Banquet: Short and Fast Signatures from AES*

Carsten Baum[1], Cyprien Delpech de Saint Guilhem[2], Daniel Kales[3], Emmanuela Orsini[2], Peter Scholl[1], and Greg Zaverucha[4]

[1] Dept. Computer Science, Aarhus University, Aarhus, Denmark.
[2] imec-COSIC, KU Leuven, Leuven, Belgium.
[3] Graz University of Technology, Graz, Austria.
[4] Microsoft Research

Abstract. This work introduces Banquet, a digital signature scheme with post-quantum security, constructed using only symmetric-key primitives. The design is based on the MPC-in-head paradigm also used by Picnic (CCS 2017) and BBQ (SAC 2019). Like BBQ, Banquet uses only standardized primitives, namely AES and SHA-3, but signatures are more than 50% shorter, making them competitive with Picnic (which uses a non-standard block cipher to improve performance). The MPC protocol in Banquet uses a new technique to verify correctness of the AES S-box computations, which is efficient because the cost is amortized with a batch verification strategy. Our implementation and benchmarks also show that both signing and verification can be done in under 10ms on a current x64 CPU. We also explore the parameter space to show the range of trade-offs that are possible with the Banquet design, and show that Banquet can nearly match the signature sizes possible with Picnic (albeit with slower, but still practical run times) or have speed within a factor of two of Picnic (at the cost of larger signatures).

1 Introduction

Digital signatures such as RSA and ECDSA form the backbone of authentication mechanisms on the Internet, and are central to many uses of cryptography. While being highly efficient, it is also known that in the presence of quantum computers, nearly all such currently deployed constructions will be rendered insecure.

This has led to a search for efficient, plausibly post-quantum-secure constructions of digital signatures that do not rely on assumptions such as factoring or the hardness of the discrete logarithm problem. There are a multitude of candidates with different design ideas: some rely on plausibly hard problems over (structured) lattices [24, 33, 21], others rely on the hardness of certain problems from multivariate cryptography [13] or isogenies [15].

While these signature schemes all rely on number-theoretic, or structured hardness assumptions, digital signatures can be built from symmetric-key primitives alone, offering the possibility of relying on "less structured" and better-studied assumptions compared with public-key encryption, where this is not

* The full version of this paper is available as IACR ePrint Report 2021/068 [4].

known to be possible. The Picnic signature scheme [14, 36] is one example which uses the following paradigm: the public key is a plaintext-ciphertext pair (x, y) while the secret key k defines a one-way function F_k, instantiated with a block cipher. A signature is then a non-interactive zero-knowledge proof showing that the prover (the signer) knows a key k such that $y = F_k(x)$.

As the proof size depends on proving statements about the chosen block cipher in zero knowledge, the authors of Picnic chose a special block cipher called LowMC [1] that is optimized for a low number of AND gates in its Boolean circuit representation. While LowMC has so far withstood cryptanalytic efforts in the context of Picnic, its security is still less established than standards like AES, and there have been attacks on older versions of LowMC [18, 35] or in applications where more than one plaintext-ciphertext pair is available [32].

MPC (in the Head). The zero-knowledge proof of the Picnic scheme is based on the so-called MPC-in-the-Head (MPCitH) paradigm [27]. We will now explain this paradigm in more detail before we continue to describe our contributions.

We consider interactive multi-party computation (MPC) protocols, where a set of N parties $\mathcal{P}_1, \ldots, \mathcal{P}_N$ securely and correctly evaluate a function f on a secret input x. In order to do so, each $\mathcal{P}_i$ obtains as input a secret share of the input x. Then, all parties together run an interactive computation where they exchange messages with the goal of ending up with a secret-sharing of the output, $f(x)$. These shares are then sent to whomever should learn the output. Assuming an adversary controls at most t of the N parties (i.e., a corruption threshold t), the MPC protocol guarantees that no information beyond $f(x)$ is leaked. For the types of MPCitH protocols related to this work, we only consider the case of passive security, where all parties are assumed to follow the protocol. Typically, the function f is described as a Boolean circuit or an arithmetic circuit over a finite field $\mathbb{F}$. We say that the MPC protocol uses preprocessing if, before the input x is specified, $\mathcal{P}_1, \ldots, \mathcal{P}_N$ are given secret-shares of correlated, random values that are sampled according to some known distribution.

The idea of MPCitH is that such an MPC scheme can be turned into a zero-knowledge proof, in the following way: the witness of a statement is secret-shared as the input x, while f will be a verification function that outputs 0 iff x is a valid witness. The prover *simulates* all N parties locally (hence the name *in the head*) and sends the verifier commitments to each parties' input shares, secret random tapes, and all communicated messages. Additionally, it sends the output shares of the parties (that should reconstruct to 0) to the verifier. Then, the verifier randomly chooses t of the parties' commitments to be opened, and verifies that the committed messages are consistent with an honest execution of the MPC protocol according to the opened input shares and random tapes, and the previously received output shares. Since only t parties are opened, the verifier learns nothing about the secret input x, while the random choice of the opened parties ensures that enough simulations of the other parties must also be correct (with some probability), ensuring soundness of the proof. Typically, to make the soundness error small enough, several parallel repetitions of the basic protocol are carried out.

Digital Signatures from MPCitH. To obtain a signature scheme, Picnic uses an MPCitH protocol to prove knowledge of a secret key k such that $y = F_k(x)$, for a one-way function F_k. This gives a zero-knowledge identification protocol, which can be turned into a signature scheme with the Fiat–Shamir transform. Making this approach practical requires an MPC protocol that efficiently evaluates the one-way function since the communication complexity of most MPC protocols scales with the number of AND (or multiplication) gates in the circuit, and multiple parallel repetitions are required.

As mentioned earlier, Picnic uses a one-way function based on the LowMC cipher, which has a relatively low AND gate count (around 600, at the 128-bit security level). However, this means that Picnic relies inherently on a non-standard assumption. A direct instantiation using, say, AES instead of LowMC would remove this assumption, but increase the signature size by around 4–5x due to the much larger AND gate count in the AES circuit. Since AES has a relatively compact description using operations over $\mathbb{F}_{2^8}$, it is natural to also consider using MPC protocols tailored for this field instead of $\mathbb{F}_2$. This was the approach recently taken in the BBQ scheme [16], which reduced AES-based signature sizes by 40% compared to the binary circuit approach. However, AES-based signatures are still about 2-3x larger than LowMC-based signatures.

1.1 Our Contributions

In this work, we present Banquet, a novel approach to constructing AES-based identification and signature schemes. Compared with the previous AES-based approach in BBQ [16], Banquet reduces signature size, and our implementation results show that Banquet can be made almost as efficient as Picnic.

First, we leverage the prover's full knowledge of the secret MPC input in an MPCitH protocol to design a protocol that does not "recompute" the AES function, but "verifies" it instead. As it turns out, verifying that $y = \mathsf{AES}_k(x)$ for public x, y is cheaper than encrypting x under k, as the prover knows all AES evaluation intermediate states, which must then be shown to be consistent.

To achieve this, we construct a new test for batch verification of multiple intermediate states in the AES encryption circuit at once, instead of verifying them individually. This batch test also allows us to avoid a preprocessing step, unlike previous work (BBQ and Picnic), reducing costs of the entire protocol. Our amortization technique does come with higher computational costs due to polynomial operations, and an increase in the number of rounds from 3 or 5 in previous MPCitH constructions up to 7, which affects concrete security bounds when using Fiat–Shamir.

We show that our approach reduces signatures sizes when compared with BBQ by around 50%, for the same security level. Interestingly, these signatures are now very close in size to Picnic[5], meaning that Banquet is comparable in signature size to the state of the art of MPCitH-based signatures while using a standard block cipher.

See Table 1 for a high-level comparison of the signature size and run times.

[5] Using the Picnic3 parameter sets, see [30].

Protocol	N	Sign (ms)	Verify (ms)	Size (bytes)
Picnic2	64	41.16	18.21	12 347
	16	10.42	5.00	13 831
Picnic3	16	5.33	4.03	12 466
AES Bin	64	-	-	51 876
BBQ	64	-	-	31 876
Banquet	16	6.36	4.86	19 776
	107	21.13	18.96	14 784

Table 1. Signature size (in bytes) and run times (if available) for Picnic2, Picnic3, AES Binary, BBQ and Banquet for comparable MPCitH parameters and 128 bit security, where N denotes the number of MPCitH parties.

Moreover, we provide a full prototype implementation of Banquet that works for a wide variety of parameter sets, and give a detailed analysis of how to securely choose parameters. At the 128-bit classical security level (corresponding to NIST's L1 category), our optimized implementation can sign messages in 6.4ms, about 1.2x slower than the 5.3ms required for Picnic3 (the latest version of Picnic), while our signature size is 19.8kB, around 50% larger than Picnic3's 12.5kB. At higher security levels (NIST's L3 and L5), Banquet's signing speed ranges from 1.5–10x slower than Picnic, and signatures are 1.2–2x larger, depending on the choices of parameters.

At all security levels, our signature size is around 40-50% smaller than the AES-based scheme BBQ, which is itself much smaller than a naive AES-based scheme instantiated with a binary circuit and MPCitH [31] (denoted AES Bin in Table 1). Note that there is no reported implementation of BBQ, so we cannot compare runtimes here.

SPHINCS+ [6] is a hash-based signature scheme based on standard, symmetric-key primitives, like Banquet. At NIST's L1 security level, we show that Banquet can outperform the SPHINCS+ -fast parameters in both signature size and signing time, but with slower verification. At higher security levels SPHINCS+ obtains smaller signatures, but sometimes with slower signing time.

1.2 Our Techniques

Our main building block to achieve smaller signatures in Banquet is a new amortized inverse verification check, which replaces the solution from BBQ. Recall that in AES, the only non-linear operations are the S-box evaluations, which can be seen as inversions in $\mathbb{F}_{2^8}$, with the difference that 0 maps to 0. In BBQ, these were evaluated with preprocessed multiplication triples based on techniques from MPC, with a modification to the AES key generation phase to ensure that 0 is never inverted (which reduces the key space by 1 bit).

At a high level, our approach for verifying inverses works in four steps.

Inject inverses. We provide the outputs of all m S-box evaluations already as part of the witness, and only verify that the product of the input and output for each of the m S-box instances is 1. Already, this optimization simplifies the

original BBQ construction and reduces its main communication cost down from 4 to 3 field elements per S-box (per MPC execution).

Amortize the verification of multiplicative relations. Then, we observe that it is not necessary to multiply all m S-box inputs with all outputs that are claimed to be correct independently. Instead, one multiplication of secret shared values is sufficient. This is done by arranging all inputs and outputs of the S-boxes into polynomials $S(\cdot)$ and $T(\cdot)$ (which can be done using free linear operations), providing a product polynomial P (that guarantees the required relation) as part of the witness and checking that $P(\cdot)$ is indeed the product of $S(\cdot)$ and $T(\cdot)$ by performing only one multiplication by evaluating at a random point. The prover additionally needs to send m field elements to specify the polynomial $P(\cdot)$.

Boost the soundness. The product test, which relies on the Schwartz–Zippel Lemma, has soundness that depends on the gap between the degree of $P(\cdot)$ and the size of the underlying field. In $\mathbb{F}_{2^8}$ this gives quite poor soundness in practice, so we boost soundness by lifting all polynomials to an extension field and performing the test there.

Reduce communication with linear combinations. Finally, we split our test from one polynomial triple $S(\cdot), T(\cdot), P(\cdot)$ with S, T of degree m into $\approx \sqrt{m}$ polynomial triples with factors of degree $\approx \sqrt{m}$ each. This stage is inspired by a recent zero-knowledge proof technique which was used to verify a batch of low-degree relations [10]. To verify these polynomial products, we then take a random linear combination of the polynomials to reduce the verification of all S-box evaluations down to a single check on polynomials of degree $\approx \sqrt{m}$, which reduces communication further.

