Sharing the LUOV: Threshold Post-Quantum Signatures

Daniele Cozzo[1] [0000−0001−5289−3769] and Nigel P. Smart[1,2] [0000−0003−3567−3304]

[1] imec-COSIC, KU Leuven, Leuven, Belgium.
[2] University of Bristol, Bristol, UK.
daniele.cozzo@kuleuven.be, nigel.smart@kuleuven.be

Abstract. We examine all of the signature submissions to Round-2 of the NIST PQC "competition" in the context of whether one can transform them into threshold signature schemes in a relatively straight forward manner. We conclude that all schemes, except the ones in the MQ family, have significant issues when one wishes to convert them using relatively generic MPC techniques. The lattice based schemes are hampered by requiring a mix of operations which are suited to both linear secret shared schemes (LSSS)-based and garbled circuits (GC)-based MPC techniques (thus requiring costly transfers between the two paradigms). The Picnic and SPHINCS+ algorithms are hampered by the need to compute a large number of hash function queries on secret data. Of the nine submissions the two which would appear to be most suitable for using in a threshold like manner are Rainbow and LUOV, with LUOV requiring less rounds and less data storage.

1 Introduction

Ever since the late 1980s there has been interest in threshold cryptography [12]. Constructions for threshold signatures have received particular interest; these allow the distribution of signing power to several authorities using different access structures. For example, the 1990s and early 2000s saw work on threshold RSA signatures [10, 45] and DSA/EC-DSA signatures [16, 35].

The case of distributed EC-DSA signature gathered renewed interest [15, 30–32], due to applications in blockchain. Furthermore, general distributed solutions for decryption and signature operations are attracting more attention, such as the recent NIST workshop in this space[3].

However, solutions for distributed RSA and EC-DSA signatures do not provide resistance against quantum computers. Thus if one is to provide threshold signatures in a post-quantum world, then one needs to examine how to "thresholdize" post-quantum signatures. The techniques to create threshold versions of RSA and EC-DSA signatures make strong use of the number-theoretic structure of such schemes; however this structure is not available for many of the proposed post-quantum signature algorithms.

[3] https://www.nist.gov/news-events/events/2019/03/nist-threshold-cryptography-workshop-2019.

The NIST post-quantum cryptography “competition” aims to find replacement public key encryption and signature algorithms for the current number-theoretic solutions based on integer factoring and discrete logarithms. There are nine solutions which have been selected for the second round of this process, and these can be divided into four classes, according to the underlying hard problem on which they are based:

- Lattice-based: there are three submissions in this category; Dilithium [34], qTesla [4], and Falconal [41]
- Hash-based: there is one submission in this category, SPHINCS+ [24].
- MPC-in-the-Head (MPC-in-H)-based: here there is also one submission Picnic [48].
- Multivariate Quadratic-based: here we have four submissions GeMSS [5], LUOV [3], MQDSS [44] and Rainbow [13].

Generic MPC techniniques are now developed enough that one could simply apply them here in a black-box manner, but not all proposed post-quantum schemes would be equally suited to this approach. In this work we therefore examine the proposed post-quantum signature schemes submitted to Round 2 of the NIST project in the context of this problem.

Our Contribution: Looking only at the underlying assumptions one would suspect Picnic would be the algorithm which best lends itself to being converted into an MPC threshold version; after all it is based on MPC-in-the-Head. However, closer examination reveals that this is not the case. Indeed we examine all the post-quantum signature submissions from the point of view of whether one can easily turn them into threshold versions. It turns out that the ones which are most amenable to “thresholdizing” are those based on the MQ family of problems, in particular Rainbow and LUOV, see Table 1 for a summary.

In this version we discuss in detail Crystal-Dilithium, Picnic, SPHINCS+, Rainbow and LUOV. In the full version we cover the other Round-2 submissions.

The main issues with lattice-based techniques are the need to perform rejection sampling, which means intermediate values need to be kept secret until after the rejection sampling has been accomplished, and they need to be compared to given constants. This results in a number of operations suitable for garbled circuit operation to be performed. However, the rest of the algorithms require operations which are linear. Thus one has both a large number of garbled circuits (GC) -based operations to perform, as well as conversions to-and-from linear secret sharing scheme (LSSS) based representations to help mitigate the number of GC operations needed. This conversion turns out to be a major bottleneck.

Picnic on the other hand requires the signer to privately evaluate a set of PRFs and then reveal the associated keys for a given subset of the PRFs when obtaining the challenge value. This means that the PRFs need to be securely evaluated in a threshold manner. Since the PRFs used in Picnic are not specifically designed to be evaluated in this way one is left applying generic MPC

Name	Underlying Assumption	Issues in Obtaining a Threshold Variant
Dilithium	Lattice	A mix of linear operations (suitable for LSSS-based MPC) and non-linear operations (suitable for GC-based MPC) requires costly transferring between the two representations. We expect this to take around 12s to execute.
qTesla	Lattice	A mix of linear operations (suitable for LSSS-based MPC) and non-linear operations (suitable for GC-based MPC) requires costly transferring between the two representations. We expect to take at least 16s to execute.
Falcon	Lattice	A mix of linear operations (suitable for LSSS-based MPC) and non-linear operations (suitable for GC-based MPC) requires costly transferring between the two representations. We expect to take at least 6s to execute.
Picnic	MPC-in-H	Applying SHA-3 to obtain the necessary randomness in the views of the MPC parties.
SPHINCS+	Hash	Applying SHA-3 to obtain the data structures needed.
MQDSS	MQ	Applying SHA-3 to obtain the commitments.
GeMSS	MQ	Potential for threshold implementation, implementation is tricky due to need to extract polynomial roots via Berlekamp algorithm
Rainbow	MQ	Simple LSSS based MPC solution which requires 12 rounds of communication. We expect a signature can be generated in around three seconds
LUOV	MQ	Simple LSSS based MPC solution which requires 6 rounds of communication. We expect a signature can be generated in just over a second

Table 1. Summary of NIST Round 2 Post-Quantum Signature Schemes

techniques. These become very expensive due to the gate counts of the underlying PRFs specified by the proposal.

The hash-based signature scheme SPHINCS+ has a similar issue in that one needs to securely evaluate the underlying hash functions in a threshold manner; this again leads to huge gate counts.

One of the MQ schemes (MQDSS) also requires to evaluate hash functions on secret data, and so suffers from the same problems as the previous schemes. One MQ scheme (GeMSS) is a plausible candidate to be implemented via MPC, but any implementation would be highly non-trivial due to the need to evaluate Berlekamps' algorithm to extract roots of a univariate polynomial.

This leaves us with the two remaining MQ schemes (LUOV and Rainbow). These are based on the FDH signature construction and hence the main issue is implementing generic MPC for arithmetic circuits over the given finite fields. This would lead to threshold variants relatively easily. In this case Rainbow requires more rounds of interaction than LUOV, on the other hand Rainbow requires less secure multiplications. In addition, LUOV requires less data to store the shared secret key state.

In all of our analyses we try to give a best estimate as to the *minimum* amount of time a threshold implementation would take for each of the candidates. This is assuming the current best run-times for evaluating the SHA-3 internal function in an MPC system. These estimates are given for the schemes at the security level denoted Level 3 by NIST; when a scheme does not have parameters at Level 3 we pick the set at Level 4. Level 3 corresponds to the difficulty of breaking AES-192 on a quantum computer. This provides less than 192-bits of quantum security (due to Grover's algorithm), and hence seems a reasonable compromise since current (classical) security levels are usually picked to be equivalent to AES-128.

2 Preliminaries

In this section we define various notations and notions which will be needed in future sections. In particular we describe the underlying MPC systems which we will assume 'as given'. In particular our focus will be on MPC solutions which are actively secure (with abort) against static corruptions.

We assume that all involved parties are probabilistic polynomial time Turing machines. Given a positive integer n, we denote by $[n]$ the set $\{1, \ldots, n\}$. We let $x \leftarrow X$ denote the uniformly random assignment to the variable x from the set X, assuming a uniform distribution over X. We also write $x \leftarrow y$ as shorthand for $x \leftarrow \{y\}$. If $\mathcal{D}$ is a probability distribution over a set X, then we let $x \leftarrow \mathcal{D}$ denote sampling from X with respect to the distribution $\mathcal{D}$. If A is a (probabilistic) algorithm then we denote by $a \leftarrow A$ the assignment of the output of A where the probability distribution is over the random tape of A.

Signature schemes: Digital signature schemes which are defined by

Definition 2.1. *A* digital signature scheme *is given by a tuple of probabilistic algorithms* $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$*:*

- $\mathsf{KeyGen}\left(1^\lambda\right)$ *is a randomized algorithm that takes as input the security parameter and returns the public key* pk *and the private key* sk*.*
- $\mathsf{Sign}\left(\mathsf{sk}, \mu\right)$ *is a randomized signing algorithm that takes as inputs the private key and a message and returns a signature on the message.*
- $\mathsf{Verify}\left(\mathsf{pk}, (\sigma, \mu)\right)$ *is a deterministic verification algorithm that takes as inputs the public key and a signature* σ *on a message* μ *and outputs a bit which is equal to one if and only if the signature on* μ *is valid.*

Correctness and security (EU-CMA) are defined in the usual manner, and all signature scheme submitted to NIST in Round-2 meet this security definition.

