Shorter Signatures from MQ

William Wang

ww@priv.pub

Abstract. We describe a simple MPC protocol in the preprocessing model for computing multivariate quadratic maps. This yields Mesquite, a KKW-style signature scheme which, to our knowledge, produces the shortest signatures of any based on the MQ assumption. For example, our compact parameter set targeting NIST security level I has an average signature size of 8.8KB and runtimes on par with Picnic3 L1.

Keywords: multivariate cryptography, MPC-in-the-head, post-quantum digital signatures

1 Introduction

Traditionally, multivariate quadratic cryptography has relied on inverting special classes of maps engineered to have a trapdoor. However, these are ad hoc assumptions and recent attacks have reduced the security of well-known constructions [Beu21; Beu22]. Another line of research focuses on the comparatively well-studied MQ problem, which asks to invert a random multivariate quadratic map. In particular, MQDSS [Sam+19] and MUDFISH [Beu20] are two signature schemes based on this hardness assumption.

MPC-in-the-head, introduced by [Ish+07], is a paradigm for obtaining zero-knowledge proofs from multiparty computation (MPC). To prove knowledge for some NP relation, the prover simulates an MPC protocol which verifies the witness. They commit to each party's view, and the verifier challenges the prover to open a subset of views. Correctness and zero-knowledge follow from the protocol's correctness and security guarantees against semi-honest parties. The soundness error is determined by the prover's ability to covertly inject inconsistent views.

To obtain shorter proofs, [KKW18] proposed using MPC protocols in the preprocessing model, i.e. with trusted setup. This is converted into a proof via "cut-and-choose" methods: for each commited execution, the verifier may choose to open all views after preprocessing, or a subset after the full MPC protocol. This idea was generalized in [Beu20] to arbitrary sigma protocols, including the one used by MUDFISH. Our approach is similar, but fully adopts the KKW construction.

1.1 Preliminaries

Commitment schemes. We use a non-interactive commitment scheme, denoted Com. As described in [KKW18], we do not need additional randomness; a proof

of security is given in [Zav+20, §6.1]. In practice, we use a hash function $H : \{0,1\}^* \to \{0,1\}^{2\kappa}$, where κ is the security parameter.

Merkle and seed trees. The KKW construction employs tree-based strategies to reduce signature size, which we briefly describe here. We use Merkle trees to open a τ-sized subset of M commitments, which requires revealing at most $\tau \log \frac{M}{\tau}$ nodes.

Seed trees are a similar construction which allow revealing all but a certain subset of pseudorandomly generated seed values. The idea is to first assign a seed value to the root of a binary tree. We then recursively expand each internal node to its children with a hash function $H : \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$. The leaves now function as independent seed values, and to conceal a subset we reveal nodes analogous to a Merkle tree.

Multivariate quadratics. A multivariate quadratic map $\mathcal{F} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ is a system of m quadratic polynomials in n variables defined over some finite field $\mathbb{F}_q$. The $\mathsf{MQ}_{n,m,q}$ problem is, for uniformly random $\mathcal{F} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ and $s \in \mathbb{F}_q^n$, to find s given $\mathcal{F}$ and $\mathcal{F}(s)$. Without loss in hardness, we assume $\mathcal{F}$ has no constant term, i.e. $\mathcal{F}(0) = 0$. In this case, the polar form

$$\mathcal{G}(x, y) := \mathcal{F}(x + y) - \mathcal{F}(x) - \mathcal{F}(y)$$

is bilinear. We use the hardness analysis of MQ from [Sam+19, §2].

2 Proof of knowledge

Let $\mathcal{F} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ be a multivariate quadratic map of m polynomials in n variables, and $\mathcal{G}$ be its polar form. Notice the identity

$$\mathcal{F}(s) - \mathcal{F}(s - r) = \mathcal{G}(r, s - r) + \mathcal{F}(r).$$

We describe an MPC protocol in the preprocessing model for evaluating $\mathcal{F}(s)$. In the preprocessing phase, the parties are dealt additive secret shares u_i and v_i for r and $\mathcal{F}(r)$ respectively, where $r \in \mathbb{F}_q^n$ is uniform. During protocol execution, the parties begin with $s - r$. They locally compute $\mathcal{G}(u_i, s - r) + v_i$, which constitute shares of $\mathcal{F}(s) - \mathcal{F}(s - r)$. These are recombined and the parties add $\mathcal{F}(s - r)$ to obtain $\mathcal{F}(s)$. It is straightforward to show the protocol is correct and secure against semi-honest corruption of all-but-one parties.