1.3 Related Work

MPCitH was introduced in [27] and first shown to be feasible in practice with the ZKBoo protocol [25] and its follow-up work ZKB++ [14]. Later, Katz et al. introduced preprocessing to MPCitH-type protocols [31], which was further developed by Baum and Nof [3]. This paradigm of constructing signatures from MPCitH was first used in Picnic [14] and further developed in several later works [31, 28, 7, 8, 30].

Other paradigms for generic zero-knowledge proofs from symmetric-key primitives could also be used to instantiate our approach, such as Ligero [2] or Aurora [5]. These systems are not particularly well-suited to signatures, however, since despite good asymptotic performance, their proof sizes are quite large for small circuits like AES. For instance, a recent variant of Ligero was suggested for constructing post-quantum signatures [9], but when using AES as the one-way function, it achieves a signature size of 224KB and signing time of 256ms, which (although on different hardware) are both significantly worse than Banquet.

A recent, alternative suggestion for MPCitH is to use the Legendre symbol as a one-way function [8]. This improves upon Picnic in terms of both speed and

signature size, but also relies on a somewhat esoteric assumption, which has not seen much quantum cryptanalysis.

In addition to Picnic, there are several other signature schemes in the 3rd round of the NIST Post Quantum Cryptography Standardization Process[6]. The current finalists are the two lattice-based proposals Dilithium [21] and Falcon [23], as well as Rainbow [17], based on multivariate polynomials. "Alternate Candidates" are, in addition to Picnic, the SPHINCS+ [6] framework which is hash-based and GeMSS [13] which also uses multivariate polynomials. We will compare Banquet to Picnic, BBQ and SPHINCS+ since they are all based on symmetric key primitives.

2 Preliminaries

We let κ denote the computational security parameter. We denote by $[n]$ the set of integers $\{1, \ldots, n\}$ and by $[0, n]$ the set $\{0, \ldots, n\}$ with 0 included. We use L1, L3 and L5 to refer to the three security levels defined by NIST [34] in the call for post-quantum proposals, which match or exceed the brute-force security of AES-128, AES-192 and AES-256, respectively.

2.1 Definitions

We use the standard security notion for digital signature schemes, namely, existential unforgeability under adaptive chosen-message attacks (EUF-CMA) [26]. As a stepping stone to EUF-CMA security, we will first prove security of our scheme against key-only attacks (EUF-KO) where the adversary is given the public key but no access to a signing oracle. See the full version [4] for formal definitions.

2.2 MPC in the Head

We now recall the MPC-in-the-Head (MPCitH) scheme used in Picnic and other previous works. In zero-knowledge proofs based on the MPCitH paradigm, the prover simulates N parties $\mathcal{P}_1, \ldots, \mathcal{P}_N$ for a computation over a field $\mathbb{F}$. As a single computation usually does not achieve a negligible soundness error, the prover evaluates τ independent instances of the MPC scheme we describe below.

For simplicity, we focus on one MPC instance only. We describe the protocol from the view of the individual parties, although it is actually the prover simulating each such party who performs these actions.

First, each $\mathcal{P}_i$ obtains a tape of private randomness derived from a seed $\mathsf{sd}^{(i)}$. These seeds are used whenever a party needs to generate pseudorandom values or randomness for commitments. In order to secret-share a value $x \in \mathbb{F}$, the parties $\mathcal{P}_1, \ldots, \mathcal{P}_N$ first locally generate a pseudo-random share $x^{(i)} \in \mathbb{F}$, which $\mathcal{P}_1$ adjusts to $x^{(1)} \leftarrow x - \sum_{i=1}^{N} x^{(i)}$, so that $x^{(1)}, \ldots, x^{(N)}$ form the secret-sharing $\langle x \rangle$ of x. Observe that all parties can easily, without communication, generate a

[6] https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions

secret-sharing $\langle r \rangle$ of a pseudorandom value r by each deterministically deriving shares from $\mathsf{sd}^{(i)}$. In order to open a secret-shared value $\langle x \rangle$, each $\mathcal{P}_i$ simply broadcasts its share $x^{(i)}$ of x to all other parties.

Initially, before the parties obtain the secret-shares of the secret input, they may first obtain shares of random triples a, b, c, such that $c = a \cdot b$. The process of establishing shares of such random triples is usually part of a preprocessing step. Each party commits to all shares from the preprocessing.

If all parties have enough shares of random triples and also obtained shares of the input (to which they also commit) then they can perform the computation. Let C be a circuit over $\mathbb{F}$ the parties evaluate as follows:

- For the secret-shared values $\langle x \rangle, \langle y \rangle$ the parties can compute a secret sharing $\langle x + y \rangle$ or $\langle \alpha \cdot x + \beta \rangle$ for publicly known $\alpha, \beta \in \mathbb{F}$ by using the linearly homomorphic property of the secret-sharing scheme. The parties can therefore compute shares of the outcome of the operation *without interaction*.
- To compute a secret-sharing $\langle x \cdot y \rangle$, from secret-shared values $\langle x \rangle, \langle y \rangle$, the parties take an unused sharing of a random triple $\langle a \rangle, \langle b \rangle, \langle c \rangle$, open the two values $\langle \delta \rangle = \langle a - x \rangle, \langle \epsilon \rangle = \langle b - y \rangle$, locally reconstruct δ and ϵ and then use the communication-free linear operations to compute a sharing of $\langle z \rangle = \delta \cdot \langle y \rangle + \epsilon \cdot \langle x \rangle - \delta \cdot \epsilon - \langle c \rangle$, which is correct if all shares were opened correctly and a, b, c indeed form a correct multiplicative triple.

When broadcasting values, each party locally commits to the values that it sends to and receives from other parties. Finally, each party locally sends its share of the output of the computation to the verifier, together with all the aforementioned commitments.

The verifier chooses a party $\mathcal{P}_i$ which should not be opened, whereupon the prover will send the verifier the openings for all commitments and seeds of the remaining $N-1$ parties, as well as all values that the un-opened party broadcast in the process. The verifier then re-computes all messages that are sent by each opened party and checks if these are consistent with the commitments as well as the pseudorandom seeds. It accepts if all these checks go through. In order to also verify the preprocessing, one would have also have to verify the preprocessing independently [31, 3]. We do not describe such checks here since our signature scheme, in comparison to BBQ [16], does not require preprocessed data.

One can see that the output, which the verifier obtains as shares from all N parties, must be correct with probability at least $1/N$, as the prover could have cheated in at most one party (the one which is not opened to the verifier). But in order to do so it would have had to anticipate the verifier's choice before making the commitments as it cannot undo cheating later. As mentioned before, the soundness error can be reduced by running the aforementioned scheme τ times in parallel. Note that by only revealing the secret states of $N - 1$ parties, the protocol we described above does not leak any information about the secret input to the verifier, as every $N - 1$ out of N shares appear pseudorandom and the remaining shares are hidden inside commitments.

2.3 AES

AES is a 128-bit block-cipher based on a substitution-permutation network (SPN). It allows key lengths of 128, 192 or 256 bits, where the SPN uses 10, 12 or 14 rounds respectively. The state of AES always consists of 128 bits and can be considered as a 4×4 matrix of elements in $\mathbb{F}_{2^8}$. The round function is composed of four operations on the state, of which only one called *SubBytes* is non-linear. SubBytes transforms each of the 16 bytes of the state by applying an S-box to each byte. The AES S-box computes a multiplicative inverse in the field $\mathbb{F}_{2^8}$, followed by an invertible affine transformation, and can be seen as

$$S : s \mapsto \phi^{-1}\left(\mathbf{A} \cdot \phi\left(s^{-1}\right) + \mathbf{b}\right), \tag{1}$$

where $\phi : \mathbb{F}_{2^8} \to (\mathbb{F}_2)^8$ is an isomorphism of vector spaces and $\mathbf{A} \in (\mathbb{F}_2)^{8\times 8}$ and $\mathbf{b} \in (\mathbb{F}_2)^8$ are the public parameters of the affine transformation. For completeness, $s^{-1} = 0$ if $s = 0$. The key schedule function is mostly linear, except for the application of the same S-box to up to four bytes of the round key.

2.4 The BBQ Signature Scheme

The BBQ signature scheme [16] follows the blueprint of Picnic [14], except for two main differences: 1) While Picnic uses LowMC as a one-way function (OWF), BBQ relies on the well-studied AES; 2) BBQ does not consider AES as a binary circuit with AND and XOR gates, but rather as an arithmetic circuit over $\mathbb{F}_{2^8}$ where the only non-linear operation is the S-box inversion.

The reason why AES naturally fits to be evaluated inside MPCitH, is that all its operations can either be expressed as linear transformations or field operations over $\mathbb{F}_{2^8}$ (as mentioned in Section 2.3). If one uses the MPCitH scheme that was outlined in Section 2.2 over the field $\mathbb{F}_{2^8}$, then only evaluating the S-boxes requires communication between the parties (beyond obtaining shares of the input). The S-box evaluation is therefore the only additional contribution to the size of the signature.

The approach of BBQ for evaluating the S-box inside MPCitH is as follows:

1: A random multiplication triple $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$ comes from preprocessing.
2: $\langle s\rangle$ is held by the parties.
3: $\langle r\rangle$ is sampled at random from $\mathbb{F}_{2^8}$.
4: $\langle t\rangle \leftarrow \langle s\rangle \cdot \langle r\rangle$ which is computed using $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$.
5: $t \leftarrow \mathsf{Open}(\langle t\rangle)$.
6: $\langle s^{-1}\rangle \leftarrow t^{-1} \cdot \langle r\rangle = \langle s^{-1} \cdot r^{-1} \cdot r\rangle$.

This approach, in the ideal case, only communicates 4 field elements per S-box: one field element for the preprocessed multiplication triple where only $\langle c\rangle$ is non-random, two for the multiplication $\langle s\rangle \cdot \langle r\rangle$ and one for the opening of $\langle t\rangle$. Two crucial properties of this protocol are:

1. It requires that s is invertible for every S-box as a computation that branches based on s being 0 or not would be much more costly.
2. It is necessary that $r \neq 0$, as the inversion does not work if $r = 0$.

The first drawback can be easily worked around, as shown in [16], by restricting key generation to choose an AES key and input such that S-box inputs are always non-zero. The authors show that this only reduces the key-space by 1.1 to 2.9 bits for AES-128 and AES-256 (respectively).

3 Identification Scheme with Amortized Verification of Inverses

In this section we present our novel technique for efficient amortized verification of inverses in MPC in more detail. This technique builds upon the sacrificing-based multiplication verification of [3], but improves upon their work in a novel and interesting way, by adapting a recent polynomial-based technique for verifying multiplications in distributed zero-knowledge proofs [10].

Our first observation is that we can replace the MPC-oriented inversion algorithm by an MPCitH-friendly version instead. In BBQ, the parties in the MPCitH first compute a masked version $\langle t \rangle$ of the input $\langle s \rangle$ to the S-box (i.e. $\langle t \rangle \leftarrow \langle s \rangle \cdot \langle r \rangle$, then invert t in the clear and finally remove the mask. This computes an inversion of $\langle s \rangle$ as one would do in regular MPC protocols, but it does not take the additional knowledge of the prover into account. Namely, the prover can directly inject a sharing of the inverse $\langle t \rangle = \langle s^{-1} \rangle$ into the protocol execution. This is possible because in an MPCitH protocol, the prover knows the evaluated circuit as well as all secret-shared values in advance. This also removes the random value $\langle r \rangle$ that was used in the inversion algorithm before, which is an advantage as BBQ required extra costs to ensure that this random $\langle r \rangle$ was non-zero in order for the inversion to be correct. However, the prover could still share a wrong value for $\langle t \rangle$, so the MPCitH parties must now run a checking procedure to ensure that the inverse values are correct.

