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

(Preliminary Draft)

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

Greg Zaverucha
Microsoft Research
gregz@microsoft.com

October 29, 2021

1 Introduction

Picnic [CDG+17, ZCD+20] is a family of signature schemes built using zero-knowledge proof systems, relying solely on the security of symmetric-key primitives. It is an alternate candidate in the third round of the NIST post-quantum standardization project. During key-generation, a one-way function (OWF) is used to generate a keypair, the input to the one-way function is the signer's secret key and the output is the public key. A Picnic signature is a non-interactive proof of knowledge that one knows the secret input to the one-way function leading to the public output, with the message to be signed included in the generation of the challenge for the zero-knowledge proof.

Concretely, Picnic uses a block cipher as a one-way function: the block cipher key is the input to the OWF, the plaintext is chosen at random, and the public key is the corresponding plaintext-ciphertext pair. Importantly, the zero-knowledge proof needs to resist quantum attacks. One of the few known constructions that achieve this are so called "MPC-in-the-head" (MPCitH) [IKOS07] proofs. These proofs work by having the prover simulate an MPC protocol (with semi-honest security). The protocol computes the function that defines the relation (the OWF in the case of Picnic) and the prover commits to the internal state of all parties. Later, a subset of the parties are revealed based on a random challenge from the verifier, who can verify the consistency of the revealed parties and gain some assurance that the unopened parties also behave honestly, which in turn provides assurance that the prover knows the secret. Due to the properties of the MPC protocol, revealing the state of a subset of the parties does not leak any information about the secret input. Since the MPC protocol is information-theoretically secure and the proof additionally only uses a hash function, MPCitH proofs are resistant to known quantum attacks. Thus Picnic also has post-quantum security provided the OWF does.

The theoretical framework for MPCitH proofs was given in [IKOS07], and a first practical instantiation was presented in ZKBoo [GMO16]. An improved version of ZKBoo, ZKB++ [CDG+17] was used in the first version of Picnic. Since then, the area of MPCitH proof systems has seen multiple new instantiations with

different improvements and tradeoffs. The KKW proof system [KKW18] uses an MPC protocol with a preprocessing phase, allowing for more communication efficient online phases and a variable number of parties. Picnic later added parameter sets based on KKW, named Picnic2 and Picnic3.

While ZKB++ and KKW focus on binary circuits, the MPCitH framework also allows for arithmetic circuits over larger fields. This was for example used in constructions focusing on signatures with (variants of) AES as the OWF, such as BBQ [dDOS19], Banquet [BdK+21] and Rainier [DKR+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.

The BN proof system supports generic arithmetic circuits, but has very large proof sizes for OWFs defined over small fields. 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[1], and 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 $\mathbb{F}_{2^{8\lambda}}$ to increase the soundness. However, this also increases the size of the proof, since it must include elements of the larger field. The Limbo proof system [dSGOT21] also addresses this problem with lifting, and in [DKR+21] the authors explore alternative OWFs that are defined over large fields. 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 (used in Picnic).

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 LowMC we could for example lift each bit to a byte. But the lift is somewhat trivial, and the overhead, called the *rate*, is high (eight in this example), and $\mathbb{F}_8$ 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. A natural question is whether we can do better.

A *reverse multiplication friendly embedding* (RMFE) allows us to encode multiple field elements into 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 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 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 an efficient checking protocol is possible. We explore different constructions of RMFEs suitable for this application, and leverage a representation of the LowMC S-box that can be implemented with one $\mathbb{F}_{2^3}$ multiplication (rather than three ANDs). For Picnic and LowMC we end up using an optimal RMFE with

[1] This comes mainly from the fact that a random field element is chosen as a challenge.

rate 1.89, but our construction allows other RMFEs to be used as well.

1.1 Contributions

With the BN proof system [BN20] as our starting point, we design an improved version and apply it to a possible fourth version of Picnic, called Picnic4.

BN++ and Helium We optimize the checking protocol from [BN20] (where it is called the sacrificing-based proof protocol). At first we ignore the lifting problem, and reduce proof sizes by about 2.5x. We call this new protocol with optimizations BN++. Then we provide a soundness analysis for BN++ in the non-interactive case, in order to choose concrete parameters for signature schemes. Then we present *Helium*: a proof protocol that handles the small field case, improving upon BN++ by using RMFEs instead of trivial lifting.

Picnic4 For Picnic signatures, we focus on the LowMC circuit, first presenting an equivalent representation of the LowMC S-box, lifting three GF(2) multiplications to a single GF(2^3) multiplication. We then instantiate a Helium-based variant of Picnic. We give concrete parameters and implementation results. The resulting signature size is 7.987 KB at NIST security level L1, 1.6x shorter than Picnic3 (which has 12.5 KB signatures), the most recent version of Picnic. The implementation is ongoing so the running times are preliminary. Based on our reference implementation and experience with Banquet and Rainier, running times should be competitive with Picnic3, perhaps slightly higher.