A threshold signature scheme with respect to some access structure Γ is defined by the following definition

Definition 2.2. *A* threshold digital signature scheme *is given by a tuple of probabilistic algorithms* (KeyGen, Sign, Verify)*:*

- KeyGen $\left(1^\lambda\right)$ *is a randomized algorithm that takes as input the security parameter and returns the public key* pk *and a set of secret keys* sk_i*, one secret key for every party in the access structure.*
- Sign (sk, μ) *is a randomized signing algorithm that takes as inputs a* qualified set *of private keys and a message and returns a signature on the message.*
- Verify $(\mathsf{pk}, (\sigma, \mu))$ *is a deterministic verification algorithm that takes as inputs the public key and a signature* σ *on a message* μ *and outputs a bit which is equal to one if and only if the signature on* μ *is valid.*

Informally security for a threshold signature scheme is that an unqualified set of parties cannot produce a signature. An additional requirement is often that a valid output signature should be indistinguishable from the signature produced by the signing algorithm of equivalent the non-thresholdized scheme with the same public key.

Multi-Party Computation: As mentioned above we consider actively secure (with abort) MPC for static adversaries in this work. We assume a generic black box for MPC, abstracted in the functionality $\mathcal{F}_{\mathsf{MPC}}$ of Figure 1, which defines MPC over a given finite field $\mathbb{K}$ (or indeed sometimes a finite ring). When instantiating this abstract MPC functionality with state-of-the-art protocols one needs to consider aspects such as the access structure, the field used $\mathbb{K}$, and the computational/communication model. We summarize many of the state-of-the-art of MPC protocols in Table 2.

The ideal functionality $\mathcal{F}_{\mathsf{MPC}}$ for MPC over $\mathbb{F}_q$

Initialize: On input $(init, \mathbb{K})$ from all parties, the functionality stores $(domain, \mathbb{K})$.

Input: On input $(input, P_i, varid, x)$ from P_i and $(input, P_i, varid, ?)$ from all other parties, with $varid$ a fresh identifier, the functionality stores $(varid, x)$.

Random: On input of $(random, varid)$, if $varid$ is not present in memory then the functionality picks a random value in $\mathbb{K}$ and stores it in $varid$.

Add: On command $(add, varid_1, varid_2, varid_3)$ from all parties, if $varid_1$ and $varid_2$ are present in memory and $varid_3$ is not then the functionality retrieves $(varid_1, x)$ and $(varid_2, y)$ and stores $(varid_3, x + y)$. Otherwise does nothing.

Multiply: On input $(multiply, varid_1, varid_2, varid_3)$ from all parties, if $varid_1$ and $varid_2$ are present in memory and $varid_3$ is not then retrieve $(varid_1, x)$ and $(varid_2, y)$ and store $(varid_3, x \cdot y)$. Otherwise do nothing.

Output: On input $(output, varid, i)$ from all honest parties, if $varid$ is present in memory then retrieve $(varid, y)$ and output it to the environment. It then waits for an input from the environment. If this input is Deliver then y is output to all players if $i = 0$, or y is output to party i if $i \neq 0$. If the input is not equal to Deliver then $\perp$ is output to all parties.

Figure 1. The ideal functionality $\mathcal{F}_{\mathsf{MPC}}$ for MPC over $\mathbb{F}_q$

Protocol Name	Field $\mathbb{K}$	Access Structure	Pre-Proc Model	Rounds	Example Reference
SPDZ family	Large $\mathbb{K}$	Full Threshold	✓	$\approx \mathsf{depth}(C)$	[11]
Tiny-OT family	$\mathbb{F}_2$	Full Threshold	✓	$\approx \mathsf{depth}(C)$	[29, 39]
SPDZ-2k	$(\mathbb{Z}/2^k\mathbb{Z})$	Full Threshold	✓	$\approx \mathsf{depth}(C)$	[8]
n-party GC family	$\mathbb{F}_2$	Full Threshold	✓	constant	[22, 47]
General Q2	Any	Q2	✓	$\approx \mathsf{depth}(C)$	[46]
General Q2	Any	Q2	-	$\approx \mathsf{depth}(C)$	[7]
Special GC	$\mathbb{F}_2$	$(t, n) = (1, 3)$	-	constant	[38]

Table 2. Summary of main practical MPC protocols

In terms of access structures the main ones in use are full threshold (for example in the SPDZ protocol family [11]) and Q2-access structures (which includes standard threshold protocols for which $t < n/2$). A Q2-access structure is one in which the union of no two unqualified sets cover the entire set of players. In terms of the field $\mathbb{K}$ for evaluating binary circuits one usually utilizes MPC over $\mathbb{K} = \mathbb{F}_2$. However, for some applications (in particular the MQ signature schemes considered later) it is better to take $\mathbb{K}$ to be a specific finite field taylored to the application. Some protocols are tailored to very specific access structures (for example using threshold $(t, n) = (1, 3)$).

The functionality has a command to produce random values in $\mathbb{K}$. This can always be achieved using interaction (via the input command), however, for LSSS based protocols in the Q2 setting (with small numbers of parties) such a command can be executed for free using a PRSS.

To make it simpler to describe MPC protocols, in what follows we use the notation $\langle x \rangle$ for $x \in \mathbb{K}$ to denote a value x stored in the MPC engine (the reader can think of this as $\langle x \rangle$ being the secret sharing of x). The MPC functionality Figure 1 enables one to compute $\langle x \rangle + \langle y \rangle$, $\lambda \cdot \langle x \rangle$ and $\langle x \rangle \cdot \langle y \rangle$. We extend this notation to vectors and matrices of elements in $\mathbb{K}$ in the obvious manner.

In terms of computational model we find the set of practical MPC protocols divided into distinct classes. In some protocols there is a function-independent offline phase, and then a fast offline phase. Other protocols have no offline phase but then pay a small cost in the online phase. In some instances one can choose which class one wants to be in. For example, for Q2 access structures over a general finite field one can use the protocol of [46] if one wishes to utilize an offline phase, but the protocol in [7] if ones wants to avoid the offline phase (but have a slightly slower "online" phase). Although the performance of [7] degrades considerably for small finite fields, whereas that of [46] does not degrade at all (however [46]'s offline phase performance degrades if the finite field is small). Note [7] is expressed in terms of $t < n/2$ threshold adversaries but it can be trivially extended to any Q2 access structure.

The communication model also plays a part with protocols based on Garbled Circuits using a constant number of rounds, whereas protocols based on linear-secret sharing (LSSS) requiring rounds (roughly) proportional to the circuit depth. In all cases the total amount of communication, and computation, is roughly proportional to the number of multiplcition gates with the arithmetic circuit over $\mathbb{K}$ which represents the function to be computed[4]. The LSSS based protocols cost (essentially) one round of communication per each multiplicative depth, and communication cost linear in the number of multplication gates.

It is possible to mix GC and LSSS based MPC in one application, and pass between the two representations. For special access structures one can define special protocols for this purpose, see [27] for example. For general access structures one can apply the technique of doubly-authenticated bits (so called daBits) introduced in [43]. This latter method however comes with a cost. Assuming we are converting ℓ bit numbers, then not only does one need to generate (at least) ℓ daBits, but when transforming from the LSSS to the GC world one requires to evaluate a garbled circuit with roughly $3 \cdot \ell$ AND gates. The more expensive part is actually computing the daBits themselves. The paper [27] claims a cost of 0.163ms per daBit, for fields of size 2^{128}. Whilst the fields used in the lattice based post-quantum signature algorithms are much smaller (of the order of 2^{20}) we use the same estimate[5].

Of course one could execute bitwise operations in an LSSS-based MPC for an odd modulus q using the methods described in [6, 9]. But these are generally slower than performing the conversion to a garbled circuit representation and then performing the garbled circuit based operation. Especially when different operations are needed to be performed on the same bit of data.

MPC of Standard Functionalities: A number of the signature schemes submitted to NIST make use of keyed (and unkeyed) symmetric functions which need to be applied in any threshold implementation to secret data. Thus any threshold implementation will need to also enable a threshold variant of these symmetric primitives. Here we recap, from the literature, the best timings and costs one can achieve for such primitives. We will use these estimates to examine potential performance in our discussions which follow.

In [27] the authors give details, also within the context of thresholdizing a NIST PQC submission (this time an encryption algorithm), of an MPC implementation of the SHA-3 round function (within the context of executing the KMAC algorithm). The round function f for SHA-3 requires a total of 38,400 AND gates, and using a variant of the three party honest majority method from [38], the authors were able to achieve a latency of 16ms per execution of f, for a LAN style setting. This equates to around 0.4μs per AND gate. Any actual application of SHA-3 requires multiple executions of the round function f; depending on how much data is being absorbed and how much is being squeezed.

In [47] give timings for a full-threshold garbled circuit based evaluation of various functions. Concentrating on the case of AES and SHA-256, and three

[4] This is not strictly true as one often does not represent the function as a pure arithmetic circuit. But as a first order approximation this holds

[5] Arithmetic modulo a prime of size 2^{20} is faster, but on the other hand one then has to perform more work to obtain the same level of active security.

party protocols, the authors obtain a latency of 95ms (13ms online) for the 6800 AND gate AES circuit, and 618ms (111ms online) for the 90825 AND gate SHA-256 circuit, again for a LAN setting. These times correspond to between 1μ and 2μs per AND gate, thus the three party full threshold setting is slightly slower than the honest majority setting (as is to be expected).