After completing the circuit for evaluating AES, the parties now hold shares for $2m$ values, where m denotes the number of S-boxes: the inputs $\langle s_1 \rangle, \ldots, \langle s_m \rangle$ and their corresponding alleged inverses $\langle t_1 \rangle, \ldots, \langle t_m \rangle$. These $2m$ values should all pairwise multiply to 1, which can be verified using the standard multiplication verification procedure that uses a random multiplication triple for each pair $\langle s_\ell \rangle, \langle t_\ell \rangle, \ell \in [m]$. In the following we will show an alternative method to check inverses that requires less communication, contributing to shorter signatures. Here we will again exploit that the prover knows the entire secret inputs of the MPCitH protocol and can provide additional, although possibly faulty, secret inputs to the computation.

3.1 Polynomial-based Checking of Inverses

Consider a random polynomial $S(x) \in \mathbb{F}_{2^8}[x]$ of degree m that has $S(\ell - 1) = s_\ell$, for all $\ell \in [m]$. Similarly, consider such a polynomial $T(x) \in \mathbb{F}_{2^8}[x]$ with $T(\ell - 1) = t_\ell$, also of degree m. We can construct these polynomials by interpolating the m points (ℓ, s_ℓ), and one extra, random point.

By defining the polynomial $P = S \cdot T$ of degree $2m$, it holds that $P(\ell - 1) = 1$ for $\ell \in [m]$ if all the $\langle t_i \rangle$ were correctly provided. All $S(\cdot), T(\cdot), P(\cdot)$

are known to the prover at proof time. To perform the amortized test, observe that the MPCitH parties (1) can compute shares of the polynomials $S(\cdot)$, $T(\cdot)$ and $P(\cdot)$, which the prover will have committed to, without communication using Lagrange interpolation, and (2) can check that $P = S \cdot T$ by receiving a random value R from the verifier, locally evaluating the three polynomials at R, jointly reconstructing $P(R)$, $S(R)$, $T(R)$ and allowing the verifier to check that $P(R) = S(R) \cdot T(R)$.

Both $S(\cdot), T(\cdot)$ are already provided in the form of the shares $\langle s_\ell \rangle, \langle t_\ell \rangle$ and no extra communication is necessary[7] to interpolate these. For $P(\cdot)$, observe that m of its evaluations can be hard-coded as equal to 1, as we want to achieve that $P(0) = \cdots = P(m-1) = 1$. The proof therefore only has to communicate $m+1$ further evaluation points of $P(\cdot)$ in secret-shared form. The prover may cheat doing so, but the multiplication test on a random R chosen by the verifier detects such cheating behavior. The probability that this check fails to reveal cheating when verifying $P(\cdot)$ can be bounded using the Schwartz–Zippel Lemma, which we can use to bound the number of zeroes of the polynomial $Q = P - S \cdot T$, which is non-zero only when $P \neq S \cdot T$.

Lemma 1 (Schwartz–Zippel Lemma). *Let $Q(x) \in \mathbb{F}[x]$ be a non-zero polynomial of degree $d \geq 0$. For any finite subset S of $\mathbb{F}$, $\Pr_{r \xleftarrow{\$} S}[Q(r) = 0] \leq \frac{d}{|S|}$.*

In summary, in addition to the m injected inverses, the prover will additionally have to provide the remaining $m+1$ points that define $P(\cdot)$ fully as part of the proof, and will have to open $S(R), T(R), P(R)$ to the verifier.

3.2 Generalized Polynomial-based Checking

We now generalize the protocol from the previous subsection to reduce the number of field elements that need to be opened by the MPCitH parties. This generalization reduces the transcript size from around $2m$ elements to $m + O(\sqrt{m})$ by instead checking $O(\sqrt{m})$ inner products of size $\sqrt{m}$. It is based on a zero-knowledge proof from [10, Section 4.2] (also described in [12, Section 4.1]).

The idea of the improved check is to first compute $O(\sqrt{m})$ random inner products and use these to define $O(\sqrt{m})$ pairs of polynomials $S_j(\cdot)$ and $T_j(\cdot)$. These are then used to define a single random polynomial $P(\cdot)$, still subjected to a constraint based on the inverse relation, which is then checked against its constituent polynomials as before. We also modify this test by lifting the random polynomial evaluations to an extension field, which improves the soundness bound from the Schwartz-Zippel lemma.

Let λ be a lifting parameter and let $m = m_1 \cdot m_2$ where $m_2 < 8\lambda$. Fix an injective homomorphism to lift shares from $\mathbb{F}_{2^8}$ to $\mathbb{F}_{2^{8\lambda}}$, which can be computed locally by each of the parties of the MPCitH protocol. We also write $f(k)$ for

[7] Both $S(\cdot), T(\cdot)$ are actually defined by m shares and one share of a random value only known to the prover, which will later simplify the simulation of the protocol for its zero-knowledge property. This sharing of a random value can be obtained for free in MPCitH.

a polynomial $f(\cdot)$ and an integer k, to mean f evaluated at the k-th element of $\mathbb{F}_{2^{s\lambda}}$ (according to a fixed ordering). The following procedure describes the non-secret-shared verification protocol.

Given inputs $s_1, \ldots, s_m$ and $t_1, \ldots, t_m \in \mathbb{F}_{2^s}$:

1: Lift $s_1, \ldots, s_m$ and $t_1, \ldots, t_m$ to $\mathbb{F}_{2^{s\lambda}}$.
2: For $j \in [m_1]$, sample additional random points $\bar{s}_j, \bar{t}_j \leftarrow \mathbb{F}_{2^{s\lambda}}$.
3: Sample $r_j \leftarrow \mathbb{F}_{2^{s\lambda}}$, for $j \in [m_1]$. ▷ Randomizing the inner products.
4: Compute $s'_{j,k} = r_j \cdot s_{j+m_1 k}$ and write $t'_{j,k} = t_{j+m_1 k}$, for $j \in [m_1], k \in [0, m_2 - 1]$. Observe that when all $s_i t_i = 1$,

$$\left(s'_{1,k} \cdots s'_{m_1,k}\right) \begin{pmatrix} t'_{1,k} \\ \vdots \\ t'_{m_1,k} \end{pmatrix} = \sum_{j \in [m_1]} r_j \qquad \text{for } k \in [0, m_2 - 1].$$

5: Define polynomials $S_j(\cdot), T_j(\cdot)$ such that

$$\begin{aligned} S_j(k) &= s'_{j,k}, & T_j(k) &= t'_{j,k} \qquad \text{for } k \in [0, m_2 - 1], \\ S_j(m_2) &= \bar{s}_j, & T_j(m_2) &= \bar{t}_j. \end{aligned}$$

By definition each such polynomial has degree m_2.
6: Let $P = \sum_j S_j \cdot T_j$ be of degree $2m_2$, where it is guaranteed that $P(k) = \sum_j r_j$, for $k \in [0, m_2 - 1]$ and $j \in [m_1]$.
7: Sample $R \leftarrow \mathbb{F}_{2^{s\lambda}} \setminus [0, m_2 - 1]$. ▷ Schwartz–Zippel challenge.
8: Compute $P(R)$ as well as $S_j(R)$ and $T_j(R)$, for $j \in [m_1]$, and verify whether $P(R) = \sum_j S_j(R) \cdot T_j(R)$.

Secret-shared Checking. Since polynomial interpolation is linear, the parties in the MPCitH protocol can interpolate the shares of the coefficients of polynomials $S_j(\cdot)$ and $T_j(\cdot)$ from shares of $s'_{j,k}$ and $t'_{j,k}$ as well as $\bar{s}_j, \bar{t}_j$. This does not require any communication as the challenges r_j are public and $\bar{s}_j, \bar{t}_j$ can be freely generated by the MPCitH scheme. From this, each party can compute their share of $S_j(R)$ and $T_j(R)$ for each j and then reveal it in order to perform the check.

As mentioned above, the only non-linear part is the computation of the polynomial $P(\cdot)$. Here the prover adjusts the shares for m_2+1 values of P, as another m_2 values are already specified by the publicly known $P(k) = \sum_j r_j$ for each $k \in [0, m_2 - 1]$, in the same way that it injects a correction value for the inverses. With those $2m_2 + 1$ shares, the parties locally interpolate shares of the coefficients of P and then compute and reveal their shares of $P(R)$.

In summary, each party $\mathcal{P}_i$ performs the following operations.

1: Lift $s_1^{(i)}, \ldots, s_m^{(i)}$ and $t_1^{(i)}, \ldots, t_m^{(i)}$ to $\mathbb{F}_{2^{s\lambda}}$.
2: For $j \in [m_1]$, sample $\bar{s}_j^{(i)}, \bar{t}_j^{(i)} \leftarrow \mathbb{F}_{2^{s\lambda}}$.
3: Receive challenges $r_j \in \mathbb{F}_{2^{s\lambda}}$ for $j \in [m_1]$.
4: Compute $s'^{(i)}_{j,k} = r_j \cdot s^{(i)}_{j+m_1 k}$ and write $t'^{(i)}_{j,k} = t^{(i)}_{j+m_1 k}$ for $j \in [m_1], k \in [0, m_2 - 1]$.

5: Interpolate polynomials $S_j^{(i)}(\cdot)$ and $T_j^{(i)}(\cdot)$ such that:

$$S_j^{(i)}(m_2) = \bar{s}_j^{(i)}, \quad T_j^{(i)}(m_2) = \bar{t}_j^{(i)}$$

and

$$S_j^{(i)}(k) = \bar{s}_{j,k}'^{(i)}, \quad T_j^{(i)}(k) = \bar{t}_{j,k}'^{(i)} \text{ for } k \in [0, m_2 - 1].$$

6: Receive $m_2 + 1$ shares $P^{(i)}(k)$, for $k \in \{m_2, \ldots, 2m_2\}$ ($k = m_2$ is included as parties cannot compute $\sum_j \bar{s}_j \cdot \bar{t}_j$ locally) from the prover. For $k \in [0, m_2 - 1]$, if $i = 1$, set $P^{(i)}(k) = \sum_j r_j$; if $i \neq 1$, set $P^{(i)}(k) = 0$. Interpolate $P^{(i)}$ from those $2m_2 + 1$ points.

7: Receive the challenge $R \in \mathbb{F}_{2^{8\lambda}} \setminus [0, m_2 - 1]$ from the verifier.

8: Compute $P^{(i)}(R), S_j^{(i)}(R)$ and $T_j^{(i)}(R)$ and open all these $2m_1 + 1$ values.

9: Check that $P^{(i)}(R) = \sum_{j \in [m_1]} S_j^{(i)}(R) \cdot T_j^{(i)}(R)$.

Soundness Error of the Generalized Check. Assume that each party honestly follows the protocol but there exists $\ell \in [m]$ such that $s_\ell \cdot t_\ell \neq 1$. Since the embedding into $\mathbb{F}_{2^{8\lambda}}$ is an injective homomorphism, it must then also hold that $s_\ell \cdot t_\ell \neq 1$ over $\mathbb{F}_{2^{8\lambda}}$. Assuming that the above protocol succeeds, then one of the following conditions must hold:

1. In Step 2, the values $r_1, \ldots, r_{m_1}$ were sampled such that $\sum_{j \in [m_1]} S_j(k) \cdot T_j(k) = 1$ for $k \in [0, m_2 - 1]$.
2. In Step 6 a value R was chosen such that $P(R) = \sum_j S_j(R) \cdot T_j(R)$ while $P \neq \sum_j S_j \cdot T_j$.

For the first condition, by assumption we have that $\exists j, k$ such that $\sum_j r_j = \sum_j s'_{j,k} \cdot t'_{j,k}$ while $r_j \neq s'_{j,k} \cdot t'_{j,k}$. By the choice of r_j this will happen with probability at most $2^{-8\lambda}$. In the second case, the polynomials on both sides are of degree $2m_2$ and can have at most $2m_2$ points in common. By Lemma 1, the probability of choosing such a value of R is at most $2m_2/(2^{8\lambda} - m_2)$.