Using the KKW construction, we obtain a three-round proof which is summarized below. To simulate an execution of the MPC protocol:

1. Given a seed value, generate an N-leaf seed tree.
2. For each $i \in [N]$, expand the i-th leaf to $u_i \in \mathbb{F}_q^n$. This constitutes a secret sharing for $r := \sum_{i=1}^N u_i$.
3. For each $i \in [N - 1]$, additionally expand the i-th leaf to $v_i \in \mathbb{F}_q^m$. This constitutes a secret sharing for $\mathcal{F}(r)$, where the final share is $v_N := \mathcal{F}(r) - \sum_{i=1}^{N-1} v_i$.

4. For each $i \in [N]$, commit to the i-th party's view with

$$\mathsf{com}_i := \begin{cases} \mathsf{Com}(i\text{-th leaf}) & i \in [N-1] \\ \mathsf{Com}(N\text{-th leaf} \parallel v_N) & i = N. \end{cases}$$

5. Commit to the online phase with

$$\mathsf{com}^* := \mathsf{Com}(s - r \parallel \mathcal{G}(u_1, s-r) + v_1 \parallel \cdots \parallel \mathcal{G}(u_N, s-r) + v_N).$$

The proof of knowledge:

1. Given a seed value, the prover generates an M-leaf seed tree. For each $j \in [M]$, they simulate an execution with the j-th leaf. They construct an M-leaf Merkle tree whose j-th leaf is com^* from the j-th execution. The prover hashes the com_i values from each execution along with the Merkle tree's root, then sends this to the verifier.
2. The verifier challenges with a uniform τ-sized subset $C \subset [M]$ along with a uniform party index $p_j \in [N]$ for each $j \in C$.
3. For each $j \in C$, the prover sends from the j-th execution:
   - the nodes required to reveal all but the p_j-th leaf from the N-leaf seed tree;
   - com_{p_j};
   - $s - r$;
   - and v_N if $p_j \neq N$.

   Additionally, they send the nodes required to reveal all but the C-th leaves from the M-leaf seed tree and open the C-th leaves from the Merkle tree.
4. For each $j \in C$, the verifier recomputes from the j-th execution:
   - $(\mathsf{com}_i, u_i, v_i)_{i \neq p_j}$ using the partially reconstructed N-leaf seed tree;
   - the p_j-th party's broadcast

$$\mathcal{F}(s) - \mathcal{F}(s-r) - \sum_{i \neq p_j} [\mathcal{G}(u_i, s-r) + v_i],$$

     and subsequently com^*.

   Using the partially reconstructed M-leaf seed tree, they recompute the j-th execution's com_i values for each $j \notin C$. Finally, the verifier derives the Merkle tree's root, and checks that their results match the prover's commit.

The soundness error is proven in [BN20, §3.2] to be

$$\max_{0 \leq e \leq \tau} \frac{\binom{M-e}{\tau-e}}{\binom{M}{\tau} N^{\tau-e}}.$$

Comparison with MUDFISH. We briefly describe MUDFISH [Beu20], a similar MQ-based proof, from the perspective of MPC with preprocessing. The prover first samples $r \in \mathbb{F}_q^n$ and, for each $c \in \mathbb{F}_q$, secret sharings $u_1 + u_{2,c} = cr$ and $v_1 + v_{2,c} = c\mathcal{F}(r)$. Note that u_1, v_1 are identical across all sharings.

During the online phase, the prover simulates q two-party computations, which are related in the sense that the first party's broadcast is identical. The verifier's challenge is to reveal one of the second party's views, and hence the soundness error is $1/q$. The advantage of MUDFISH is that both prover and verifier only need to evaluate $\mathcal{G}(\cdot, s - r)$ once, improving runtime. The primary disadvantage is that soundness error is bounded by the finite field's size. With our proof the soundness error is $1/N$, and tuning it lower can significantly reduce signature size. Furthermore, we achieve shorter signatures by virtue of revealing nodes in a seed tree to open views, whereas MUDFISH must use a Merkle tree.

It is also possible to view the two as limiting cases of a more general protocol. The idea is now to perform q related N-party computations, where all-but-one parties have fixed shares. Based on our analysis this yields larger signatures without improving runtime, so we did not pursue the idea further.