For general arithmetic circuits the estimates in [7] in the honest majority three party setting for a 61-bit prime field give a time of 826ms to evaluate a depth 20 circuit with one million multiplication gates, in a LAN setting. Thus we see that when using arithmetic circuits over such a finite field one can deal we obtain a similar throughput, in terms of multiplications per second, as one has when looking at binary circuits using garbled circuit techniques. However, the fields are of course much larger and we are performing more "bit operations" per second in some sense.

However, the protocol in [7] for 61-bit prime fields assumes a statistical security of 61-bits, i.e. the adversary can pass one of the checks on secure multiplications probability $1/2^{61} = 1/\#\mathbb{K}$. For smaller finite fields the performance degrades as one needs to perform more checks. A back-of-the-envelope calculation reveals one would expect a throughput of roughly 250,000 multiplications per second in the case of $\mathbb{F}_{2^8}$.

Whilst these times are comparing apples and oranges, they do give an order of magnitude estimate of the time needed to compute these functions. Generally speaking, one is looking for operations which involve a few number of multiplications. Almost all NIST signature submissions make use of SHAKE-256, as a randomness expander, or hash-function. The SHAKE-256 algorithm is based on SHA-3. Recall that an application of SHA-3/SHAKE-256 on an input of ℓ_i bits, to produce an output of ℓ_o bits, will require (ignoring issues when extra padding results in more blocks being processed) a total of

$$\mathsf{rounds}(\ell_i, \ell_o) := \left\lceil \frac{\ell_i}{1088} \right\rceil + \left\lceil \frac{\ell_o}{1088} \right\rceil - 1$$

iterations of the main Keccak round function, since the rate of SHA3-256 is $r = 1088$ bits. In what follows we use the current best MPC evaluation time for this function (of 16ms from [27]) to obtain an estimate of how a specific application of SHAKE-256/SHA-3 will take.

3 Lattice Based Schemes

Lattice based signature schemes have a long history, going back to the early days of lattice based cryptography. Early examples such as NTRUSign [23] were quickly shown to be insecure due to each signature leaking information about the private key [18]. In recent years following the work of Lyubashevsky [33] the standard defence against such problems has been to adopt a methodology of Fiat–Shamir-with-aborts. All of the three lattice based submissions to NIST Round-2 follow this paradigm. However, we shall see that this means that they are all not particularly tuned to turning into threshold variants; for roughly the same reasons; although Falcon is slightly better in this regard. In all our lattice descriptions we will make use of a ring R_q, which one can take to be the cyclotomic ring $\mathbb{Z}[X]/(X^N + 1)$ reduced modulo q.

3.1 Crystals-Dilithium

The Dilithium Signature Algorithm

1. $z \leftarrow \perp$
2. While ($z = \perp$) do
   (a) Sample a "short" $y \in R_q^l$ with $\|y\|_\infty \leq \gamma_1$.
   (b) $v \leftarrow A \cdot y$.
   (c) Let w be the topbits of v.
   (d) $c \leftarrow H(\mu \| w) \in R_q$.
   (e) $z \leftarrow y + c \cdot s_1$.
   (f) If z or the lower bits of $v - c \cdot s_2$ are too big then set $z \leftarrow \perp$.
3. Return $\sigma \leftarrow (z, c)$.

Figure 2. The Dilithium Signature Algorithm

The Dilithium [34] signature scheme is based on the Module-LWE problem. The secret key is two "short" vectors (s_1, s_2) with $s_1 \in R_q^l$ and $s_2 \in R_q^k$, and the public key is a matrix $A \in R_q^{k \times l}$ and a vector $t \in R_q^k$ such that $t = A \cdot s_1 + s_2$. The high-level view of the signature algorithm for signing a message μ is given in Figure 2, for precise details see the main Dilithium specification. We do not discuss the optimization in the Dilithium specification of the MakeHint function, to incorporate this will involve a few more AND gates in our discussion below. To aid exposition we concentrate on the basic signature scheme above. At the Level-3 security level the main parameters are set to be $N = \deg R_q = 256$, $q = 2^{23} - 2^{13} + 1$ and $(k, l) = (5, 4)$. There are a number of other parameters which are derived from these, in particular $\gamma_1 = (q-1)/16$ and $\gamma_2 = (q-1)/32$.

From our point of view we see that Dilithium is a signature scheme in the Fiat-Shamir-with-aborts family. If we did not have the while-loop in the signature algorithm, then the values of z and $v - c \cdot s_2$ would leak information to the adversary. Thus it is clear that *any* distributed version of Dilithium signatures should maintain the secrecy of these intermediate values. Only values values which pass the size check and are output as a valid signature, can be revealed.

The parameters of the algorithm are selected so that the probability of needing two iterations of the while loop is less than one percent. Thus we can concentrate on the case of only executing one iteration of the loop. We assume that the secret key has been shared in an LSSS scheme over $\mathbb{F}_q$ which supports one of the MPC algorithms for LSSS schemes discussed in the introduction. The main issue with Dilithium, and indeed all the other lattice based schemes, is that some operations are best suited to linear secret sharing based MPC over the underlying finite field $\mathbb{F}_q$ (e.g. lines 2b and 2e), whereas some are more suited to evaluation using a binary circuit (e.g. lines 2c and 2d). The main cost therefore comes in switching between the two types of MPC systems. For full details see the full version.

3.2 qTesla

qTesla is a signature scheme based on the ring-LWE problem, and like the previous one it too uses rejection sampling to avoid information leakage from signatures. The secret key is a pair $s, e \in R_q$, where e is small and R_q has degree N. The public key is a value $a \in R_q$ in R_q along with the value $t = a \cdot s + e$. The high level view of the signature algorithm is given in Figure 3. For the Level-3 security level we have the parameters $N = \deg R_q = 1024$, $B = 2^{21} - 1$, $q = 8404993$, and $d = 22$.

The qTesla Signature Algorithm

1. $z \leftarrow \perp$.
2. While ($z = \perp$) do
   (a) Sample a "short" $y \in R_q$ with $\|y\|_\infty \leq B$.
   (b) $b \leftarrow [a \cdot y]_M \in R_q$.
   (c) $c \leftarrow H(b \| G(\mu)) \in R_q$.
   (d) $z \leftarrow y + s \cdot c$.
   (e) If z is not short or $a \cdot y - e \cdot c$ is not "well-rounded" then set $z \leftarrow \perp$.
3. Return $\sigma \leftarrow (z, c)$.

Figure 3. The qTesla Signature Algorithm

The operation $[x]_M$ applied to $x \in R_q$ provides a rounding operation akin to taking the top $(\log_2 q - d)$ bits of x in each coefficient. We define $[x]_M = (x \pmod{q} - x \pmod{2^d})/2^d$ where the two modular operations perform a centered reduction (i.e. in the range $(-q/2, \ldots, q/2]$. The values of $[x]_M$ are stored in one byte per coefficient.

The function G is a hash function which maps messages to 512 bit values, and H is a hash function which maps elements in $R_q \times \{0,1\}^{512}$ to a 512-bit string c, which is then treated as a trinary polynomial. The functions H and G being variants of SHAKE-256 (or SHAKE-128 for the low security variants). Again much like the Dilithium, due to the rejection sampling the computation of y and the evaluation of H must be done in shared format.

The analysis of the cost of qTesla in a threshold system follows much the same as the analysis done above for Crystals-Dilithium, thus we leave the full discussion to the full version.

3.3 Falcon

Falcon [41] is another lattice based scheme, and the only one to have NTRU-like public keys. It is based on the GPV framework [17]. The private key is a set of four "short" polynomials $f, g, F, G \in R_q$ such that $f \cdot G = g \cdot F$ in the ring R_q.

The public key is the polynomial $h \leftarrow g/f$, which will have "large" coefficients in general. Associated to the private key is the private lattice basis in the FFT domain

$$\overline{B} = \begin{pmatrix} \mathsf{FFT}(g) & -\mathsf{FFT}(f) \\ \mathsf{FFT}(G) & -\mathsf{FFT}(F) \end{pmatrix}.$$

There is also a data structure T, called the *Falcon Tree* associated to the private key, which can be thought of as a set of elements in the ring R_q. At the Level-3 security level one has $N = \deg R_q = 768$ and $q = 18435$. A high level view of the signature algorithm is given in Figure 4.

The Falcon Signature Algorithm

1. $r \leftarrow \{0,1\}^{320}$.
2. $c \leftarrow H(r \| \mu)$.
3. $t \leftarrow (\mathsf{FFT}(c), \mathsf{FFT}(0)) \cdot \overline{B}^{-1}$.
4. $z \leftarrow \perp$.
5. While $(z = \perp)$ do
   (a) $z \leftarrow \mathsf{ffSampling}_n(t, T)$.
   (b) $s \leftarrow (t - z) \cdot \overline{B}$.
   (c) If s is not short then set $z = \perp$.
6. $(s_1, s_2) \leftarrow \mathsf{FFT}^{-1}(s)$.
7. $s \leftarrow \mathsf{Compress}(s_2)$.
8. Return $\sigma = (r, s)$.