The overall soundness error is therefore at most $2^{-8\lambda} + 2m_2/(2^{8\lambda} - m_2)$.

Simulatability of the Generalized Check. We now construct a simulator for the aforementioned protocol to argue that it does not leak any information. The simulator obtains $N - 1$ shares of $\{s_\ell, t_\ell\}_{\ell \in [m]}$ as inputs, w.l.o.g. it misses the N-th share. It first chooses $r_1, \ldots, r_{m_1}$ uniformly at random from $\mathbb{F}_{2^{8\lambda}}$ and R uniformly from the appropriate set.

For $i \in [N - 1]$, it samples $\bar{s}_j^{(i)}$ and $\bar{t}_j^{(i)}$ as in the protocol and interpolates $S_j^{(i)}$ and $T_j^{(i)}$ for $j \in [m_1]$. It then samples $S_j(R)$ and $T_j(R)$ at random. It fixes m pairs of shares $\{s_\ell^{(N)}, t_\ell^{(N)}\}$ arbitrarily, sets $S_j^{(N)}(R) = S_j(R) - \sum_{i=1}^{N-1} S_j^{(i)}(R)$ and similarly for $T_j^{(N)}(R)$, and interpolates $S_j^{(N)}$ and $T_j^{(N)}$ using these values (instead of sampling $\bar{s}_j^{(N)}$ and $\bar{t}_j^{(N)}$ at random). Because the uniform sampling of $\bar{s}_j$ and $\bar{t}_j$ in the protocol imply a uniform distribution for $S_j(R)$ and $T_j(R)$,

the values produced by the simulator (including the opened shares $S_j^{(N)}(R)$ and $T_j^{(N)}(R)$), are identically distributed.

The simulator then computes the product polynomial P defined by the shared S_j and T_j polynomials it interpolated before and honestly samples the $m_2 + 1$ shares for each party. Instead of opening $P^{(N)}(R)$ honestly, the simulator computes $P^{(N)}(R) = P(R) - \sum_{i=1} N - 1P^{(i)}(R)$ and opens that instead. Because $P(R)$ is computed honestly from $S_j(R)$ and $T_j(R)$, the computed $P^{(N)}(R)$ is distributed identically to an honest run of the protocol.

By construction, all opened values are consistent with the protocol definition and are perfectly indistinguishable from a real run. Furthermore, all secret-shared values by linearity generate a correct transcript.

Communication Cost. Steps 1–4 and 6 are local given public knowledge of r_j and R. The only parts which add communication are Step 5, which introduces $8\lambda(m_2 + 1)$ bits into the transcript unless $\mathcal{P}_1$ is opened (as injecting shares can be done with $\mathcal{P}_2, \ldots, \mathcal{P}_N$ sampling random shares and $\mathcal{P}_1$ receiving a correction share), and Step 7, which always adds $8\lambda(2m_1+1)$ bits. This gives a total average overhead per S-box of $8\lambda \left(\frac{N-1}{N}(m_2 + 1) + 2m_1 + 1\right)/m$ bits.

For instance, in our concrete instantiation at the L1 security level (see Section 6.1), we use $m = 200$, with $N = 16$ parties, $\lambda = 4$, $m_1 = 10$ and $m_2 = 20$ to obtain an overhead of 6.51 bits per S-box. This is significantly less than the direct approach of checking with m MPC multiplications, which would require around 16 bits per multiply (8 for the last party's share of the multiplication triple, and 8 for the unopened party's broadcast value).

4 The Banquet Signature Scheme

In this section we present the Banquet signature scheme obtained by using the zero-knowledge identification scheme from the previous section with the Fiat–Shamir transform [22]. We state and prove its security theorem in section 5.

At a high level, Banquet works exactly as previous MPCitH-based signatures, like Picnic and BBQ. Given a key pair consisting of a random secret key $\mathsf{sk} = k$ and a public key $\mathsf{pk} = (x, y)$ such that $\mathsf{AES}_k(x) = y$, and to sign a message μ one generates a non-interactive zero-knowledge proof of knowledge of k in a way that binds μ to the proof. More concretely, we start with a 7-round, interactive identification scheme for proving knowledge of k, and compile this to a non-interactive proof using the Fiat–Shamir transform. The verifier's three random challenges are generated using random oracles, with the message and public key used as initial inputs to bind these to the signature.

Additionally, to sample the MPC parties' randomness we use a deterministic seed expansion function, Expand, which we instantiate using SHAKE. Each seed s used as input to Expand defines a distinct random tape t, such that every call to $\mathsf{Sample}(t)$ reads the next bits of the tape defined by t, keeping track of the current position on the tape. As introduced in [31], to reduce communication when opening seeds that were randomly challenged, we generate all seeds as the

leaves of a binary tree derived from a master seed at the root. This allows $N-1$ of N seeds to be communicated by revealing only $\log N$ seeds.

Parameters. The scheme depends on the following values: the security parameter κ, the total number of parties N in the underlying MPC protocol, the total number of S-boxes m, the number of parallel repetitions τ, and m_1, m_2, and λ be as described in the previous section.

Key Generation. $\mathsf{Gen}(1^\kappa)$ samples $k, x \leftarrow \{0,1\}^\kappa$, computes $\mathsf{AES}_k(x) = y$, and repeats this until there are no S-boxes in the computation of y with input 0. It then sets $\mathsf{sk} = k$ and $\mathsf{pk} = (x, y)$, and returns $(\mathsf{pk}, \mathsf{sk})$.

We use e to index executions, i to index parties, and ℓ to index S-boxes.
$\mathsf{Sign}(\mathsf{sk}, \mu)$:
Phase 1: Committing to the seeds and the execution views of the parties.

1: Sample a random salt $\mathsf{st} \xleftarrow{\$} \{0,1\}^{2\kappa}$.
2: **for** each parallel execution e **do**
3: Sample a root seed: $\mathsf{sd}_e \xleftarrow{\$} \{0,1\}^\kappa$.
4: Compute parties' seeds $\mathsf{sd}_e^{(1)}, \ldots, \mathsf{sd}_e^{(N)}$ as leaves of binary tree from sd_e.
5: **for** each party i **do**
6: Commit to seed: $\mathsf{C}_e^{(i)} \leftarrow \mathsf{Commit}(\mathsf{st}, e, i, \mathsf{sd}_e^{(i)})$.
7: Expand random tape: $\mathsf{tape}_e^{(i)} \leftarrow \mathsf{Expand}(\mathsf{st}, e, i, \mathsf{sd}_e^{(i)})$
8: Sample witness share: $\mathsf{sk}_e^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
9: Compute witness offset: $\Delta\mathsf{sk}_e \leftarrow \mathsf{sk} - \sum_i \mathsf{sk}_e^{(i)}$.
10: Adjust first share: $\mathsf{sk}_e^{(1)} \leftarrow \mathsf{sk}_e^{(1)} + \Delta\mathsf{sk}_e$.
11: **for** each S-box ℓ **do**
12: For each party i, compute the local linear operations to obtain the share $s_{e,\ell}^{(i)}$ of the S-box input $s_{e,\ell}$.
13: Compute the S-box output: $t_{e,\ell} = \left(\sum_i s_{e,\ell}^{(i)}\right)^{-1}$.
14: For each party i, sample the share of the output: $t_{e,\ell}^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
15: Compute output offset: $\Delta t_{e,\ell} = t_{e,\ell} - \sum_i t_{e,\ell}^{(i)}$.
16: Adjust first share: $t_{e,\ell}^{(1)} \leftarrow t_{e,\ell}^{(1)} + \Delta t_{e,\ell}$.
17: Broadcast each party's share $\mathsf{ct}_e^{(i)}$ of the output.
18: Set $\sigma_1 \leftarrow (\mathsf{st}, (\mathsf{C}_e^{(i)})_{i\in[N]}, (\mathsf{ct}_e^{(i)})_{i\in[N]}, \Delta\mathsf{sk}_e, (\Delta t_{e,\ell})_{\ell\in[m]})_{e\in[\tau]}$.

Fig. 1. Signature scheme - Phase 1. Commitment to executions of AES.

The security of the key generation directly follows from the security analysis in [16], which we review in Section 5.

We recall that for security reasons the key generation algorithm requires the block-size and the key-size to be equal [14]. While this is true for AES-128, this is not the case for AES-192 and AES-256. Two solutions to this problem were proposed in [16]: the first one relies on the use of the Rijndael cipher with 192-bit (resp. 256-bit) blocks and keys; the second on the combination of two copies of AES-192 (resp. AES-256) in ECB mode.

Phase 2: Challenging the multiplications.

Compute challenge hash: $h_1 \leftarrow H_1(\mu, \mathsf{pk}, \sigma_1)$.
Expand hash: $(r_{e,j})_{e\in[\tau],j\in[m_1]} \leftarrow \mathsf{Expand}(h_1)$ where $r_{e,j} \in \mathbb{F}_{2^{8\lambda}}$.

Phase 3: Committing to the checking polynomials.

for each execution e **do**
  for each party i **do**
    Lift $s^{(i)}_{e,\ell}, t^{(i)}_{e,\ell} \hookrightarrow \mathbb{F}_{2^{8\lambda}}$, for $\ell \in [m]$.
    for $j \in [m_1]$ **do**
      Set $s'^{(i)}_{e,j,k} \leftarrow r_{e,j} \cdot s^{(i)}_{e,j+m_1 k}$ and $t'^{(i)}_{e,j,k} \leftarrow t^{(i)}_{e,j+m_1 k}$, for $k \in [0, m_2]$.
      Sample additional random points: $\bar{s}^{(i)}_{e,j}, \bar{t}^{(i)}_{e,j} \leftarrow \mathsf{Sample}(\mathsf{tape}^{(i)}_e)$.
      Define $S^{(i)}_{e,j}(k) = s'^{(i)}_{e,j,k}$ and $T^{(i)}_{e,j}(k) = t'^{(i)}_{e,j,k}$ for $k \in [0, m_2 - 1]$ as well as $S^{(i)}_{e,j}(m_2) = \bar{s}^{(i)}_{e,j}$ and $T^{(i)}_{e,j}(m_2) = \bar{t}^{(i)}_{e,j}$.
      Interpolate polynomials $S^{(i)}_{e,j}(\cdot)$ and $T^{(i)}_{e,j}(\cdot)$ of degree m_2 using defined $m_2 + 1$ points.
  Compute product polynomial: $P_e \leftarrow \sum_{j\in[m_1]} \left(\sum_i S^{(i)}_{e,j}\right) \cdot \left(\sum_i T^{(i)}_{e,j}\right)$.
  for each party i **do**
    For $k \in [0, m_2 - 1]$: $P^{(i)}_e(k) = \begin{cases} \sum_j r_{e,j} & \text{if } i = 1 \\ 0 & \text{if } i \neq 1 \end{cases}$
    For $k \in \{m_2, \ldots, 2m_2\}$, sample $P^{(i)}_e(k) \leftarrow \mathsf{Sample}(\mathsf{tape}^{(i)}_e)$.
  for $k \in \{m_2, \ldots, 2m_2\}$ **do**
    Compute offset: $\Delta P_e(k) = P_e(k) - \sum_i P^{(i)}_e(k)$.
    Adjust first share: $P^{(1)}_e(k) \leftarrow P^{(1)}_e(k) + \Delta P_e(k)$.
  For each party i, interpolate $P^{(i)}_e$ using defined $2m_2 + 1$ points.
Set $\sigma_2 \leftarrow \left((\Delta P_e(k))_{k\in\{m_2,\ldots,2m_2\}}\right)_{e\in[\tau]}$.

Fig. 2. Signature scheme - Phases 2 and 3. Computation of randomized inner product checking polynomials.

Signature Generation and Verification Algorithm. The $\mathsf{Sign}(\mathsf{sk}, \mathsf{pk}, \mu)$ algorithm is formally described in Figures 1,2 and 3. We describe the protocol in phases and give the rationale behind each one. The challenges h_1, h_2 and h_3 are generated using three random oracles H_1, H_2 and H_3, respectively. The MPC-in-the-Head computation is divided into two steps: the AES *execution* and then the *verification* of the inverse injections.