1.2 Status of this work

This is a draft of a work-in-progress. The focus is Picnic and LowMC, to preview possible improvements to Picnic should NIST select it for the fourth round of its Post-Quantum Cryptography Standardization Process. [2] The implementation is preliminary. It confirms that the scheme works as expected and that the signature sizes are correct, but it is not optimized so we do not yet have good benchmarks for signing and verification time. The full version will have a broader scope, including an investigation of other Picnic-like signature schemes constructed by applying Helium to other OWFs.

1.3 Notation

In Table 1 we define some of the notation we use frequently in this document. 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" to correct the sharing:

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

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

[2] See `https://csrc.nist.gov/projects/post-quantum-cryptography`.

κ	Security parameter
N	Number of parties
$[x]$	The set $\{1, \ldots, x\}$
τ	Number of parallel repetitions
e	Index of repetition $e = 1, \ldots, \tau$
i	Index of party P_i, $i = 1, \ldots, N$
$\overline{i}, \overline{i}_e$	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
$\mathbb{F}$	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
ϕ	RMFE encoding function $\phi : \mathbb{F}^k \to \mathbb{K}$
ψ	RMFE decoding function $\psi : \mathbb{K} \to \mathbb{F}^k$

Table 1: Notation used frequently in this document.

2 Improving the BN Zero-Knowledge Proof

In this section we first review the Baum-Nof zero-knowledge protocol, as presented in [BN20], then give a series of four optimizations aimed at reducing the proof size. Later we refer to the variant using all our optimizations as BN++.

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 the approach by Katz, Kolesnikov and Wang [KKW18] of using an MPC protocol with a pre-processing step. The protocol in [BN20] uses an MPC protocol with standard multiplication triples (or Beaver triples [Bea92]), but instead of revealing the pre-processing phase of some iterations to gain assurance that unrevealed ones are also correct, they instead use a common technique from multi-party computation and show that a triple is correct by "sacrificing" another one. A special verifier party provides a random challenge, which is natural in the context of MPCitH proof protocols. In a traditional interactive MPC, the parties can jointly generate the challenge.

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 α and β 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 α and β.

The signature scheme based on the [BN20] protocol is depicted in Figure 1. The circuit is assumed to be a one-way function, with input sk and output ct (sk is the signing key and ct is the public key). For soundness the base protocol is repeated τ times in parallel. Several hash functions: Commit, H_1 and H_2 are required; as well as the two pseudorandom generators: Expand and $\mathsf{ExpandTape}$. All of these functions can be instantiated with the SHAKE128 extensible output function (or SHAKE256 for larger security levels), with different constants added for domain separation. The helper function $\mathsf{Sample}(t)$ samples elements from a random tape t that was output by $\mathsf{ExpandTape}$, 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.

$\mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$: **Phase 1: Committing to the seeds and views of the parties.**

Sample a random salt $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{2\kappa}$.
for each parallel repetition e **do**
  Sample a root seed: $\mathsf{seed}_e \xleftarrow{\$} \{0,1\}^{\kappa}$.
  Derive $\mathsf{seed}_e^{(1)}, \ldots, \mathsf{seed}_e^{(N)}$ as leaves of binary tree from seed_e.
  for each party i **do**
    Commit to seed: $\mathsf{com}_e^{(i)} \leftarrow \mathsf{Commit}(\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)})$.
    Expand random tape: $\mathsf{tape}_e^{(i)} \leftarrow \mathsf{ExpandTape}(\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)})$
    Sample witness share: $\mathsf{sk}_e^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
  Compute witness offset: $\Delta\mathsf{sk}_e \leftarrow \mathsf{sk} - \sum_i \mathsf{sk}_e^{(i)}$.
  Adjust first share: $\mathsf{sk}_e^{(1)} \leftarrow \mathsf{sk}_e^{(1)} + \Delta\mathsf{sk}_e$.
  for each multiplication gate with index $\ell \in [C]$ **do** ▷ Pre-processing triples
    For each party i, set $(a_{e,\ell}^{(i)}, b_{e,\ell}^{(i)}, c_{e,\ell}^{(i)}) \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$
    Compute $a_{e,\ell} = \sum_{i=1}^{N} a_{e,\ell}^{(i)}$, $b_{e,\ell} = \sum_{i=1}^{N} b_{e,\ell}^{(i)}$, $c_{e,\ell} = \sum_{i=1}^{N} c_{e,\ell}^{(i)}$
    Compute offset $\Delta c_{e,\ell} = a_{e,\ell} \cdot b_{e,\ell} - c_{e,\ell}$.
    Adjust first share: $c_{e,\ell}^{(1)} \leftarrow c_{e,\ell}^{(1)} + \Delta c_{e,\ell}$
  for each gate g in $\mathcal{C}$ with index ℓ **do** ▷ Online simulation
    if g is an addition gate with inputs (x, y) **then**
      The parties locally compute the output share $z^{(i)} = x^{(i)} + y^{(i)}$
    if g is a multiplication gate with inputs $(x_{e,\ell}, y_{e,\ell})$ **then**
      Compute output shares $z_{e,\ell}^{(i)} = \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
      Compute offset $\Delta z_{e,\ell} = x_{e,\ell} \cdot y_{e,\ell} - \sum_{i=1}^{N} z_{e,\ell}^{(i)}$.
      Adjust first share $z_{e,\ell}^{(1)} \leftarrow z_{e,\ell}^{(1)} + \Delta z_{e,\ell}$.
  Let $\mathsf{ct}_e^{(i)}$ be the output shares of online simulation.
Set $\sigma_1 \leftarrow (\mathsf{salt}, ((\mathsf{com}_e^{(i)})_{i\in[N]}, (\mathsf{ct}_e^{(i)})_{i\in[N]}, \Delta\mathsf{sk}_e, (\Delta c_{e,\ell}, \Delta z_{e,\ell})_{\ell\in[C]})_{e\in[\tau]}$.