3 Signature scheme

We present the signature scheme Mesquite, which is obtained via the Fiat-Shamir transform. The key generation algorithm samples a multivariate quadratic map $\mathcal{F}$ and vector $s \in \mathbb{F}_q^n$, then computes $t = \mathcal{F}(s)$. We employ the standard practice of expanding a seed into $\mathcal{F}$ with a pseudorandom generator. The public key is the seed with t, and the private key additionally includes s. The maximum size of a signature is

$$2\kappa + 3\kappa \left\lceil \tau \log \frac{M}{\tau} \right\rceil + \tau \cdot (\kappa \lceil \log N \rceil + 2\kappa + (n+m) \log q)$$

bits, although in practice the average is smaller.

Multi-target attacks. To prevent multi-target attacks described in [DN19], the signing algorithm must employ domain separation and counters to invoke the hash function. Furthermore, a unique "salt" value must be used per signature to key the hash function. The Picnic specification addresses this by generating a new salt per signature. We use an alternative method which does not increase signature size. The idea is for the salt to be derived from the public key and message, in conjunction with deterministic signing (i.e. coins are derived from the secret key and message).

Optimizing performance. Evaluating multivariate quadratics and their polar forms is relatively expensive. Naively simulating an MPC execution as described above is slow, but this can be optimized.

– The prover does not need to explicitly compute $\mathcal{F}(r)$, since

$$v_N = \mathcal{F}(s) - \mathcal{F}(s-r) - \sum_{i=1}^{N} \mathcal{G}(u_i, s-r) - \sum_{i=1}^{N-1} v_i.$$

– We amortize the cost of evaluating the linear map $\mathcal{G}(\cdot, s-r)$ by caching its matrix representation. Concretely, each quadratic in $\mathcal{F}$ has the matrix form

$$f_i(x) = x^\top A_i x + b_i^\top x,$$

and the corresponding map in $\mathcal{G}$ is $g_i(x,y) = x^\top (A_i + A_i^\top) y$. Thus we store the column vectors $\{(A_i + A_i^\top)(s-r)\}_{i \in [n]}$.

For each execution, the signer computes $\mathcal{F}(s-r)$ and $\mathcal{G}(\cdot, s-r)$, then performs N evaluations of $\mathcal{G}(\cdot, s-r)$. To challenge the online phase, the verifier does the same except with $N-1$ evaluations. To challenge the preprocessing phase, the verifier only computes $\mathcal{F}(r)$.

Finally, for some applications storing the full representation of $\mathcal{F}$ in memory is not feasible. We can instead evaluate $\mathcal{F}$ over all inputs in one pass of the pseudorandom generator.

Parameter selection. We first fix $m = n$ to maximize hardness of MQ. Unlike MQDSS and MUDFISH, Mesquite does not rely on the finite field's size for soundness. The main tradeoff arises from memory representation. For a given security level, n decreases as q increases — but $n \log q$ (the size of a vector) will slightly increase. Hence smaller q yields shorter signatures. On the other hand, larger q dramatically reduces the size of a multivariate quadratic map, which requires $\approx \frac{1}{2} n^3 \log q$ bits. Concretely, we found $\mathbb{F}_4$ to yield faster runtimes than $\mathbb{F}_2$, even though multiplication requires more bitwise operations. It may be worth considering even larger fields for specialized platforms (e.g. hardware instructions).

We adapt the $\mathbb{F}_4$ parameters proposed in the MQDSS specification [Sam+19, §8.2]. We propose "fast" and "compact" parameter sets at each security level, where $N = 8, 16$ respectively. Table 2 gives the values of M and τ, which are chosen to nearly optimize signature size. Finally, the SHAKE extensible output function doubles as a hash function and pseudorandom generator. We use SHAKE128 for NIST security level I and SHAKE256 otherwise.

				Classical attack	Quantum attack	
Level	q	$n = m$	|pk|	gates	gates	depth
I	4	88	38B	2^{156}	2^{93}	2^{83}
III	4	128	56B	2^{230}	2^{129}	2^{119}
V	4	160	72B	2^{290}	2^{158}	2^{147}

Table 1. Proposed parameters for the MQ problem targeting NIST security levels. We reference the best known classical and quantum attacks estimated in [Sam+19, §2.2].

4 Implementation

We implemented Mesquite in Rust to benchmark performance; this is available at `https://priv.pub/mesquite`. We used the standard library's portable SIMD module to represent elements of $\mathbb{F}_4^n$. For SHAKE we used the `tiny-keccak` crate[1], which is comparable in performance with the extended Keccak Code Package's `generic64` implementation[2]. Note that Mesquite and other KKW-style signatures stand to benefit most from faster hash functions (Haraka, KangarooTwelve [Köl+16; Ber+18]) and specialized implementations (AVX, NEON), compared to sacrificing-based MPC-in-the-head protocols [BN20].