In Phase 1 (Figure 1) the prover commits to τ executions of AES, generating the first signature component σ_1, which consists of a 2κ-bit salt st, commitments to the seeds $\{\mathsf{C}^{(i)}_e\}_{i\in[N]}$, the injection of the secret key sk, the injection of the m inverse values $t_{e,\ell}$ and the broadcast of the output ct_e, for each execution $e \in [\tau]$ and S-box $\ell \in [m]$. These values suffice to uniquely determine the output values of the distributed AES circuit. The first challenge, h_1, is generated in Phase 2 (Figure 2) by hashing together σ_1, the message μ and the public key pk. The next phases are devoted to the verification of the AES executions. In particular, in Phase 3 (Figure 2) the prover generates the commitments to the checking polynomial $P_e(\cdot)$, for each execution $e \in [\tau]$, as described in Section 3.2, and

Phase 4: Challenging the checking polynomials.

1: Compute challenge hash: $h_2 \leftarrow H_2(h_1, \sigma_2)$.
2: Expand hash: $(R_e)_{e\in[\tau]} \leftarrow \mathsf{Expand}(h_2)$ where $R_e \in \mathbb{F}_{2^{8\lambda}} \setminus [0, m_2 - 1]$.

Phase 5: Committing to the views of the checking protocol.

1: **for** each execution e **do**
2: **for** each party i **do**
3: For $j \in [m_1]$, compute: $a_{e,j}^{(i)} \leftarrow S_{e,j}^{(i)}(R_e)$ and $b_{e,j}^{(i)} \leftarrow T_{e,j}^{(i)}(R_e)$.
4: Compute: $c_e^{(i)} \leftarrow P_e^{(i)}(R_e)$.
5: Open c_e, and $a_{e,j}$, $b_{e,j}$ for $j \in [m_1]$.
6: Set $\sigma_3 \leftarrow (((a_{e,j}^{(i)}, b_{e,j}^{(i)})_{j\in[m_1]}, c_e^{(i)})_{i\in[N]})_{e\in[\tau]}$.

Phase 6: Challenging the views of the checking protocol.

1: Compute challenge hash: $h_3 \leftarrow H_3(h_2, \sigma_3)$.
2: Expand hash: $(\bar{i}_e)_{e\in[\tau]} \leftarrow \mathsf{Expand}(h_3)$ where $\bar{i}_e \in [N]$.

Phase 7: Opening the views of the checking protocol.

1: **for** each execution e **do**
2: $\mathsf{seeds}_e \leftarrow \{\log_2(N)$ nodes needed to compute $\mathsf{sd}_{e,i}$ for $i \in [N] \setminus \{\bar{i}_e\}\}$.
3: Output $\sigma \leftarrow (\mathsf{st}, h_1, h_3, (\mathsf{seeds}_e, \mathsf{C}_e^{(\bar{i}_e)}, \Delta\mathsf{sk}_e, (\Delta t_{e,\ell})_{\ell\in[m]},$
$(\Delta P_e(k))_{k\in\{m_2,\ldots,2m_2\}}, P_e(R_e), (S_{e,j}(R_e), T_{e,j}(R_e))_{j\in[m_1]})_{e\in[\tau]})$.

Fig. 3. Signature scheme - Phases 4-7. Computation of the views of the randomized check openings, challenging and opening of the views of the checking protocol.

computes a challenge for them in Phase 4. Phases 5 and 6 (Figure 3) describe the computation of the challenge for the committed checking polynomial and the commitments to the views of the check openings for each execution, respectively. Finally, Phase 7 outputs the signature σ.

The verification algorithm $\mathsf{Verify}(\mathsf{pk}, \mu, \sigma)$ is described in Figure 4. It performs similar computations to those made during generation of the signature, thereby checking the consistency of $N - 1$ of the N parties in each of the τ instances.

5 Security Proof

We prove Banquet is EUF-CMA-secure (unforgeability against chosen-message attacks, Theorem 1) by first proving that Banquet is EUF-KO-secure (unforgeability against key-only attacks) in Lemma 2.

Lemma 2. *Let* $\mathsf{Commit}, H_1, H_2$ *and* H_3 *be modeled as random oracles,* Expand *be modeled as a random function, and let* (N, τ, m_2, λ) *be parameters for the Banquet scheme. Let* $\mathcal{A}$ *be a probabilistic* $\mathsf{poly}(\kappa)$*-time adversary against the* EUF-KO *security of the signature scheme that makes* Q_c, Q_1, Q_2 *and* Q_3 *queries to the respective oracles. Then there exists a probabilistic* $\mathsf{poly}(\kappa)$*-time adversary* $\mathcal{B}$ *against the* OWF *security of* $f_\mathbf{x}$ *such that*

$$\mathbf{Adv}_{\mathcal{B}}^{\mathsf{OWF}}(1^\kappa) \geq \mathbf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}KO}}(1^\kappa) - \varepsilon(Q_\mathsf{c}, Q_1, Q_2, Q_3),$$

with

$$\varepsilon(Q_\mathsf{c}, Q_1, Q_2, Q_3) = \frac{(\tau N + 1)(Q_\mathsf{c} + Q_1 + Q_2 + Q_3)^2}{2^{2\kappa}} + \Pr[X + Y + Z = \tau]$$

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

1: Parse $\sigma \leftarrow (\mathsf{st}, h_1, h_3, (\mathsf{seeds}_e, \mathsf{C}_e^{(\bar{i}_e)}, \Delta\mathsf{sk}_e, (\Delta t_{e,\ell})_{\ell\in[m]},$
$(\Delta P_e(k))_{k\in\{m_2,\ldots,2m_2\}}, P_e(R_e), (S_{e,j}(R_e), T_{e,j}(R_e))_{j\in[m_1]})_{e\in[\tau]})$.
2: Compute $h_2' \leftarrow H_2(h_1, ((\Delta P_e(k))_{k\in\{m_2,\ldots,2m_2\}})_{e\in[\tau]})$.
3: Expand hashes as $(r_{e,j})_{e\in[M],j\in[m_1]} \leftarrow \mathsf{Expand}(h_1)$, $(R_e)_{e\in[M]} \leftarrow \mathsf{Expand}(h_2')$ and $(\bar{i}_e)_{e\in[M]} \leftarrow \mathsf{Expand}(h_3)$.
4: **for** each execution e **do**
5: Use seeds_e to compute $\mathsf{sd}_e^{(i)}$ for $i \in [N] \setminus \bar{i}_e$.
6: **for** each party $i \in [N] \setminus \bar{i}_e$ **do**
7: Recompute $\mathsf{C}_e^{(i)} \leftarrow \mathsf{Commit}(\mathsf{st}, e, i, \mathsf{sd}_e^{(i)})$, $\mathsf{tape}_e^{(i)} \leftarrow \mathsf{Expand}(\mathsf{st}, e, i, \mathsf{sd}_e^{(i)})$ and $\mathsf{sk}_e^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
8: **if** $i \stackrel{?}{=} 1$ **then**
9: Adjust first share: $\mathsf{sk}_e^{(i)} \leftarrow \mathsf{sk}_e^{(i)} + \Delta\mathsf{sk}_e$.
10: **for** each S-box ℓ **do**
11: Compute local linear operations to obtain $s_{e,\ell}^{(i)}$.
12: Sample output share: $t_{e,\ell}^{(i)} \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
13: **if** $i \stackrel{?}{=} 1$ **then**
14: Adjust first share: $t_{e,\ell}^{(i)} \leftarrow t_{e,\ell}^{(i)} + \Delta t_{e,\ell}$.
15: Recompute output broadcast $\mathsf{ct}_e^{(i)}$ and missing $\mathsf{ct}_e^{(\bar{i}_e)} = \mathsf{ct} - \sum_{i\neq\bar{i}_e} \mathsf{ct}_e^{(i)}$.
16: Do as in Phase 3, lines 3–8 to interpolate $S_{e,j}^{(i)}, T_{e,j}^{(i)}$ for $j \in [m_1]$.
17: **for** k from 0 to $m_2 - 1$ **do**
18: If $i \stackrel{?}{=} 1$, set $P_e^{(i)}(k) = \sum_j r_{e,j}$; otherwise set $P_e^{(i)}(k) = 0$.
19: **for** k from m_2 to $2m_2$ **do**
20: Sample share: $P_e^{(i)}(k) \leftarrow \mathsf{Sample}(\mathsf{tape}_e^{(i)})$.
21: **if** $i \stackrel{?}{=} 1$ **then**
22: Adjust first share: $P_e^{(i)}(k) \leftarrow P_e^{(i)}(k) + \Delta P_e(k)$.
23: Interpolate $P_e^{(i)}$ and compute $c_e^{(i)} \leftarrow P_e^{(i)}(R_e)$.
24: For $j \in [m_1]$, compute $a_{e,j}^{(i)} \leftarrow S_{e,j}^{(i)}(R_e)$ and $b_{e,j}^{(i)} \leftarrow T_{e,j}^{(i)}(R_e)$.
25: Compute missing shares $c_e^{(\bar{i}_e)} \leftarrow P_e(R_e) - \sum_{i\neq\bar{i}_e} c_e^{(i)}$ and for $j \in [m_1]$:

$$a_{e,j}^{(\bar{i}_e)} \leftarrow S_{e,j}(R_e) - \sum\nolimits_{i\neq\bar{i}_e} a_{e,j}^{(i)} \text{ and } b_{e,j}^{(\bar{i}_e)} \leftarrow T_{e,j}(R_e) - \sum\nolimits_{i\neq\bar{i}_e} b_{e,j}^{(i)}$$

26: Set $h_1' \leftarrow H_1(\mathsf{st}, ((\mathsf{C}_e^{(i)})_{i\in[N]}, (\mathsf{ct}_e^{(i)})_{i\in[N]}, \Delta\mathsf{sk}_e, (\Delta t_{e,\ell})_{\ell\in[m]})_{e\in[\tau]})$.
27: Set $h_3' \leftarrow H_3\left(\begin{array}{c} h_2', (P_e(R_e), (c_e^{(i)})_{i\in[N]}, \\ (S_{e,j}(R_e), T_{e,j}(R_e), (a_{e,j}^{(i)}, b_{e,j}^{(i)})_{i\in[N]})_{j\in[m_1]})_{e\in[\tau]} \end{array}\right)$.
28: Output accept iff $h_1' \stackrel{?}{=} h_1$, $h_3' \stackrel{?}{=} h_3$ and for all executions e it holds that $P_e(R_e) \stackrel{?}{=} \sum_j S_{e,j}(R_e) \cdot T_{e,j}(R_e)$.

Fig. 4. Verification algorithm.

where $X = \max_{q_1 \in \mathcal{Q}_1}\{X_{q_1}\}$ *with* $X_{q_1} \sim \mathfrak{B}(\tau, 1/2^{8\lambda})$, $Y = \max_{q_2 \in \mathcal{Q}_2}\{Y_{q_2}\}$ *with* $Y_{q_2} \sim \mathfrak{B}(\tau - X, 2m_2/(2^{8\lambda} - m_2))\forall q_2$ *and* $Z = \max_{q_3 \in \mathcal{Q}_3}\{Z_{q_3}\}$ *with* $Z_{q_3} \sim \mathfrak{B}(\tau - X - Y, 1/N)\forall q_3$*, where* $\mathfrak{B}(n, p)$ *denotes the binomial probability distribution with* n *samples each with probability* p *of success.*

Remark 1. Due to the mix of different distributions, we do not express the second term of $\varepsilon(Q_{\mathsf{c}}, Q_1, Q_2, Q_3)$ as a closed function; we will later compute parameters τ, m_2, κ and N such that ε is negligible in λ. This will then imply that if $\mathcal{A}$ has a non-negligible advantage in the EUF-KO game with these parameters, then $\mathcal{B}$ also has non-negligible advantage in the corresponding OWF game.