Phase 2: Challenging the checking protocol.

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

Phase 3: Commit to simulation of checking protocol.

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

Phase 4: Challenging the views of the MPC protocol.

Compute challenge hash: $h_2 \leftarrow H_2(h_1, \sigma_2)$.
Expand hash: $(\bar{i}_e)_{e\in[\tau]} \leftarrow \mathsf{Expand}(h_2)$ where $\bar{i}_e \in [N]$.

Phase 5: Opening the views of the checking protocol.

for each repetition e **do**
  $\mathsf{seeds}_e \leftarrow \{\log_2(N) \text{ nodes needed to compute } \mathsf{seed}_{e,i} \text{ for } i \in [N] \setminus \{\bar{i}_e\}\}$.
Output $\sigma \leftarrow$
$(\mathsf{salt}, h_1, h_2, (\mathsf{seeds}_e, \mathsf{com}_e^{(\bar{i}_e)}, \Delta\mathsf{sk}_e, \mathsf{ct}_e^{(\bar{i}_e)}, (\Delta c_{e,\ell}, \Delta z_{e,\ell}, \alpha_{e,\ell}^{(\bar{i}_e)}, \beta_{e,\ell}^{(\bar{i}_e)}, v_{e,\ell}^{(\bar{i}_e)})_{\ell\in[C]})_{e\in[\tau]})$.

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

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 in this section reduce the proof size.

The total proof size of the protocol in Figure 1 is

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

where $\mathsf{M}(C) = 5C \log_2(|\mathbb{F}|)$ is the size of the checking protocol to ensure the multiplications are correct. Referring to Figure 1, the five field elements per multiplication are $(\Delta c_{e,\ell}, \Delta z_{e,\ell}, \alpha_{e,\ell}^{(\bar{i}_e)}, \beta_{e,\ell}^{(\bar{i}_e)}, v_{e,\ell}^{(\bar{i}_e)})_{\ell \in [C]}$. Also note that this assumes that the output of $\mathcal{C}$ is κ bits (as will be the case for Picnic). After Optimization 1, the signature size will be

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

Optimizations 2,3,4 will each reduce $\mathsf{M}(C)$, as summarized in Table 2. In Table 2 we also include the $\mathsf{M}(C)$ for the simple lifting (BN++ with lifting, Section 3.1) and Helium (Section 3.2) zero-knowledge proofs. The sizes of the additional parameters for these protocols are given in their respective sections.

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

Table 2: Summary of proof sizes after successive optimizations building up to BN++, and for the simple lifting (Section 3.1) and Helium (Section 3.2) zero-knowledge proofs. Here we give the size of the checking protocol, $\mathsf{M}(C)$, only. To get the full proof size, substitute the value of $\mathsf{M}(C)$ into Eq. (1). $\mathbb{K}$ is an extension field of $\mathbb{F}$, where the size depends on the construction and k is a parameter of the RMFE construction mapping $\mathbb{F}^k \to \mathbb{K}$.

2.3 Optimization 1: Removing the Output Broadcast

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

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

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

Intuitively, since all $\mathsf{ct}_e^{(i)}$ are input to the hash function in Phase 2, this ensures that the verifier is using the same shares of ct as the prover. This optimization saves including the unopened party's output broadcast message $\mathsf{ct}_e^{(\bar{i}_e)}$ in the proof, saving one output value per repetition. The proof size formula is given in Eq. (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 utilizes 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, $\mathsf{M}(C) = 4C \log_2(|\mathbb{F}|)$, and together with Eq. (1) we get the total proof size.

2.5 Optimization 3: Removing the Broadcast of β.

In the standard triple verification procedure from Section 2.1 the parties need to broadcast both α and β. 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 Δc 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. 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 $\alpha^{(i)} = \epsilon \cdot x^{(i)} + a^{(i)}$.
3. The parties open α by broadcasting their shares.
4. The parties locally set $v^{(i)} = \alpha \cdot y^{(i)} - \epsilon \cdot z^{(i)} - c^{(i)}$.

