} \leftarrow \text{Sample}(\text{tape}_e^{(i)})$ .
- 19:       Compute  $a_{e,\ell} = \sum_{i=1}^N a_{e,\ell}^{(i)}$  and set  $b_{e,\ell} = y_{e,\ell}$ .
- 20:     Compute  $c_e^{(i)} \leftarrow \text{Sample}(\text{tape}_e^{(i)})$ .
- 21:     Compute offset  $\Delta c_e = \left( \sum_{\ell=1}^{|\mathcal{C}|} a_{e,\ell} \cdot b_{e,\ell} \right) - c_e$ .
- 22:     Adjust first share:  $c_e^{(1)} \leftarrow c_e^{(1)} + \Delta c_e$
- 23:     Let  $\text{ct}_e^{(i)}$  be the output shares of online simulation.
- 24:    Set  $\sigma_1 \leftarrow (\text{salt}, ((\text{com}_e^{(i)}, \text{ct}_e^{(i)})_{i \in [N]}, \Delta \text{sk}_e, \Delta c_e, (\Delta z_{e,\ell})_{\ell \in [C]})_{e \in [\tau]}$ .

**Phase 2: Challenging the checking protocol.**

- 1: Compute challenge hash:  $h_1 \leftarrow H_1(\text{salt}, \text{msg}, \sigma_1)$ .
- 2: Expand hash:  $((\epsilon_{e,\ell})_{\ell \in [C]})_{e \in [\tau]} \leftarrow \text{Expand}(h_1)$  where  $\epsilon_{e,\ell} \in \mathbb{F}$ .

**Phase 3: Commit to simulation of the checking protocol.**

- 1: **for** each repetition  $e$  **do**  
    For the set of multiplication gates in  $\mathcal{C}$ , simulate the triple checking protocol
- 2:   as defined in §2.6 for all parties with challenge  $(\epsilon_{e,\ell})_{\ell \in [C]}$ . The inputs are  $(x_{e,\ell}^{(i)}, y_{e,\ell}^{(i)}, z_{e,\ell}^{(i)}, a_{e,\ell}^{(i)}, b_{e,\ell}^{(i)}, c_e^{(i)})$ , and let  $\alpha_{e,\ell}^{(i)}$  and  $v_e^{(i)}$  be the broadcast values.
- 3:   Set  $\sigma_2 \leftarrow (\text{salt}, ((\alpha_{e,\ell}^{(i)})_{\ell \in [C]}, v_e^{(i)})_{i \in [N]})_{e \in [\tau]}$ .

**Phase 4: Challenging the views of the MPC protocol.**

- 1: Compute challenge hash:  $h_2 \leftarrow H_2(h_1, \sigma_2)$ .
- 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 MPC and checking protocols.**

- 1: **for** each repetition  $e$  **do**
- 2:    $\text{seeds}_e \leftarrow \{\log_2(N)$  nodes to compute  $\text{seed}_e^{(i)}$  for  $i \in [N] \setminus \{\bar{i}_e\}\}$ .
- 3:   Output  $\sigma \leftarrow (\text{salt}, h_1, h_2, (\text{seeds}_e, \text{com}_e^{(\bar{i}_e)}, \Delta \text{sk}_e, \Delta c_e, (\Delta z_{e,\ell}, \alpha_{e,\ell}^{(\bar{i}_e)})_{\ell \in [C]})_{e \in [\tau]}$ .

Figure 2: BN++ signature scheme, signing algorithm.

Verify(pk, msg,  $\sigma$ ) :

- 1: Parse  $\sigma$  as (salt,  $h_1$ ,  $h_2$ , (seeds $_e$ , com $_e^{(\bar{i}_e)}$ ,  $\Delta sk_e$ ,  $\Delta c_e$ , ( $\Delta z_{e,\ell}$ ,  $\alpha_{e,\ell}^{(\bar{i}_e)}$ ) $_{\ell \in [C]}$ ) $_{e \in [\tau]}$ ).
- 2: Expand hashes: ( $\epsilon_{e,\ell}$ ) $_{e \in [\tau], \ell \in [C]} \leftarrow \text{Expand}(h_1)$ , and ( $\bar{i}_e$ ) $_{e \in [\tau]} \leftarrow \text{Expand}(h_2)$ .
- 3: **for** each repetition  $e$  **do**
- 4:     Use seeds $_e$  to recompute seed $_e^{(i)}$  for  $i \in [N] \setminus \bar{i}_e$ .
- 5:     **for** each party  $i \in [N] \setminus \bar{i}_e$  **do**
- 6:         Recompute com $_e^{(i)} \leftarrow \text{Commit}(\text{salt}, e, i, \text{seed}_e^{(i)})$ ,
- 7:         tape $_e^{(i)} \leftarrow \text{ExpandTape}(\text{salt}, e, i, \text{seed}_e^{(i)})$ , and
- 8:         sk $_e^{(i)} \leftarrow \text{Sample}(\text{tape}_e^{(i)})$ .
- 9:         **if**  $i = 1$  **then** adjust first share: sk $_e^{(i)} \leftarrow \text{sk}_e^{(i)} + \Delta sk_e$ .
- 10:         **for** each gate  $g$  in  $\mathcal{C}$  with index  $\ell$  **do**
- 11:             **if**  $g$  is an addition gate with inputs  $(x^{(i)}, y^{(i)})$  **then**
- 12:                 Compute the output share  $z^{(i)} = x^{(i)} + y^{(i)}$
- 13:             **if**  $g$  is a mult. gate, with inputs  $(x_{e,\ell}^{(i)}, y_{e,\ell}^{(i)})$  **then**
- 14:                 Compute output share  $z_{e,\ell}^{(i)} = \text{Sample}(\text{tape}_e^{(i)})$ .
- 15:                 **if**  $i = 1$  **then**
- 16:                     Adjust first share  $z_{e,\ell}^{(i)} \leftarrow z_{e,\ell}^{(i)} + \Delta z_{e,\ell}$ .
- 17:                     Set  $a_{e,\ell}^{(i)} \leftarrow \text{Sample}(\text{tape}_e^{(i)})$ , and  $b_{e,\ell}^{(i)} = y_{e,\ell}^{(i)}$
- 18:                 Set  $c_e^{(i)} \leftarrow \text{Sample}(\text{tape}_e^{(i)})$
- 19:                 **if**  $i = 1$  **then** adjust first share  $c_e^{(i)} \leftarrow c_e^{(i)} + \Delta c_e$ .
- 20:                 Let ct $_e^{(i)}$  be party  $i$ 's share of the circuit output.
- 21:             Compute ct $_e^{(\bar{i}_e)} = \text{ct} - \sum_{i \neq \bar{i}_e} \text{ct}_e^{(i)}$
- 22: Set  $\sigma_1 \leftarrow (\text{salt}, ((\text{com}_e^{(i)}, \text{ct}_e^{(i)})_{i \in [N]}, \Delta sk_e, \Delta c_e, (\Delta z_{e,\ell})_{\ell \in [C]})_{e \in [\tau]}$ .
- 23: Set  $h'_1 = H_1(\text{salt}, \text{msg}, \sigma_1)$
- 24: **for** each repetition  $e$  **do**
- 25:     **for** each party  $i \in [N] \setminus \bar{i}_e$  **do**
- 26:         For the set of multiplication gates in  $\mathcal{C}$ , simulate the triple verification procedure as defined in §2.6 for party  $i$  with challenge ( $\epsilon_{e,\ell}$ ) $_{\ell \in [C]}$ . The inputs are  $(x_{e,\ell}^{(i)}, y_{e,\ell}^{(i)}, z_{e,\ell}^{(i)}, a_{e,\ell}^{(i)}, b_{e,\ell}^{(i)}, c_e^{(i)})$ , and let  $\alpha_{e,\ell}^{(i)}$  and  $v_e^{(i)}$  be the broadcast values.
- 27: Compute  $v_e^{(\bar{i}_e)} = 0 - \sum_{i \neq \bar{i}_e} v_e^{(i)}$
- 28: Set  $\sigma_2 \leftarrow (\text{salt}, ((\alpha_{e,\ell}^{(i)})_{\ell \in [C]}, v_e^{(i)})_{i \in [N]})_{e \in [\tau]}$ .
- 29: Set  $h'_2 = H_2(h'_1, \sigma_2)$ .
- 30: Output accept iff  $h'_1 \stackrel{?}{=} h_1$  and  $h'_2 \stackrel{?}{=} h_2$ .

Figure 3: BN++ signature scheme, verification algorithm.

The cost of the attack is given by

$$\text{Cost}(\kappa, N, \tau) = \frac{1}{\text{SPMF}(\tau, \tau_1, 1/|\mathbb{F}|)} + N^{\tau_2},$$

where 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 cost gives the optimal attack

$$\tau_1 = \arg \min_{0 \leq \tau' \leq \tau} \frac{1}{\text{SPMF}(\tau, \tau', 1/|\mathbb{F}|)} + N^{\tau - \tau'}.$$

To select secure parameters, we fix  $\kappa$  and  $N$  and  $\mathbb{F}$ , then increase  $\tau$  until the cost of the best attack exceeds  $2^\kappa$ . A script implementing this formula was used to generate the concrete parameters given in the next section. Similarly to other MPCitH proofs, BN++ offers a size-speed tradeoff, as increasing  $N$  allows us to reduce  $\tau$ , which in turn reduces the proof size, but requires more computation due to the increased number of parties.

### 3 Handling Small Fields Efficiently

The BN++ protocol performs well for circuits defined over large fields. For example, the Rain block cipher is the basis for the Rainier signature scheme [DKR<sup>+</sup>21], and its nonlinear operations are defined over  $\text{GF}(2^{128})$ . The proof size of BN++ with Rain is slightly smaller than Rainier, and the implementation of BN++ is arguably simpler, not requiring polynomial interpolation or arithmetic.

However, for the LowMC OWF used in Picnic, we have 516 binary AND gates, and signatures with BN++ are estimated to be about 62 KB ( $\tau = 202, N = 256$ ). The number of parallel repetitions  $\tau$  is very high because the soundness of the multiplication check in each repetition is only 1/2. This compares to 12.5 KB for the Picnic3 parameters (based on the KKW [KKW18] proof system). As a second example, for AES-128 we must check 200 multiplications in  $\mathbb{F}_{2^8}$ , and BN++ proofs are 20 KB ( $\tau = 35, N = 256$ ), compared to 13.2 KB for Banquet ( $\tau = 21, N = 255$ ). Thus the BN++ proof system is not competitive with existing solutions when the field size is small.

In this section we address this limitation with four strategies. The first is called *simple lifting*, which lifts values from the small field to a larger field before the multiplication check, increasing soundness. Alternatively, we can repeat the multiplication check multiple times per repetition, leading to nearly identical sizes when compared to simple lifting. We can also combine the two strategies, in order to target a specific preferred field in which to perform most of the field arithmetic, which can vary depending on the platform. The third strategy is to lift groups of triples to a larger field using a reverse multiplication-friendly embedding. This strategy can also be combined with the previous two.

Our last approach to lifting uses alternative, but equivalent circuit representations over a larger field. This is often the case for S-box designs, which can

Binary LowMC (516 ANDs):	61.7 KB without lifting 26.6 KB lifting to $\mathbb{K} = \text{GF}(2^8)$
$\text{GF}(2^3)$ LowMC (172 mults):	22.7 KB without lifting 14.0 KB lifting to $\mathbb{K} = \text{GF}(2^{12})$
AES-128 (200 $\mathbb{F}_{2^8}$ mults):	20.2 KB without lifting 19.5 KB lifting to $\mathbb{K} = \text{GF}(2^{16})$

Table 3: BN++ proof size estimates for LowMC and AES with the simple lifting strategy, with  $N = 256$  parties.

be evaluated as a binary circuit, but also have an efficient representation over a larger field.

### 3.1 Simple Lifting

For BN++, we can prove circuits over small fields by first lifting them to a larger field, as was done in Banquet [BdK<sup>+</sup>21] and Limbo [dOT21]. This lifting takes a value in  $\mathbb{F}$  and lifts it to an extension field  $\mathbb{K}$  using an injective homomorphism. In particular, we execute the circuit over  $\mathbb{F}$ , getting the shares of the multiplication gates, and then lift these shares to  $\mathbb{K}$  for the multiplication checking protocol. (The parties can lift their shares using only local operations.) The challenge  $\epsilon$  must be in  $\mathbb{K}$  to improve soundness, which means that  $a_\ell$  must also be in  $\mathbb{K}$  to ensure that  $x_\ell$  is hidden in the computation of the public value  $\alpha_\ell$ . Because of this, the dot product triple  $(\mathbf{a}, \mathbf{b}, c)$  is also in  $\mathbb{K}$ . Recall that in BN++ we must communicate  $(\Delta z_{e,\ell}, \alpha_{e,\ell}^{(i)})$  once per gate and  $\Delta c_e$  once per repetition. From the discussion above,  $\Delta z_{e,\ell} \in \mathbb{F}$ ,  $\alpha_{e,\ell}^{(i)}$  and  $\Delta c_e$  are in  $\mathbb{K}$ .

Then the proof/signature size formula given in Equation (1) has

$$\mathcal{M}(C) = C(\log_2(|\mathbb{F}|) + \log_2(|\mathbb{K}|)) + \log_2(|\mathbb{K}|).$$

We give some examples when  $N = 256$ , in Table 3 to illustrate the size improvements possible with this lifting strategy.

The improvements in the table make sense intuitively, when looking at the “rate” of the lift, a measure of the overhead. In binary LowMC we lift each bit to 8 bits, for a rate of 8. When moving from  $\mathbb{F}_2$  to  $\mathbb{F}_{2^3}$  each group of 3 bits are lifted to one  $\text{GF}(2^3)$  value, for a rate of 1, giving nearly a 3x decrease in proof size. When we lift from  $\text{GF}(2^3)$  to  $\text{GF}(2^{12})$  (rate 4) we get the best signature size, about 1.5 KB more than Picnic3.

The AES example shows that lifting is not always effective: when starting from  $\mathbb{F}_{2^8}$  lifting to the next smallest field,  $\mathbb{F}_{2^{16}}$  we see only a small decrease in signature size, and lifting to larger fields increases signature size. As the number of parties increases, lifting to large fields can show a somewhat greater improvement.

### 3.2 Multiple Checks Per Repetition

Another approach to handle small fields is possible in BN++ since we can instead repeat the checking protocol  $M$  times per repetition, at the same communication

cost. As a lifting strategy, this can be seen as lifting to  $\mathbb{F}^M$  (the direct product of  $\mathbb{F}$ ,  $M$  times) with rate  $M$ . For example, instead of lifting from  $\mathbb{F} = \mathbb{F}_2$  to  $\mathbb{K} = \mathbb{F}_{2^M}$ , we can instead do  $M$  checks over  $\mathbb{F}_2$  for the same communication cost. The advantage of this is that the checking arithmetic is in  $\mathbb{F}_2$ , which may be faster than any extension field depending on the platform (and when the base field is not  $\mathbb{F}_2$ , smaller fields generally have faster arithmetic).

The description of BN++ in Figure 2 corresponds to the case  $M = 1$ . When  $M > 1$ , the checking protocol of Section 2.6 is repeated  $M$  times. This requires the parties compute  $M$  check values per repetition  $v_{e,m}$  each using a fresh challenge and broadcast values of the parties; this protocol can be seen as being done with vectors over  $\mathbb{F}^M$ . The input triples  $(x_\ell, y_\ell, z_\ell)_{\ell \in [C]}$  remain in  $\mathbb{F}$ , only the random dot-product  $((a_\ell, b_\ell)_{\ell \in [C]}, c)$  is in  $\mathbb{F}^M$ . Therefore the proof size with  $M$  checks is given in Equation (1) with

$$\mathcal{M}(C) = C \log_2(|\mathbb{F}|) + M(C + 1) \log_2(|\mathbb{F}|) .$$

The optimal values of  $M$  for the circuits in Table 3 are  $M = 8$  for binary LowMC,  $M = 4$  for  $\text{GF}(2^3)$  LowMC, and  $M = 2$  for AES-128; since these make  $|\mathbb{F}^M| = |\mathbb{K}|$  and we arrive at identical proof sizes.

### 3.3 Lifting with RMFEs

When looking at an example of the simple lifting approach, say from  $\text{GF}(2)$  to  $\text{GF}(2^8)$  this means we take one bit and lift it to an 8-bit field, then run the checking protocol the larger field. The *rate* of this lift is 8, since 8 bits must be communicated in place of one. While this does help soundness, the lift is somewhat trivial, and resulting rate is high. A natural question is if we can do better.