Proof. The proof follows a similar strategy to that of [8, Lemma 2, Appendix B] and is included in the full version [4].

Theorem 1. *The Banquet signature scheme is* EUF-CMA*-secure, assuming that* Commit*,* H_1*,* H_2 *and* H_3 *are modelled as random oracles,* Expand *is a PRG with output computationally* ϵ_{PRG}*-close to uniform, the seed tree construction is computationally hiding, the* (N, τ, m_2, λ) *parameters are appropriately chosen, and the key generation function* $f_{\mathsf{x}} : \mathsf{sk} \mapsto \mathsf{pk}$ *is a one-way function.*

Proof. Fix an attacker $\mathcal{A}$. We define a sequence of games where the first 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 EUF-CMA game to $\mathcal{A}$. (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 witness and can compute signatures honestly;
G_1**:** $\mathcal{B}$ replaces real signatures with simulated ones which no longer use sk;
$\mathcal{B}$ then uses the EUF-KO challenge pk^* in its simulation to $\mathcal{A}$.

We note that $\mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A}} = \mathsf{G}_0 = (\mathsf{G}_0 - \mathsf{G}_1) + \mathsf{G}_1$ and we obtain a bound on G_0 by bounding first $\mathsf{G}_0 - \mathsf{G}_1$ and then G_1

Hopping to game G_1*.* When $\mathcal{A}$ queries the signing oracle, $\mathcal{B}$ simulates a signature σ by sampling a random witness, picking a party P_{i^*} and cheating in the verification phase and in the broadcast of the output shares $\mathsf{ct_e}^{(i)}$ such that the circuit still outputs the correct AES ciphertext, and finally ensuring that the values observed by $\mathcal{A}$ are sampled independently of the incorrect witness and with the same distribution as in a real signature. It programs both the second random oracle to return the R_e values that it sampled, and also the third random oracle to hide the party for which it has cheated in the verification and opening phases.

We now argue that simulating the signatures as in G_1 is indistinguishable from signatures in G_0. We list a series of (sub) game hops which begins with G_0, where the witness is known and signatures are created honestly, and ends with G_1, where signatures are simulated without using the witness. 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 witness 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 $\mathsf{EUF\text{-}CMA}$ game as long as it does not abort.
2. Before beginning, the next $\mathcal{B}$ samples h_3 at random and expands it to obtain $(i_e^*)_{e\in[\tau]}$; these are the unopened parties, which $\mathcal{B}$ will use for cheating. It proceeds as before and programs the random oracle H_3 so that it outputs h_3 when queried in Phase 6. If that query has already been made, $\mathcal{B}$ aborts the simulation.
3. In Phase 1, the next $\mathcal{B}$ replaces $\mathsf{sd}_e^{(i^*)}$ in the binary tree, for each $e \in [\tau]$, by a randomly sampled one. This is indistinguishable from the previous hybrid assuming that the tree structure is hiding.
4. The next $\mathcal{B}$ replaces all outputs of $\mathsf{Expand}(\mathsf{st}, e, i^*, \mathsf{sd}_e^{(i^*)})$ by random outputs (independent of the seed). This is indistinguishable from the previous reduction assuming that Expand is indistinguishable from a random function.
5. The next $\mathcal{B}$ replaces the commitments of the unopened parties $C_e^{(i^*)}$ with random values (i.e., without querying Commit).
6. Before starting Phase 3, the next $\mathcal{B}$ samples h_2 at random and expands it to obtain $(R_e)_{e\in[\tau]}$; this will enable it to sample the checking values at random. It then proceeds as before and programs the random oracle H_2 to output h_2 in Phase 4. If that query has already been made, $\mathcal{B}$ aborts the simulation.
7. In Phase 3, the next $\mathcal{B}$ interpolates $S_{e,j}^{(i)}$ for $i \in [N] \setminus \{i^*\}$, samples the values $S_{e,j}(R_e)$ at random, computes $S_{e,j}^{(i^*)}(R_e) = S_{e,j}(R_e) - \sum_{i\neq i^*} S_{e,j}^{(i)}$ and interpolates $S_{e,j}^{(i^*)}$ using $k \in \{0, \ldots, m2-1\} \cup \{R_e\}$. It does the same for the T polynomials and computes P_e and the offsets according to the protocol. As the uniform distribution of honestly generated $S_{e,j}(R_e)$ and $T_{e,j}(R_e)$ (opened in Phase 5) comes from the uniform distribution of $\bar{s}_{e,j}$ and $\bar{t}_{e,j}$ given by the random function (and which are hidden from $\mathcal{A}$ as $\mathsf{sd}_e^{(i^*)}$ is no longer used), this is indistinguishable from the previous hybrid. The same holds for the shares of party $\mathcal{P}_{i^*}$ that are opened in Phase 5. The distribution of the ΔP_e offsets is therefore also indistinguishable from a real signature as they are computed honestly from indistinguishable elements. (At this stage the P_e polynomials always satisfy the check since $\mathcal{B}$ is still using a correct witness.)
8. In Phase 5, the next $\mathcal{B}$ replaces $c_e^{(i^*)} \leftarrow P_e^{(i^*)}(R_e)$ with $c_e^{(i)} \leftarrow P_e(R_e) - \sum_{i\neq i^*} P_e^{(i)}(R_e)$. This is indistinguishable because the $P_e^{(i)}(R_e)$ values, for $i \neq i^*$, are computed honestly, and the $P_e(R_e)$ value is distributed identically to an honest signature (because $S_{e,j}$ and $T_{e,j}$ are). From now on, the Schwartz–Zippel check always passes, even if the product relation doesn't hold, and

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

9. The final $\mathcal{B}$ replaces the real sk by a fake witness sk^* and cheats on the broadcast of party P_{i^*}'s output share $\mathsf{ct}_e^{(i^*)}$ such that it matches what is expected, given the $N-1$ other shares. As $\mathsf{sk}_e^{(i^*)}$ is independent from the seeds $\mathcal{A}$ observes, the distribution of $\Delta\mathsf{sk}_e^*$ is identical and $\mathcal{A}$ has no information about sk^*. As $\mathcal{P}_{i^*}$ is never opened, $\mathcal{B}$'s cheating on $\mathsf{ct}_e^{(i^*)}$ 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 $\mathsf{EUF\text{-}CMA}$ 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 commitment it replaces is queried, or if its queries to H_2 and H_3 have been made previously. Let Q_{st} 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 $Q_{\mathsf{st}}/2^{2\kappa}$ 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 $\mathsf{sd}_e^{(i^*)}$ is a uniformly random κ-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_2, resp. H_3, $\mathcal{B}$ has a maximum probability of $Q_2/2^{2\kappa}$, resp. $Q_3/2^{2\kappa}$ of aborting, where Q_2 and Q_3 denote the number of queries made to each random oracle during the game. Note that $\mathcal{B}$ samples one salt, replaces τ commitments and makes one query to both H_2 and H_3 for each signature query.

Therefore

$$\mathsf{G}_0 - \mathsf{G}_1 \leq Q_s \cdot \left(\tau \cdot \epsilon_{\mathrm{PRG}} + \mathbf{Adv}^{\mathsf{Hiding}}_{\mathsf{Tree}} + \Pr[\mathcal{B} \text{ aborts}]\right)$$

where

$$\begin{aligned}\Pr[\mathcal{B} \text{ aborts}] &\leq Q_{\mathsf{st}}/2^{2\kappa} + Q_c/2^{\kappa} + Q_2/2^{2\kappa} + Q_3/2^{2\kappa} \\ &= (Q_{\mathsf{st}} + Q_2 + Q_3)/2^{2\kappa} + Q_c/2^{\kappa}\ .\end{aligned}$$

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 pk with and instance pk^* of the $\mathsf{EUF\text{-}KO}$ 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 $\mathsf{EUF\text{-}KO}$ game, and so we have

$$\mathsf{G}_1 \leq \epsilon_{\mathrm{KO}} \leq \epsilon_{\mathrm{OWF}} + \varepsilon(Q_{\mathsf{c}}, Q_1, Q_2, Q_3)$$

where the bound on advantage ϵ_{KO} of a $\mathsf{EUF\text{-}KO}$ attacker follows from Lemma 2. By a union bound, we have that

$$\begin{aligned}\mathbf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A}} &\leq \epsilon_{\mathrm{OWF}} + \varepsilon(Q_{\mathsf{c}}, Q_1, Q_2, Q_3) \\ &\quad + Q_s \cdot \left(\tau \cdot \epsilon_{\mathrm{PRG}} + \mathbf{Adv}^{\mathsf{Hiding}}_{\mathsf{Tree}} + \frac{Q_{\mathsf{st}} + Q_2 + Q_3}{2^{2\kappa}} + \frac{Q_c}{2^{\kappa}}\right).\end{aligned}$$

Assuming that Expand is a PRG, that the seed tree construction is hiding, that $f_{\mathbf{x}}$ is a one-way function and that parameters (N, τ, m_2, λ) are appropriately chosen implies that $\mathbf{Adv}_{\mathcal{A}}^{\mathsf{EUF\text{-}CMA}}$ is negligible in κ.

Strong unforgeability. Our analysis uses the EUF-CMA definition and therefore does not rule out the case that an attacker can find a new signature for a previously signed message (for instance, by mauling a signature output by the signer in such a way that it remains valid on the same message). Intuitively, since the intermediate seeds from the tree are the only part of the signature that the verifier does not hash directly during verification (the verifier only hashes the leaf seeds of the tree), as a minimum we require that (i) it is difficult to find a set of intermediate seeds in the tree that derive the same leaf seeds used in a signature, and (ii) it is difficult to find a different seed that produces the same outputs when input to Expand as a seed used in the signature. Since we use cryptographic hash functions to derive seeds and to instantiate Expand, these requirements should be met under the assumption that the hash function is 2nd-preimage resistant (or is a random oracle). A formal proof of strong unforgeability is nevertheless an interesting open question.

QROM Security. The quantum ROM is a stronger version of the ROM that allows attackers to make superposition queries to the RO; this models attacks on PQ primitives which make use of a quantum computer. As Banquet aims to provide post-quantum security, whether our analysis holds in the QROM is a natural question. The most promising approach seems to be the general results for multi-round Fiat–Shamir type signatures [19, 20], since our current reduction makes essential use of the RO query history, ruling out the "history-free" approach of [11]. However, in order to apply the QROM EUF-CMA result [19, Theorem 23] would require that we formulate Banquet as Σ-protocol, and prove multiple properties about it, which is beyond the scope of the current paper. Finally we note that the amount of assurance provided by QROM for signature schemes is debatable, as there are no known non-contrived schemes that are secure in the ROM, but insecure in the QROM.

6 Parameters, Implementation and Performance

We first describe how we chose parameters for Banquet, and give some options. We then describe our implementation[8] and the optimizations we use to improve performance of Sign and Verify, which can be improved significantly over a direct implementation of the scheme from Section 4. We then compare Banquet to some other post-quantum signature schemes, and finally discuss some other features of the design. In this section all times are given in milliseconds, by averaging over 100 runs, on an Intel Xeon W-2133 CPU @ 3.60GHz, unless noted otherwise.

[8] The implementation is publicly available at https://github.com/dkales/banquet.

6.1 Parameter Selection

The soundness error of the signature scheme from Section 4 depends on the parameters $(\kappa, N, \lambda, m_1, m_2, \tau)$. Our analysis of Banquet is similar to the analysis of the 7-round ID schemes in [8] and the five-round schemes in [29]. We can bound the probability of cheating by assuming that the attacker can cheat by guessing any one of the challenges in a given parallel repetition. Let τ_1 be the number of repetitions for which the attacker will guess the first challenge, and τ_2 be the number for which she will guess the second challenge. Since the attacker is successful in a repetition by being successful in any of the challenges, the number of repetitions where the challenge must be guessed in the third round is $\tau_3 = \tau - \tau_1 - \tau_2$. The cost of the attack is given by

$$C = 1/P_1 + 1/P_2 + 1/P_3$$

where P_i is the probability of correctly guessing τ_i sub-challenges in challenge step i. We call a triple (τ_1, τ_2, τ_3) an attack *strategy*. Our goal is to choose parameters such that $C > 2^\kappa$ for all strategies.