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 $(\alpha_{e,\ell}^{(\bar{i})}, \Delta c_{e,\ell}, \Delta z_{e,\ell})$ for each of the C multiplication gates, so the size of the checking protocol is $\mathsf{M}(C) = 3C\log_2(|\mathbb{F}|)$, which we can plug into the proof size formula given in Eq. (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 α and β, 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 $\ell = 1, \ldots, C$ 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 $(\epsilon_1, \ldots, \epsilon_C) \in \mathbb{F}^C$.
2. The parties locally set $\alpha_\ell^{(i)} = \epsilon \cdot x_\ell^{(i)} + a_\ell^{(i)}$.
3. The parties open $(\alpha_1, \ldots, \alpha_C)$ by broadcasting their shares.
4. The parties locally set

$$\begin{aligned} v^{(i)} = {} & \epsilon_1 z_1^{(i)} - \alpha_1 b_1^{(i)} \\ & + \epsilon_2 z_2^{(i)} - \alpha_2 b_2^{(i)} \\ & \cdots \\ & + \epsilon_C \cdot z_C^{(i)} - \alpha_C \cdot b_C^{(i)} \\ & - c^{(i)} \end{aligned}$$

   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 [dSGOT21, Lemma 4.1]. When compared to the multiplication check protocol in [dSGOT21], our protocol uses independent random challenges ϵ_i, rather than $(R, R^2, \ldots, R^{C-1})$ 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.[3]

First we recall the multivariate version of the Schwartz-Zippel lemma.

Lemma 1 (General Schwartz-Zippel lemma)**.** *Let $P(x_1, \ldots, x_n)$ 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,*

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

The total degree of P is the largest sum of the exponents in a term of P. For example, $P(x_1, x_2) = 4{x_1}^2{x_2}^3 + x_1{x_2}^2 + 1$ 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. Note that the protocol in Optimization 3 is 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:

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

Define a multivariate polynomial

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

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

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

Remark 3. When C and $|\mathbb{F}|$ are small, we can compute v exhaustively for all possible inputs (x, y, z, a, c) and get the exact probability that the protocol above succeeds. We find that the bound of Lemma 2 can be off by a factor of two

[3] Whether Limbo 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 [dSGOT21], 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.

which could lead to a big difference in the number of repetitions required for a sound protocol. For example, when $C = 1$ (multiplication triples are verified individually) and $\mathbb{F} = \mathrm{GF}(2)$, we found that the parties accept with probability 0.25, rather than 0.5 and with $N = 256$ parties this means that only 106 parallel repetitions are required, rather than 202, leading to proofs that are roughly twice as large. As a second example, when $\mathbb{F} = \mathbb{F}_3$ and $C = 2$, the exact probability is 0.274, slightly less than 1/3. We observe that, as C and/or $|\mathbb{F}|$ get larger, the empirically observed success probability (over many runs), is extremely close to $1/|\mathbb{F}|$ as guaranteed by Lemma 2.

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 $\bar{i}$. 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_ℓ is a uniform random value, and the simulated value of α_ℓ 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.

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

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

which together with Eq. (1) gives the proof size.

2.7 BN++: New Protocol with all Optimizations

In Fig. 2 we describe the new signing algorithm using optimizations 1, 2, 3 and 4. In Fig. 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.

$\mathsf{Sign}(\mathsf{sk}, \mathsf{msg})$: **Phase 1: Committing to the seeds and views of the parties.**

Sample a random salt $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{2\kappa}$.
for each parallel repetition e **do**
  Sample a root seed: $\mathsf{seed}_e \xleftarrow{\$} \{0,1\}^{\kappa}$.
  Derive $\mathsf{seed}_e^{(1)}, \ldots, \mathsf{seed}_e^{(N)}$ as leaves of binary tree from seed_e.
  for each party i **do**
    Commit to seed: $\mathsf{com}_e^{(i)} \leftarrow \mathsf{Commit}(\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)})$.
    Expand random tape: $\mathsf{tape}_e^{(i)} \leftarrow \mathsf{ExpandTape}(\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)})$
    Sample witness share: $\mathsf{sk}_e^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
  Compute witness offset: $\Delta\mathsf{sk}_e \leftarrow \mathsf{sk} - \sum_i \mathsf{sk}_e^{(i)}$.
  Adjust first share: $\mathsf{sk}_e^{(1)} \leftarrow \mathsf{sk}_e^{(1)} + \Delta\mathsf{sk}_e$.
  for each gate g in $\mathcal{C}$ with index ℓ **do**
    if g is an addition gate with inputs (x, y) **then**
      The parties locally compute the output share $z^{(i)} = x^{(i)} + y^{(i)}$
    if g is a multiplication gate with inputs $(x_{e,\ell}, y_{e,\ell})$ **then**
      Compute output shares $z_{e,\ell}^{(i)} = \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
      Compute offset $\Delta z_{e,\ell} = x_{e,\ell} \cdot y_{e,\ell} - \sum_{i=1}^{N} z_{e,\ell}^{(i)}$.
      Adjust first share $z_{e,\ell}^{(1)} \leftarrow z_{e,\ell}^{(1)} + \Delta z_{e,\ell}$.
      For each party i, set $a_{e,\ell}^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$
      Compute $a_{e,\ell} = \sum_{i=1}^{N} a_{e,\ell}^{(i)}$
      Set $b_{e,\ell} = y_{e,\ell}$
  Compute $c_e^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$
  Compute offset $\Delta c_e = \left(\sum_{\ell=1}^{|C|} a_{e,\ell} \cdot b_{e,\ell}\right) - c_e$.
  Adjust first share: $c_e^{(1)} \leftarrow c_e^{(1)} + \Delta c_e$
  Let $\mathsf{ct}_e^{(i)}$ be the output shares of online simulation.
Set $\sigma_1 \leftarrow (\mathsf{salt}, ((\mathsf{com}_e^{(i)}, \mathsf{ct}_e^{(i)})_{i\in[N]}, \Delta\mathsf{sk}_e, \Delta c_e, (\Delta z_{e,\ell})_{\ell\in[C]})_{e\in[\tau]}$.