A *reverse multiplication friendly embedding* allows us to encode multiple bits into a field extension with better rate. For example, the  $(3, 5)_2$ -RMFE allows us to lift a batch of 3 elements of  $\mathbb{F}_2$  into  $\mathbb{F}_{2^5}$ , with rate  $5/3 = 1.6$  and there exist RMFEs with larger base and extension fields. Once we have encoded groups of bits  $(x_1, x_2, x_3)$  and  $(y_1, y_2, y_3)$  we can multiply the encoded values in  $\mathbb{F}_{2^5}$ , then decode to get the three ANDs  $(x_1 \cdot y_1, x_2 \cdot y_2, x_3 \cdot y_3)$ . Importantly, the encoding and decoding operations are linear, meaning the parties can compute the maps on their shares locally to obtain shares of the encoded value. There are somewhat strict limitations on the arithmetic operations one can perform on encoded values such that the decoded values are correct, but we show that the sacrificing check is possible.

#### 3.3.1 RMFE Preliminaries

First a definition, adapted from [CCXY18].

**Definition 3.** Let  $k$  and  $m$  be positive integers and  $q$  be a prime power that defines the field  $\mathbb{F}_q$ . Define a pair of mappings:

- $\phi : (\mathbb{F}_q)^k \rightarrow \mathbb{F}_{q^m}$  that maps vectors over the base field to the extension field, and
- $\psi : \mathbb{F}_{q^m} \rightarrow (\mathbb{F}_q)^k$  which does the reverse.

We say that  $(\phi, \psi)$  is a *reverse multiplication friendly embedding*, denoted  $(k, m)_q$ -RMFE, if

1.  $\phi$  and  $\psi$  are  $\mathbb{F}_q$ -linear, and
2. For any pair of vectors  $\mathbf{x}, \mathbf{y} \in (\mathbb{F}_q)^k$ , we have

$$\mathbf{x} * \mathbf{y} = \psi(\phi(\mathbf{x}) \cdot \phi(\mathbf{y}))$$

where  $*$  denotes component-wise multiplication.

The *rate* of the RMFE is  $m/k$ . When the sizes are clear from the context, we refer to  $\mathbb{F}_q$  as  $\mathbb{F}$  and  $\mathbb{F}_{q^m}$  as  $\mathbb{K}$ , so that  $\phi : \mathbb{F}^k \rightarrow \mathbb{K}$  and  $\psi : \mathbb{K} \rightarrow \mathbb{F}^k$ .

We call out an important limitation of the RMFE property. Suppose  $X = \phi(\mathbf{x})$  and  $Y = \phi(\mathbf{y})$ , and  $\mathbf{x} * \mathbf{y} = \mathbf{z}$ . It is not guaranteed that  $XY = \phi(\mathbf{z})$ , only that  $\psi(XY) = \mathbf{z}$ . One way to see why is to note that the representation of  $\mathbf{z}$  in  $\mathbb{K}$  is redundant; there are many  $Z' \in \mathbb{K}$  such that  $\psi(Z') = \mathbf{z}$ , and only one possible result for  $\phi(\mathbf{z})$ .

**RMFE Constructions** We experimented with two constructions of RMFEs from [CCXY18]. The first is based on interpolation codes [CCXY18, Lemma 4], has the lowest rate, and gives the shortest proof sizes in **BN++** (for the circuits over small fields). The construction gives us a  $(k, 2k - 1)_q$ -RMFE, with  $1 \leq k \leq q + 1$ . The constraint on  $k$  means the size of  $\mathbb{K}$  is limited by  $q$  (limiting the soundness of the multiplication check, when  $q$  is small, where we need it the most). We can then use the concatenation construction of [CCXY18, Lemma 5] to get larger  $\mathbb{K}$ , but at the expense of higher rate. Other options exist, e.g., starting from  $\mathbb{F}_2$ , but the rates of other constructions we investigated are higher (and thus led to larger proof sizes), starting around rate 3.

A construction with improved rate and similarly sized  $\mathbb{K}$  would immediately give shorter signatures (e.g., in the case of LowMC, a rate 1.6 RMFE would reduce signature sizes by about 1 KB). Unfortunately [CCXY18, Lemma 4] is optimal.<sup>4</sup> That said, how we make use of this optimal RMFE may not be optimal, and an interesting question is whether the flexibility of the MPC-in-the-head paradigm can be used to further reduce communication. In Section 5.1.1 we show some of the concrete RMFEs we investigated, along with resulting signature size estimates.

In terms of implementation, once the RMFE parameters are fixed, we can derive matrix representations for  $\phi$  and  $\psi$ , then encoding and decoding can each be done with a matrix multiplication. This is mainly done for efficiency, since the construction of [CCXY18, Lemma 4] is using linear interpolation codes. We can use this linearity to construct a matrix representation of the RMFE encoding step by evaluating the RMFE encoding function on the unit basis vectors (and repeat the same for the decoding step). This only has to be done once for a set of RMFE parameters and both simplifies and speeds up the performance of the RMFE operations.

---

<sup>4</sup>Personal communication from Ignacio Cascudo; co-author of [CCXY18].

### 3.3.2 BN++RMFE: Multiplication Checking with RMFEs

Recall the checking protocol of Section 2.6. The input is  $(x_\ell, y_\ell, z_\ell)_{\ell=1}^C$  such that  $x_\ell \cdot y_\ell = z_\ell$ , and  $((a_\ell)_{\ell=1}^C, c)$  such that  $c = \sum a_\ell \cdot y_\ell$ . All of these elements are over  $\mathbb{F}$ , and assume for the moment that the number of triples is a multiple of  $k$ , so that we have  $C/k$  groups of elements to map to the extension field  $\mathbb{K}$ .

**Prover operations** The main change when lifting with an RMFE is how the prover prepares the inputs to the checking protocol. Once the inputs are prepared, the protocol happens over the extension field  $\mathbb{K}$ . We try to use capital letter variables for elements in  $\mathbb{K}$  and lower case variables for elements in  $\mathbb{F}$ .

1. The prover executes the circuit normally over  $\mathbb{F}$ , to obtain the multiplication gate inputs/outputs  $(x_\ell, y_\ell, z_\ell)$  for  $\ell = 1, \dots, C$ . We group these into vectors from  $\mathbb{F}^k$ , denoted  $(\mathbf{x}_j, \mathbf{y}_j, \mathbf{z}_j)$  for  $j = 1, \dots, C/k$ .
2. Then the prover computes  $X_j = \phi(\mathbf{x}_j)$  and  $Y_j = \phi(\mathbf{y}_j)$ , then  $Z_j = X_j \cdot Y_j$ . The parties have shares of  $\mathbf{x}_j$  and  $\mathbf{y}_j$  and can compute shares of  $X_j$  and  $Y_j$  on their own because  $\phi$  is linear. In BN++ the prover provides  $\Delta z_j$  in order to inject the result of the multiplication gate, but in BN++RMFE the prover will inject shares of  $Z_j$ . More precisely, the prover samples shares of  $Z_j$  from the random tapes, and adjusts the first party's share with  $\Delta Z_j = Z_j - \sum_{i=1}^N Z_j^{(i)}$ . From their shares of  $Z_j$  the parties can obtain their shares of  $\mathbf{z}_j$  as  $\psi(Z_j^{(i)})$ , which they need for the computation of the circuit.
3. The prover then chooses  $A_j$  at random from  $\mathbb{K}$ , sets  $B_j = Y_j$  then computes  $S = \sum A_j Y_j$  and injects the sharing of  $S$ , by computing  $\Delta S \in K$  (as she did for  $\Delta Z$ ).

Now all inputs for dot-product check are in  $\mathbb{K}$ , and the protocol proceeds as in Section 2.6 but the computation of  $\mathcal{A} = \epsilon_j X_j + A_j$  and  $V$  happen over  $\mathbb{K}$ :

1. The verifier provides a random challenge  $\epsilon \in \mathbb{K}^{C/k}$ .
2. The parties locally set  $\mathcal{A}_j^{(i)} = \epsilon_j \cdot X_j^{(i)} + A_j^{(i)}$ .
3. The parties open  $(\mathcal{A}_1, \dots, \mathcal{A}_{C/k})$  by broadcasting their shares.
4. Party  $i$  locally computes

$$V^{(i)} = -S^{(i)} + \sum_{j=1}^{C/k} \left( \mathcal{A}_j Y_j^{(i)} - \epsilon_j Z_j^{(i)} \right)$$

5. The parties open  $V$  by broadcasting  $V^{(i)}$  and output ACC iff  $V = 0$

**Lemma 4.** *If the secret shared input  $(x_i, y_i, z_i)_{i=1}^C$  contains an incorrect multiplication triple, or the shares of  $((A_i, Y_i)_{i=1}^{C/k}, S)$  form an incorrect dot product, then the parties output ACC in the sub-protocol with probability at most  $1/|\mathbb{K}|$ .*

*Proof.* Lemma 2 ensures that  $(X_j, Y_j, Z_j)$  are all valid multiplication triples in  $\mathbb{K}$ , and that  $(A_j, Y_j, S)$  is a valid dot product in  $\mathbb{K}$  with probability  $1/|\mathbb{K}|$ . We must show that this implies  $(x_i, y_i, z_i)$  are all valid multiplication triples in  $\mathbb{F}$ .

If  $(X, Y, Z)$  is a valid multiplication triple in  $\mathbb{K}$ , where  $X = \phi(\mathbf{x})$  and  $Y = \phi(\mathbf{y})$  then  $\psi(X \cdot Y) = \mathbf{x} * \mathbf{y} = \mathbf{z}$  by the RMFE property of the maps  $(\phi, \psi)$  required by Definition 3. Thus given that  $(X, Y, Z)$  is a valid triple in  $\mathbb{K}$  ensures that  $(\mathbf{x}, \mathbf{y}, \mathbf{z})$  are valid in  $\mathbb{F}$  with probability 1, so the result follows.  $\square$

Note that it is important to compute (shares of)  $\mathbf{z}$  with  $\psi$  as we do in the protocol, since it is not guaranteed that  $\phi(\mathbf{x}) \cdot \phi(\mathbf{y}) = \phi(\mathbf{z})$  even though  $\mathbf{z} = \mathbf{x} * \mathbf{y}$ . This is one of the limitations of RMFEs.

**Batching inputs to RMFE encoding** When  $C/k$  does not divide evenly, we encode  $\lceil C/k \rceil$  groups of elements from  $\mathbb{F}$  where the last group is padded with zeros. In the interactive MPC setting, it can be difficult to batch all  $C$  multiplications arbitrarily, since only some triples may be ready at a given time. For example consider a block cipher like AES or LowMC: the multiplication gate inputs at round  $i$  depend on the multiplication gate outputs of round  $i - 1$ . Fortunately in the MPCitH setting, the prover can first compute the entire circuit to get all triples. Then the checking protocol is subsequently run on the entire batch, giving full flexibility over how the RMFE encoding is done. In MPCitH we could even potentially have a  $(C, m)_q$ -RMFE, so that all multiplication triples are encoded as a single batch into one very large field element.

**Proof size** For a circuit defined over  $\mathbb{F}$  with  $C$  multiplication gates, using an RMFE from  $\mathbb{F}^k \rightarrow \mathbb{K}$ , the size of a BN++ proof is given by the formula:

$$3\kappa + \tau \cdot (3\kappa + \kappa \cdot \lceil \log_2(N) \rceil) + (2 \lceil C/k \rceil + 1) \log_2(|\mathbb{K}|).$$

We must communicate  $(\Delta Z_{e,j}, \mathcal{A}_{e,j}^{(i)})$  for each of the  $C/k$  batched multiplication gates, and  $\Delta S_e$  once per repetition.

The size formula for BN++RMFE is difficult to compare to BN++ as it depends on the concrete choice of RMFE (for the parameters  $\mathbb{K}$  and  $k$ ). We can compare the number of bits communicated by the checking protocol for each bit of soundness to asymptotically evaluate the impact of using an RMFE on the size (see Appendix C.1). Since the rate  $m/k$  is constant (as shown in [CCXY18, Theorem 5]) we can argue that BN++RMFE is asymptotically better for nontrivial RMFEs when compared to simple lifting or no lifting. However, in our experience with concrete small circuits, experimentation with different parameter choices is still helpful, as different RMFEs can lead to significant changes in size.

While we are explicit about the combination of RMFE and BN++ in this section, one could also think about them as separate steps: use the RMFE to *lift* a circuit over a small field to a circuit over a larger field, and use BN++ to prove the validity of a circuit executed over a larger field  $\mathbb{K}$ . This view will simplify security analysis, as we will not need to treat BN++RMFE as a separate protocol.

**Inverse checking with RMFEs** It is also possible to check inversion gates with BN++, which is required for the AES circuit (with the efficient representation of S-boxes over  $\mathbb{F}_{2^8}$ , as discussed in Section 3.4). The cost is increased by an additional  $\tau \lceil C/k \rceil$  elements of  $\mathbb{K}$ . Since Helium gives much shorter proofs for AES, we only describe inverse checking briefly in Appendix C.2.

## 3.4 Alternative Circuit Representations

While not strictly a type of lifting, for some circuits it is possible to find an equivalent representation over larger fields. We give two concrete examples. The first, for the AES S-box is fairly well-known, it can be represented over  $\mathbb{F}_2$  or  $\mathbb{F}_{2^8}$ . The second, for LowMC is similar but novel.

### 3.4.1 The AES S-Box

The best known binary circuit representation of the AES S-box optimized for lowest AND count has 32 AND gates, given by [BP09]. Using this binary circuit would mean that a prover would have to lift each of the 32 multiplication triples over  $\mathbb{F}_2$  to a larger field. However, the AES S-box can also be built using an inversion over  $\mathbb{F}_{2^8}$ , which can be implemented using 4 multiplications and 7 squarings (a linear operation in binary fields) in  $\mathbb{F}_{2^8}$ . In fact, if the zero to zero transition is forbidden, a trick originating in BBQ [dDOS19], its cost can be reduced to a single multiplication in  $\mathbb{F}_{2^8}$ . These alternate representations allow us to start in a larger field (boosting soundness) and allow for more efficient RMFEs constructions. They also reduce the number of multiplication triples from 32 to 4 or even 1 (but over a larger field).

### 3.4.2 Alternative Representation of the LowMC S-box

The 3-bit S-box in LowMC is defined (over  $\mathbb{F}_2$ ) as

$$S(a, b, c) = (a + bc, a + b + ac, a + b + c + ab). \quad (2)$$

We now show an alternate representation of the S-box that uses a single multiplication in  $\mathbb{F}_{2^3} \cong \mathbb{F}_2[X]/(X^3 + X + 1)$  instead of 3 multiplications in  $\mathbb{F}_2$ .

---

#### Algorithm 1

LowMC S-box with a multiplication in  $\mathbb{F}_{2^3} \cong \mathbb{F}_2[X]/(X^3 + X + 1)$

---

**Input:**  $a, b, c$

**Output:**  $S(a, b, c)$

$t_1 \leftarrow aX^2 + bX + c$	▷ interpret as element of $\mathbb{F}_{2^3}$
$t_2 \leftarrow (a + b)X^2 + aX + c$	▷ interpret as element of $\mathbb{F}_{2^3}$
$t \leftarrow t_1 \cdot t_2$	▷ multiplication in $\mathbb{F}_{2^3}$
$dX^2 + eX + f \leftarrow t$	▷ extract coefficients
<b>return</b> $(d, d + e, f)$	▷ final linear transformation

---

It can easily be verified that Equation (2) and Algorithm 1 are equivalent by enumeration of the eight possible inputs. Furthermore, and importantly for its use in an MPC protocol using linear secret sharing, all of the operations except the multiplication are linear and therefore do not require any communication between the parties.

The benefit of this representation is that we can use the checking protocol over the field  $\mathbb{F}_{2^3}$  rather than  $\mathbb{F}_2$  which increases the soundness of the multiplication check. In this sense we get a “free lift” to the larger field, when compared to protocols like Banquet [BdK<sup>+</sup>21] and Limbo [dOT21] which use simple lifting as described above.

Additionally, when we use an RMFE in BN++ and Helium to further increase the field size (and soundness), the constructions available when the starting field

is  $\mathbb{F}_{2^3}$  have significantly better rate, about 1.88, than when the starting field is  $\mathbb{F}_2$ , where the best known rate is about 2.83 (to arrive at a similar sized  $\mathbb{K}$ ).

## 4 The Helium Proof System

In this section we describe the Helium proof system. The proof system is a variant of Rainier, used in [DKR<sup>+</sup>21] (which itself is a simplified version of Banquet [BdK<sup>+</sup>21]), with some of the lifting techniques of Section 3 applied. To recap Rainier briefly, the prover injects the results of multiplications as in BN++, however, the checking protocol uses polynomials, and works well for checking inverses, namely  $s_i \cdot t_i = 1$  for  $i \in [C]$ , since the S-box in the AES and Rain one-way functions are field inverses. The parties interpolate the points  $(i, s_i)$  and  $(i, t_i)$  to get polynomials  $S(X)$  and  $T(X)$  in  $\mathbb{F}[X]$ . Then the prover computes  $P = S \cdot T$  and communicates shares of  $P$  to the parties. Note that  $P(i) = S(i)T(i) = s_i \cdot t_i = 1$  for  $i \in [C]$ , so that the parties can use these points  $(i, 1)$  with an additional  $C - 2$  points from the prover to recover  $P$  by interpolation. When we must check generic multiplication triples  $(s_i, t_i, p_i)$ , the parties use their shares of  $p_i$  plus an  $C - 2$  additional points from the prover to interpolate  $P$ . As the prover is injecting the multiplication of  $ST$ , the final step is to check that  $ST = P$ . This is done by checking  $P(R) - S(R)T(R) = 0$  for a randomly selected  $R \in \mathbb{F}$ , an idea from traditional MPC, originating in [BFO12].

In Banquet and Rainier, given shares of polynomials  $(S, T, P)$  and a public challenge  $R$  from the verifier, the parties can locally compute shares of  $(S(R), T(R), P(R))$ , then broadcast to open them. To prevent the opening step from leaking information about  $S$  and  $T$ , an additional random point  $(s_r, t_r, p_r)$  is included by the prover. Conversely, one could keep the shares of  $(S(R), T(R), P(R))$  secret, and use a checking protocol like in Section 2.1.<sup>5</sup>

The advantage of making the triple  $(S(R), T(R), P(R))$  public is that the checking step is very simple, and communication-efficient. On the other hand, we must randomize  $S$  and  $T$ , and the random values must be from the field where  $R$  is sampled from. When lifting to a large field, this means that  $P$ , and the  $C$  points the prover must communicate are from the large field. By contrast, if  $S$  and  $T$  are not randomized, then  $P$  may be communicated in the small field, and the lifting can be delayed until just before the polynomials are evaluated at  $R$ . However, this option requires makes the checking protocol nontrivial since it must be done with shares. Another consideration is that to use  $M$  checks requires  $M$  random values when the  $M$  triples are public, increasing the degree of  $P$ , whereas when the triple remains shared this is not a concern.

We estimated signature sizes for both of the above options, and found that the savings of sending the  $C$  points of  $P$  from the small field justify the more complex checking protocol. Also, since the checking can be done in a very large field, and the values depending on the circuit are communicated in the small field, we found  $M = 1$  to be preferable (i.e., checking a single triple in a field of size  $O(2^\kappa)$  was more efficient than checking  $M > 1$  triples in a smaller field). For the final check we use the checking protocol from BN++.

<sup>5</sup>This was also done in concurrent work [FJR22], together with simple lifting,  $M$ -times checking and our dot-product optimization from Section 2.6.

**Helium-AES** Before we can use Helium with AES-128, we must make one change: since  $C = 200$  and  $\deg P = 398$ , we cannot represent  $P$  by points for interpolation, as a list  $(i, P(i))$  since  $i \in \mathbb{F}_{2^8}$  limits us to 256 values (polynomials of degree 255). Below we give some general approaches to working around this, since it will often be the case that  $2C > |\mathbb{F}|$  when  $\mathbb{F}$  is small. For AES we found it most efficient to use two polynomials  $P_1$  and  $P_2$ , each encoding half of the triples. This doubles our multiplication checking costs, but keeps the  $O(C)$  part of the prover’s communication in  $\mathbb{F}_{2^8}$ .

**Helium-LowMC** For LowMC the base field  $\mathbb{F}_{2^3}$  is too small to contain  $P$  (as in AES). In this case, rather than using more than one polynomial, we use the  $(2, 3)_8$ -RMFE to replace our 172 triples in  $\mathbb{F}_{2^3}$  with 86 triples in  $\mathbb{F}_{2^9}$ . As in Section 3.3, the parties compute  $\phi(\mathbf{s})$  to get an element of the larger field  $\mathbb{F}_{2^9}$ , then the prover injects shares of the product in  $\mathbb{F}_{2^9}$ . At this point the parties have shares of the triples in  $\mathbb{F}_{2^9}$ , and the protocol proceeds, and we apply a second lifting step, this time simple lifting, going from  $\mathbb{F}_{2^9}$  to  $\mathbb{K} = \mathbb{F}_{2^{144}}$  when we compute the final triple  $(S(R), T(R), P(R))$ .

**Helium with arbitrary circuits** Generalizing to arbitrary circuits, we have multiple options when using Helium. The main restriction is that the degree of the polynomials needs to be less than the field size, so we have enough points for the interpolation step, where the critical polynomial is  $P$ , of degree  $2C - 2$ . We now present three options to handle this:

1. Using a simple lift from  $\mathbb{F}$  to an extension field  $\mathbb{F}'$  where  $\deg(P) < |\mathbb{F}'|$ .
2. Partitioning the multiplication triples into batches so that for each batch and its corresponding polynomial  $P_i$ ,  $\deg(P_i) < |\mathbb{F}|$ . This is a generalization of the Helium + AES case. (Here  $\mathbb{F}' = \mathbb{F}$ .)
3. Using an RMFE to lift batches of multiplication triples to  $\mathbb{F}'$ , where  $\deg(P) < |\mathbb{F}'|$ . In contrast to the simple lifting protocol, this also reduces the number of multiplications in  $\mathbb{F}'$ . This is a generalization of the Helium + LowMC case.

We note that this step is separate from the checking protocols, and like in the above examples, the evaluation of the polynomial and the multiplication check can happen in an even larger extension field to boost the soundness of these checks.

As a concrete example, we look to circuits for AES with higher security levels. Taking the options presented in BBQ [dDOS19], we execute two instances of AES-192 (AES-256) with a shared key-schedule, resulting in 416 (500) multiplications. For both security levels, splitting the multiplications in 4 batches of 104 (125) results in  $\deg(P_i) < |\mathbb{F}_{2^8}|$ , again allowing us to keep the  $O(C)$  part of the prover’s communication in  $\mathbb{F}_{2^8}$ .

**Signature Size** The signature size is dominated by the  $C$  multiplication outputs  $\Delta z$  in  $\mathbb{F}$  and the  $C - 1$  values in  $\mathbb{F}'$  to communicate  $P \in \mathbb{F}'[X]$ . The integer  $n_p$  denotes the number of polynomials used, for LowMC  $n_p = 1$  and for AES-128  $n_p = 2$ . The signature size can be estimated with the formula:

$$6\kappa + \tau (\lceil \log_2(N) \rceil \kappa + 3\kappa + C|\mathbb{F}| + (C - n_p)|\mathbb{F}'| + (n_p + 1)|\mathbb{K}|) .$$

**Soundness and parameters** We need to choose the number of parallel repetitions  $\tau$  for a given number of parties  $N$ , so that the noninteractive Helium protocol provides  $\kappa$  bits of soundness. As a seven-round protocol, we can use the general formula of [BdK<sup>+</sup>21, §6.1] to bound the cost of an attack on  $\tau$  repetitions where the attacker’s strategy is to guess the first challenge in  $\tau_1$  repetitions, the second in  $\tau_2$  and the third in  $\tau_3$  repetitions (such that  $\tau = \tau_1 + \tau_2 + \tau_3$ ). Then for a given  $N$  we find  $\tau$  such that the cost of an attack is at least  $2^\kappa$ , regardless of the strategy. The probability  $p_1$ , that an adversary guesses the first challenge correctly is  $(2L - 2)/|\mathbb{K}|$  (by the univariate Schwartz-Zippel lemma) where  $L = \lceil C/n_p \rceil$ . For the second challenge,  $p_2 = 1/|\mathbb{K}|$  and the third is  $p_3 = 1/N$ . These probabilities, together with the formula in [BdK<sup>+</sup>21, §6.1] allows us to compute the cost of an attack given  $(N, \tau, \tau_1, \tau_2, \tau_3)$ . Using a script we exhaustively check all strategies to find  $\tau$ . Intuitively, since we choose  $|\mathbb{K}| > 2^\kappa$ , the limiting factor for the soundness is  $1/N$ , so we end up with  $(N, \tau)$  such that  $N^\tau \geq 2^\kappa$ .

#### 4.1 Helium Protocol Description

In Figures 4 and 5 we describe the signing algorithm of Helium, and Figure 6 describes verification. The hash functions, and helper functions to expand random tapes and seeds are as in Section 2.7. We describe Helium for circuits with addition and multiplication gates in  $\mathbb{F}$ , but explain how inversions in  $\mathbb{F}$  can be checked instead, using nearly the same protocol. This is useful as the AES circuit uses additions and inversions, while the LowMC circuit uses additions and multiplications. For inversion gates  $(s, t, 1)$ , the prover injects shares of  $t$ , and parties use these shares to interpolate the polynomial  $T$ . When interpolating the polynomial  $P$ , the public value 1 is used in place of shares of the multiplication output  $z$ , for  $P(i)$  for  $i \in [C]$ . Thus the communication costs and checking steps are the same. We remark that for circuits using inversion gates, we assume that key-generation takes care that no zero inputs to inverse gates exist, as proposed by [dDOS19].

## 5 Using BN<sup>++</sup> and Helium in Signature Schemes

In this section we demonstrate the performance of using BN<sup>++</sup> and Helium to construct signature schemes with different OWFs. All benchmarks given in the paper were computed on an Intel Xeon W-2133 @ 3.6 GHz. We prove that BN<sup>++</sup> is a secure signature in the ROM, assuming the OWF used for key generation is secure in Section 5.5. To keep the scope of this section manageable we restrict our attention to the 128-bit security level (NIST level L1) and focus on three particular one-way function designs used in previous works: AES-128, LowMC [ARS<sup>+</sup>15] and Rain [DKR<sup>+</sup>21].

**Other OWFs.** In [DGH<sup>+</sup>21], Dinur et al. give a OWF construction based on mixed moduli circuits. Simplified, their construction first performs a matrix multiplication modulo 2, interprets the output as elements of  $\mathbb{Z}_3$  and performs a second matrix multiplication modulo 3. This mixture of different fields provides resistance against cryptanalytic attacks while keeping the resulting construction simple and efficient. However, it is not obvious how to support an efficient

**Sign(sk, msg): Phase 1: Committing to the seeds, the execution views and interpolated polynomials of the parties.**

- 1: Sample a random salt:  $\text{salt} \xleftarrow{\$} \{0, 1\}^{2\kappa}$ .
- 2: **for** each parallel repetition  $e$  **do**
- 3:     Sample a root seed:  $\text{seed}_e \xleftarrow{\$} \{0, 1\}^\kappa$ .
- 4:     Derive  $\text{seed}_e^{(1)}, \dots, \text{seed}_e^{(N)}$  as leaves of a binary tree from  $\text{seed}_e$ .
- 5:     **for** each party  $i$  **do**
- 6:         Commit to seed:  $\text{com}_e^{(i)} \leftarrow \text{Commit}(\text{salt}, e, i, \text{seed}_e^{(i)})$ .
- 7:         Expand random tape:  $\text{tape}_e^{(i)} \leftarrow \text{ExpandTape}(\text{salt}, e, i, \text{seed}_e^{(i)})$
- 8:         Sample witness share:  $\text{sk}_e^{(i)} \leftarrow \text{Sample}(\text{tape}_e^{(i)})$ .
- 9:     Compute witness offset:  $\Delta\text{sk}_e \leftarrow \text{sk} - \sum_i \text{sk}_e^{(i)}$ .
- 10:     Adjust first share:  $\text{sk}_e^{(1)} \leftarrow \text{sk}_e^{(1)} + \Delta\text{sk}_e$ .
- 11:     **for** each gate  $g$  in  $\mathcal{C}$  with index  $\ell$  **do**
- 12:         **if**  $g$  is an addition gate with inputs  $(x, y)$  **then**
- 13:             Party  $i$  locally computes the output share  $z^{(i)} = x^{(i)} + y^{(i)}$ .
- 14:         **if**  $g$  is a multiplication gate with inputs  $(x_{e,\ell}, y_{e,\ell})$  **then**
- 15:             Compute output shares  $z_{e,\ell}^{(i)} = \text{Sample}(\text{tape}_e^{(i)})$ .
- 16:             Compute offset  $\Delta z_{e,\ell} = x_{e,\ell} \cdot y_{e,\ell} - \sum_{i=1}^N z_{e,\ell}^{(i)}$ .
- 17:             Adjust first share  $z_{e,\ell}^{(1)} \leftarrow z_{e,\ell}^{(1)} + \Delta z_{e,\ell}$ .
- 18:     Let  $\text{ct}_e^{(i)}$  denote each party's share of the output.
- 19:     **for** each party  $i$  **do**
- 20:         Define  $S_e^{(i)}(\ell) = x_{e,\ell}^{(i)}$  and  $T_e^{(i)}(\ell) = y_{e,\ell}^{(i)}$  for  $\ell \in [0, C-1]$
- 21:         Interpolate shares of polynomials  $S_e^{(i)}(\cdot)$  and  $T_e^{(i)}(\cdot)$  of degree  $C-1$  using the  $C$  points.
- 22:     Compute the product polynomial:  $P_e \leftarrow \sum S_e^{(i)} \cdot \sum T_e^{(i)}$ .
- 23:     **for** each party  $i$  **do**
- 24:         For  $k \in [0, C-1]$ :  $P_e^{(i)}(k) = \begin{cases} 1 & \text{if } i = 1 \\ 0 & \text{if } i \neq 1 \end{cases}$
- 25:         For  $k \in [C, 2C-2]$ , sample  $P_e^{(i)}(k) \leftarrow \text{Sample}(\text{tape}_e^{(i)})$ .
- 26:     **for**  $k \in [C, 2C-2]$  **do**
- 27:         Compute offset:  $\Delta P_e(k) = P_e(k) - \sum_i P_e^{(i)}(k)$ .
- 28:         Adjust first share:  $P_e^{(1)}(k) \leftarrow P_e^{(1)}(k) + \Delta P_e(k)$ .
- 29:     For each party  $i$ , interpolate  $P_e^{(i)}$  using the defined  $2C-2$  points.
- 30: Set  $\sigma_1 \leftarrow (\text{salt}, ((\text{com}_e^{(i)})_{i \in [N]}, (\text{ct}_e^{(i)})_{i \in [N]}), \Delta\text{sk}_e, ((\Delta z_{e,\ell})_{\ell \in [0, C-1]}, \Delta P_e(k))_{k \in [C, 2C-2]})_{e \in [\tau]}$ .

Figure 4: Helium signature scheme, Phase 1. Commitment to executions of Helium and the interpolated polynomials. We use  $e$  to index the  $\tau$  parallel repetitions,  $i$  to index the  $N$  parties, and  $\ell$  to index the  $C$  multiplications.

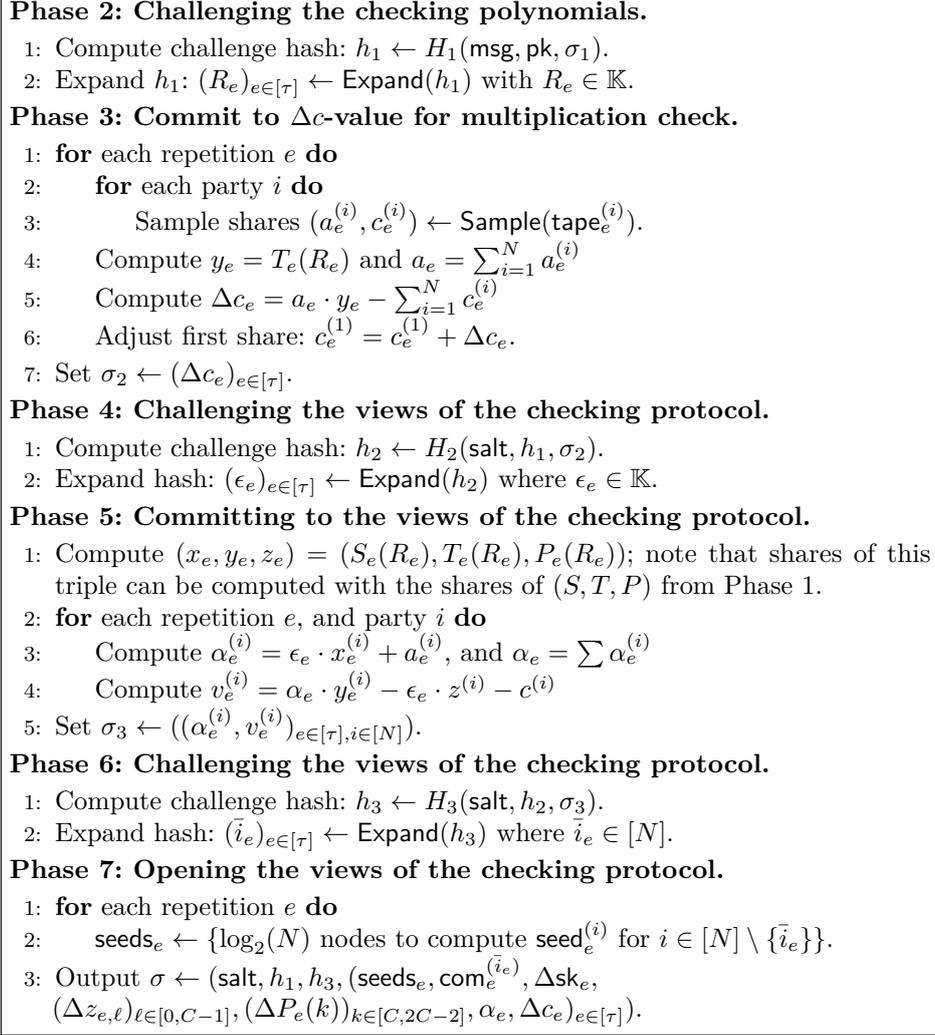


Figure 5: Helium signature Scheme, Phases 2-7. Computation of the checking protocol, challenging and opening of the views of the checking protocol.

Verify(pk, msg,  $\sigma$ ) :

- 1: Parse  $\sigma \leftarrow (\text{salt}, h_1, h_3, (\text{seeds}_e, \text{com}_e^{(\bar{i}_e)}, \Delta \text{sk}_e, (\Delta z_{e,\ell})_{\ell \in [C]}, (\Delta P_e(k))_{k \in [C; 2C-2]}, \alpha_e, \Delta c_e)_{e \in [\tau]})$ .
- 2: Set  $h_2 \leftarrow H_2(\text{salt}, h_1, (\Delta c_e)_{e \in [\tau]})$ .
- 3: Expand hashes as  $(R_e)_{e \in [\tau]} \leftarrow \text{Expand}(h_1)$ ,  $(\epsilon_e)_{e \in [\tau]} \leftarrow \text{Expand}(h_2)$  and  $(\bar{i}_e)_{e \in [\tau]} \leftarrow \text{Expand}(h_3)$ .
- 4: **for** each repetition  $e$  **do**
- 5:     Use  $\text{seeds}_e$  to recompute  $\text{seed}_e^{(i)}$  for  $i \in [N] \setminus \bar{i}_e$ .
- 6:     **for** each party  $i \in [N] \setminus \bar{i}_e$  **do**
- 7:         Recompute  $\text{com}_e^{(i)} \leftarrow \text{Commit}(\text{salt}, e, i, \text{seed}_e^{(i)})$ ,  $\text{tape}_e^{(i)} \leftarrow \text{ExpandTape}(\text{salt}, e, i, \text{seed}_e^{(i)})$  and  $\text{sk}_e^{(i)} \leftarrow \text{Sample}(\text{tape}_e^{(i)})$ .
- 8:         If  $i \stackrel{?}{=} 1$ , set  $\text{sk}_e^{(i)} \leftarrow \text{sk}_e^{(i)} + \Delta \text{sk}_e$ .
- 9:         **for** each gate  $g \in \mathcal{C}$  with index  $\ell$  **do**
- 10:             **if**  $g$  is an addition gate with inputs  $(x, y)$  **then**
- 11:                 Party  $i$  locally computes the output share  $z^{(i)} = x^{(i)} + y^{(i)}$ .
- 12:             **if**  $g$  is a mult. gate with inputs  $(x_{e,\ell}, y_{e,\ell})$  **then**
- 13:                 Compute output shares  $z_{e,\ell}^{(i)} = \text{Sample}(\text{tape}_e^{(i)})$ .
- 14:                 If  $i \stackrel{?}{=} 1$ , set  $z_{e,\ell}^{(i)} \leftarrow z_{e,\ell}^{(i)} + \Delta z_{e,\ell}$ .
- 15:             Let  $\text{ct}_e^{(i)}$  be party  $i$ 's share of the circuit output.
- 16:             Do as in Phase 1, Lines 19–21 to interpolate  $S_e^{(i)}, T_e^{(i)}$ .
- 17:             **for**  $\ell$  from 0 to  $C - 1$  **do**
- 18:                 Set  $P_e^{(i)}(\ell) = z_{e,\ell}^{(i)}$ .
- 19:             **for**  $k$  from  $C$  to  $2C - 2$  **do**
- 20:                 Sample share:  $P_e^{(i)}(k) \leftarrow \text{Sample}(\text{tape}_e^{(i)})$ .
- 21:                 If  $i \stackrel{?}{=} 1$ , set  $P_e^{(i)}(k) \leftarrow P_e^{(i)}(k) + \Delta P_e(k)$ .
- 22:             Interpolate  $P_e^{(i)}$  from the shares of the  $2C - 2$  points.
- 23:             Compute  $(x_e^{(i)}, y_e^{(i)}, z_e^{(i)}) = (S_e^{(i)}(R_e), T_e^{(i)}(R_e), P_e^{(i)}(R_e))$ .
- 24:             Compute  $c_e^{(i)} = \text{Sample}(\text{tape}_e^{(i)})$
- 25:             If  $i \stackrel{?}{=} 1$ , set  $c_e^{(i)} \leftarrow c_e^{(i)} + \Delta c_e$ .
- 26:             Compute  $\alpha_e^{(i)} = \epsilon_e \cdot x_e^{(i)} + a_e^{(i)}$  and  $v_e^{(i)} = \alpha_e \cdot y_e^{(i)} - \epsilon_e \cdot z_e^{(i)} - c_e^{(i)}$ .
- 27:             Compute missing shares  $\text{ct}_e^{(\bar{i}_e)} = \text{ct} - \sum_{i \neq \bar{i}_e} \text{ct}_e^{(i)}$ ,  $\alpha_e^{(\bar{i}_e)} \leftarrow \alpha_e - \sum_{i \neq \bar{i}_e} \alpha_e^{(i)}$ , and  $v_e^{(\bar{i}_e)} \leftarrow 0 - \sum_{i \neq \bar{i}_e} v_e^{(i)}$ .
- 28: Set  $h'_1 \leftarrow H_1 \left( \text{msg}, \text{pk}, \text{salt}, ((\text{com}_e^{(i)})_{i \in [N]}, (\text{ct}_e^{(i)})_{i \in [N]}), (\Delta \text{sk}_e, (\Delta z_{e,\ell})_{\ell \in [C]}, (\Delta P_e(k))_{k \in [C; 2C-2]})_{e \in [\tau]} \right)$ .
- 29: Set  $h'_3 \leftarrow H_3 \left( \text{salt}, h'_2, (\alpha_e^{(i)}, v_e^{(i)})_{e \in [\tau], i \in [N]} \right)$ .
- 30: Output accept iff  $h'_1 \stackrel{?}{=} h_1$  and  $h'_3 \stackrel{?}{=} h_3$ .

Figure 6: Helium verification algorithm.

conversion function from  $\mathbb{Z}_2$  to  $\mathbb{Z}_3$  in our proof systems. In [DGH<sup>+</sup>21], the authors design a custom MPC protocol based on KKW [KKW18] to use this OWF in a signature construction. We aim to investigate the suitability of mixed moduli constructions for our proof system in future work.

The LegRoast signature scheme [Bd20] is using a PRF based on the Legendre symbol or its generalized power residue form. It uses the BN [BN20] proof system internally, and proves the correctness of a linear combination of the multiplication triples to reduce the proof size. We can still apply our optimizations in Section 2.5 to save one field item per repetition and reduce their sizes further, although the gains are minor, at about 0.5 KB.

Some AES-based OWFs are introduced in [DKR<sup>+</sup>21] with the goal of reducing signature sizes. The EM one-way functions reduces the number of S-boxes required by using the block cipher in single-key Even-Mansour (EM) mode, which does not need the key schedule (this can be applied to AES and LowMC, denoted EM-AES and EM-LowMC). LSAES is a modified version of AES, which replaces the 8-bit S-boxes with 32-bit S-boxes, also defined as field inversion, but in  $\mathbb{F}_{2^{32}}$  instead of  $\mathbb{F}_{2^8}$ . From the perspective of BN++, this is like lifting four 8-bit multiplications to one 32-bit multiplication, with rate 1 – better than the rate 1.8 of the  $(5, 9)_{256}$ -RMFE we used for AES. We will give some size estimates for Helium with these OWFs.

The EM mode can also be used with LowMC, reducing the cost of computing the circuit by saving the (relatively expensive) key schedule. For Picnic3 EM-LowMC is not significantly faster due to the optimizations of [KZ20b], which make circuit evaluation costs independent of  $N$ . The circuit evaluation costs in BN++ and Helium do depend on  $N$ , and based on our benchmarking we estimate that using EM mode would reduce signing and verification times for BN++ given in Table 5 by about 7% in our current implementation when  $N = 256$ , and by about 28% for the variant of LowMC using partial S-box layers. With EM-LowMC the size optimization of Appendix B.1.2 may also be applied, reducing signature size slightly (e.g, by 574 bytes when  $N = 256$ ).

## 5.1 BN++LowMC

This section describes how the BN++ proof system can be used to instantiate a signature scheme using the LowMC block cipher as a one-way function. We use the alternate representation of the LowMC S-box that is better suited to BN++, and investigate the choice of RMFE. A description of LowMC is given in Appendix D. At a high level, the circuit is divided into rounds, and each round has a linear layer and a nonlinear layer that are applied to the state. The nonlinear layer can be either *partial* or *full*, depending on how much of the state is transformed by S-boxes. To simplify presentation, we focus on the case of a full nonlinear layer, and use the same parameters as in Picnic [CDG<sup>+</sup>20b] for our implementation and size estimates.

### 5.1.1 Concrete Parameters and Signature Size

We estimated signature sizes with multiple RMFE constructions, and show some of the options in Table 4 for the L1 security level with the full S-box layer LowMC parameters. The construction with the lowest rate is [CCXY18, Lemma 4], but the size of  $\mathbb{K}$  is somewhat limited. The construction gives us a  $(k, 2k - 1)_q$ -RMFE,

RMFE	Rate	$\mathbb{F}$	$\mathbb{K}$	$M$	$\lceil C/k \rceil$	$N$	$\tau$	Size
None	1	$\mathbb{F}_2$	$\mathbb{F}_2$	8	516	256	35	26 609
$(3, 5)_2$	1.6	$\mathbb{F}_2$	$\mathbb{F}_{2^5}$	3	172	246	25	15 258
$(30, 95)_2$	3.16	$\mathbb{F}_2$	$\mathbb{F}_{2^{95}}$	1	18	256	17	10 534
$(30, 95)_2$	3.16	$\mathbb{F}_2$	$\mathbb{F}_{2^{95}}$	1	18	1535	13	8 697
$(18, 51)_2$	2.83	$\mathbb{F}_2$	$\mathbb{F}_{2^{51}}$	1	29	246	18	10 009
$(18, 51)_2$	2.83	$\mathbb{F}_2$	$\mathbb{F}_{2^{51}}$	1	29	1535	14	8 473
$(3, 5)_8$	1.6	$\mathbb{F}_{2^3}$	$\mathbb{F}_{2^{15}}$	1	58	256	25	9 981
$(8, 15)_8$	1.88	$\mathbb{F}_{2^3}$	$\mathbb{F}_{2^{45}}$	1	22	254	19	8 250
$(9, 17)_8$	1.88	$\mathbb{F}_{2^3}$	$\mathbb{F}_{2^{51}}$	1	20	57	24	9 849
$(9, 17)_8$	1.88	$\mathbb{F}_{2^3}$	$\mathbb{F}_{2^{51}}$	1	20	256	18	7 987
$(9, 17)_8$	1.88	$\mathbb{F}_{2^3}$	$\mathbb{F}_{2^{51}}$	1	20	1626	14	6 906
$(9, 17)_8$	1.88	$\mathbb{F}_{2^3}$	$\mathbb{F}_{2^{51}}$	1	20	$2^{16}$	10	5 760

Table 4: Parameter options for BN++LowMC, using LowMC with a full S-box layer (as used in Picnic3) which require 516 AND gates, or 172 GF( $2^3$ ) multiplies. Sizes in bytes.

with  $1 \leq k \leq q + 1$ . So when  $q = 2$  (binary LowMC) we get the  $(3, 5)_2$ -RMFE mentioned above, and  $\mathbb{K} = \mathbb{F}_{2^5}$ . The rate is very small but so is  $\mathbb{K}$ , forcing us to use a relatively large number of parallel repetitions and the signature size is about 15 KB (when  $N = 256$ ). This RMFE saves us about 10 KB against the baseline of  $M = 8$  checks (equivalent to simple lifting to  $\mathbb{K} = \mathbb{F}_{2^8}$ ).

We can then use the concatenation construction of [CCXY18, Lemma 5] to get larger  $\mathbb{K}$ , but at the expense of higher rate. With the  $(30, 95)_2$ -RMFE from [CCXY18, Remark 7] we get about 10 KB signatures with rate 3.16 and since  $\mathbb{K}$  is quite large, we can increase  $N$  to 1535 to decrease signatures to 8.7 KB. Another option from the concatenation construction is  $(18, 51)_2$ -RMFE with rate 2.8 but smaller  $\mathbb{K}$ . This ends up being slightly better than the rate 3.16 option, and  $\mathbb{K}$  fits in a 64-bit word.

Next we have the RMFE options with the GF( $2^3$ ) representation of the S-box. The [CCXY18, Lemma 4] construction can achieve larger  $\mathbb{K}$  when  $\mathbb{F}$  is larger and going from 2 to 8 lets us increase our largest  $|\mathbb{K}|$  from 5 bits to 51 bits while keeping the rate low at 1.88 (as we do not need to use the concatenation construction). The second half of Table 4 shows that larger fields are better, and we show a range of options with the largest possible field given by the  $(9, 17)_8$ -RMFE. The signature size is just under 8 KB with  $N = 256$ , this is about 4.5 KB shorter than Picnic3. As the number of parties increases the signature size decreases, to 5.7 KB when  $N = 2^{16}$  and 5 KB at  $N = 2^{32}$ . Such large number of parties are currently not practical due to the high CPU costs of signing and verification (e.g., 3s when  $N = 2^{16}$  in our implementation), and we mention them only to show the limits of the construction.

We also considered using the concatenation construction with  $q = 2^3$  in order to have  $\mathbb{K}$  be larger than 51 bits, but the increased rate (2.7 or greater) offset the increased soundness of larger  $\mathbb{K}$  and gave strictly larger signatures.

Finally, we note that with an additional circuit-specific optimization, see Appendix B.1, the sizes in the first half of Table 4 can be reduced slightly, but the parameters in the second half of the table still give shorter signatures.

**Implementation Aspects** During the implementation we are working with field elements whose sizes are not a multiple of a full byte (e.g., 51 bits). To optimize the concrete signature sizes, we densely pack these field elements together using shifts and bit operations during serialization. However, this might still lead to the blob of all packed elements not being a byte multiple, in which case we pad it with zero bits. However, during the deserialization we have to take care to verify that these bits are indeed zero to prevent the signature from being malleable which is an undesirable property. In Section 3.3, we discuss representing the RMFE en- and decoding as a matrix multiplication in  $\mathbb{F}_2$ . Since the decoding part is applied right after the S-box layer in LowMC, which is followed by the linear layer (a multiplication with a dense matrix in  $\mathbb{F}_2$ ), we can combine the two matrix multiplications into a single one once the RMFE parameters are fixed. However, our implementation does not use this strategy at this point.

## 5.2 Helium+LowMC

To use LowMC with Helium, we again first use the alternative S-box representation of Section 3.4.2. However, for Helium, we need to work with polynomials of degrees of up to  $2C - 2$ . To achieve this, we use the strategy outlined in Section 4, using RMFEs to first lift to  $\mathbb{F}_{2^9}$  and then perform the final check in  $\mathbb{F}_{2^{144}}$ .

In Table 5, we present benchmarks of our implementation of Helium+LowMC and compare them to the BN++LowMC variant and existing Picnic signatures. We can observe that the additional interpolation required in Helium as well as the usage of a smaller RMFE is contributing to larger computational cost compared to BN++. However, increasing the number of parties has a larger impact on signature size in Helium, since we require fewer repetitions due to the smaller soundness error and more of the field elements in the signature are from smaller fields compared to BN++.

In particular, when  $N = 256$ , both schemes have nearly identical performance, however the signature of Helium is  $\approx 20\%$  smaller than BN++. When compared to existing Picnic3 signatures, both variants can provide much smaller signatures but with longer signing and verification times. Our Helium variant can have signatures that are 1.9x smaller with a  $\approx 4.4$  increase in signing times. We remark that our implementations are not as well optimized as the Picnic3 implementation, but we can still instantiate parameter sets that are strictly better than Picnic3 in both size and signing speed.

## 5.3 Helium+AES

When using AES-128 with Helium, we split the 200 S-boxes into two polynomials, as described in Section 4, and perform the two checks in  $\mathbb{F}_{2^{144}}$ . In contrast to combining BN++ and AES, our Helium+AES variant leads to much smaller signatures, 30% smaller than the previously best AES-based signatures of Banquet and Limbo. We implemented Helium+AES and give benchmarks in Table 6. We see that our Helium+AES variant offers both smaller and faster signatures than both Banquet and Limbo especially when the number of parties  $N$  increases. At  $N = 256$ , Helium+AES results in the first AES-based signatures below 10KB, being 2.4x (1.7x) faster and 25% (32%) smaller than Banquet (Limbo) instances with the same number of parties, respectively.

Scheme	$N$	$\tau$	Sign	Verify	Size
BN++LowMC	16	34	2.11	2.05	12 825
	57	24	5.03	4.96	9 849
	107	21	8.41	8.46	8 966
	256	18	17.40	17.91	7 987
	1626	14	88.55	91.56	6 906
Helium+LowMC	11	36	5.30	5.03	12 386
	57	22	7.15	7.08	8 311
	107	19	9.93	10.32	7 495
	256	16	17.38	17.14	6 582
	1625	12	76.94	76.97	5 537
Picnic1-full			0.86	0.69	30 821
Picnic3			4.01	2.98	12 595

Table 5: Benchmarks for BN++ and Helium with LowMC, with various choices of  $N$  and current versions of Picnic for reference. Times are in ms and sizes are in bytes. All Picnic instances use the LowMC parameters with a full S-box layer. BN++ instances use the  $(9, 17)_8$ -RMFE from Table 4, while Helium instances use the  $(2, 3)_8$ -RMFE.

We briefly mention Helium coupled with the other AES-based OWFs (EM-AES and (EM-)LSAES) when  $N = 256$ . For EM-AES, LSAES, and EM-LSAES signature sizes are 8 608, 9 632 and 8 352 bytes respectively. The options using LSAES provide only a small improvement (EM-LSAES) or none at all (LSAES) when compared to EM-AES, and even AES. An interesting combination is EM-AES with  $N = 107$  since it matches the size of AES with  $N = 256$  from Table 6 but would be almost twice as fast (since  $N$  is halved).

**Implementation Aspects** The cost of polynomial interpolation is significant for a standard implementation of Helium. However, since the  $x$ -values of the interpolations are fixed once a concrete circuit is chosen, we can precompute the Lagrange interpolation polynomials  $L_i(x)$  for these given  $x$ -values. Furthermore, we need to evaluate the polynomials at a given point  $R$  for each party. Instead of interpolating the shares of the polynomials for each party, we can evaluate the Lagrange polynomials at  $R$  once, and then just evaluate the expression  $P(R) = \sum_i L_i(R) \cdot y_i$ , reducing the per-party work required for interpolation and evaluation to a single dot-product of the evaluated Lagrange polynomials and the parties shares of the  $y$ -values. We apply this optimization for both the AES and LowMC-based Helium implementations. We also investigated the choice of using  $\mathbb{F}_{2^{128}}$  as used by AES-GCM as the big field where we perform our checks. However, even though the field arithmetic is faster in  $\mathbb{F}_{2^{128}}$  compared to  $\mathbb{F}_{2^{144}}$ , the lower soundness leads to a larger number of parallel repetitions, leading to larger signatures at the cost of only slightly improved runtimes. In our tests, we observed that instances using  $\mathbb{F}_{2^{144}}$ , but using a lower number of parties  $N$ , produced both faster and smaller signatures than using  $\mathbb{F}_{2^{128}}$  and a larger  $N$ .

Scheme	$N$	$\tau$	Sign	Verify	Size
Helium+AES	17	31	6.36	5.78	17 580
	57	22	7.54	7.17	12 856
	107	19	9.87	9.60	11 420
	256	16	16.53	16.47	9 888
Banquet [BdK <sup>+</sup> 21]	16	41	5.98	4.50	19 776
	57	31	13.08	11.35	15 968
	107	24	19.38	17.43	14 784
	255	21	40.6	37.96	13 284
Limbo [dOT21]	16	40	2.70	2.00	21 520
	57	29	7.30	6.70	16 574
	107	28	11.10	10.00	15 216
	255	24	29.00	27.00	14 512

Table 6: Benchmarks for Helium+AES with various choices of  $N$  and other AES-based signatures for reference. Times are in ms and sizes are in bytes. Limbo numbers are taken directly from [dOT21], since no public implementation is available.

## 5.4 BN++Rain

We can improve the Rainier [DKR<sup>+</sup>21] signature scheme by replacing the proof system with BN++. The OWF of Rainier, called Rain, uses a blockcipher-based design, where the S-boxes are inversion in the field  $\mathbb{F}_{2^\kappa}$ , for  $\kappa = 128, 192$  and 256. Therefore, when combining it with BN++, no lifting is necessary and we can directly work in  $\mathbb{F}_{2^\kappa}$ . We use Rain<sub>4</sub>, the recommended 4-round instance of Rain from [DKR<sup>+</sup>21], described in Appendix D. Based on the publicly available implementation in [DKR<sup>+</sup>21] we have implemented our BN++Rain instance and give a performance comparison to Rainier in Table 7. We see that our BN++Rain instance produces both smaller and faster signatures than Rainier for the same proof system parameters. Additionally, we present some circuit-specific optimizations that allow us to reduce the proof size for the Rain block cipher further in Appendix B.2 and Appendix B.1.2. Combining all of these optimizations, we can reduce the signature sizes of Rainier by about 15%, while also reducing signing and verification times by about the same amount.

We remark that using Helium for Rain is also possible, however, the optimization in Appendix B.1.2 is only applicable to BN++, which also does not require polynomial interpolation, making BN++ preferable in terms of both size and performance.

## 5.5 Signature Scheme Security

In Appendix A we prove unforgeability of BN++ signatures. Since the general structure of BN++ is similar to previous MPCitH-based signatures (such as Picnic [CDG<sup>+</sup>20a], Banquet [BdK<sup>+</sup>21] and Rainier [DKR<sup>+</sup>21]) the analysis is similar, and identical in parts. The main difference is the way that BN++ checks multiplication triples, which is largely covered by Lemma 2 and the simulation-based argument for  $(N - 1)$ -privacy of the checking protocol in Section 2.6.

Instance	$N$	$\tau$	Sign	Verify	Size
Rainier <sub>4</sub>	57	23	1.94	1.87	7 456
Rainier <sub>4</sub>	107	20	2.85	2.73	6 816
Rainier <sub>4</sub>	256	17	5.68	5.67	6 080
Rainier <sub>4</sub>	1625	13	28.07	28.07	5 296
BN++Rain <sub>4</sub>	57	23	1.49	1.43	6 720
BN++Rain <sub>4</sub>	107	20	2.50	2.38	6 176
BN++Rain <sub>4</sub>	256	17	5.15	4.89	5 536
BN++Rain <sub>4</sub>	1615	13	25.24	24.30	4 880
BN++Rain <sub>4</sub> + §B.2, B.1.2	57	23	1.59	1.49	5 984
BN++Rain <sub>4</sub> + §B.2, B.1.2	107	20	2.52	2.36	5 536
BN++Rain <sub>4</sub> + §B.2, B.1.2	256	17	4.79	4.53	4 992
BN++Rain <sub>4</sub> + §B.2, B.1.2	1615	13	25.26	24.36	4 464

Table 7: Comparison of Rainier<sub>4</sub> [DKR<sup>+</sup>21] and BN++Rain at the 128-bit security level. Times in ms, sizes in bytes.

Security reduces to the difficulty of inverting the OWF used for key generation. The reduction is given a OWF output as input. Signatures can be simulated without the corresponding private key, in the standard way (for schemes based on the Fiat-Shamir transform). The reduction first chooses a random challenge, then programs the random oracle so that signature verification passes on the simulated signature. Then, once a valid signature is output by the adversary, by the soundness of the proof protocol, the hash queries for one of the parallel repetitions must have sufficient information to extract the secret key. In particular, since the per-party seeds are committed to using a hash function, the reduction obtains the seeds from the list of RO queries made by the adversary. From the seeds it is straightforward to compute the key shares of all  $N$  parties and recover the key. We give the full EUF-CMA security proof for BN++ in Appendix A.

As the structure of Helium is inherited from Banquet, it also admits a similar security analysis, with the main differences being: Helium uses a simplified polynomial encoding of triples, and the checking protocol from BN++. As this analysis is very repetitive of BN++, we omit it.

As for the QROM, there are generic results by Don et al. [DFM20] for multi-round Fiat-Shamir proofs that might apply directly (or be adapted) to BN++ and Helium. Since both are commit-and-open proofs, the QROM analogue of the strategy just described for the ROM (reading the secret key shares from the RO query transcript), recently developed in [DFMS21, §5] for  $\Sigma$ -protocols, seems like the most direct approach to a QROM analysis of BN++ and Helium (assuming [DFMS21] can be generalized from three to five and seven round protocols).

## 5.6 Comparison to Other Post-Quantum Signature Schemes

Table 8 gives benchmarks for the PQ signature schemes that are candidates in the third round of the NIST PQC process (the first section of the table; implementations from SUPERCOP[eBA22] version 20220213, except for Picnic [DGK<sup>+</sup>22] and SPHINCS<sup>+</sup> [FNR<sup>+</sup>22].) along with three existing AES and Rain based signatures from the literature (middle section) and our new schemes

(in the last section). All schemes are targeting the NIST L1 security level (128 bits of classical security<sup>6</sup>). The schemes making structured hardness assumptions (Dilithium, Falcon, Rainbow and GeMSS) have a large performance advantage in running time, when compared to our new schemes, though the new schemes have much shorter keys and rely on more conservative assumptions. SPHINCS+ is the other candidate besides Picnic (which we compared to throughout §5) making limited assumptions, and comparison here generally favors Helium, except for verify times, where SPHINCS+ is always significantly faster. For example, signatures with Helium+LowMC can be 1.2x shorter, and 6x faster to create (comparing to the “small” SPHINCS+ parameters). Alternatively, Helium+AES signatures are 1.2x larger, with similarly fast signing.

We can also compare performance to SPHINCS+ when signature length is held constant. When compared to the small variant, Helium+LowMC ( $N = 57, \tau = 22$ ) is 16x faster to sign. When compared to the fast variant, (i) BN++LowMC (with  $N = 8, \tau = 45$ ) is 4.6x faster to sign, and (ii) Helium+AES (with  $N = 19, \tau = 30$ ) has equal signing speed.

Scheme	pk	sig	Sign	Verify
Picnic1-L1-full [ZCD+20]	32	30 821	0.86	0.69
Picnic3-L1 [ZCD+20]	32	12 468	4.01	2.98
sphincss128sha256simple [HBD+20]	32	7 856	111.57	0.19
sphincsf128sha256simple [HBD+20]	32	17 088	6.32	0.50
Dilithium2 [LDK+20]	1 312	2 420	0.06	0.03
Falcon-512 [PFH+20]	897	666	0.12	0.03
Rainbow IIIa-Classic [DCP+20]	882 080	164	0.11	0.07
GeMSS128v2 [CFM+20]	352 188	33	242.41	0.08
Banquet-AES-128 [BdK+21]	32	13 284	40.6	37.96
Limbo-Sign AES-128 [dOT21]	32	14 512	29.00	27.00
Rainier <sub>4</sub> [DKR+21]	32	6 080	5.68	5.67
Helium+AES	32	11 420	9.87	9.60
Helium+AES	32	9 888	16.53	16.47
Helium+LowMC	32	7 495	9.93	10.32
Helium+LowMC	32	6 582	17.38	17.14
BN++LowMC	32	11 417	2.72	2.66
BN++Rain <sub>4</sub>	32	5 536	2.52	2.36
BN++Rain <sub>4</sub>	32	4 992	4.79	4.53

Table 8: 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 [dOT21], as no public implementation is available.

<sup>6</sup>We used the L3 parameters for Rainbow as they appear to provide L1 security [Beu22].

## 6 Conclusion

In this work, we presented various techniques to improve the concrete efficiency of MPCitH proof systems where the soundness of the protocol is dependent on the field size. By carefully applying lifting techniques like RMFEs at optimal points in protocols, we produce the shortest proofs (and therefore, signatures) for common choices of one-way functions, showing that proof systems originally intended for use with circuits over larger fields can also be competitive for small fields like  $\mathbb{F}_2$  and  $\mathbb{F}_{2^8}$ . Our methods allow for more flexibility, adding the choice of RMFE (or overall lifting strategy), as another variable in the parameter selection. Future improvements in the design and implementation of RMFEs can be directly applied using our methods.

Now that the soundness error deriving from the field size is much reduced, the number of parties  $N$  becomes a more significant bottleneck for performance. As we’ve shown, the signature size can be reduced even further by increasing  $N$ , motivating optimizations (either at the implementation or protocol level) that allow efficiently scaling to more parties. A significant part of the per-party costs are calls to SHAKE for hashing or generating random tapes, especially for BN++Rain ( $\approx 65\%$ ), and less so for BN++ and Helium with LowMC and AES ( $\approx 20\text{--}30\%$ ). The other per-party costs are much higher in BN++ and Helium: the linear layer in LowMC is particularly expensive, and the polynomial interpolations in Helium is another large part of the cost.

While our optimizations improve the concrete efficiency of proofs and signatures, our multiplication checks still incur a per-gate-per-repetition communication cost, i.e.,  $\mathcal{O}(\tau C)$ . It is an open question if some information can be shared between repetitions to reduce the cost to  $\mathcal{O}(\tau + C)$  instead.

An interesting direction for future work is to develop a general theory about the types of circuits that admit alternate representations over larger fields (as discussed for AES and LowMC in Section 3.4), and also how to find these representations in an automated way. An example question here is whether there exists a representation of two LowMC S-boxes over  $\mathbb{F}_{2^6}$  that could be used to reduce signature size further.

**Acknowledgments** We thank Jonathan Katz for helpful comments on an earlier draft of this work, Christian Rechberger for helpful discussions, and Ignacio Cascudo for answering our questions about RMFEs.

## References

- [ACE<sup>+</sup>21] Mark Abspoel, Ronald Cramer, Daniel Escudero, Ivan Damgård, and Chaoping Xing. Improved single-round secure multiplication using regenerating codes. Cryptology ePrint Archive, Report 2021/253, 2021. <https://eprint.iacr.org/2021/253>.
- [AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkatasubramanian. Liger: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.

- [ARS<sup>+</sup>15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, Heidelberg, April 2015.
- [BCR<sup>+</sup>19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Heidelberg, May 2019.
- [Bd20] Ward Beullens and Cyprien de Saint Guilhem. LegRoast: Efficient post-quantum signatures from the Legendre PRF. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 130–150. Springer, Heidelberg, 2020.
- [BdK<sup>+</sup>21] Carsten Baum, Cyprien de Saint Guilhem, Daniel Kales, Emmanuela Orsini, Peter Scholl, and Greg Zaverucha. Banquet: Short and fast signatures from AES. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 266–297. Springer, Heidelberg, May 2021.
- [Bea92] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Heidelberg, August 1992.
- [Beu22] Ward Beullens. Breaking rainbow takes a weekend on a laptop. Cryptology ePrint Archive, Report 2022/214, 2022. <https://ia.cr/2022/214>.
- [BFH<sup>+</sup>20] Rishabh Bhaduria, Zhiyong Fang, Carmit Hazay, Muthuramkrishnan Venkatasubramanian, Tiancheng Xie, and Yupeng Zhang. Liger++: A new optimized sublinear IOP. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 2025–2038. ACM Press, November 2020.
- [BFO12] Eli Ben-Sasson, Serge Fehr, and Rafail Ostrovsky. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 663–680. Springer, Heidelberg, August 2012.
- [BMN18] Alexander R. Block, Hemanta K. Maji, and Hai H. Nguyen. Secure computation with constant communication overhead using multiplication embeddings. In Debrup Chakraborty and Tetsu Iwata, editors, *INDOCRYPT 2018*, volume 11356 of *LNCS*, pages 375–398. Springer, Heidelberg, December 2018.
- [BN20] Carsten Baum and Ariel Nof. Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In Aggelos Kiayias, Markulf Kohlweiss, Petros

- Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 495–526. Springer, Heidelberg, May 2020.
- [BNO19] Aner Ben-Efraim, Michael Nielsen, and Eran Omri. Turbospeedz: Double your online SPDZ! Improving SPDZ using function dependent preprocessing. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 530–549. Springer, Heidelberg, June 2019.
- [BP09] Joan Boyar and Rene Peralta. New logic minimization techniques with applications to cryptology. Cryptology ePrint Archive, Report 2009/191, 2009. <https://eprint.iacr.org/2009/191>.
- [BS20] Dan Boneh and Victor Shoup. A graduate course in applied cryptography, 2020. Available online <https://crypto.stanford.edu/~dabo/cryptobook/>.
- [CCXY18] Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure MPC revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 395–426. Springer, Heidelberg, August 2018.
- [CDG<sup>+</sup>17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1825–1842. ACM Press, October / November 2017.
- [CDG<sup>+</sup>20a] Melissa Chase, David Derler, Steven Goldfeder, Jonathan Katz, Vlad Kolesnikov, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Xiao Wang, and Greg Zaverucha. The Picnic Signature Scheme Design Document (ver 3.0), 2020.
- [CDG<sup>+</sup>20b] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Daniel Kales, Jonathan Katz, Vladimir Kolesnikov, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Xiao Wang, and Greg Zaverucha. The Picnic Signature Algorithm Specification: Version 3.0, April 2020. <https://github.com/microsoft/Picnic/blob/master/spec/>.
- [CFM<sup>+</sup>20] A. Casanova, J.-C. Faugère, G. Macario-Rat, J. Patarin, L. Perret, and J. Ryckeghem. GeMSS. Technical report, National Institute of Standards and Technology, 2020. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions>.
- [CG20] Ignacio Cascudo and Jaron Skovsted Gundersen. A secret-sharing based MPC protocol for boolean circuits with good amortized complexity. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 652–682. Springer, Heidelberg, November 2020.

- [CG21] Ignacio Cascudo and Emanuele Giunta. On interactive oracle proofs for boolean R1CS statements. *Cryptology ePrint Archive*, Report 2021/694, 2021. <https://eprint.iacr.org/2021/694>.
- [DCP<sup>+</sup>20] Jintai Ding, Ming-Shing Chen, Albrecht Petzoldt, Dieter Schmidt, Bo-Yin Yang, Matthias Kannwischer, and Jacques Patarin. Rainbow. Technical report, National Institute of Standards and Technology, 2020. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions>.
- [dDOS19] Cyprien de Saint Guilhem, Lauren De Meyer, Emmanuela Orsini, and Nigel P. Smart. BBQ: Using AES in picnic signatures. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 669–692. Springer, Heidelberg, August 2019.
- [DFM20] Jelle Don, Serge Fehr, and Christian Majenz. The measure-and-reprogram technique 2.0: Multi-round fiat-shamir and more. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 602–631. Springer, Heidelberg, August 2020.
- [DFMS21] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Online-extractability in the quantum random-oracle model. *Cryptology ePrint Archive*, Report 2021/280, 2021. <https://ia.cr/2021/280>.
- [DGH<sup>+</sup>21] Itai Dinur, Steven Goldfeder, Tzipora Halevi, Yuval Ishai, Mahimna Kelkar, Vivek Sharma, and Greg Zaverucha. MPC-friendly symmetric cryptography from alternating moduli: Candidates, protocols, and applications. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 517–547, Virtual Event, August 2021. Springer, Heidelberg.
- [DGK<sup>+</sup>22] David Derler, Alexander Grass, Daniel Kales, Angela Promitzer, and Sebastian Ramacher. Picnic optimized implementation, April 2022. <https://github.com/IAIK/Picnic>.
- [DKR<sup>+</sup>21] Christoph Dobraunig, Daniel Kales, Christian Rechberger, Markus Schafneggger, and Greg Zaverucha. Shorter signatures based on tailor-made minimalist symmetric-key crypto. *IACR Cryptol. ePrint Arch. Report 2021/692*, 2021. <https://eprint.iacr.org/2021/692>.
- [DL77] Richard A DeMillo and Richard J Lipton. A probabilistic remark on algebraic program testing. Technical report, Georgia Institute of Technology, Atlanta School of Information and Computer Science, 1977.
- [DLN19] Ivan Damgård, Kasper Green Larsen, and Jesper Buus Nielsen. Communication lower bounds for statistically secure MPC, with or without preprocessing. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 61–84. Springer, Heidelberg, August 2019.

- [DLS20] Anders P. K. Dalskov, Eysa Lee, and Eduardo Soria-Vazquez. Circuit amortization friendly encodings and their application to statistically secure multiparty computation. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 213–243. Springer, Heidelberg, December 2020.
- [dOT21] Cyprien de Saint Guilhem, Emmanuela Orsini, and Titouan Tanguy. Limbo: Efficient zero-knowledge MPC with H-based arguments. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 3022–3036. ACM Press, November 2021.
- [eBA22] eBACS: ECRYPT Benchmarking of Cryptographic Systems. SUPER COP, Version 20220213, April 2022. <https://bench.cr.yp.to/supercop.html>.
- [FJR22] Thibault Feneuil, Antoine Joux, and Matthieu Rivain. Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. Cryptology ePrint Archive, Report 2022/188, 2022. <https://ia.cr/2022/188>.
- [FNR<sup>+</sup>22] Scott Fluhrer, Ruben Niederhagen, Joost Rijneveld, Peter Schwabe, and Bas Westerbaan et al. SPHINCS<sup>+</sup> sha256-avx2 implementation, May 2022. <https://github.com/sphincs/sphincsplus>.
- [GHS<sup>+</sup>21] Yaron Gvili, Julie Ha, Sarah Scheffler, Mayank Varia, Ziling Yang, and Xinyuan Zhang. TurboIKOS: Improved non-interactive zero knowledge and post-quantum signatures. In *Applied Cryptography and Network Security*, pages 365–395. Springer, 2021.
- [GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 1069–1083. USENIX Association, August 2016.
- [Gol07] Oded Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, 2007.
- [HBD<sup>+</sup>20] Andreas Hulsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. SPHINCS+. Technical report, National Institute of Standards and Technology, 2020. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions>.
- [IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.
- [Kat10] Jonathan Katz. *Digital signatures*. Springer Science & Business Media, 2010.

- [KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 525–537. ACM Press, October 2018.
- [KZ20a] Daniel Kales and Greg Zaverucha. An attack on some signature schemes constructed from five-pass identification schemes. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20*, volume 12579 of *LNCS*, pages 3–22. Springer, Heidelberg, December 2020.
- [KZ20b] Daniel Kales and Greg Zaverucha. Improving the performance of the Picnic signature scheme. *IACR TCHES*, 2020(4):154–188, 2020. <https://tches.iacr.org/index.php/TCHES/article/view/8680>.
- [LDK<sup>+</sup>20] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2020. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions>.
- [MV04] David A. McGrew and John Viega. The security and performance of the Galois/counter mode (GCM) of operation. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT 2004*, volume 3348 of *LNCS*, pages 343–355. Springer, Heidelberg, December 2004.
- [NIS15] NIST. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. National Institute of Standards and Technology (NIST), FIPS PUB 202, U.S. Department of Commerce, 2015.
- [PFH<sup>+</sup>20] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2020. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions>.
- [Sch80] Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980.
- [ZCD<sup>+</sup>20] Greg Zaverucha, Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Jonathan Katz, Xiao Wang, Vladimir Kolesnikov, and Daniel Kales. Picnic. Technical report, National Institute of Standards and Technology, 2020. available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions>.
- [Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *International symposium on symbolic and algebraic manipulation*, pages 216–226. Springer, 1979.

## A Security Proof of BN++ Signatures

In Theorem 7, we prove that BN++ 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 Lemma 5 that BN++ 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 7, 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 5.** *Let Commit,  $H_1$  and  $H_2$  be modeled as random oracles, Expand be modeled as a random function, and let  $(N, \tau, \kappa, \mathbb{F})$  be parameters of the BN++ signature scheme. Let  $\mathcal{A}$  be an adversary against the EUF-KO security of BN++ 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*

$$\varepsilon_{\text{KO}} \leq \varepsilon_{\text{OWF}} + \frac{(\tau N + 1)Q^2}{2^{2\kappa}} + \Pr[X + Y = \tau],$$

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

*Remark 6.* We do not express  $\Pr[X + Y = \tau]$  as a closed function; we must choose parameters  $(N, \tau, \mathbb{F})$  for BN++ such that it is negligible in  $\kappa$ .

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

Algorithm  $\mathcal{B}$  simulates the EUF-KO game using the random oracles  $H_c$  (shorthand for Commit),  $H_1$  and  $H_2$  and query lists  $\mathcal{Q}_c$ ,  $\mathcal{Q}_1$  and  $\mathcal{Q}_2$ . In addition,  $\mathcal{B}$  maintains two tables  $\mathcal{T}_{\text{sh}}$ ,  $\mathcal{T}_{\text{in}}$  to store shares of the parties, inputs to the MPC protocol and openings of the multiplication check protocol that it recovers from  $\mathcal{A}$ ’s RO queries.  $\mathcal{B}$  also maintains a set **Bad** to keep track of the outputs of all three random oracles. We also ignore calls to Expand() in our analysis, since they are used to expand outputs from  $H_1$  and  $H_2$  when Expand is a random function this is equivalent to increasing the output lengths of  $H_1$  and  $H_2$ .

*Behavior of  $\mathcal{B}$ .* On input  $\text{pk}$ , a OWF challenge, algorithm  $\mathcal{B}$  forwards it to  $\mathcal{A}$  as a BN++ public key for the EUF-KO game. It lets  $\mathcal{A}$  run and answers its random oracle queries in the following way. We assume (wlog.) that Algorithm 2, Algorithm 3 and Algorithm 4 only consider queries that are correctly formed, and ignore duplicate queries.)

- $H_c$ : When  $\mathcal{A}$  queries the commitment random oracle,  $\mathcal{B}$  records the query to learn which commitment corresponds to which seed. See Algorithm 2.
- $H_1$ : When  $\mathcal{A}$  commits to seeds and sends the offsets for the secret key and the inverse values,  $\mathcal{B}$  checks whether the commitments were output by its

---

**Algorithm 2**  $H_c(q_c = (\text{salt}, e, i, \text{seed}))$ :

---

- 1:  $x \xleftarrow{\$} \{0, 1\}^{2\kappa}$ .
  - 2: **if**  $x \in \text{Bad}$  **then** abort. ▷ Check if  $x$  is fresh.
  - 3:  $x \rightarrow \text{Bad}$ .
  - 4:  $(q_c, x) \rightarrow \mathcal{Q}_c$ .
  - 5: Return  $x$ .
- 

simulation of  $H_c$ . If any were for some  $e$  and  $i$ , then  $\mathcal{B}$  is able to reconstruct the shares for party  $i$  in repetition  $e$ . If  $\mathcal{B}$  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  $\mathcal{A}$  in that repetition. We also recover all other values that are part of the protocol in addition to  $\text{sk}_e$ , however this is not strictly necessary and only aims to make the following probability analysis of the case  $\Pr[\mathcal{A} \text{ wins} \mid \mathcal{B} \text{ outputs } \perp]$  easier to follow. See Algorithm 3.

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

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

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

$$\begin{aligned}
\Pr[\mathcal{A} \text{ wins}] &= \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{B} \text{ aborts}] + \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{B} \text{ outputs } \perp] \\
&\quad + \Pr[\mathcal{A} \text{ wins} \wedge \mathcal{B} \text{ outputs sk}] \\
&\leq \Pr[\mathcal{B} \text{ aborts}] + \Pr[\mathcal{A} \text{ wins} \mid \mathcal{B} \text{ outputs } \perp] \\
&\quad + \Pr[\mathcal{B} \text{ outputs sk}]. \tag{3}
\end{aligned}$$

Let  $Q_{\text{com}}, Q_1$  and  $Q_2$  denote the number of queries made by  $\mathcal{A}$  to each respective random oracle. Given the way in which values are added to  $\text{Bad}$ , we have:

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

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

*Cheating in the first round.* For any query  $q_1 \in \mathcal{Q}_1$ , and its corresponding expanded answer  $h_1 = ((\epsilon_{e,\ell})_{\ell \in [C]})_{e \in [\tau]}$ , let  $G_1(q_1, h_1)$  be the set of indices  $e \in [\tau]$  of “good executions” where both  $\mathcal{T}_{\text{in}}[q_1, e]$  is non-empty and it holds that  $v_e = 0$ . If there does not exist such a  $q_1$ , let  $G_1(q_1, h_1) = \emptyset$ .

---

**Algorithm 3**  $H_1(q_1 = \sigma_1)$ :

---

Parse  $\sigma_1$  as  $(\text{salt}, ((\text{com}_e^{(i)}, \text{ct}_e^{(i)})_{i \in [N]}, \Delta \text{sk}_e, \Delta c_e, (\Delta z_{e,\ell})_{\ell \in [C]})_{e \in [\tau]})$

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  $\mathcal{B}$  records the shares of the secret key and of the multiplication output values for that party, derived from that seed and the offsets committed to in  $\sigma_1$ :

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

3:  $\text{sk}_e^{(i)}, (z_{e,\ell}^{(i)})_{\ell \in [C]}, (a_{e,\ell}^{(i)})_{\ell \in [C]}, c_e^{(i)} \leftarrow \text{ExpandTape}(\text{salt}, e, i, \text{seed}_e^{(i)})$ .

4: **if**  $i \stackrel{?}{=} 1$  **then**

5:  $\text{sk}_e^{(i)} \leftarrow \text{sk}_e^{(i)} + \Delta \text{sk}_e, (z_{e,\ell}^{(i)} \leftarrow z_{e,\ell}^{(i)} + \Delta z_{e,\ell})_{\ell \in [C]}$  and  $c_e^{(i)} \leftarrow c_e^{(i)} + \Delta c_e$ .

6:  $(\text{sk}_e^{(i)}, (z_{e,\ell}^{(i)})_{\ell \in [C]}, (a_{e,\ell}^{(i)})_{\ell \in [C]}, c_e^{(i)}) \rightarrow \mathcal{T}_{\text{sh}}[q_1, e, i]$ .

If the shares of the various elements are known for every party in that repetition,  $\mathcal{B}$  records the resulting secret key, multiplication inputs and dot-product triple:

7: **for** each  $e : \forall i, \mathcal{T}_{\text{sh}}[q_1, e, i] \neq \emptyset$  **do**

8:  $\text{sk}_e \leftarrow \sum_i \text{sk}_e^{(i)}, (z_{e,\ell} \leftarrow \sum_i z_{e,\ell}^{(i)})_{\ell \in [C]}, (a_{e,\ell} \leftarrow \sum_i a_{e,\ell}^{(i)})_{\ell \in [C]}$  and

$c_e \leftarrow \sum_i c_e^{(i)}$ .

9: Using  $\text{sk}_e$ , execute the circuit to recover the mult. inputs  $(x_{e,\ell}, y_{e,\ell})_{\ell \in [C]}$ .

10:  $(\text{sk}_e) \rightarrow \mathcal{T}_{\text{in}}[q_1, e]$ .

11:  $x \stackrel{\$}{\leftarrow} \{0, 1\}^{2\kappa}$ .

12: **if**  $x \in \text{Bad}$  **then** abort.

13:  $x \rightarrow \text{Bad}$ .

14:  $(q_1, x) \rightarrow \mathcal{Q}_1$ .

Compute the multiplication check protocol values.

15:  $(\epsilon_{e,\ell})_{\ell \in [C]} \leftarrow \text{Expand}(x)$ .

16: **for** each  $e : \mathcal{T}_{\text{in}}[q_1, e] \neq \emptyset$  **do**

17:  $(\alpha_{e,\ell} \leftarrow \epsilon_{e,\ell} \cdot x_{e,\ell} - a_{e,\ell})_{\ell \in [C]}$ .

18:  $v_e \leftarrow \left( \sum_{\ell \in [C]} \epsilon_{e,\ell} \cdot z_{e,\ell} - \alpha_{e,\ell} \cdot y_{e,\ell} \right) - c_{e,\ell}$ .

19: Return  $x$ .

---

---

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

---

- 1:  $h_1 \rightarrow \text{Bad}$ .
  - 2:  $x \xleftarrow{\$} \{0, 1\}^{2\kappa}$ .
  - 3: **if**  $x \in \text{Bad}$  **then** abort.
  - 4:  $x \rightarrow \text{Bad}$ .
  - 5:  $(q_2, x) \rightarrow \mathcal{Q}_2$ .
  - 6: Return  $x$ .
- 

For any such good execution  $e \in G_1(q_1, h_1)$ , since  $\mathcal{B}$  outputs  $\perp$  but  $\mathcal{A}$  wins, this implies that either

$$\forall \ell \in [C] : x_{e,\ell} \cdot y_{e,\ell} = z_{e,\ell} \text{ and } \sum_{\ell \in [C]} a_{e,\ell} \cdot y_{e,\ell} = c_e$$

held (in which case any values of  $(\epsilon_{e,\ell})_{\ell \in [C]}$  passes the check), or the challenges  $(\epsilon_{e,\ell})_{\ell \in [C]}$  were sampled such that the multiplication check presented in Section 2.6 output ACC, missing the incorrect value(s). Conditioning on the first event not happening (since then no cheating in the first round would be required), Lemma 2 gives us that the second happens with probability at most  $p_1 := 1/|\mathbb{F}|$ , given that  $h_1$  is distributed uniformly at random (which holds assuming  $H_1$  and 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  $\mathcal{B}$  outputs  $\perp$ . We therefore have that  $\#G_1(q_1, h_1) \mid_{\perp} \sim X_{q_1}$  where  $X_{q_1} = \mathfrak{B}(\tau, p_1)$ , where  $\mathfrak{B}(\tau, p_1)$  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

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

*Cheating in the second round.* Each second round query  $q_2 = (h_1, \sigma_2)$  that  $\mathcal{A}$  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  $e \in [\tau] \setminus G_1(q_1, h_1)$ , either verification failed, in which case  $\mathcal{A}$  couldn’t have won, or the verification passed, despite  $\sum_i v_e^{(i)} = 0$  or  $\sum_i \text{ct}_e^{(i)} = \text{pk}$  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. Additionally, verification would also fail if more than one party’s  $\text{com}_e^{(i)}$  was not a valid commitment to a seed, so we also implicitly handle the case where our reduction could not recover the shares of all parties in Algorithm 3.

Since the expanded  $h_2 = (\tilde{i}_e)_{e \in [\tau]} \in [N]^\tau$  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

$$\Pr[\mathcal{A} \text{ wins} \mid \#G_1(q_{\text{best}_1}, h_{\text{best}_1}) = \tau_1] \leq 1 - \left(1 - \left(\frac{1}{N}\right)^{\tau - \tau_1}\right)^{Q_2}.$$

Finally conditioning on  $\mathcal{B}$  outputting  $\perp$  and summing over all values of  $\tau_1$ , we have that

$$\Pr[\mathcal{A} \text{ wins} \mid \mathcal{B} \text{ outputs } \perp] \leq \Pr[X + Y = \tau] \quad (5)$$

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  $\mathfrak{B}(\tau - X, 1/N)$ .

*Conclusion.* Bringing Equation (3), Equation (4) and Equation (5) together, we obtain the following.

$$\begin{aligned} \Pr[\mathcal{A} \text{ wins}] &\leq \frac{(\tau N + 1)(Q_{\text{com}} + Q_1 + Q_2)^2}{2^{2\kappa}} + \Pr[X + Y = \tau] \\ &\quad + \Pr[\mathcal{B} \text{ outputs sk}] \end{aligned}$$

Assuming  $\text{KeyGen}$  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.  $\square$

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<sup>+</sup>20a, Section 6.3] it is shown that this holds when the hash function used to derived seeds is modeled as a random oracle. Secure one-way functions are defined in [Gol07, Section 2.2].

**Theorem 7.** *The  $BN^{++}$  signature scheme is EUF-CMA-secure, assuming that  $\text{Commit}$ ,  $H_1$ ,  $H_2$  and  $\text{Expand}$  are modeled as random oracles,  $\text{ExpandTape}$  is a secure PRG, the seed tree construction is computationally hiding, the  $(N, \tau, \mathbb{K})$  parameters are appropriately chosen, and that  $\text{KeyGen}$  is a secure one-way function.*

*Proof.* Fix an attacker  $\mathcal{A}$ . We define a sequence of games where the first game corresponds to  $\mathcal{A}$  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  $\mathcal{B}$  denote an adversary against the EUF-KO game that acts as a simulator of the EUF-CMA game to  $\mathcal{A}$ . As we're in the random oracle model: when  $\mathcal{A}$  queries one of its random oracles,  $\mathcal{B}$  first checks if that query has been recorded before; if so, then it responds with the recorded answer; if not,  $\mathcal{B}$  forwards the query to its corresponding random oracle, records the query and the answer it receives and forwards the answer to  $\mathcal{A}$ . Let  $G_i$  denote the probability that  $\mathcal{A}$  succeeds in game  $G_i$ . At a high level, the sequence of games is as follows:

$G_0$ :  $\mathcal{B}$  knows a real secret key  $\text{sk}$  and can compute signatures honestly;

$G_1$ :  $\mathcal{B}$  replaces real signatures with simulated ones which no longer use  $\text{sk}$ ;

$\mathcal{B}$  then uses the EUF-KO challenge  $\text{pk}^*$  in its simulation with  $\mathcal{A}$ .

We note that  $\mathcal{A}$ '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  $\mathcal{A}$  queries the signing oracle,  $\mathcal{B}$  simulates a signature by sampling a random secret key  $\text{sk}^*$ , choosing a party  $\mathcal{P}_{i^*}$  at random and cheating in the verification phase and in the broadcast of the output shares  $\text{ct}_e^{(i)}$  such that the circuit still outputs the correct ciphertext, and finally ensuring that the values observed by  $\mathcal{A}$  are sampled independently of  $\text{sk}^*$  and with a distribution that is computationally indistinguishable from a real signature.  $\mathcal{B}$  programs  $H_1$  to return the randomly sampled challenges  $((\epsilon_{e,\ell})_{\ell \in [C]})_{e \in [\tau]}$  and  $H_2$  to return a randomly sampled challenge  $(\bar{i}_e)_{e \in [\tau]}$ .

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  $\text{sk}$  is known and signatures are created honestly, and ends with  $G_1$ , where signatures are simulated without using  $\text{sk}$ . With each change to  $\mathcal{B}$ 's behavior, we give an argument as to why the simulation remains indistinguishable, and quantify these below.

1. The initial  $\mathcal{B}$  knows the real  $\text{sk}$  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,  $\mathcal{B}$  is indistinguishable from the real EUF-CMA game as long as it does not abort.
2. Before beginning, the next  $\mathcal{B}$  samples  $h_2$  at random and expands it to obtain  $(\bar{i}_e)_{e \in [\tau]}$ ; these are the unopened parties, which  $\mathcal{B}$  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 4. If that query has already been made,  $\mathcal{B}$  aborts the simulation.
3. In Phase 1, the next  $\mathcal{B}$  replaces  $\text{seed}_e^{(\bar{i})}$  (the seed of the cheating party) in the binary tree, for each  $e \in [\tau]$ , by a randomly sampled one. This is indistinguishable from the previous version of  $\mathcal{B}$  assuming that the tree structure is hiding.
4. The next  $\mathcal{B}$  replaces the random tapes for party  $\bar{i}_e$ , i.e., the outputs of  $\text{ExpandTape}(\text{salt}, e, \bar{i}_e, \text{seed}_e^{(\bar{i}_e)})$ , 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  $\mathcal{B}$  replaces the commitments of the unopened parties  $\text{com}_e^{(\bar{i}_e)}$  with random values (i.e., without querying  $\text{Commit}$ ).  $\mathcal{B}$  aborts if  $\mathcal{A}$  queries  $x$  such that  $\text{Commit}(x)$  was output by  $\mathcal{B}$ .
6. Before starting Phase 2, the next  $\mathcal{B}$  samples  $h_1$  at random and expands it to obtain  $((\epsilon_{e,\ell})_{\ell \in [C]})_{e \in [\tau]}$ ; 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,  $\mathcal{B}$  aborts the simulation.
7. In Phase 3, the next  $\mathcal{B}$  replaces  $\alpha_{e,\ell}^{(\bar{i}_e)}$  with a uniformly random value. Furthermore, it sets  $v_e^{(\bar{i}_e)} \leftarrow -\sum_{i \neq \bar{i}_e} v_e^{(i)}$ . As these were previously computed based on uniform elements read from the random tapes, this is indistinguishable from the previous hop. From now on, the multiplications check always passes, even if the inputs are not valid triples according to Lemma 2,

and the distribution of everything that  $\mathcal{A}$  can observe is indistinguishable from an honest signature and independent of hidden values.

8. In Phase 1, the next  $\mathcal{B}$  sets  $\Delta z_{e,\ell}$  and  $\Delta c_e$  to random values. As these were previously computed based on uniform elements read from the random tapes, this is indistinguishable from the previous hop.
9. The final  $\mathcal{B}$  replaces the real  $\text{sk}$  by a random key  $\text{sk}^*$  and cheats on the broadcast of party  $\mathcal{P}_{i_e}$ 's output share  $\text{ct}_e^{(i_e)}$  such that it matches what is expected, given the  $N - 1$  other shares. As  $\text{sk}_e^{(i_e)}$  is independent from the seeds  $\mathcal{A}$  observes, the distribution of  $\Delta \text{sk}_e$  is identical and  $\mathcal{A}$  has no information about  $\text{sk}^*$ . As  $\mathcal{P}_{i_e}$  is never opened,  $\mathcal{B}$ 's cheating on  $\text{ct}_e^{(i_e)}$  can't be detected.

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

There are four points at which  $\mathcal{B}$  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  $\mathcal{A}$  and  $\mathcal{B}$ ); each time  $\mathcal{B}$  simulates a signature, it has a maximum probability of  $Q_{\text{salt}}/2^{2\kappa}$  of selecting an existing salt and aborting. Let  $Q_c$  denote the number of queries made to Commit by  $\mathcal{A}$ , including those made during signature queries. Since Commit is a random oracle, and  $\text{seed}_e^{(i_e)}$  is a uniformly random  $\kappa$ -bit value not used by  $\mathcal{B}$  elsewhere, each time  $\mathcal{B}$  attempts a new signature, it has a maximum probability of  $Q_c/2^\kappa$  of replacing an existing commitment and aborting.

Similarly for  $H_1$ , resp.  $H_2$ ,  $\mathcal{B}$  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  $\mathcal{B}$  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

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

where

$$\begin{aligned} \Pr[\mathcal{B} \text{ aborts}] &\leq Q_{\text{salt}}/2^{2\kappa} + Q_c/2^\kappa + Q_1/2^{2\kappa} + Q_2/2^{2\kappa} \\ &= (Q_{\text{salt}} + Q_1 + Q_2)/2^{2\kappa} + Q_c/2^\kappa \\ &\leq (Q_1 + Q_2)/2^{2\kappa-1} + Q_c/2^\kappa \quad (\text{Since } Q_{\text{salt}} \leq Q_1 + Q_2), \\ &\leq Q/2^\kappa \quad (\text{where } Q = Q_1 + Q_2 + Q_c). \end{aligned}$$

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

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

where the bound on the advantage  $\varepsilon_{\text{KO}}$  of a EUF-KO attacker follows from Lemma 5. By a union bound, we have that

$$\begin{aligned} \varepsilon_{\text{CMA}} \leq & \varepsilon_{\text{OWF}} + \frac{(\tau N + 1)Q^2}{2^{2\kappa}} + \Pr[X + Y = \tau] \\ & + Q_s \cdot (\tau \cdot \varepsilon_{\text{PRG}} + \varepsilon_{\text{TREE}} + Q/2^\kappa). \end{aligned}$$

Assuming that `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  $(N, \tau, \mathbb{K})$  are appropriately chosen implies that  $\varepsilon_{\text{CMA}}$  is negligible in  $\kappa$ .  $\square$

## B Other Optimizations

In this section we describe some additional optimizations that we used in our implementations to reduce signature size and improve performance. They depend somewhat on the circuit or parameters of the MPCitH protocol, but are sufficiently generic that we expect them to be of independent interest.

### B.1 Optimization: Repeated Multipliers

When some of the multiplication triples in the batch to be verified have the same multiplier, e.g.,  $(x_1, y, z_1), (x_2, y, z_2)$  we can batch the  $\alpha$  value in the multiplication checking protocol. Instead of computing  $\alpha_1 = \epsilon_1 x_1 + a_1$  and  $\alpha_2 = \epsilon_2 x_2 + a_2$  and broadcasting shares of both  $\alpha_1$  and  $\alpha_2$ , we can instead compute  $\alpha = \epsilon_1 x_1 + \epsilon_2 x_2 + a$ , broadcast it and then compute

$$v = \alpha \cdot y - \epsilon_1 \cdot z_1 - \epsilon_2 \cdot z_2 - c$$

where  $c = y \cdot a$  (instead of  $c = y_1 a_1 + y_2 a_2$  as in §2.6). Of course this optimization can only be applied if the repeated multiplier is a result of the structure of the circuit to be computed (i.e., the same wire is an input to two gates), and not just because two wires happen to have the same value for the given input, as this might leak information about the input.

This optimization is applicable to `BN++` but not `Helium`, since in `Helium` there is only one  $\alpha$ -value per repetition: the checking protocol is only applied to one multiplication triple based on the polynomial encoding of all multiplication gates.

#### B.1.1 Application to Binary LowMC

Recall that the LowMC S-box over  $\mathbb{F}_2$  is computed as

$$S(a, b, c) = (a + bc, a + b + ac, a + b + c + ab).$$

The input bit  $c$  is the multiplier in two of the multiplications, meaning we only need two  $\alpha$  values per S-box rather than three. The `BN++` proof size for binary LowMC can be reduced to

$$3\kappa + \tau \cdot (3\kappa + \kappa \cdot \lceil \log_2(N) \rceil) + \mathcal{M}(C) + \log_2(|\mathbb{K}|).$$

where  $\mathcal{M}(C) = \frac{5}{3} \lceil C/k \rceil \cdot (\log_2(|\mathbb{K}|))$ . Comparing to Table 3, when applied to the binary circuit for LowMC, the signature size goes from 26.6 KB to 21.1 KB when lifting to  $\mathbb{F}_{2^8}$ . For BN++RMFE with the  $(18, 51)_2$ -RMFE with  $N = 246$  and  $\tau = 18$  (see Table 4) the signature size goes from 10 KB to 9.15 KB. In conclusion, while useful, the optimization is not enough for binary LowMC to be competitive with LowMC over  $\mathbb{F}_{2^3}$  (Section 3.4.2).

### B.1.2 Application to Rain and EM ciphers

Let  $(s_i, t_i)$  denote the S-box inputs and outputs in Rain, and with the recommended number of rounds we have four such pairs.

Note that  $s_1 = p + k + c_1$  and  $t_4 = c + k$ , where  $c_1$  is the round constant, and  $(p, c)$  are from the public key (it helps to refer to Figure 3 in the Rain paper). We can substitute these expressions into the checks  $s_1 t_1 \stackrel{?}{=} 1$  and  $s_4 t_4 \stackrel{?}{=} 1$  and arrive at the following expressions:

$$\begin{aligned} (p + k + c_1) \cdot t_1 &\stackrel{?}{=} 1 \text{ iff } t_1 \cdot k \stackrel{?}{=} 1 - (p + c_1)t_1, \\ s_4 \cdot (k + c) &\stackrel{?}{=} 1 \text{ iff } s_4 \cdot k \stackrel{?}{=} 1 - s_4 c. \end{aligned}$$

The new triples on the right have the same multiplier,  $k$ , and we can make use of the repeated multipliers optimization, saving one  $\alpha$  per repetition, for a total of  $\tau|\mathbb{F}|$  bits.

This generalizes to OWFs based on Even-Mansour (EM) block ciphers. Namely, when  $E(k, p) = k + \pi(k + p)$  for a public permutation  $\pi$ , the output(s) of the last multiplication gate(s) of  $\pi$  can be written as a linear combination of  $k$  and constants, similarly the inputs to the first multiplication gate(s) depend only on  $k, p$  and public constants.

## B.2 Optimization: Multiplications with Public Output

In the BN++ protocol, for every multiplication gate we communicate a value  $\Delta z$ , to inject the output. Of course when the output  $z$  is a public value, then  $\Delta z$  does not need to be sent. We aren't aware of many circuits where multiplication outputs are public, one example is an RSA modulus, but none of the OWFs we consider in this paper have this property. However, the circuits we consider have multiplication gates where a linear function of the output is public, namely, the public ciphertext (OWF output) is computed linearly from the last multiplication gate(s). We show how to avoid sending  $\Delta z$  in this case. We explain the idea using the Rain OWF, since there an S-box is the width of the full state and it's most simple to explain, but this is also applicable to LowMC, AES and the other AES-based OWFs considered in this paper (as well as their Even-Mansour variants).

At the moment, for the last round of  $\text{Rain}_4$ , we inject the S-box output  $t_4$  via  $\Delta t_4$  and calculate the S-box input  $s_4$  from the last round. After making sure that  $s_4 \cdot t_4 = 1$  using the multiplication check, we calculate forward using  $t_4$  and make sure that it is indeed equal to the correct output by doing the last key-addition and then broadcasting the internal state to compare it to the expected ciphertext. However, we can compute shares of  $t_4$  using our shares of the key and ciphertext, since  $t_4 + k_4 = c$ . First we create a (trivial) sharing of

the public ciphertext  $c$  as

$$c^{(i)} = \begin{cases} c & \text{if } i = 1, \\ 0 & \text{otherwise.} \end{cases}$$

Then, we can calculate shares of  $t_4$  locally:  $t_4^{(i)} = c^{(i)} - k_4^{(i)}$ . This leads to a valid sharing of  $t_4$  without the need for the prover to inject it. We still need to check that  $s_4 \cdot t_4 = 1$  using a multiplication check (so we must still communicate the associated  $\alpha$  values). For Rain, this saves  $\tau|\mathbb{F}|$  values, for LowMC with a full nonlinear layer this saves one full round worth of  $z$  values, in each case the width of the state (16 bytes at the 128-bit security level).

The optimization also applies to Helium, but is not compatible with BN++RMFE, since we cannot compute shares of  $T \in \mathbb{K}$  from the shares  $\mathbf{t} \in \mathbb{F}^k$  we would compute above, due to the general limitations we have with RMFEs.

### B.3 Optimization: TurboIKOS Compression

The TurboIKOS paper [GHS<sup>+</sup>21] also presents a series of optimizations to the BN protocol, first reducing  $\mathcal{M}(C)$  to  $3C \log_2(|\mathbb{F}|)$ , then presenting a final optimization (in Section 3.4). In this optimization (described in our notation), we can save sending the  $C$   $\alpha$  values, but must instead send  $N$  elements of  $\mathbb{F}$  and one hash digest. Thus when this optimization is applied to BN++, we change  $\mathcal{M}(C)$  from  $(2C + 1) \log_2(|\mathbb{F}|)$  to  $(C + 1 + N) \log_2(|\mathbb{F}|) + 2\kappa$ .

When  $C$  is larger than  $N$ , this is smaller. However, smaller  $N$  decreases soundness and means we must use more parallel repetitions, potentially offsetting the benefit of this optimization. To determine whether this optimization will be competitive, we searched the parameter space for the concrete circuits and security levels we consider in this paper.

First, for Rain, since  $C = 4$ , this is not a useful optimization, and with Helium we have only one or two triples to verify so we consider applying the TurboIKOS optimization to BN++ with AES and LowMC, where  $C$  is 172–200. For LowMC, since we can apply the  $(9, 17)_8$ -RMFE, the number of multiplications we need to check (in the larger field  $\mathbb{K}$ ) is only 20, and so we get shorter proofs without this optimization.

For AES we see a small window where  $N$  is large enough for soundness, but small relative to  $C$  and the BN++ with the TurboIKOS optimization has its shortest proof sizes, and then as  $N$  increases proof sizes steadily increase (which unfortunately means parameter choices are limited). There are some cases where this optimization is helpful, as shown in Table 9. We show some size estimates for AES-128, first showing the window around  $N = 32$  where BN++TurboIKOS is most competitive (note that we need  $M = 3$  checks per repetition for these sizes or a simple lift from  $\mathbb{F}_{2^8}$  to  $\mathbb{F}_{2^{24}}$ ). Then we compare BN++TurboIKOS's shortest size (14.6 KB, when  $N = 31$ ) to Banquet and Helium (parameterized to have size  $\approx 14.6$  KB). In applications where this size of signature is acceptable, the CPU costs of BN++/TurboIKOS should be lower, since when compared to Banquet and Helium, it uses fewer parties and requires no lifting to extension fields and no polynomial arithmetic. That said, 14.6 KB is significantly larger than the 9.9 KB of Helium+AES (obtained when  $N = 256, \tau = 16$ , see Table 6).

Scheme	$N$	$\tau$	$M$	Size
BN++	8	49	3	17 491
	16	38	3	15 106
	31	32	3	14 688
	56	28	3	15 412
Banquet	107	24		14 784
Helium	35	25		14 596

Table 9: AES-128 signature size estimates for BN++ with the TurboIKOS optimization, compared to Banquet and Helium for signatures of approximately the same size.

## C Additional Details of RMFE Lifting

In this section we give details of the comparison of BN++RMFE and BN++ on proof size, and also describe how to adapt BN++RMFE to inverse checking (instead of multiplication checking).

### C.1 Size Comparison

Does lifting with an RMFE always give the shortest proof sizes? As discussed in Section 3.3, it is difficult to compare the size of BN++RMFE and BN++ as it depends on the concrete choice of RMFE, for the parameters  $\mathbb{K}$  and  $k$ . We can compare the number of bits communicated by the checking protocol for each bit of soundness to asymptotically evaluate the impact of using an RMFE on the size (see Appendix C.1).

For BN++ we must communicate  $\mathcal{M}(C)/\log_2(\mathbb{F}) = 2C + 1$  bits per bit of soundness. In BN++RMFE,

$$\begin{aligned}\mathcal{M}(C) &= (2 \lceil C/k \rceil + 1) \log_2(|\mathbb{K}|) \\ &= (2 \lceil C/k \rceil + 1) \log_2(|\mathbb{F}|^m)\end{aligned}$$

and we must communicate

$$\mathcal{M}(C)/\log_2(|\mathbb{F}|^m) = 2 \lceil C/k \rceil + 1 \tag{6}$$

bits per bit of soundness, making this better as long as  $k > 1$ . This is always better than plain BN++, subject to some caveats. First, we must have  $|\mathbb{F}|^m < 2^\kappa$ . If this is not the case, then it might be worse in practice, as we’re paying in communication for unnecessary additional soundness.

Second, this comparison does not consider BN++ with simple lifting, which is a more natural baseline (but complicates comparison). For BN++ with simple lifting, we have

$$\begin{aligned}\mathcal{M}(C) &= C \log_2(|\mathbb{F}|) + (C + 1) \log_2(|\mathbb{K}|) \\ &= C \log_2(|\mathbb{F}|) + (C + 1) \log_2(|\mathbb{F}|^m) \\ &= C \log_2(|\mathbb{F}|) + m(C + 1) \log_2(|\mathbb{F}|) \\ &= (C + mC + m) \log_2(|\mathbb{F}|)\end{aligned}$$

For simple lifting we must communicate

$$\mathcal{M}(C)/\log_2(|\mathbb{F}|^m) = C/m + C + 1 \tag{7}$$

bits per bit of soundness. So we can say as long as  $C/m + C > 2 \lceil C/k \rceil$ , BN++RMFE is better than BN++ with simple lifting.

We next relate Equations (6) and (7) on  $r = m/k$ , assuming (i)  $k$  divides  $C$  evenly, and (ii) the rate is a constant independent of  $C$  (as shown in [CCXY18, Theorem 5]) to get

$$\begin{aligned} C/m + C &> 2 \lceil C/k \rceil \\ (C + Cm)/m &> 2Cr/m \\ C + Cm &> 2Cr \\ m + 1 &> 2r \\ (m + 1)/2 &> r \end{aligned}$$

For our concrete choices,  $m = 17, r = 1.88$ , so this holds easily. For the smallest possible RMFE,  $m = 3, k = 2, r = 1.5$ , this still holds, so RMFE is always better than simple lifting in this analysis for BN++.

Another caveat remains in this comparison, since we've assumed that both the RMFE and simple lifting strategies use the same parameter  $m$ , but the optimal choices may be different. Letting  $m_1$  be the choice for BN++ with simple lifting and  $m_2$  be the choice for BN++RMFE, using an analysis similar as above we find that BN++RMFE is better if  $r < (m_2 + m_1 m_2)/(2m_1)$ . Looking at the smallest choice of RMFE,  $m_2 = 3$  and letting  $m_1$  be larger,  $r$  must be below 1.5, so the methods are tied. But once we allow  $m_2$  to be larger, e.g.,  $m_2 = 17$ , we must have  $r < 8.5$ , which always holds by [CCXY18, Theorem 5].

## C.2 Applying RMFE Lifting to Inverse Checking

The OWFs based on AES use only inversion in their nonlinear layers, and have no multiplications. An inverse pair  $(s, t)$  such that  $st = 1$  is a special type of multiplication triple  $(s, t, 1)$  so we can use a multiplication checking protocol for these OWFs. In this section we describe how to handle inversion triples (such as those in AES of the form  $(s, t, 1)$ , with  $st = 1$ ) in a way that allows the RMFE lifting technique to be used. When trying to apply the technique from Section 3.3 directly, we are faced with the following problems: The operands  $s$  and  $t$  would be mapped to  $\mathbb{K}$  as  $S = \phi(s)$  and  $T = \phi(t)$  and then the prover would compute  $S' = ST \in \mathbb{K}$  and inject shares of the product in  $\mathbb{K}$ . In contrast to traditional multiplication gates where the two inputs are already known to the parties, the prover also needs to inject shares of  $t$ , increasing the required communication. Additionally, even though we know that the result of the multiplications should be 1, we cannot make use of this fact and still need to communicate  $\Delta S'$ .

Since  $s$  and  $t$  are the same for all repetitions of the circuit, one could try to make  $S'$  public, since this would save the broadcast of  $\Delta S'$  in each repetition. However, due to the properties of the RMFE,  $S'$  leaks some information about  $s$  and  $t$ , and therefore we can only work with it in a secret-shared form, requiring the use of a per-repetition  $\Delta S'$ . The problem is that revealing  $S'$  reveals more than the fact that  $st = 1$ , since there are many  $S' \in \mathbb{K}$  such that  $\psi(X) = 1$ , and revealing a specific one leaks information about  $S$  and  $T$ .

The protocol proceeds as follows:

1. For each batch of  $k$  inversion gates with secret shared input  $\mathbf{s} = (s_\ell, \dots, s_{\ell+k-1})$ :

- (a) The prover computes  $S = \phi(\mathbf{s})$ .
  - (b) The prover injects  $\Delta t_j$ , so that the shares  $t_j^{(i)}$  read from the parties random tapes fulfill  $s_j t_j = 1$ .
  - (c) The prover computes  $T = \phi(\mathbf{t}) = \phi((t_\ell, \dots, t_{\ell+k-1}))$ .
  - (d) The prover computes  $S' = ST$ , and injects  $\Delta S'$ , so that the shares  $S'^{(i)}$  read from the parties' random tapes sum to  $S'$ .
2. The parties ensure the circuit output is correct:
    - (a) For each  $S'$ , calculate  $\mathbf{s}' = \psi(S')$  and check that  $\mathbf{s}' = (\mathbf{1}, \dots, \mathbf{1})$ . (This step is free using Opt. 2 from Section 2.4)
  3. Now the checking step runs, to ensure that for each RMFE-batched multiplication,  $S' = S \cdot T$ :
    - (a) This uses the batched checking protocol in Section 2.6. The simple lifting and multiple check strategies can also be applied directly to boost soundness.

In total, this protocol leads to larger signatures than alternative approaches such as Banquet [BdK<sup>+</sup>21], Limbo [dOT21] and our Helium-based variant in Section 5.3. When compared to checking regular multiplication triples with BN++ using RMFEs, checking inversions increases the size by an additional  $C$  field elements per repetition, as we must communicate  $\Delta t \in \mathbb{F}$  per inversion gate. For example, with AES-128,  $N = 256$  parties and the  $(5, 9)_{256}$ -RMFE we need  $\tau = 17$  repetitions, and signatures are estimated to be 18 881 bytes.

## D Descriptions of LowMC and Rain

In this section we review the core primitives used in the LowMC and Rain one-way functions.

### D.1 LowMC

LowMC [ARS<sup>+</sup>15] is a block cipher that is highly parameterizable, designed to have a small number of AND gates (low multiplicative complexity), to be suitable for MPC applications. When used as a OWF in Picnic-like signatures, as with other block cipher-based OWFs, key generation chooses a random key  $k$ , and a random plaintext block  $p$ , then computes  $c = E_k(p)$  and outputs  $k$  as the secret signing key and  $(c, p)$  as the public verification key.

Let  $n$  be the block size and key size,  $s$  be the number of S-boxes per round, and  $r$  the number of rounds. The parameters  $n$  and  $s$  may be chosen to suit the application. The LowMC specification defines random and independent constants for each set of parameter choices. Specifically,

- round constants  $R_i \in \mathbb{F}_2^n$ ,
- linear layer matrices  $L_i \in \mathbb{F}_2^{n \times n}$  (of full rank), and
- key matrices  $K_i \in \mathbb{F}_2^{n \times n}$  for the computation of round keys.

There are  $r$  round and linear layer constants  $R_i, L_i$ , and  $r + 1$  key matrices  $K_i$ . Keys are sampled uniformly at random from  $\mathbb{F}_2^n$ .

Encryption first adds a round key to the plaintext, which is followed by  $r$  rounds. Each round key is computed by multiplying the key with the key matrix  $K_i$ . A round is composed of an S-box layer, a linear layer, addition with constants, and addition of the round key as shown in Algorithm 5. The S-box layer applies the same 3-bit S-box on the first  $3 \cdot s$  bits of the state. The S-box is defined as  $S(a, b, c) = (a \oplus bc, a \oplus b \oplus ac, a \oplus b \oplus c \oplus ab)$ . The other layers only consist of  $\mathbb{F}_2$ -vector space arithmetic, all local operations in our MPC setting.

---

**Algorithm 5** LowMC encryption.

Parameters  $K_i, L_i$  and  $R_i$  are as described in the text.

---

**Input:** plaintext  $p \in \mathbb{F}_2^n$  and key  $k \in \mathbb{F}_2^n$   
state  $\leftarrow K_0 \cdot k \oplus p$   
**for**  $i \in [1, r]$  **do**  
state  $\leftarrow$  SBOXLAYER(state)  
state  $\leftarrow L_i \cdot$  state  $\triangleright$  LINEARLAYER  
state  $\leftarrow R_i \oplus$  state  $\triangleright$  CONSTANTADDITION  
state  $\leftarrow K_i \cdot k \oplus$  state  $\triangleright$  KEYADDITION  
**return** state

---

## D.2 Rain

Rain is a one-way function with a block-cipher based design [DKR<sup>+</sup>21]. It has a single large S-box per round, defined as inversion in  $\mathbb{F}_{2^n}$ , applied to the entire  $n$ -bit state, and the linear layer is multiplication by a randomly chosen matrix  $M \in \mathbb{F}_2^{n \times n}$ . As Rain is a OWF and not a classical block cipher, its design assumes that the attacker is given a single known (input, output) pair.

Rain is a keyed permutation  $F_k(x)$  with nonlinear operation  $S$  and a constant addition over a large field  $\mathbb{F}_{2^n}$ , and the linear layer. The S-box is  $S : \mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^n}$  such that

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

Let  $c^{(i)}$  denote the round constant in round  $i$ , and  $M_i \in (\mathbb{F}_2)^{n \times n}$  denote the linear layer matrix over  $\mathbb{F}_2$  used in round  $i$ . The permutation can be described by the round functions  $R_i$ , where each round function  $R_i$ , for  $i < r$  is defined as

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

and the final round  $R_r$  is defined as

$$R_r(x) = k + S \left( x + k + c^{(r)} \right).$$

A graphical overview of the construction is shown in Figure 7. For the 128-bit security level, Rain uses  $r = 4$  rounds, and the field is  $\mathbb{F}_{2^{128}}$  with the reduction polynomial  $X^{128} + X^7 + X^2 + X + 1$ . This polynomial is also used in AES-GCM [MV04]. The details of other parameters, and for the generation of the round constants and matrices are given in [DKR<sup>+</sup>21].

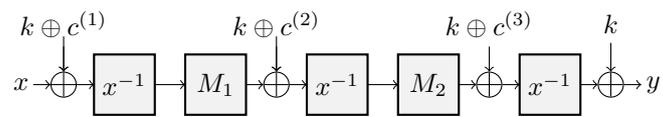


Figure 7: The Rain permutation with  $r = 3$  rounds.  $M_i$  denotes multiplication with the linear layer matrix over  $\mathbb{F}_2$  in round  $i$ .