Figure 4. The Falcon Signature Algorithm

Again, we assume that the secret key has been shared in an LSSS scheme over $\mathbb{F}_q$, and we go through each line in turn. The specification document says that how the discrete Gaussian is evaluated is "arbitrary" and "outside the scope of this specification" [41], bar needing to be close in terms of the Rényi divergence. However, a Gaussian sampler is defined within the specification, [41][Section 4.4], for use in the reference implementation. It turns out that this sampler is the main impediment to producing an efficient threshold version of Falcon. We leave the details to the full version.

4 MPC-in-the-Head Based Scheme

The MPC-in-the-Head paradigm for producing zero-knowledge proofs was developed in [25]. The prover, to prove knowledge of a preimage x of some function $\Phi(x) = y$ (where Φ and y are public), simulates an MPC protocol to compute the functionality Φ, with the input x shared among the simulated parties. The prover executes the protocol (in it's head), then commits to the state and the transcripts of all players. Then it sends the verifier these commitments and randomly opens a (non-qualified) subset of them (the precise subset is chosen by the verifier). The verifier checks that the simulated protocol was correctly executed using the opened values. If everything is consistent, it then accepts the statement that the prover knows x, otherwise it rejects. Typically, the proof has to repeated several times in ordar to achieve high security. Clearly to obtain a signature scheme we apply the Fiat–Shamir transform so that the verifier's choices are obtained by hashing the commitments with the message.

4.1 Picnic

Picnic is a digital signature scheme whose security entirely relies on the security only of symmetric key primitives, in particular the security of SHA-3 and a low-complexity block cipher called Low-MC [1]. The core construction is a zero-knowledge proof of knowledge of a preimage for a one-way function $y = f_k(x)$, where f is the Low-MC block cipher, the values x and y are public and the key k is the value being proved. Using the Fiat–Shamir and MPC-in-the-Head paradigms we obtain a signature scheme with public key (x, y) and private key k.

In this paper we concentrate on Picnic-1, but a similar discussion also applies to the Picnic-2 construction. The specific proof system that implements the MPC-in-the-Head for Picnic-1 is ZKBoo++ [1], which is itself an extension of the original ZKBoo framework from [19]. The simulated MPC protocol is between three parties, and is executed at a high level as in Figure 5

The Picnic Signature Algorithm (High Level)

1. Generate $3 \cdot T$ secret seeds $\mathsf{seed}_{i,j}$ for $i = 0, \ldots, T-1$ and $j = 0, 1, 2$.
2. Using a KDF expand the $\mathsf{seed}_{i,j}$ values to a sequence of random tapes $\mathsf{rand}_{i,j}$.
3. For each i use the three random tapes $\mathsf{rand}_{i,j}$ as the random input to a player P_j for an MPC protocol to evaluate the function $f_k(x)$.
4. Commit to the resulting views, and hash them with a message to obtain a set of challenges $e_0, \ldots, e_t \in \{0, 1, 2\}$.
5. Reveal all seeds $\mathsf{seed}_{i,j}$ bar seed_{i,e_i}.

Figure 5. The Picnic Signature Algorithm (High Level)

In our analysis we will ignore any hashing needed to produce commitments and the challenge, and we will simply examine the operation of the key derivation in step 2 of Figure 5. It is clear that in the MPC-in-the-Head paradigm the seeds need to be kept secret until the final reveal phase, thus the derivation of the random tape from the seed needs to be done in a secure manner in a any threshold implementation.

In Picnic the precise method used to derive the random tape is to use

$$\mathsf{rand}_{i,j} = \mathsf{KDF}\left(H_2\left(\mathsf{seed}_{i,j}\right) \| \mathsf{salt} \| i \| j \| \mathsf{length}\right)$$

where

- The seeds are S bits long.
- The salt is 256 bits long.

- The integers i, j and length are encoded as 16-bit values.
- The output length (length), of the KDF, is $n + 3 \cdot r \cdot s$ when $j = 0, 1$ and $3 \cdot r \cdot s$ when $j = 2$.

We again concentrate on the NIST security Level-3, which is instantiated with the parameters $S = n = 192$, $T = 329$, $s = 10$ and $r = 30$. The hash function H_2 is SHAKE-256 based with an output length of 384 bits. Thus the execution of H_2 requires only two executions of the SHA-3 round function. Each KDF operation is also cheap, requiring either two or three rounds. The problem is we need to execute these operations so many times. The total number of executions of the round function of SHA-3 is given by

$$\begin{aligned} T \cdot \Big(& 2 + 2 \cdot \mathsf{rounds}(384 + 256 + 32 + 32, n + 3 \cdot r \cdot s) \\ & \qquad + \mathsf{rounds}(384 + 256 + 32 + 32, 3 \cdot r \cdot s)\Big) \\ &= 329 \cdot \Big(2 + 2 \cdot \mathsf{rounds}(704, 1092) + \mathsf{rounds}(704, 900)\Big) \\ &= 329 \cdot (2 + 2 \cdot (3 - 1) + (2 - 1)) = 2303. \end{aligned}$$

Thus given our estimate of a minimum of 16ms for a SHA-3 round execution in MPC we see that even this part of the Picnic algorithm is expected to take $16 \cdot 3290$ ms, i.e. 37 seconds!

5 Hash Based Scheme

Hash based signatures have a long history going back to the initial one-time signature scheme of Lamport [28]. A more efficient variant of the one-time signature attributed to Winternitz is given in [37], where a method is also given to turn the one-time signatures into many-time signatures via so-called Merkle-trees. The problem with these purely Merkle tree based constructions is that they are strictly a statefull signature scheme. The signer needs to maintain a changing state between each signature issued, and the number of signatures able to be issued is bounded as a function of the height of the Merkle tree.

To overcome these issues with state the SPHINCS signature scheme was introduced in 2015 [2], which itself builds upon ideas of Goldreich elaborated in [21], and going back to [20]. In the SPHINCS construction messages are still signed by Winternitz one-time signatures, but the public keys of such signatures are then authenticated via another (similar) structure called a Forest of Random Subsets (which is itself based on earlier work in [42]).

5.1 SPHINCS+

The only hash based signature scheme to make it into the second round of the NIST competition is SPHINCS+ [24]. We refer the reader to the design document [24] for a full description. For our purposes we recall that messages are signed using Winternitz one-time signatures which are then authenticated using a FORS tree. The parameters which are of interest to us are: n the security parameter in bytes, w a parameter related to the underlying Winternitz signature, h the height of the hypertree, d the number of layers in the hypertree, k the number of trees in a FORS, t the number of leaves in a FORS tree. From these two length functions are defined[6]

$$\mathsf{len}_1 = \left\lceil \frac{8 \cdot n}{\log_2 w} \right\rceil, \qquad \mathsf{len}_2 = \left\lfloor \frac{\log(\mathsf{len}_1 \cdot (w-1))}{\log w} \right\rfloor + 1, \quad \text{and} \quad \mathsf{len} = \mathsf{len}_1 + \mathsf{len}_2.$$

The scheme uses (essentially) four hash functions labelled $\mathbf{F}$, $\mathbf{H}$, $\mathbf{PRF}$ and T_{len}. The function $\mathbf{F}$ is used as the main function in the Winternitz signature scheme, as well as the FORs signature. The underlying expansion the secret key into secret keys of the trees is done via the function $\mathbf{PRF}$. The function $\mathbf{H}$ is used to construct a root of the associated binary trees, where as T_{len} is used to compress the len Winternitz public key values into a single n-bit value for use as a leaf in the Merkle tree. The evaluation of the $\mathbf{F}$ and $\mathbf{PRF}$ calls within a single signature needs to be performed on secret data, even though eventually some of the input/outputs become part of the public signature. The calls to $\mathbf{H}$ and T_{len} appear to be able to be performed on public data, and will not concern us here.

In what follows we concentrate on the SHAKE-256 based instantiation of SPHINCS+ (to be comparable with other signature schemes in this paper). In the SHAKE instantiation the execution of the function $\mathbf{F}$ requires two calls to the underlying SHA-3 permutation, where as $\mathbf{H}$ requires three calls to the underlying SHA-3 permutation, and $\mathbf{PRF}$ requires one call to the SHA-3 permutation.

To sign a message requires $k \cdot t + d \cdot w \cdot \mathsf{len} \cdot 2^{h/d}$ calls to $\mathbf{F}$ and $k \cdot t + d \cdot \mathsf{len} \cdot 2^{h/d} + 1$ calls to $\mathbf{PRF}$. When instantiated with the parameters at the NIST Level-3 security level (for fast signing) we have $(n, w, h, d, k, t) = (24, 16, 66, 22, 33, 256)$. Leading to $\mathsf{len}_1 = 48$, $\mathsf{len}_2 = 3$ and $\mathsf{len} = 51$. This leads to a grand total of 152064 calls to $\mathbf{F}$ and 17425 calls to $\mathbf{PRF}$. This leads to a total of 321553 calls to the SHA-3 internal permutation which need to be performed securely. With current best garbled circuit implementations this on its own would require 85 minutes to execute. Of course a complete threshold implementation would take longer as we have not looked at other aspects of the signature algorithm.

6 MQ Based Schemes

The history of MQ cryptography, is almost as old as that of hash-based signatures. The first MQ based scheme was presented in 1988 [36]. In terms of signature schemes based on the MQ problem, the original works were due to Patarin and were given the name "Oil and Vinegar" [26, 40]. The basic idea is to define a set of multivariate quadratic equations (hence the name MQ) $P : \mathbb{F}_q^m \longrightarrow \mathbb{F}_q^n$

[6] Note the definition of len_1 in the specification is wrong and need correcting which we do below

and the hard problem is to invert this map, where q is a power of two[7]. The intuition being that inverting this map is (for a general quadratic map P) is an instance of the circuit satisfiability problem, which is known to be NP-Complete.

In three of the NIST candidate signature schemes the function P is generated so that there is an efficient trapdoor algorithm which allows the key holder to invert the map P using the secret key. In such situations the secret key is usually chosen to be two affine transforms $S : \mathbb{F}_q^n \longrightarrow \mathbb{F}_q^n$ and $T : \mathbb{F}_q^m \longrightarrow \mathbb{F}_q^m$, plus an easy to invert map $P' : \mathbb{F}_q^m \longrightarrow \mathbb{F}_q^n$ consisting of quadratic functions (note any function can be expressed in terms of quadratic functions by simple term rewriting). Then the public map is defined by $P = S \circ P' \circ T$. Of course the precise definition of this construction implies that one is not using a generic circuit satisfiability problem. However, for specific choices of P', q, n and m the construction is believed to provide a trapdoor one-way function.

Given we have a trapdoor one way function the standard Full Domain Hash construction gives us a signature scheme. Namely to sign a message μ, the signer hashes μ to an element $y \in \mathbb{F}_q^m$ and then exhibits a preimage of y under P as the signature s. To verify the signature the verifier simply checks that $P(s) = y$. Note, that many preimages can exist for y under P, thus every message could have multiple valid signatures. From this basic outline one can define a number of signature scheme depending on the definition of the "central map" P'. All of the Round-2 MQ based signaure schemes, with the exception of MQDSS, follow this general construction method. In this version we discuss Rainbow and LUOV, leaving MQDSS and GeMSS to the full version.

Inverting Linear Systems in MPC Before proceeding we present a trick which enables us to efficiently solve linear systems in an LSSS based MPC system. We will use this in our analysis of two of the submissions, so we present it here first. Suppose we have a shared $n \times n$ matrix $\langle A \rangle$ over $\mathbb{F}_q$ and an n-dimensional shared vector $\langle \mathbf{b} \rangle$. We would like to determine $\langle \mathbf{x} \rangle$ such that $A \cdot \mathbf{x} = \mathbf{b}$. We do this using the algorithm in Figure 6. This algorithm either returns the secret shared solution or the $\perp$ symbol. This latter either happens because the input matrix has determinant zero, or the random matrix used in the algorithm has determinant zero (which occures with probability $1/q$). The algorithm requires a total of three rounds of communication and $n^3 + n^2$ secure multiplications.

6.1 Rainbow

The Rainbow signature scheme can be seen as a multilayer version of the original UOV. In its original version, the number of layers is determined by a parameter u. For $u = 1$ this is just the basic UOV scheme, whereas the candidate submission chooses $u = 2$. As described earlier we pick for the secret key two affine transforms $\mathcal{S} : \mathbb{F}_q^m \to \mathbb{F}_q^m$ and $\mathcal{T} : \mathbb{F}_q^n \to \mathbb{F}_q^n$. Along with a function $\mathcal{F}$, called

[7] To enable comparison with the NIST submissions we use the same notation in the sections which follow as used in the submissions. We hope this does not confuse the reader.

Method for solving $\langle A \rangle \cdot \langle \mathbf{x} \rangle = \langle \mathbf{b} \rangle$

Input: $\langle A \rangle, \langle \mathbf{b} \rangle$ with $A \in F_q^{n \times n}$ and $\mathbf{b} \in \mathbb{F}_q^n$.
Output: $\perp$ or $\langle \mathbf{x} \rangle$ such that $A \cdot \mathbf{x} = \mathbf{b}$.

1. Generate a random $n \times n$ shared matrix $\langle R \rangle$. Generation of random elements in LSSS based MPC systems can *usually* be done for free in the online phase with no communication costs.
2. Compute $\langle T \rangle \leftarrow \langle A \rangle \cdot \langle R \rangle$. This requires one round of communication and the secure multiplication of n^3 elements.
3. Open the matrix $\langle T \rangle$. This requires one round of communication.
4. In the clear, compute T^{-1}. If $\det(T) = 0$ then we return $\perp$.
5. Compute $\langle \mathbf{t} \rangle \leftarrow T^{-1} \cdot \langle \mathbf{b} \rangle$, which is a linear operation and hence free.
6. Finally compute $\langle \mathbf{x} \rangle \leftarrow \langle R \rangle \cdot \langle \mathbf{t} \rangle = \langle R \cdot T^{-1} \cdot \mathbf{b} \rangle = \langle R \cdot R^{-1} \cdot A^{-1} \cdot \mathbf{b} \rangle = \langle A^{-1} \cdot \mathbf{b} \rangle$. This step requires one round of communication, and n^2 secure multiplications.

Figure 6. Method for solving $\langle A \rangle \cdot \langle \mathbf{x} \rangle = \langle \mathbf{b} \rangle$

the central map, which can be defined by quadratic functions. The public key is then the map $\mathcal{P} = \mathcal{S} \circ \mathcal{F} \circ \mathcal{T} : \mathbb{F}_q^n \to \mathbb{F}_q^m$.

In the Rainbow specification the affine maps $\mathcal{S}$ and $\mathcal{T}$ are chosen to be given by matrix multiplication by upper triangular matrices S and T. This means that the inverse matrices S^{-1} and T^{-1} are also upper triangular. In particular the inverses are selected to have the following block form

$$S^{-1} = \begin{pmatrix} \mathbf{1}_{o_1} & S_{o_1 \times o_2} \\ \mathbf{0}_{o_2 \times o_1} & \mathbf{1}_{o_2} \end{pmatrix} \qquad \text{and} \qquad T^{-1} = \begin{pmatrix} \mathbf{1}_{v_1} & T_{v_1 \times o_1} & T'_{v_1 \times o_2} \\ \mathbf{0}_{o_2 \times v_1} & \mathbf{1}_{o_1} & T''_{o_1 \times o_2} \\ \mathbf{0}_{o_2 \times v_1} & \mathbf{0}_{o_2 \times o_1} & \mathbf{1}_{o_2} \end{pmatrix}$$

where $S_{a \times b}$ etc denotes a matrix of dimension $a \times b$, $\mathbf{0}_{a \times b}$ denotes the zero matrix of dimension $a \times b$ and $\mathbf{1}_a$ denotes the identity matrix of dimension a.

To define the central map we define three constants (v_1, o_1, o_2), which at the Level-3 security level are chosen to be $(68, 36, 36)$. From these we define further parameters given by $v_2 = v_1 + o_1, n = v_3 = v_2 + o_2$ and $m = o_1 + o_2$. Note this means that $n = v_1 + m$. We then define the sets $V_i = \{1, \ldots, v_i\}$ and $O_i = \{v_i + 1, \ldots, v_{i+1}\}$, for $i = 1, 2$, which will be referred to as the vinegar (resp. oil) variables of the ith layer.

The Rainbow central map $\mathcal{F} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ can then be defined by the set of m quadratic polynomials $f^{(v_1+1)}, \ldots, f^{(n)}$ having the form

$$f^{(k)} = \begin{cases} \sum_{i,j \in V_1, i \leq j} \alpha_{ij}^{(k)} x_i \cdot x_j + \sum_{i \in V_1} \sum_{j \in O_1} \beta_{ij}^{(k)} x_i \cdot x_j & k = v_1 + 1, \ldots, v_2, \\ \sum_{i,j \in V_2, i \leq j} \alpha_{ij}^{(k)} x_i \cdot x_j + \sum_{i \in V_2} \sum_{j \in O_2} \beta_{ij}^{(k)} x_i \cdot x_j & k = v_2 + 1, \ldots, n, \end{cases}$$

where the coefficients $\alpha_{i,j}^{(k)}, \beta_{i,j}^{(k)}$ are randomly chosen from $\mathbb{F}_q$.

Signature generation (for the EUF-CMA scheme) is done by the steps

1. Compute the hash value $\mathbf{h} \leftarrow H(H(\mu) \| \mathsf{salt}) \in \mathbb{F}_q^m$, where μ is the message, salt is a random l-bit string and $H : \{0,1\} \to \mathbb{F}_q^m$ is an hash function.

2. Compute $\mathbf{x} \leftarrow S^{-1} \cdot \mathbf{h} \in \mathbb{F}_q^m$,
3. Compute a preimage $\mathbf{y} \in \mathbb{F}_q^n$ of $\mathbf{x}$ under the central map $\mathcal{F}$.
4. Compute $\mathbf{z} \leftarrow T^{-1} \cdot \mathbf{y} \in \mathbb{F}_q^n$.
5. Output $(\mathbf{z}, \mathsf{salt})$.

Inversion of the Rainbow central map

Input: The central map $\mathcal{F} = \left(f^{(v_1+1)}, \ldots, f^{(n)}\right)$, a vector $\mathbf{x} \in \mathbb{F}_q^m$
Output: A vector $\mathbf{y} \in \mathbb{F}_q^n$ satisfying $\mathcal{F}(\mathbf{y}) = \mathbf{x}$.

1. Choose random values for the variables $\hat{y}_1, \ldots, \hat{y}_{v_1}$ and substitute these values into the polynomials $f^{(v_1+1)}, \ldots, f^{(v_2)}$.
2. Perform Gaussian elimination on the system
$$\begin{aligned} f^{(v_1+1)}(\hat{y}_1, \ldots, \hat{y}_{v_1}, y_{v_1+1}, \ldots, y_n) &= x_{v_1+1} \\ &\vdots \\ f^{(v_2)}(\hat{y}_1, \ldots, \hat{y}_{v_1}, y_{v_1+1}, \ldots, y_n) &= x_{v_2} \end{aligned}$$
to obtain the values of the variables $y_{v_1+1}, \ldots, y_{v_2}$, say $\hat{y}_{v_1+1}, \ldots, \hat{y}_{v_2}$.
3. Substitute the values $\hat{y}_{v_1}, \ldots, \hat{y}_{v_2}$ into the polynomials $f^{(v_2+1)}, \ldots, f^{(n)}$.
4. Perform Gaussian elimination on the system
$$\begin{aligned} f^{(v_2)}(\hat{y}_1, \ldots, \hat{y}_{v_2}, y_{v_2+1}, \ldots, y_n) &= x_{v_2+1} \\ &\vdots \\ f^{(n)}(\hat{y}_1, \ldots, \hat{y}_{v_2}, y_{v_2+1}, \ldots, y_n) &= x_n \end{aligned}$$
to obtain the values of the variables $y_{v_2+1}, \ldots, y_n$, say $\hat{y}_{v_2+1}, \ldots, \hat{y}_n$.
5. Return $\mathbf{y} = (\hat{y}_1, \ldots, \hat{y}_n)$.

Figure 7. Inversion of the Rainbow central map

The main work of the signing algorithm occurs in step 3 which is done using the method described in Figure 7. As all the components $f^{(k)}$ of the central map are homogeneous polynomials of degree two, we can represent them using matrices. Specifically, substituting the first layer of the vinegar variables $\hat{y}_1, \ldots, \hat{y}_{v_1}$ into the first o_1 components of $\mathcal{F}$ is equivalent to computing

$$(\hat{y}_1, \ldots, \hat{y}_{v_1}) \cdot \begin{pmatrix} \alpha_{11}^{(k)} & \cdots & \alpha_{1v_1}^{(k)} \\ & \ddots & \vdots \\ & & \alpha_{v_1v_1}^{(k)} \end{pmatrix} \cdot \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_{v_1} \end{pmatrix}$$
$$+ (\hat{y}_1, \ldots, \hat{y}_{v_1}) \cdot \begin{pmatrix} \beta_{1v_1+1}^{(k)} & \cdots & \beta_{1v_2}^{(k)} \\ \vdots & & \vdots \\ \beta_{v_1v_1+1}^{(k)} & \cdots & \beta_{v_1v_2}^{(k)} \end{pmatrix} \cdot \begin{pmatrix} y_{v_1+1} \\ \vdots \\ y_{v_2} \end{pmatrix},$$

for $k = v_1 + 1, \ldots, v_2$. With a similar equation occuring for the second layer, namely,

$$(\hat{y}_1, \ldots, \hat{y}_{v_2}) \cdot \begin{pmatrix} \alpha_{11}^{(k)} & \ldots & \alpha_{1v_2}^{(k)} \\ & \ddots & \vdots \\ & & \alpha_{v_1 v_2}^{(k)} \end{pmatrix} \cdot \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_{v_2} \end{pmatrix}$$

$$+ (\hat{y}_1, \ldots, \hat{y}_{v_2}) \cdot \begin{pmatrix} \beta_{1v_2+1}^{(k)} & \ldots & \beta_{1n}^{(k)} \\ \vdots & & \vdots \\ \beta_{v_2 v_2+1}^{(k)} & \ldots & \beta_{v_2 n}^{(k)} \end{pmatrix} \cdot \begin{pmatrix} y_{v_2+1} \\ \vdots \\ y_n \end{pmatrix}$$

for $k = v_2+1, \ldots, n$. We call the $2 \cdot (n - v_1)$ matrices in these equations $A^{(k)}, B^{(k)}$. So (abusing notation a bit) we write the equations as $f_k = \hat{\mathbf{y}} \cdot A^{(k)} \cdot \hat{\mathbf{y}}^\mathsf{T} + \hat{\mathbf{y}} \cdot A^{(k)} \cdot \mathbf{y}^\mathsf{T}$. Recall at any stage we know $\hat{\mathbf{y}}$ and we want to solve the equations for $\mathbf{y}$.

It is clear that signing, given $\mathbf{h} \in \mathbb{F}_q^m$, is a purely algebraic operation over $\mathbb{F}_q$. Thus it can be accomplished in a threshold manner via any LSSS based MPC protocol which evaluates arithmetic circuits over $\mathbb{F}_q$, such as those mentioned earlier. We assume that the private key already exists in secret shared form, i.e. we have sharings $\langle S^{-1} \rangle$, $\langle T^{-1} \rangle$, $\langle \alpha_{i,j}^{(k)} \rangle$ and $\langle \beta_{i,j}^{(k)} \rangle$.

We now look at the signing algorithm's complexity from the point of view of MPC evaluation. We count both the multiplicative depth, as well the number of secure $\mathbb{F}_q$ multiplications needed.

- The first two operations of the signing algorithm come for free, as they are a public hash calculation, followed by the linear opeation $\langle \mathbf{x} \rangle \leftarrow \langle S^{-1} \rangle \cdot \mathbf{h}$.
- We then need to evaluate the map $\mathcal{F}$. This executes in a number of phases.
  - We generate shared values $\langle y_1 \rangle, \ldots, \langle y_{v_1} \rangle$ at random.
  - We then translate the first level of o_2 equations $f_k = \mathbf{x}^{(k)}$ for $k = v_1 + 1, \ldots, v_2$ into a linear system to solve for $\mathbf{y}_1 = (y_{v_1+1}, \ldots, y_{v_2})$. Thus we find an $o_1 \times o_1$ shared matrix $\langle C \rangle$ and a vector $\langle \mathbf{b} \rangle$ such that $C \cdot \mathbf{y}_1 = \mathbf{b}$. To *determine* this system requires two rounds of communication and

$$\begin{aligned} M_1 &= o_1 \cdot \left(\sum_{i=1}^{v_1} i + v_1 + (v_2 - v_1) \cdot v_1 \right) \\ &= o_1 \cdot (v_1 \cdot (v_1 + 1)/2 + v_1 + o_1 \cdot v_1) \\ &= o_1 \cdot (o_1 \cdot v_1 + v_1 \cdot (v_1 + 3)/2) = 175032 \end{aligned}$$

    secure multiplications.
  - Solving our linear system to obtain $\langle \mathbf{y}_1 \rangle$ using our method from Figure 6, which requires three rounds of communication and $M_2 = o_1^3 + o_1^2 = 47952$ secure multiplications.

  - We then repeat with the second layer of the central map, which requires

$$\begin{aligned} M_3 &= o_2 \cdot \left(\sum_{i=1}^{v_2} i + v_2 + (n - v_2) \cdot v_2 \right) \\ &= o_2 \cdot (v_2 \cdot (v_2 + 1)/2 + v_2 + o_2 \cdot v_2) \\ &= o_2 \cdot (o_2 \cdot v_2 + v_2 \cdot (v_2 + 3)/2) = 335088. \end{aligned}$$

    secure multiplications, and another two rounds of communication.
  - We now solve this new linear system to obtain $\langle \mathbf{y}_2 \rangle$ using Figure 6. Again this requires three rounds of communication and $M_4 = M_2$ secure multiplications.
- We then compute $\langle \mathbf{z} \rangle \leftarrow \langle T^{-1} \rangle \cdot \langle \mathbf{y} \rangle$. This requires one round of communication, and due to the special form of T^{-1} it requires $M_5 = v_1 \cdot (o_1 + o_2) + o_1 \cdot o_2 = 6192$ secure multiplications.
- Finally we need to open $\langle \mathbf{z} \rangle$ to obtain the signature in the clear which takes one round of communication.

In summary we require $2 + 3 + 2 + 3 + 1 + 1 = 12$ rounds of communication and $M_1 + M_2 + M_3 + M_4 + M_5 = 612216$ secure multiplications. Note the last two steps could be computed by opening the last o_2 variables (one round), and then computing $v_1 \cdot o_1 = 2448$ secure multiplications (one round), with another round of communication to open the first $v_1 + o_1$ variables. In practice we expect the extra round to be more costly than the extra multiplications.

If the above algorithm aborts, which can happen if the linear systems have zero determinant, or the random matrices in the trick to solve the linear systems also have zero determinant, then we simply repeat the signing algorithm again. The probability of an abort is bounded by $4/q$. The Rainbow specification uses $q = 2^8$, thus we expect to need to repeat the signing process with probability about 1.5 percent. As mentioned in the introduction a LSSS based MPC protocol can process at least a 250,000 secure multiplications per second over the field $\mathbb{F}_{2^8}$ in the honest majority setting. Thus we expact an implementation of a threshold version of Rainbow to take around three seconds. A *major* disadvantage of this threshold variant of Rainbow is the need to store so much data in secret shared form, namely $\langle S^{-1} \rangle$, $\langle T^{-1} \rangle$, $\langle \alpha_{i,j}^{(k)} \rangle$ and $\langle \beta_{i,j}^{(k)} \rangle$.

6.2 LUOV

Here we present the LUOV signature scheme [3]. As we shall see this is almost entirely made up of low depth algebraic operations, making this scheme a prefect candidate for a threshold variant. The main non-linear component is a map $\mathcal{F} : \mathbb{F}_{2^r}^n \to \mathbb{F}_{2^r}^m$ with components $(f_1, \ldots, f_m)$ where

$$f_k(\mathbf{x}) = \sum_{i=1}^{v} \sum_{j=1}^{n} \alpha_{i,j,k} x_i x_j + \sum_{i=1}^{n} \beta_{i,k} x_i + \gamma_k,$$

with the coefficients $\alpha_{i,j,k}, \beta_{i,k}$ and γ_k being chosen from the field $\mathbb{F}_{2^r}$ by expanding a seed which forms part of the secret key. The integers n, m and v are related by the $v = n - m$. The elements in $\{x_1, \ldots, x_v\}$ are called the *vinegar* variables and that the ones in $\{x_{v+1}, \ldots, x_n\}$ are the *oil* variables. Note that the polynomials $f_1, \ldots, f_m$ contain contain no quadratic terms $x_i \cdot x_j$ with both x_i and x_j oil variables.

The central map $\mathcal{F}$ has to be secret and in order to hide the structue of $\mathcal{F}$ in the public key, one composes $\mathcal{F}$ with an affine map $\mathcal{T} : \mathbb{F}_{2^r}^n \to \mathbb{F}_{2^r}^m$. The public key consisting of composition $\mathcal{P} = \mathcal{F} \circ \mathcal{T} : \mathbb{F}_{2^r}^n \to \mathbb{F}_{2^r}^m$, and the private key being $\mathcal{P}$. At the Level-4 security level (Level-3 is not provided for LUOV) there are two sets of parameters $(r, m, v) = (8, 82, 323)$ and $(64, 61, 302)$.

The LUOV signature generation

Input: The message to be signed μ, and the data $\mathbf{C}, \mathbf{L}, \mathbf{Q}_1$ and $\mathbf{T}$.
Output: A signature $(\mathbf{s}, \mathsf{salt})$ on the message μ.

1. $\mathsf{salt} \leftarrow \{0,1\}^{16 \cdot 8}$.
2. $\mathbf{h} \leftarrow H\left(\mu \| \mathsf{0x00} \| \mathsf{salt}\right)$
3. While no solution $\mathbf{s}'$ for the system $\mathcal{F}(\mathbf{s}) = \mathbf{h}$ is found
   (a) $\mathbf{v} \leftarrow \mathbb{F}_{2^r}^v$.
   (b) $\mathbf{RHS} \| \mathbf{LHS} \leftarrow \text{BuildAugmentedMatrix}\left(\mathbf{C}, \mathbf{L}, \mathbf{Q}_1, \mathbf{T}, \mathbf{h}, \mathbf{v}\right)$
   (c) If $\det(\mathbf{LHS}) \neq 0$ set $\mathbf{o} \leftarrow \mathbf{LHS}^{-1} \cdot \mathbf{RHS}$.
4. $\mathbf{s} \leftarrow \begin{pmatrix} \mathbf{1}_v & -\mathbf{T} \\ \mathbf{0} & \mathbf{1}_m \end{pmatrix} \cdot \begin{pmatrix} \mathbf{v} \\ \mathbf{o} \end{pmatrix}$
5. Return $(\mathbf{s}, \mathsf{salt})$.

Figure 8. The LUOV signature generation

The LUOV public and private keys are in practice exapanded from a random seed to define the actual data defining the various maps. However, for our threshold variant we assume this expansion has already happened and we have the following data values $\mathbf{C} \in \mathbb{F}_{2^r}^m$, $\mathbf{L} \in \mathbb{F}_{2^r}^{m \times n}$, $\mathbf{Q}_1 \in \mathbb{F}_{2^r}^{m \times (\frac{v(v+1)}{2} + v \cdot m)}$, and $\mathbf{T} \in \mathbb{F}_{2^r}^{v \times m}$, where $\mathbf{C}$, $\mathbf{L}$, and $\mathbf{Q}_1$ are public values and the matrix $\mathbf{T}$ is a secret parameter. In our threshold variant the parameter $\mathbf{T}$ will be held in secret shared form $\langle \mathbf{T} \rangle$. There is another matrix $\mathbf{Q}_2$, but that will not concern us as it is only related to verification. The signing algorithm is given in Figure 8, and makes use of an auxillary algorithm given in Figure 9 and a hash function $H : \{0,1\}^* \longrightarrow \mathbb{F}_{2^r}^m$ The auxiliary algorithm builds a linear system $\mathbf{LHS} \cdot \mathbf{o} = \mathbf{RHS}$ which we solve to obtain the oil variables.

We now examine the above algorithm from the point of view of how one could implement it in a threshold manner given a generic MPC functionality for arithmetic circuits over $\mathbb{F}_{2^r}$. We assume that the secret matrix $\mathbf{T}$ is presented in secret shared form $\langle \mathbf{T} \rangle$. First note that the hash function is only called to compute the hash digest, which does not involve any shared input.

In the main while loop we assume the vinegar variables are generated in a shared manner in secret shared form $\langle \mathbf{v} \rangle$. Thus the main call to the BuiltAug-

BuildAugmentedMatrix

Input: The data $\mathbf{C}, \mathbf{L}, \mathbf{Q}_1$ and $\mathbf{T}$, the hashed message $\mathbf{h} \in \mathbb{F}_{2^r}^m$, and an assignment to the vinegar variables $\mathbf{v} \in \mathbb{F}_{2^r}^v$.
Output: $\mathbf{LHS} \| \mathbf{RHS} \in \mathbb{F}_{2^r}^{m \times m+1}$, the augmented matrix for $\mathcal{F}(\mathbf{v}\|\mathbf{o}) = \mathbf{h}$.

1. $\mathbf{RHS} \leftarrow \mathbf{h} - \mathbf{C} - \mathbf{L} \cdot \begin{pmatrix} \mathbf{v} \\ \mathbf{0} \end{pmatrix}$
2. $\mathbf{LHS} \leftarrow \mathbf{L} \cdot \begin{pmatrix} -\mathbf{T} \\ \mathbf{1}_m \end{pmatrix}$
3. For k from 1 to m
   (a) From $\mathbf{Q}_1$ build a public matrix $\mathbf{P}_{k,1} \in \mathbb{F}_2^{v \times v}$ (for details see the LUOV specification).
   (b) From $\mathbf{Q}_1$ build $\mathbf{P}_{k,2} \in \mathbb{F}_2^{v \times m}$ (again see the specification).
   (c) $\mathbf{RHS}[k] \leftarrow \mathbf{RHS}[k] - \mathbf{v}^\mathsf{T} \cdot \mathbf{P}_{k,1} \cdot \mathbf{v}$.
   (d) $\mathbf{F}_{k,2} \leftarrow -\left(\mathbf{P}_{k,1} + \mathbf{P}_{k,1}^\mathsf{T}\right) \cdot \mathbf{T} + \mathbf{P}_{k,2}$.
   (e) $\mathbf{LHS}[k] \leftarrow \mathbf{LHS}[k] + \mathbf{v} \cdot \mathbf{F}_{k,2}$.
4. Return $\mathbf{LHS} \| \mathbf{RHS}$

Figure 9. BuildAugmentedMatrix

mentedMatrix routine has two secret shared input $\langle \mathbf{T} \rangle$ and $\langle \mathbf{v} \rangle$, with the other values being public. The key lines in this algorithm then requiring secure multiplications are lines 3c and 3e to compute $\langle \mathbf{v} \rangle^\mathsf{T} \cdot \mathbf{P}_{k,1} \cdot \langle \mathbf{v} \rangle$ and $\langle \mathbf{v} \rangle \cdot \mathbf{F}_{k,2}$ respectively. The first of these takes v secure multiplications, whereas the latter requires $v \cdot m$ secure multiplications. Giving a total of $v \cdot m \cdot (m+1)$ secure multiplications in total, which can be performed in parallel in one round of communication.

Solving the nonlinear system in line 3c is done using the method in Figure 6, which requires three rounds of interaction and $m^3 + m^2$ secure multiplications. Note the probability that this procedure fails is roughly 2^{-r+1}, which can be essentially ignored for the parameter set with $r = 64$ and is under one percent for the parameter set with $r = 8$. But if it does fail, then we simply repeat the signing algorithm with new shared vinegar variables.

We then need to compute the matrix multiplication $\langle \mathbf{T} \rangle \cdot \langle \mathbf{o} \rangle$. However, note that we can save some secure multiplications by opening the oil variables $\mathbf{o}$ after the matrix inversion (since they are going to be released in any case in the clear). This will require anyway a rouond of interaction, but we are then able to save the $v \cdot m$ secure multiplications required to multiply $\mathbf{T}$ by $\mathbf{o}$, since that operation then becomes a linear operation. Finally, we open the resulting shared signature in order to transmit it in the clear. This requires one round of interaction.

Thus the overall cost of LUOV signature algorithm is $1+3+1+1 = 6$ rounds of interaction and $(m^3+m^2)+v \cdot m \cdot (m+1)$ secure multiplications. Choosing the Level-4 parameter set with $(r, m, v) = (8, 82, 323)$ this gives a total of 2756430 secure multiplications. Whereas for the parameter set $(r, m, v) = (64, 61, 302)$ this gives us 1372866 secure multiplications. In the former case, where arithmetic is over $\mathbb{F}_{2^8}$ and we expect to perform 250,000 secure multiplications per second, signing will take about 10 seconds. In the latter case, where arithmetic is over $\mathbb{F}_{2^{64}}$ and we expect to perform 1,000,000 secure multiplications per second, signing will take about 1.3 seconds. Another advantage of LUOV is that our threshold variant requires less storage of secret key material. We only need to store $\langle \mathbf{T} \rangle$ in secret shared form.

It is worth mentioning that at the NIST second PQC Standardization Conference a new attack has been presented [14] against LUOV. This attack crucially exploits the existence intermediate subfields in $\mathbb{F}_{2^r}$. Consequentely, the authors proposed new sets of parameters to ensure the extension degree r is prime. Our expected run times above are likely to be similar for the new prime power finite fields. However, the other parameters are now a little smaller, resulting in the need for fewer secure multiplications. Thus we expect the new version of LUOV will be more efficient as a threshold variant.

Acknowledgments

This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT, by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contracts No. N66001-15-C-4070 and FA8750-19-C-0502, and by the FWO under an Odysseus project GOH9718N. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the ERC, United States Air Force, DARPA or FWO. The authors would like to thank Cyprien Delpech de Saint Guilhem and Dragos Rotaru for helpful discussions whilst this work was carried out.

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. Bernstein, D.J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., Wilcox-O'Hearn, Z.: SPHINCS: Practical stateless hash-based signatures. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 368–397. Springer, Heidelberg (Apr 2015)
3. Beullens, W., Preneel, B., Szepieniec, A., Vercauteren, F.: LUOV (2019), Submission to NIST PQC "competition" Round-2.
4. Bindel, N., Akleylek, S., Alkim, E., Barreto, P.S., Buchmann, J., Eaton, E., Gutoski, G., Kramer, J., Longa, P., Polat, H., Ricardini, J.E., Zanon, G.: Lattice-based digital signature scheme qTESLA (2019), Submission to NIST PQC "competition" Round-2.
5. Casanova, A., Faugère, J.C., Patarin, G.M.R.J., Perret, L., Ryckeghem, J.: GeMSS: A great multivariate short signature (2019), Submission to NIST PQC "competition" Round-2.
6. Catrina, O., de Hoogh, S.: Improved primitives for secure multiparty integer computation. In: Garay, J.A., Prisco, R.D. (eds.) SCN 10. LNCS, vol. 6280, pp. 182–199. Springer, Heidelberg (Sep 2010)

7. Chida, K., Genkin, D., Hamada, K., Ikarashi, D., Kikuchi, R., Lindell, Y., Nof, A.: Fast large-scale honest-majority MPC for malicious adversaries. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 34–64. Springer, Heidelberg (Aug 2018)
8. Cramer, R., Damgård, I., Escudero, D., Scholl, P., Xing, C.: SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 769–798. Springer, Heidelberg (Aug 2018)
9. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 285–304. Springer, Heidelberg (Mar 2006)
10. Damgård, I., Koprowski, M.: Practical threshold RSA signatures without a trusted dealer. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 152–165. Springer, Heidelberg (May 2001)
11. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (Aug 2012)
12. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 307–315. Springer, Heidelberg (Aug 1990)
13. Ding, J., Chen, M.S., Petzoldt, A., Schmidt, D., Yang, B.Y.: Rainbow (2019), Submission to NIST PQC "competition" Round-2.
14. Ding, J., Zhang, Z., Deaton, J., Schmidt, K., Vishakha, F.: New attacks on lifted unbalanced oil vinegar (2019), the 2nd NIST PQC Standardization Conference.
15. Doerner, J., Kondi, Y., Lee, E., shelat, a.: Secure two-party threshold ECDSA from ECDSA assumptions. In: 2018 IEEE Symposium on Security and Privacy. pp. 980–997. IEEE Computer Society Press (May 2018)
16. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust threshold DSS signatures. In: Maurer, U.M. (ed.) EUROCRYPT'96. LNCS, vol. 1070, pp. 354–371. Springer, Heidelberg (May 1996)
17. 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)
18. Gentry, C., Szydlo, M.: Cryptanalysis of the revised NTRU signature scheme. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 299–320. Springer, Heidelberg (Apr / May 2002)
19. 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)
20. Goldreich, O.: Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 104–110. Springer, Heidelberg (Aug 1987)
21. Goldreich, O.: Foundations of Cryptography: Basic Applications, vol. 2. Cambridge University Press, Cambridge, UK (2004)
22. Hazay, C., Scholl, P., Soria-Vazquez, E.: Low cost constant round MPC combining BMR and oblivious transfer. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 598–628. Springer, Heidelberg (Dec 2017)
23. Hoffstein, J., Howgrave-Graham, N., Pipher, J., Silverman, J.H., Whyte, W.: NTRUSIGN: digital signatures using the NTRU lattice. In: Joye, M. (ed.) Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13-17, 2003, Proceedings. Lecture

Notes in Computer Science, vol. 2612, pp. 122–140. Springer (2003), https://doi.org/10.1007/3-540-36563-X_9
24. Hulsing, A., Bernstein, D.J., Dobraunig, C., Eichlseder, M., Fluhrer, S., Gazdag, S.L., Kampanakis, P., Kolbl, S., Lange, T., Lauridsen, M.M., Mendel, F., Niederhagen, R., Rechberger, C., Rijneveld, J., Schwabe, P.: SPHINCS+ (2019), Submission to NIST PQC "competition" Round-2.
25. 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)
26. Kipnis, A., Patarin, J., Goubin, L.: Unbalanced oil and vinegar signature schemes. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 206–222. Springer, Heidelberg (May 1999)
27. Kraitsberg, M., Lindell, Y., Osheter, V., Smart, N.P., Alaoui, Y.T.: Adding distributed decryption and key generation to a Ring-LWE based CCA encryption scheme. IACR Cryptology ePrint Archive 2018, 1034 (2018), https://eprint.iacr.org/2018/1034
28. Lamport, L.: Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory (Oct 1979)
29. Larraia, E., Orsini, E., Smart, N.P.: Dishonest majority multi-party computation for binary circuits. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 495–512. Springer, Heidelberg (Aug 2014)
30. Lindell, Y.: Fast secure two-party ECDSA signing. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 613–644. Springer, Heidelberg (Aug 2017)
31. Lindell, Y., Nof, A.: Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 1837–1854. ACM Press (Oct 2018)
32. Lindell, Y., Nof, A., Ranellucci, S.: Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. IACR Cryptology ePrint Archive 2018, 987 (2018), https://eprint.iacr.org/2018/987
33. Lyubashevsky, V.: Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 598–616. Springer, Heidelberg (Dec 2009)
34. Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehle, D.: Crystals-dilithium (2019), Submission to NIST PQC "competition" Round-2.
35. MacKenzie, P.D., Reiter, M.K.: Two-party generation of DSA signatures. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 137–154. Springer, Heidelberg (Aug 2001)
36. Matsumoto, T., Imai, H.: Public quadratic polynominal-tuples for efficient signature-verification and message-encryption. In: Günther, C.G. (ed.) EUROCRYPT'88. LNCS, vol. 330, pp. 419–453. Springer, Heidelberg (May 1988)
37. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (Aug 1990)
38. Mohassel, P., Rosulek, M., Zhang, Y.: Fast and secure three-party computation: The garbled circuit approach. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015. pp. 591–602. ACM Press (Oct 2015)
39. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 681–700. Springer, Heidelberg (Aug 2012)

40. Patarin, J.: The oil and vinegar signature scheme (1997), presentation at the Dagstuhl Workshop on Cryptography
41. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon: Fast-fourier lattice-based compact signatures over NTRU (2019), Submission to NIST PQC "competition" Round-2.
42. Reyzin, L., Reyzin, N.: Better than BiBa: Short one-time signatures with fast signing and verifying. In: Batten, L.M., Seberry, J. (eds.) ACISP 02. LNCS, vol. 2384, pp. 144–153. Springer, Heidelberg (Jul 2002)
43. Rotaru, D., Wood, T.: Marbled circuits: Mixing arithmetic and boolean circuits with active security. IACR Cryptology ePrint Archive 2019, 207 (2019), `https://eprint.iacr.org/2019/207`
44. Samardjiska, S., Chen, M.S., Hulsing, A., Rijneveld, J., Schwabe, P.: MQDSS (2019), Submission to NIST PQC "competition" Round-2.
45. Shoup, V.: Practical threshold signatures. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 207–220. Springer, Heidelberg (May 2000)
46. Smart, N.P., Wood, T.: Error detection in monotone span programs with application to communication-efficient multi-party computation. In: Matsui, M. (ed.) CT-RSA 2019. LNCS, vol. 11405, pp. 210–229. Springer, Heidelberg (Mar 2019)
47. Wang, X., Ranellucci, S., Katz, J.: Global-scale secure multiparty computation. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 39–56. ACM Press (Oct / Nov 2017)
48. Zaverucha, G., Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D.: The picnic signature scheme (2019), Submission to NIST PQC "competition" Round-2.