Phase 2: Challenging the checking protocol.

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

Phase 3: Commit to simulation of the checking protocol.

for each repetition e **do**
  For the set of multiplication gates in $\mathcal{C}$, simulate the triple checking protocol 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.
Set $\sigma_2 \leftarrow (\mathsf{salt}, (((\alpha_{e,\ell}^{(i)})_{\ell\in[C]}, v_e^{(i)})_{i\in[N]})_{e\in[\tau]}$.

Phase 4: Challenging the views of the MPC protocol.

Compute challenge hash: $h_2 \leftarrow H_2(h_1, \sigma_2)$.
Expand hash: $(\bar{i}_e)_{e\in[\tau]} \leftarrow \mathsf{Expand}(h_2)$ where $\bar{i}_e \in [N]$.

Phase 5: Opening the views of the MPC and checking protocols.

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

Figure 2: BN++ Signature Scheme.

$\mathsf{Verify}(\mathsf{pk}, \mathsf{msg}, \sigma)$:

Parse σ as $(\mathsf{salt}, h_1, h_2, (\mathsf{seeds}_e, \mathsf{com}_e^{(\bar{i}_e)}, \Delta\mathsf{sk}_e, \Delta c_e, (\Delta z_{e,\ell}, \alpha_{e,\ell}^{(\bar{i}_e)})_{\ell\in[C]})_{e\in[\tau]})$.
Expand hashes: $(\epsilon_{e,\ell})_{e\in[\tau],\ell\in[C]} \leftarrow \mathsf{Expand}(h_1)$, and $(\bar{i}_e)_{e\in[\tau]} \leftarrow \mathsf{Expand}(h_2)$.
for each repetition e **do**
    Use seeds_e to recompute $\mathsf{seed}_e^{(i)}$ for $i \in [N] \setminus \bar{i}_e$.
    for each party $i \in [N] \setminus \bar{i}_e$ **do**
        Recompute $\mathsf{com}_e^{(i)} \leftarrow \mathsf{Commit}(\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)})$,
        $\mathsf{tape}_e^{(i)} \leftarrow \mathsf{ExpandTape}(\mathsf{salt}, e, i, \mathsf{seed}_e^{(i)})$, and
        $\mathsf{sk}_e^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
        if $i = 1$ **then** adjust first share: $\mathsf{sk}_e^{(i)} \leftarrow \mathsf{sk}_e^{(i)} + \Delta\mathsf{sk}_e$.
        for each gate g in $\mathcal{C}$ with index ℓ **do**
            if g is an addition gate with inputs $(x^{(i)}, y^{(i)})$ **then**
                locally compute the output share $z^{(i)} = x^{(i)} + y^{(i)}$
            if g is a multiplication gate, with inputs $(x_{e,\ell}^{(i)}, y_{e,\ell}^{(i)})$ **then**
                Compute output share $z_{e,\ell}^{(i)} = \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
                if $i = 1$ **then** adjust first share $z_{e,\ell}^{(i)} \leftarrow z_{e,\ell}^{(i)} + \Delta z_{e,\ell}$.
                Set $a_{e,\ell}^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$, and $b_{e,\ell}^{(i)} = y_{e,\ell}^{(i)}$
        Set $c_e^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$
        if $i = 1$ **then** adjust first share $c_e^{(i)} \leftarrow c_e^{(i)} + \Delta c_e$.
        Let $\mathsf{ct}_e^{(i)}$ be party i's share of the circuit output.
    Compute $\mathsf{ct}_e^{(\bar{i}_e)} = \mathsf{ct} - \sum_{i\neq\bar{i}_e} \mathsf{ct}_e^{(i)}$
Set $\sigma_1 \leftarrow (\mathsf{salt}, ((\mathsf{com}_e^{(i)}, \mathsf{ct}_e^{(i)})_{i\in[N]}, \Delta\mathsf{sk}_e, \Delta c_e, (\Delta z_{e,\ell})_{\ell\in[C]})_{e\in[\tau]}$.
Set $h_1' = H_1(\mathsf{salt}, \mathsf{msg}, \sigma_1)$
for each repetition e **do**
    for each party $i \in [N] \setminus \bar{i}_e$ **do**
        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.
Compute $v_e^{(\bar{i}_e)} = 0 - \sum_{i\neq\bar{i}_e} v_e^{(i)}$
Set $\sigma_2 \leftarrow (\mathsf{salt}, (((\alpha_{e,\ell}^{(i)})_{\ell\in[C]}, v_e^{(i)})_{i\in[N]})_{e\in[\tau]}$.
Set $h_2' = H_2(h_1, \sigma_2)$.
Output accept iff $h_1' \stackrel{?}{=} h_1$ and $h_2' \stackrel{?}{=} h_2$.

Figure 3: BN++ Verification algorithm.

3 Handling Small Field Sizes 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+21], and its nonlinear operations are defined over $\mathrm{GF}(2^{128})$. The proof size of BN++ with Rain is slightly smaller than Rainier, and the implementation of BN++ is arguably simpler, as it does not require polynomial interpolation or arithmetic, as in Rainier.

However, for Picnic and the LowMC cipher, we have 516 binary AND gates, and signatures with BN++ are estimated to be about 78 KB. This compares to 12.5 KB for the Picnic3 parameters (based on the [KKW18] proof system).

Clearly the BN++ protocol is not competitive with existing solutions when the field size is small. In this section we address this limitation.

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+21] and Limbo [dSGOT21]. 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 ϵ must be in $\mathbb{K}$ for the soundness analysis, which means that a_ℓ must also be in $\mathbb{K}$ to ensure that x_ℓ is hidden in the computation of the public value α_ℓ. Because of this, the dot product triple $(\mathbf{a}, \mathbf{b}, c)$ is also in $\mathbb{K}$. Recall that in BN++ we must communicate $(\Delta z_{e,\ell}, \alpha_{e,\ell}^{(\bar{i})})$ once per gate and Δc_e once per repetition. From the discussion above, $\Delta z_{e,\ell} \in \mathbb{F}$, $\alpha_{e,\ell}^{(\bar{i})}$ and Δc_e are in $\mathbb{K}$.

Then the proof/signature size formula given in Eq. (1) has $\mathsf{M}(C) = C(\log_2(|\mathbb{F}|) + \log_2(|\mathbb{K}|)) + \log_2(|\mathbb{K}|)$. 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, while in the $\mathrm{GF}(2^3)$ case each group of 3 bits are lifted to one $\mathrm{GF}(2^3)$ value, for a rate of 1, but the three bit field is still quite small. When we lift from $\mathrm{GF}(2^3)$ to $\mathrm{GF}(2^{12})$ (rate 4) we get the best signature size, about 2 KB more than Picnic3.

Binary LowMC (516 AND gates):	27.2 KB when lifting to $\mathbb{K} = \mathrm{GF}(2^8)$
Binary LowMC (516 AND gates):	21.1 KB when lifting to $\mathbb{K} = \mathrm{GF}(2^8)$
$\mathrm{GF}(2^3)$ LowMC (172 mults):	23.9 KB without lifting
$\mathrm{GF}(2^3)$ LowMC (172 mults):	14.5 KB when lifting to $\mathbb{K} = \mathrm{GF}(2^{12})$

Table 3: Proof size estimates for LowMC with the simple lifting protocol, with $N = 256$ parties. In the second row, we apply the optimization in Appendix B.2.

3.2 Helium: Lifting with RMFEs

In the previous section, with binary LowMC, we took one bit and lifted it to an 8-bit field, then did the checking protocol in the extension field, and called the rate of this lift 8, since 8 bits must be communicated in place of one. This helps soundness, but the lift is somewhat trivial, and resulting rate is high. A natural question is whether 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.

3.2.1 RMFE Preliminaries

We start with a definition, adapted from [CCXY18].

Definition 4. 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:

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

We say that (ϕ, ψ) is a *reverse multiplication friendly embedding*, denoted $(k,m)_q$–RMFE, if

1. ϕ and ψ 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 $\phi : \mathbb{F}^k \to \mathbb{K}$ and $\psi : \mathbb{K} \to \mathbb{F}^k$.

Concrete RMFEs For Picnic using LowMC, our goal will be to lift elements of GF(2^3) to a larger field (after representing the LowMC S-box as a GF(2^3) multiplication, see Section 4.1.). In [CCXY18, Lemma 4] a construction of RMFE based on interpolation codes is given which gives us the best (currently) known choice for our application: we have a $(9,17)_{2^3}$–RMFE with rate 1.88. 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 signature sizes). A construction with improved rate and similarly sized $\mathbb{K}$ would immediately give shorter signatures, e.g., rate 1.6 would reduce signature sizes by about 1 KB. Unfortunately

[CCXY18, Lemma 4] is optimal[4] so there is no better RMFE from GF(2^3). In Section 4.2 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 ϕ and ψ, then encoding and decoding can each be done with a matrix multiplication.

3.2.2 The Helium Proof Protocol: Multiplication Checking with RMFEs in BN++

Recall the multiplication checking protocol of BN++. 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_ℓ, y_ℓ, z_ℓ) for $\ell = 1, \ldots, C$. We group these into vectors from $\mathbb{F}^k$, denoted $(\mathbf{x}_j, \mathbf{y}_j, \mathbf{z}_j)$ for $j = 1, \ldots, C/k$.
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 ϕ is linear. In BN++ the prover provides Δz_j in order to inject the result of the multiplication gate, but in Helium 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 ΔZ).

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

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

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

[4]Personal communication from Ignacio Cascudo; co-author of [CCXY18].

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

Lemma 5. *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 (ϕ, ψ) required by Definition 4. Thus given that (X, Y, Z) are 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.

□

Note that it is important to compute (shares of) $\mathbf{z}$ with ψ as we do in the protocol, since it is not guaranteed that 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 run separately 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 $\mathbb{F}^k \to \mathbb{K}$, the size of a Helium proof is given by the formula:

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

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

3.2.3 Parameter Selection

In this section we describe how to choose parameters to instantiate secure non-interactive proofs with Helium. Since the NI proof is a canonical 5-round protocol constructed with the Fiat-Shamir transform, we can apply the existing analysis in [KZ20, §4.1], similar to Banquet [BdK+21] (though it is seven rounds) and Rainier [DKR+21].

We provide an attack strategy that minimizes the attack cost, where we cheat τ_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 $1/|\mathbb{K}|$, by Lemma 5. For the remaining τ_2

instances he must cheat in the MPC computation of one party, and hope that the selected party remains unopened.

The cost of the attack is given by

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

where SPMF is the summed probability mass function,

$$\mathsf{SPMF}(n, k, p) = \sum_{k'=k}^{n} \binom{n}{k'} p^{k'} (1-p)^{n-k'} \,,$$

where each term gives the probability of guessing correctly in k' of τ independent trials, each with success probability p. The choice of τ_1 that minimizes the attack cost gives the optimal attack

$$\tau_1 = \arg\min_{0 \le \tau' \le \tau} \frac{1}{\mathsf{SPMF}(\tau, \tau', 1/\mathbb{K})} + N^{\tau - \tau'} \,.$$

To select secure parameters, we fix κ and N and $\mathbb{K}$, then increase τ until the cost of the best attack exceeds 2^κ. A script implementing this formula was used to generate the concrete parameters given in the next section.

As with many MPCitH protocols, τ is the parameter that affects proof sizes the most and computational cost is most sensitive to the choice of N. Once $\mathbb{K}$ is moderately large (easily achieved due to the RMFE lift), N becomes the bottleneck for soundness, and τ decreases only as N increases (and the relationship is exponential). Thus a size-speed tradeoff curve is present in Helium, similar to other MPCitH proofs, where smaller signatures have large N and are slower to create and verify, and larger signatures may be much faster.

4 Picnic4

This section describes how the Helium proof system can be used to instantiate a fourth version of the Picnic signature scheme. Here $\mathcal{C}$ is the LowMC block cipher. We start by describing an alternate representation of the LowMC S-box, then give options for concrete parameter sizes.

4.1 Alternative Representation of the LowMC S-box

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

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

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 $\mathbb{F}_2$.

It can easily 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.

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
return $(d, d+e, f)$	▷ final linear transformation

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+21] and Limbo [dSGOT21] which must lift inputs of multiplication gates to larger fields by essentially padding them (as discussed in Section 3.1).

Additionally, when we use an RMFE in 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 (when $\mathbb{K}$ is large enough to be useful in our application).

4.2 Concrete Parameters and Signature Size

The remaining important parameter choice is the RMFE. 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, 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 $\mathbb{K}$, which ends up being a bottleneck, and signatures are about 20 KB and do not change much in size as N is increased or decreased.

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 1626 to decrease signatures to 8.6KB. 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 jump from 5 bits to 51 bits and keeps the rate low at 1.89. The second half of Table 4 shows the options for multiple choices of N.

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) gave strictly larger signatures.

Finally, we note that with an additional circuit-specific optimization, see Appendix B.2, the sizes in the first half of Table 4 can be reduced slightly, but

RMFE	Rate	$\mathbb{F}$	$\mathbb{K}$	$\lceil C/k \rceil$	N	τ	Size (KB)
$(3,5)_2$	1.6	$\mathbb{F}_2$	$\mathbb{F}_{2^5}$	172	255	49	20.07
$(30,95)_2$	3.16	$\mathbb{F}_2$	$\mathbb{F}_{2^{95}}$	18	256	17	10.51
$(30,95)_2$	3.16	$\mathbb{F}_2$	$\mathbb{F}_{2^{95}}$	18	1626	13	8.61
$(18,51)_2$	2.83	$\mathbb{F}_2$	$\mathbb{F}_{2^{51}}$	29	256	18	10.25
$(18,51)_2$	2.83	$\mathbb{F}_2$	$\mathbb{F}_{2^{51}}$	29	1626	14	8.58
$(9,17)_8$	1.89	$\mathbb{F}_{2^3}$	$\mathbb{F}_{2^{51}}$	20	57	24	9.87
$(9,17)_8$	1.89	$\mathbb{F}_{2^3}$	$\mathbb{F}_{2^{51}}$	20	256	18	7.99
$(9,17)_8$	1.89	$\mathbb{F}_{2^3}$	$\mathbb{F}_{2^{51}}$	20	371	17	7.82
$(9,17)_8$	1.89	$\mathbb{F}_{2^3}$	$\mathbb{F}_{2^{51}}$	20	921	15	7.15
$(9,17)_8$	1.89	$\mathbb{F}_{2^3}$	$\mathbb{F}_{2^{51}}$	20	1626	14	6.91
$(9,17)_8$	1.89	$\mathbb{F}_{2^3}$	$\mathbb{F}_{2^{51}}$	20	2^{16}	10	5.76

Table 4: Parameters for different RMFE choices and parameters for Picnic signatures implemented with the Helium proof system and the full S-box layer LowMC parameters (as used in Picnic3) which requires 516 AND gates, or 172 GF(2^3) multiplies.

the parameters in the second half of the table still give shorter signatures.

Tentative Picnic4 Parameter Sets We recommend the $(N, \tau) = (256, 18)$ parameter set with 7987 byte signatures for L1 (the highlighted row in Tables 4 and 5). Using LowMC with the partial S-box layer increases the size slightly to 8658 bytes. Until we have an optimized implementation it is hard to say what performance will be like with different choices of N. Optimistically it will be similar to Rainier [DKR+21, Table 3, page 47], and $N = 256$ will have Picnic3-like performance. Realistically the LowMC computations will probably cost more than Rain, and the performance will be somewhat worse. Table 5 shows the speed and sizes of the current implementation with multiple choices of N.

The L3 and L5 parameter choices require further investigation. One possibility for L5 uses $N = 256$, $\tau = 37$ and the RMFE in the highlighted row to get 31.347 KB signatures, an improvement over the 48.4 KB for Picnic3-L5.

4.3 Picnic4 Implementation

Our implementation is preliminary: we have an unoptimized C++ implementation demonstrating the correctness of the scheme and confirming the signature sizes. Some benchmarks from the current implementation are given in Table 5.

4.4 Signature Scheme Security

Informally, the signature scheme's security analysis (in the random oracle model (ROM)) will follow Rainier's very closely, since they are both 5-round protocols with the same structure, the main difference being the way that Helium checks multiplication triples. This difference is largely covered by Lemma 2 and the simulation-based argument for $N - 1$ privacy of the checking protocol following it.

The Rainier analysis in turn uses a similar strategy as in the Picnic3/KKW security proof. 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 ROM 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.

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 Helium. But since Helium is a commit-and-open proof, 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 Σ-protocols, seems like the most direct approach to a QROM analysis of Helium (assuming [DFMS21] can be generalized from three to five round protocols).

5 Conclusion

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

[BdK+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.

[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.

[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+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.

[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`.

[DKR+21] Christoph Dobraunig, Daniel Kales, Christian Rechberger, Markus Schofnegger, 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`.

[dSGOT21] Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, and Titouan Tanguy. Limbo: Efficient zero-knowledge mpcith-based arguments. *IACR Cryptol. ePrint Arch. Report 2021/215*, 2021. `https://eprint.iacr.org/2021/215`.

[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.

[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.

[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.

[KZ20] 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.

[ZCD+20] Greg Zaverucha, Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, Jonathan Katz, Xiao Wang, Vladmir 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`.

N	τ	Sign	Verify	Size
16	34	3.98	3.84	12 825
57	24	7.34	7.02	9 849
107	21	11.22	10.75	8 966
256	18	21.60	21.20	7 987
1626	14	98.36	98.77	6 906
`Picnic1-full`		1.0	0.8	30 821
`Picnic3`		5.17	3.96	12 595

Table 5: Benchmarks for Picnic4 with various choices of N (top half) and current versions of Picnic (lower half) for reference. Times are in ms on an Intel Xeon W-1233 @ 3.6 GHz and sizes are in bytes. All Picnic instances use the LowMC parameters with a full S-box layer.

A Some Preliminary Benchmarks

In Table 5 we provide some preliminary benchmarks showing the various sizes and running times of Picnic4 when the number of parties N varies from 16 to 1626. As mentioned earlier, we are leaning towards $N = 256$ but some of the options with smaller N still significantly improve on the sizes of Picnic1 and Picnic3, while being much faster. All instances in the table use the LowMC parameters with a full S-box layer and are at security level L1.

B Other Optimizations

In this section we describe some optimizations that we passed over in the Picnic/LowMC application, but that might be useful in other contexts.

B.1 Optimization: Multiplications with Public Outputs

In the BN++ protocol, we are communicating two elements per gate, $(\Delta z, \alpha)$. When z is a public value, then Δz does not need to be sent. We aren't aware of many circuits where multiplication outputs are public. One example is an RSA modulus.

B.2 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 α 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 α_1 and α_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).

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

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

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

$$3\kappa + \tau \cdot (2\kappa + \kappa \cdot \lceil \log_2(N) \rceil + \mathsf{M}(C) + \log_2(|\mathbb{K}|) + \kappa)\,.$$

where $\mathsf{M}(C) = \frac{5}{3} \lceil C/k \rceil \cdot (\log_2(|\mathbb{K}|))$. For the parameters with the $(18, 51)_2$–RMFE with $N = 256$ and $\tau = 18$ the signature size goes from 10.25 KB to 9.15 KB.