The first challenge space (Phase 2) has size $2^{8\lambda m_1}$, and the probability that a given challenge allows cheating is $2^{-8\lambda}$ (as shown in Section 3.2). Therefore, the probability of guessing τ_1 of τ challenges is

$$P_1 = \sum_{k=\tau_1}^{\tau} \mathsf{PMF}(k, \tau, 2^{-8\lambda}) \ ,$$

where PMF is the probability mass function:

$$\mathsf{PMF}(k, \tau, p) = \binom{\tau}{k} p^k (1-p)^{\tau-k}$$

which gives the probability of getting exactly k successes in τ independent trials each having probability p of success.

The second challenge space (Phase 4) has size $2^{8\lambda} - m_2$, and the probability that a given challenge allows cheating is $2m_2/(2^{8\lambda} - m_2)$. The probability of guessing τ_2 of $\tau - \tau_1$ challenges correctly is therefore

$$P_2(\tau_2) = \sum_{k=\tau_2}^{\tau-\tau_1} \mathsf{PMF}(k, \tau - \tau_1, 2m_2/(2^{8\lambda} - m_2)) \ .$$

The third challenge space (Phase 6) has size N, and the attacker must guess the challenge in the remaining repetitions correctly, therefore

$$P_3(\tau_3) = N^{-\tau_3} \ .$$

To choose parameters, we fix $(\kappa, N, \lambda, m_1, m_2)$, then start with $\tau = 1$, and increase it, at each step checking the attack costs for all possible strategies (τ_1, τ_2, τ_3). When we reach τ where the attack cost exceeds 2^κ for all strategies, we output τ as the number of repetitions required for κ-bit security. Since τ is always less than 100, and τ_3 is fixed once τ_1 and τ_2 are chosen, using a script to perform this exhaustive search is practical.

Choice of m_1, m_2. We found that choosing $m_1 \approx \sqrt{m}$ gave good performance in practice. For example at L1, when $m = 200$, we chose $m_1 = 10$, and $m_2 = 20$. The signature size and runtime does not change significantly for small changes in m_i (e.g., $m_1 = 8$, $m_2 = 25$ is about the same), but signature sizes increase as we move m_1 further away from $\sqrt{m}$.

Number of parties. Increasing N allows us to decrease signature size, but increases the cost of signing and verification. The choices of N we found interesting are powers of two from 16 to 256, with 64 being a sweet spot on the curve. Our search started at these powers of two, but often we were able to decrease N slightly without increasing τ, which improves sign and verify times.

Choice of λ. We generated parameters for $\lambda = 2, 3, 4, 5, 6$ so that field arithmetic happens in fields of size 16 to 48 bits, which are convenient for processor word sizes, and these values provide sufficient soundness. We benchmarked multiple parameter sets with $\lambda = 4$ and $\lambda = 6$, and in general $\lambda = 6$ is slightly faster, but has slightly larger signatures (about 0.5-1KB larger). Table 2 shows the differences.

L3 and L5 Parameters. As in the BBQ scheme, scaling the Banquet design to the 192 and 256-bit security levels presents a challenge because the block size of AES is limited to 128 bits. Simply using AES-256 with 128-bit outputs does not provide 256-bit security; intuitively, in this case there are a large number of 256-bit keys that produce the same ciphertext for any fixed plaintext, and finding any one of them allows an attacker to create a forgery.

BBQ explores two options for higher security parameters. The first is to use the Rijndael cipher, which has parameter sets with 192 and 256-bit block sizes, and the second is to make two calls to AES-192 or AES-256 in ECB mode (using two different plaintext blocks), which we refer to as AES-192x2 and AES-256x2 respectively.

The Rijndael option has 80 fewer S-boxes at the 192-bit level, and we estimate that Banquet signatures with Rijndael would be $\approx$ 2KB shorter. At L5 however, two calls to AES-256 has 60 fewer S-boxes, as the Rijndael key schedule is more expensive. Since the difference is small, we use only AES at all three levels.

Public-key uniqueness. As noted in [16], the ECB construction trivially has two public keys for each secret key, where the plaintext and ciphertext blocks in the public key are swapped. Since the Banquet design includes the public key as an input to the challenge computation (prepending the public key to the signed message), this ensures that the verifier is using the same public key as the signer.

6.2 Implementation and Optimizations

We have implemented Banquet in C++ in order to determine the running time of sign and verify operations. We began with a direct implementation of the

Scheme	N	λ	τ	Sign (ms)	Verify (ms)	Size (bytes)
AES-128	16	4	41	6.36	4.86	19776
	16	6	37	5.91	4.51	20964
	31	4	35	8.95	7.46	17456
	31	6	31	8.19	6.76	18076
	57	4	31	14.22	12.30	15968
	57	6	27	12.45	10.75	16188
	107	4	28	24.15	21.71	14880
	107	6	24	21.13	18.96	14784
	255	4	25	51.10	46.88	13696
	255	6	21	43.81	40.11	13284
AES-192x2	16	4	62	17.25	13.15	51216
	16	6	57	16.52	12.50	53936
	31	4	53	25.82	21.66	45072
	31	6	47	24.00	19.89	45624
	64	4	46	43.03	38.00	40240
	64	6	40	39.07	34.15	39808
	116	4	42	69.37	62.68	37760
	116	6	36	61.95	55.55	36704
	256	4	38	135.29	124.50	35088
	256	6	32	119.01	108.53	33408
AES-256x2	16	4	84	27.78	21.67	83488
	16	6	75	25.71	19.88	84610
	31	4	72	41.40	35.20	73888
	31	6	63	37.67	31.76	73114
	62	4	63	67.99	60.33	66688
	62	6	54	60.79	53.39	64420
	119	4	56	112.62	102.52	61088
	119	6	48	100.66	90.76	58816
	256	4	50	213.58	196.27	56160
	256	6	43	190.58	174.70	54082

Table 2. Performance metrics of different parameter sets for Banquet. All instances for AES-128 have $(m, m_1, m_2) = (200, 10, 20)$ for AES-192x2 we have $(416, 16, 26)$ and for AES-256x2 we have $(500, 20, 25)$.

scheme, as presented in Section 4, where all field and polynomial operations were done with the NTL library[9]. However, sign and verify times were on the order of multiple seconds, due to the cost of the polynomial operations in Phase 3.

As a first optimization, note that when interpolating a set of points (x_i, y_i), the x_i values are fixed, therefore, we can precompute the Lagrange coefficients, and save computation later. However, we can avoid most interpolation altogether. Rather than computing the per-party shares of the polynomials $S^{(i)}, T^{(i)}$, by interpolating shares of points, the prover can first reconstruct the points from the shares and interpolate them to get S, T and P (in unshared form). Then in Phase 4, the shares of points previously used to interpolate $S^{(i)}, T^{(i)}$ are used to compute $(a^{(i)}, b^{(i)})$. Applying this technique to polynomials S and T (and P in

[9] https://shoup.net/ntl/

a similar fashion) reduces the number of interpolations from $Nm_1\tau$ to $m_1\tau$ (of polynomials of degree m_2), while for P, the number drops from $N\tau$ to τ interpolations (of degree $2m_2+1$). We can reduce this further, observing that in each of the τ instances, the polynomials S, T are defined using the same set of points, except for the additional randomizing points $\bar{s}, \bar{t}$ (and a constant random multiplier for S). We therefore interpolate each of these only once for all instances, and then adjust to the correct polynomial in each instance by adding a multiple of the Lagrange polynomial corresponding to the last point. Incorporating this into the computation of P, we do $2m_1$ interpolations in total to compute P, however, some of the steps in the protocol still require the evaluation of the polynomials $S^{(i)}$, $T^{(i)}$ at the point R_e. For this calculation, we also do not need to interpolate the polynomials, but instead evaluate the Lagrange coefficients at the common evaluation point R_e beforehand, reducing the needed work from a full interpolation to a single inner product.

Taken together, these optimizations reduced the time of sign and verify by more than 30x, to be on the order of 100ms. Finally, we replaced NTL with a dedicated field arithmetic implementation optimized for our application (avoiding dynamic memory allocation, and other options NTL has to be flexible) and the small binary fields that we are working with. This reduced the runtime by a further factor of 4x. However, we have not invested the resources to ensure that our implementation runs in constant time, some parts of the implementation may be susceptible to timing and/or memory access leaks. In our final implementation, we make use of hardware instructions for binary field multiplication (`PCLMUL`) and the AES hardware instructions (`AES-NI`) whenever possible, although the structure of the MPC execution does not allow a straightforward application of the AES instructions due to the injection of the inverse values in the protocol. However, we can still use small tricks like implementing the MixColumns layer by a trifold application of the inverse MixColumns instruction (for the AES MixColumns matrix it holds that $M = (M^{-1})^3$), giving us a speedup of about 10% compared to a naive implementation.

At L1, the final implementation (for $N = 57, \tau = 31, \lambda = 4$) spends about 3% of the time on computing the MPC evaluation of the AES circuit(s), 38% on finite field arithmetic, 31% on hashing, and about 25% on allocating/copying memory. Our implementation uses the SHAKE extensible output function both as a hash function and PRG (details are given in [4]).

6.3 Performance

In Table 3, we compare the proof size of Banquet with BBQ [16], and a Picnic-like signature scheme with a binary circuit for AES based on the KKW protocol [31]. Our Banquet instances provide much smaller signatures than all previous MPCitH-based signatures using AES. We cannot compare the performance here since no implementation is available for BBQ or AES Bin. However, we can estimate the performance of BBQ and AES Bin based on the implementation of Picnic, where a large part of the signing and verification cost is comprised of similar operations such as the seed tree generation, hashing of messages and

Protocol	N	M	τ	Size (bytes)
AES Bin	64	343	27	51 876
BBQ	64	343	27	31 568
Banquet	16	-	41	19 776
	107	-	24	14 784
	255	-	21	13 284
AES Bin	64	570	39	149 134
BBQ	64	570	39	86 888
Banquet	16	-	62	51 216
	116	-	36	36 704
	256	-	32	33 408
AES Bin	64	803	50	233 696
BBQ	64	803	50	137 670
Banquet	16	-	84	83 488
	119	-	48	58 816
	256	-	43	54 082

Table 3. Comparison of signature sizes. N is the number of parties used in the MPC, M is the total number of MPC instances (when preprocessing is used), and τ is the number of online executions (equal to the number of parallel repetitions in Banquet). The rows are grouped by security level in order L1, L3, L5.

computation of the pre-processing phase. The only difference is that the LowMC circuit is replaced by the AES circuit, but again, the structure of the MPC operations does not allow straightforward use of AES hardware instructions. In addition, since the internal states of the parties are larger due to the larger circuit of AES, we expect most of the hashing and seed expansion to take slightly longer. Overall, we would estimate the performance of a highly optimized implementation of BBQ or AES Bin to match the performance of Picnic2 instances, however with significantly larger signatures.

In Table 4 we compare the signature size and signing/verification times for Banquet to other post-quantum signature schemes: Picnic2 and Picnic3 [30], the previous and latest version of Picnic (based on KKW and LowMC), respectively, and SPHINCS$^+$ [6]. At the L1 security level, our Banquet implementation can sign a message in 6.36ms and verify it in 4.86ms. Compared to Picnic3 this corresponds to a slowdown of factor of only 1.2x, while the signature size is about 50% larger than that of Picnic3. However, different parameter sets for Banquet can reduce its signature size at the cost of slower signing and verification speeds. When comparing to SPHINCS$^+$, Banquet can offer faster signing speeds and smaller signatures, however SPHINCS$^+$ verification is faster. At the L3 and L5 security levels, the relative performance of Picnic3 and Banquet remains similar: our 16-party Banquet instances are about 1.6x slower than Picnic3 with signatures that are about 50% larger. Banquet signing speeds are still comparable to SPHINCS$^+$, however SPHINCS$^+$ signatures are smaller.