We ran benchmarks on a Google Cloud `c2-standard-4` instance (Intel Cascade Lake, Xeon CPU @ 3.10GHz, 4 vCPUs, 16 GB memory). For comparison, we also tested the optimized implementation[3] of Picnic3 [Zav+20]. This was compiled with SIMD optimizations and the `generic64` Keccak implementation.

Table 2. Proposed parameter sets for Mesquite targeting NIST security levels. Mesquite runtimes were calculated with Rust's `test` crate. Picnic3 runtimes were taken over 1000 samples using provided scripts. The average signature sizes and standard deviations for each parameter set were taken over 1000 samples.

Level	Scheme		N	M	τ	Sign (ms)	Verify (ms)	Size (B)
I	Mesquite	Fast	8	176	51	4.97	3.89	9492 ± 196
		Compact	16	232	37	9.90	7.86	8844 ± 224
	Picnic3	-	16	252	36	8.25	6.60	12455 ± 229
III	Mesquite	Fast	8	276	74	12.45	9.12	20984 ± 357
		Compact	16	354	55	21.55	15.77	19656 ± 406
	Picnic3	-	16	419	52	17.96	14.24	27403 ± 448
V	Mesquite	Fast	8	372	98	31.88	23.70	36719 ± 551
		Compact	16	460	74	47.69	31.88	34573 ± 589
	Picnic3	-	16	601	68	29.78	21.94	48452 ± 701

Our benchmarks show that runtimes are competitive with Picnic3, and we believe there is room for improvement. For example, multivariate evaluations account for a significant fraction of signing time, ranging from 40% for L1 Compact to 75% for L5 Fast. Currently each one is done in sequence, as opposed to batches. There are also more aggressive SIMD optimizations that we have not explored.

[1] `https://github.com/debris/tiny-keccak`
[2] `https://github.com/XKCP/XKCP`
[3] `https://github.com/IAIK/Picnic`

References

[Ber+18] Guido Bertoni et al. "KangarooTwelve: Fast Hashing Based on Keccak-p". In: *ACNS 18*. Vol. 10892. July 2018, pp. 400–418.

[Beu20] Ward Beullens. "Sigma Protocols for MQ, PKP and SIS, and Fishy Signature Schemes". In: *EUROCRYPT 2020, Part III*. Vol. 12107. May 2020, pp. 183–211.

[Beu21] Ward Beullens. "Improved Cryptanalysis of UOV and Rainbow". In: *EUROCRYPT 2021, Part I*. Vol. 12696. Oct. 2021, pp. 348–373.

[Beu22] Ward Beullens. *Breaking Rainbow Takes a Weekend on a Laptop*. Cryptology ePrint Archive, Report 2022/214. `https://eprint.iacr.org/2022/214`. 2022.

[BN20] Carsten Baum and Ariel Nof. "Concretely-Efficient Zero-Knowledge Arguments for Arithmetic Circuits and Their Application to Lattice-Based Cryptography". In: *PKC 2020, Part I*. Vol. 12110. May 2020, pp. 495–526.

[DN19] Itai Dinur and Niv Nadler. "Multi-target Attacks on the Picnic Signature Scheme and Related Protocols". In: *EUROCRYPT 2019, Part III*. Vol. 11478. May 2019, pp. 699–727.

[Ish+07] Yuval Ishai et al. "Zero-knowledge from secure multiparty computation". In: *39th ACM STOC*. June 2007, pp. 21–30.

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. "Improved Non-Interactive Zero Knowledge with Applications to Post-Quantum Signatures". In: *ACM CCS 2018*. Oct. 2018, pp. 525–537.

[Köl+16] Stefan Kölbl et al. "Haraka v2 - Efficient Short-Input Hashing for Post-Quantum Applications". In: *IACR Trans. Symm. Cryptol.* 2016.2 (2016). `https://tosc.iacr.org/index.php/ToSC/article/view/563`, pp. 1–29. ISSN: 2519-173X.

[Sam+19] Simona Samardjiska et al. *MQDSS*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`. National Institute of Standards and Technology, 2019.

[Zav+20] Greg Zaverucha et al. *Picnic*. Tech. rep. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions`. National Institute of Standards and Technology, 2020.