Reduced round AES. Using the full 10 rounds of AES-128 arguably provides excessive security margin against key recovery attacks in the restricted attack

Protocol	N	M	τ	Sign (ms)	Ver (ms)	Size (bytes)
Picnic2	64	343	27	41.16	18.21	12 347
	16	252	36	10.42	5.00	13 831
Picnic3	16	252	36	5.33	4.03	12 466
SPHINCS$^+$-fast	-	-	-	14.42	1.74	16 976
SPHINCS$^+$-small	-	-	-	239.34	0.73	8 080
Banquet	16	-	41	6.36	4.86	19 776
	107	-	24	21.13	18.96	14 784
	255	-	21	43.81	40.11	13 284
Picnic2	64	570	39	123.21	41.25	27 173
	16	420	52	29.85	11.77	30 542
Picnic3	16	419	52	11.01	8.49	27 405
SPHINCS$^+$-fast	-	-	-	19.05	2.82	35 664
SPHINCS$^+$-small	-	-	-	493.17	1.12	17 064
Banquet	16	-	62	17.25	13.15	51 216
	116	-	36	61.95	55.55	36 704
	256	-	32	119.01	108.53	33 408
Picnic2	64	803	50	253	71.32	46 162
	16	604	68	61.09	21.19	52 860
Picnic3	16	601	68	18.82	13.56	48 437
SPHINCS$^+$-fast	-	-	-	38.71	2.89	49 216
SPHINCS$^+$-small	-	-	-	310.37	1.46	29 792
Banquet	16	-	84	27.78	21.67	83 488
	119	-	48	100.66	90.76	58 816
	256	-	43	190.85	174.70	54 082

Table 4. Comparison of signature sizes and run times for various MPCitH-based signature schemes and SPHINCS$^+$ (using "`sha256simple`" parameter sets). N is the number of parties, M is the total number of MPC instances (when preprocessing is used), and τ is the number of online executions (equal to the number of parallel repetitions in Banquet). The rows are grouped by security level in order L1, L3, L5.

setting applicable to Banquet. A forger that attacks the AES instance used in Banquet key generation must recover an AES secret key given only a single plaintext-ciphertext pair (the Banquet public key). It may thus be reasonable to assume that AES reduced to 7 rounds provides 128 bits of security in this restricted setting. With 7-round AES, $m_1 = 10$, $m_2 = 14$, Banquet signatures with parameters (N, λ, τ) $(64, 4, 31)$ are 13.36KB, with $(128, 5, 25)$ are 12.10KB and with $(256, 5, 22)$ are 11.01KB. As there are fewer parallel repetitions and polynomials of smaller degree we can expect faster sign and verify operations as well. The goal of this paper was to construct PQ signatures using only standardized primitives, so we did not explore this further; however it appears to be an interesting direction for future work.

Interactive Identification The signature scheme of Section 4 may be used as an interactive protocol between a prover and verifier, where the prover runs

Scheme	N	Rep.	Prover (ms)	Verifier (ms)	Communication (bytes)
Banquet	16	11	1.94	1.44	4 452
	23	9	2.06	1.60	3 804
	55	7	3.15	2.66	3 092
	109	6	4.70	4.16	2 760
Picnic3	16	$(72, 12)$	1.73	1.33	4 070
	16	$(48, 16)$	1.16	0.92	4 750

Table 5. Benchmarks of interactive identification schemes at security level L1. All Banquet parameters have $m_1 = 10$, $m_2 = 20$ and $\lambda = 2$. The column "Rep" gives the number of parallel repetitions τ used in Banquet, and (M, τ) the number of MPC instances and online executions in Picnic3.

phases 1, 3 and 5, while the verifier runs phases 2, 4 and 6. For phase 2, the prover sends a commitment to σ_1, and the verifier responds with a random bitstring h_1. Similarly, in phase 4, and 6 the prover sends commitments to σ_2, σ_3 (respectively) and the verifier responds with random bitstrings h_2, h_3.

Let p_1, p_2 and p_3 denote the probability of guessing the first, second or third challenge (respectively). Recall that $p_1 = 2^{-8\lambda}$, $p_2 = m_2/(2^{8\lambda} - m_2)$ and $p_3 = 1/N$. The soundness error of the interactive protocol is

$$\epsilon = p_1 + (1 - p_1)p_2 + (1 - p_1)(1 - p_2)p_3$$

since a dishonest prover wins if: either she gets the first challenge right, or not the first but the second or neither the first nor the second but the third. We can reduce the soundness error to ϵ^τ by running τ parallel repetitions.

To target t bits of interactive security we choose τ so that $\epsilon^\tau < 2^{-t}$. Table 5 gives some of the costs when $t = 40$. We see that Banquet needs very few parallel repetitions, and can use a very small field size $\lambda = 2$. When compared to the Picnic3 ID scheme (details in [30, §7]), the communication costs with 16 parties are about the same, and then are less with Banquet as N increases. A similar tradeoff could be made with Picnic3, which also requires less time. Finally, the reduced number of roundtrips make Picnic easier to fit into existing flows of network protocols. That said, the performance of Banquet remains competitive in this setting, and it has a more conservative security assumption.

References

1. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 430–454. Springer, Heidelberg (Apr 2015)
2. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: ACM CCS 2017. pp. 2087–2104. ACM Press (Nov 2017)
3. Baum, C., Nof, A.: Concretely-efficient zero-knowledge arguments for arithmetic circuits and their application to lattice-based cryptography. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part I. LNCS, vol. 12110, pp. 495–526. Springer, Heidelberg (May 2020)

4. Baum, C., de Saint Guilhem, C.D., Kales, D., Orsini, E., Scholl, P., Zaverucha, G.: Banquet: Short and fast signatures from AES. Cryptology ePrint Archive, Report 2021/068 (2021), full version of this paper. https://eprint.iacr.org/2021/068
5. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for R1CS. In: EUROCRYPT 2019, Part I. LNCS, vol. 11476, pp. 103–128. Springer, Heidelberg (May 2019)
6. Bernstein, D.J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P.: The SPHINCS$^+$ signature framework. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 2129–2146. ACM Press (Nov 2019)
7. Beullens, W.: Sigma protocols for MQ, PKP and SIS, and Fishy signature schemes. In: Canteaut, A., Ishai, Y. (eds.) EUROCRYPT 2020, Part III. LNCS, vol. 12107, pp. 183–211. Springer, Heidelberg (May 2020)
8. Beullens, W., de Saint Guilhem, C.: LegRoast: Efficient post-quantum signatures from the Legendre PRF. In: Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020. pp. 130–150. Springer, Heidelberg (2020)
9. Bhadauria, R., Fang, Z., Hazay, C., Venkitasubramaniam, M., Xie, T., Zhang, Y.: Ligero++: A new optimized sublinear IOP. In: CCS. pp. 2025–2038. ACM (2020)
10. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Zero-knowledge proofs on secret-shared data via fully linear PCPs. In: CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 67–97. Springer, Heidelberg (Aug 2019)
11. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 41–69. Springer, Heidelberg (Dec 2011)
12. Boyle, E., Gilboa, N., Ishai, Y., Nof, A.: Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) ACM CCS 2019. pp. 869–886. ACM Press (Nov 2019)
13. Casanova, A., Faugere, J.C., Macario-Rat, G., Patarin, J., Perret, L., Ryckeghem, J.: Gemss: A great multivariate short signature. Submission to the NIST's post-quantum cryptography standardization process (2017)
14. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1825–1842. ACM Press (Oct / Nov 2017)
15. De Feo, L., Galbraith, S.D.: SeaSign: Compact isogeny signatures from class group actions. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part III. LNCS, vol. 11478, pp. 759–789. Springer, Heidelberg (May 2019)
16. de Saint Guilhem, C., De Meyer, L., Orsini, E., Smart, N.P.: BBQ: Using AES in picnic signatures. In: Paterson, K.G., Stebila, D. (eds.) SAC 2019. LNCS, vol. 11959, pp. 669–692. Springer, Heidelberg (Aug 2019)
17. Ding, J., Schmidt, D.: Rainbow, a new multivariable polynomial signature scheme. In: Ioannidis, J., Keromytis, A., Yung, M. (eds.) ACNS 05. LNCS, vol. 3531, pp. 164–175. Springer, Heidelberg (Jun 2005)
18. Dinur, I., Liu, Y., Meier, W., Wang, Q.: Optimized interpolation attacks on LowMC. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part II. LNCS, vol. 9453, pp. 535–560. Springer, Heidelberg (Nov / Dec 2015)
19. Don, J., Fehr, S., Majenz, C.: The measure-and-reprogram technique 2.0: Multi-round Fiat-Shamir and more. In: CRYPTO 2020, Part III. LNCS, vol. 12172, pp. 602–631. Springer, Heidelberg (Aug 2020)
20. Don, J., Fehr, S., Majenz, C., Schaffner, C.: Security of the Fiat-Shamir transformation in the quantum random-oracle model. In: Boldyreva, A., Micciancio, D.

(eds.) CRYPTO 2019, Part II. LNCS, vol. 11693, pp. 356–383. Springer, Heidelberg (Aug 2019)
21. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium: A lattice-based digital signature scheme. IACR TCHES 2018(1), 238–268 (2018)
22. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987)
23. Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon: Fast-Fourier lattice-based compact signatures over NTRU. Submission to the NIST's post-quantum cryptography standardization process (2018)
24. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC. pp. 197–206. ACM Press (May 2008)
25. Giacomelli, I., Madsen, J., Orlandi, C.: ZKBoo: Faster zero-knowledge for Boolean circuits. In: Holz, T., Savage, S. (eds.) USENIX Security 2016. pp. 1069–1083. USENIX Association (Aug 2016)
26. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Computing 17(2), 281–308 (1988)
27. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Zero-knowledge from secure multiparty computation. In: Johnson, D.S., Feige, U. (eds.) 39th ACM STOC. pp. 21–30. ACM Press (Jun 2007)
28. Kales, D., Ramacher, S., Rechberger, C., Walch, R., Werner, M.: Efficient FPGA implementations of LowMC and Picnic. In: Jarecki, S. (ed.) CT-RSA 2020. LNCS, vol. 12006, pp. 417–441. Springer, Heidelberg (Feb 2020)
29. Kales, D., Zaverucha, G.: An attack on some signature schemes constructed from five-pass identification schemes. In: Krenn, S., Shulman, H., Vaudenay, S. (eds.) CANS 20. LNCS, vol. 12579, pp. 3–22. Springer, Heidelberg (Dec 2020)
30. Kales, D., Zaverucha, G.: Improving the performance of the Picnic signature scheme. IACR TCHES 2020(4), 154–188 (2020)
31. Katz, J., Kolesnikov, V., Wang, X.: Improved non-interactive zero knowledge with applications to post-quantum signatures. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 525–537. ACM Press (Oct 2018)
32. Liu, F., Isobe, T., Meier, W.: Cryptanalysis of full LowMC and LowMC-M with algebraic techniques. Cryptology ePrint Archive, Report 2020/1034 (2020)
33. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (Apr 2012)
34. National Institute of Standards and Technology: Round 3 Submissions for the NIST Post-Quantum Cryptography Project. https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions, accessed 11-11-2020 (2020)
35. Rechberger, C., Soleimany, H., Tiessen, T.: Cryptanalysis of low-data instances of full LowMCv2. IACR Trans. Symm. Cryptol. 2018(3), 163–181 (2018)
36. Zaverucha, G., Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Katz, J., Wang, X., Kolesnikov, V.: Picnic. Tech. rep., National Institute of Standards and Technology (2019), available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions