Share the MAYO: thresholdizing MAYO

Sofia Celi[1], Daniel Escudero[2], and Guilhem Niot[3]

[1] Brave Software,
cherenkov@riseup.net
[2] J.P. Morgan AI Research & J.P. Morgan AlgoCRYPT CoE,
daniel.escudero@protonmail.com
[3] PQShield, Univ Rennes, CNRS, IRISA
guilhem@gniot.fr

Abstract. We present the first comprehensive study on thresholdizing practical OV-based signature schemes, specifically focusing on MAYO and UOV. Our approach begins by addressing the challenges associated with thresholdizing algorithms that sample solutions to linear equation systems of the form $\mathbf{Ax} = \mathbf{y}$, which are fundamental to OV-based signature schemes. Previous attempts have introduced levels of leakage that we deem insecure. We propose a novel minimum-leakage solution and assess its practicality. Furthermore, we explore the thresholdization of the entire functionality of these signature schemes, demonstrating their unique applications in networks and cryptographic protocols.

Keywords: Post-quantum, threshold-cryptography, multivariate-based cryptography

This is the full version of a paper to appear in the proceedings of the 16th International Conference on Post-Quantum Cryptography (PQCrypto 2025). This version includes a discussion on applications and complete security proofs.

1 Introduction

Nowadays, threshold signatures [Des90; DF90] are an emerging field in cryptography, as they provide many real-world applications by allowing the distribution of signing power to several parties using different access structures. This distribution of signing power is often given to any subset of t parties among a set of n of signers but with the restriction that $t-1$ cannot sign. While the case for efficient EC-DSA-based or Schnorr-based threshold-signature schemes has gathered renewed interest in the recent years [Sho99; GGN16; Bol03; KG20; Lin24; Ruf+22; Gar+21; Doe+19; Doe+23] (the former for their wide applications and the latter mostly due to their applications in blockchain), they are mostly centered in a classical setting and have not focused on providing security against quantum-adversaries. Moreover, the security of these schemes is nuanced, often requiring additional mechanisms to address potential misbehavior by parties. This misbehavior can stem from parties being either dishonest or *honest but*

curious. Furthermore, from a practical setting, some threshold schemes are incompatible with widely-used standardised signature schemes, or, when they are, they tend to be inefficient or reliant upon ad-hoc assumptions.

Recently, there has been a renewed interest in providing quantum-security to threshold signatures, as "quantum resistance" is listed as an important criterion on the recent NIST standardization call [PB23, Sections 3.2 and 3.3] for threshold signatures. A starting point in research on this area has focused on actively looking at the proposed submissions for the NIST PQC standardization call [NIS22] and analyzing what it will take to have a threshold variant of them. Due to this, we have seen recently proposals of lattice-based threshold schemes [Bos+24; Pin+24; ENP24; ASY22], isogeny-based threshold schemes [DM20], and hashed-based threshold schemes [Kha+22]. Notably, Cozzo et. al [CS19] explored the viability of applying generic MPC techniques to many of the NIST PQC submissions at the time [4] in order to instantiate threshold signatures schemes.

In this work, we are interested in thresholdising multivariate-based signature schemes, as they seem to arrive at practical communicational and computational measures, and seem to be tailored and feasible for real-world applications [Wes24; Adr24]. Ostensibly, both the MAYO [Beu+23a] and UOV [Beu+23b] signature schemes seem to be highly practical, and their security relies on a well-studied problem that has withstood the test of time despite cryptanalytic efforts.

Our starting point, as noted, will be the work of Cozzo et. al [CS19]. Our goal is to explore what it will take to make threshold variants of both UOV and MAYO, as practical OV-based schemes that are part of Round-2 of the "on-ramp" post-quantum process [Ala+24]. Hence, we prioritize schemes that might be standardized (and, thus, might become widely available), grounded in a well-studied problem, and practical. We push the study further, and do not only focus on constructing threshold signing for OV-based schemes; we propose each functionality in its own: distributed key generation (DKG), multi-party signature generation, and multi-party verification. This modular approach caters to diverse application needs, allowing for tailored solutions.

Contributions. We contribute the following results in our work:

- **Threshold variant of Solving Systems of Linear Equations:** We present a minimum-leakage threshold variant for the solving systems of linear equations algorithm. We provide an in-depth discussion of its leakage characteristics and strategies for minimization.
- **Threshold variants of OV-based schemes:** We propose threshold variants of the MAYO and UOV signature schemes, addressing the demand for quantum-resistant cryptographic solutions that securely operate in multi-party settings. We highlight their "MPC-friendliness" and present a "matrix-only" perspective that facilitates efficient MPC operations.
- **Practical Framework:** Our work outlines a practical framework for constructing these threshold schemes, ensuring that the proposed schemes maintain efficiency and compatibility with existing standardized signature algorithms.

[4] At the Round-2 of the NIST post-quantum first standardization process.

We identify optimization opportunities and situate our work within an established MPC framework and model.
- **Modular Functionality:** We introduce a modular approach to threshold schemes by separately addressing critical functionalities to enhance flexibility and usability.

Outline. We start with the needed preliminaries in Section 2. Then, we provide a description of OV-based schemes in Section 3. In Section 4, Section 5 and Section 6, we present threshold protocols implementing system solving of linear equations, DKG, signing and verification algorithms. In Section 7, we discuss the costs and optimization techniques applicable to our protocol; in Section 8, we present possible applications

2 Preliminaries

Sets, functions, and distributions. For an integer $N > 0$, we note $[N] = \{0, \ldots, N-1\}$. To denote the *assign* operation, we use $y := f(x)$ when f is deterministic and $y \leftarrow f(x)$ when randomized. When S is a finite set, we note $\mathcal{U}(S)$ the uniform distribution over S, and shorthand $x \xleftarrow{\$} S$ for $x \leftarrow \mathcal{U}(S)$. We use $||$ to denote concatenation and λ as the security parameter. We use the standard Landau notation $O(\cdot)$ for asymptotics.

Finite fields $\mathbb{F}_q$, and vectors over $\mathbb{F}_q$. Given q, a power of a prime number, we denote $\mathbb{F}_q$ a finite field with q elements. We give an explicit representation for $\mathbb{F}_{16}$ as $\mathbb{Z}_2[x]/(x^4 + x + 1)$. We denote the addition and multiplication of field elements a and b as $a + b$ and ab respectively, and we denote the multiplicative inverse of a as a^{-1}. We denote by $\mathbb{F}_q^n$ the set of column vectors of length n over $\mathbb{F}_q$. Vectors are noted with small bold letters (i.e, $\mathbf{x}$), and we write $(x_0, ..., x_{n-1})$ for the coordinates of $\mathbf{x} \in \mathbb{F}_q^n$.

Matrices. We denote by $\mathbb{F}_q^{m \times n}$ the set of (zero-indexed) matrices over $\mathbb{F}_q$ with m rows and n columns. We denote by $\mathbf{I}_n \in \mathbb{F}_q^{n \times n}$ the identity matrix of size n-by-n. Abusing notation, we see a vector $\mathbf{b} \in \mathbb{F}_q^n$ as a matrix of dimension $n \times 1$. If $\mathbf{A} \in \mathbb{F}_q^{m \times n}$, we denote by $\mathbf{A}[i, j]$ the entry in the i-th row and the j-th column of $\mathbf{A}$, and the j-th column of $\mathbf{A}$ by $\mathbf{A}[:, j] \in \mathbb{F}_q^m$. We say that a matrix $\mathbf{A} \in \mathbb{F}_q^{n \times n}$ is upper triangular if $\mathbf{A}[i, j] = 0$ for all $0 \leq j < i < n$.

If $\mathbf{A}, \mathbf{B} \in \mathbb{F}_q^{m \times n}$ are matrices of same size, then we denote their (entry-wise) sum as $\mathbf{A} + \mathbf{B}$. If $\mathbf{A} \in \mathbb{F}_q^{m \times n}$ and $\mathbf{B} \in \mathbb{F}_q^{n \times k}$, then we denote the matrix product by $\mathbf{AB}$, i.e. $\mathbf{AB} \in \mathbb{F}_q^{n \times k}$ is the matrix whose entry in row i and column j is equal to $\sum_{l=0}^{n} \mathbf{A}[i, l]\mathbf{B}[l, j]$. We denote by $\mathbf{A}^\top$ the transpose of $\mathbf{A}$, i.e. the matrix in $\mathbb{F}_q^{n \times m}$ such that $\mathbf{A}^\top[i, j] = \mathbf{A}[j, i]$ for all $0 \leq j < m$ and $0 \leq i < n$.

We define the function $\mathsf{Upper} : \mathbb{F}_q^{n \times n} \to \mathbb{F}_q^{n \times n}$ that takes a square matrix $\mathbf{M}$ as input, and outputs the upper triangular matrix $\mathsf{Upper}(\mathbf{M})$, defined as $\mathsf{Upper}(\mathbf{M})_{i,i} = \mathbf{M}_{i,i}$ and $\mathsf{Upper}(\mathbf{M})_{i,j} = \mathbf{M}_{i,j} + \mathbf{M}_{j,i}$ for $i < j$.

Network model. We assume a synchronous network, as already assumed and required by modern threshold signature schemes [Kon+21; GG19; LNR18]. We also consider that the adversary can observe any message sent, and choose to deliver or block messages at will on this network.

2.1 The UC Framework: Model and Functionalities

For our ideal functionalities, adversarial model, and proofs, our scheme instantiations rely on the Universal-Composability (UC) framework introduced by Canetti [Can01]. This framework allows for the independent implementation of functionalities and to securely compose them. In the following, we give a brief overview of the framework, where we use the traditional notation to refer to protocols in uppercase and procedures in the lowercase.

UC Protocols. The UC framework is based on *Probabilistic Polynomial-Time* (PPT) *Interactive Turing machines* (ITM) entities that are used to model parties, adversaries, and simulators.

In the *real-world* experiment, we have n ITM-based parties $(P_1, \ldots, P_n)$ that execute a protocol Π, an adversary $\mathcal{A}$ that can corrupt any subset of parties, and an environment $\mathcal{Z}$ that is initialized with an advice string z. All entities are initialized with the security parameter λ and with a random tape. The environment $\mathcal{Z}$ activates the parties involved in Π, selects their inputs, receives their outputs, and communicates with the adversary $\mathcal{A}$, who may instruct the corrupted parties to deviate from Π in any manner. The real-world experiment completes when Z ceases to activate parties and produces a decision bit.

Conversely, the *ideal-world* experiment involves n ITM-based dummy parties $(P_1, \ldots, P_n)$, an ideal functionality $\mathcal{F}$, an ideal-world adversary $\mathcal{S}$ (the simulator), and an environment $\mathcal{Z}$. The dummy parties relay messages from $\mathcal{Z}$ to $\mathcal{F}$. The simulator $\mathcal{S}$ can corrupt any subset of these dummy parties and interact with $\mathcal{F}$ on their behalf communicating directly with $\mathcal{F}$ according to its specification. Throughout the experiment, $\mathcal{Z}$ and $\mathcal{S}$ interact with the simulator, the role of the simulator goal being to convince the environment that it is participating in the real experiment. The ideal-world experiment completes when $\mathcal{Z}$ stops activating parties and outputs a decision bit.

The $\mathcal{F}$-hybrid Model. When a protocol Π operates in a hybrid model of computation where parties can communicate as usual and have access to an unlimited number of instances of the ideal functionality $\mathcal{F}$, it is called the $\mathcal{F}-$hybrid model [Can+02]. In this setting, let P denote a protocol that UC-realizes $\mathcal{F}$, and let Π^{P} be the "composed protocol". Composition here means that Π^{P} is identical to Π, except that each interaction with $\mathcal{F}$ is substituted by an activation of the concrete instance of the protocol, with P's outputs serving as values provided by $\mathcal{F}$. A fundamental theorem states that in such conditions, Π and Π^{P} exhibit the same input/output behavior: P behaves just like the ideal functionality $\mathcal{F}$ even when composed with an arbitrary protocol Π. A special case of this theorem asserts that if Π UC-realizes some ideal functionality $\mathcal{G}$ in the $\mathcal{F}$-hybrid model, then Π^{P} also UC-realizes $\mathcal{G}$.

UC-Security Model. [Can01] defines a notion of security for a protocol Π by comparing it to the ideal functionality $\mathcal{F}$. The core intuition is that any attack on Π would be no more effective than an attack on $\mathcal{F}$. Let $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}(\lambda,z)}$ denote the random variable representing the output of the *ideal-world* experiment; let $\mathsf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}(\lambda,z)}$ denote the random variable representing the output of the *real-world* experiment.

Definition 1 (UC-security). *A protocol Π UC-realizes the functionality $\mathcal{F}$, if for every PPT $\mathcal{A}$, there is a simulator $\mathcal{S}$ that for every PPT "admissible" environment $\mathcal{Z}$,*

$$\{\mathsf{REAL}_{\Pi,\mathcal{Z}(\lambda,z)}\}_{\lambda\in\mathbb{N}^+,z\in\{0,1\}^{\mathsf{poly}(\lambda)}} \approx_c \{\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}(\lambda,z)}\}_{\lambda\in\mathbb{N}^+,z\in\{0,1\}^{\mathsf{poly}(\lambda)}}$$

Remark 1 (On the Difference between Procedures and Protocols). Universal Composability (UC) security is crucial not only for ensuring composability but also for facilitating a modular approach to protocol design. It enables the definition of functionalities that can be instantiated and utilized later without the need to focus on their concrete implementations. However, when developing complex protocols, it can be beneficial to create several separate "sub-protocols" that can be invoked within the main protocol, even if the specific functionalities these sub-protocols instantiate are not explicitly defined. To address this, we introduce the concept of a *procedure*. Procedures resemble protocols in that they outline the steps parties must follow, but unlike protocols, they are not intended to instantiate a specific functionality. Instead, they serve as modular blocks that can be integrated into an actual protocol that does instantiate functionality. A useful analogy is to compare procedures to "macros" in programming languages such as C or C++. Like macros, which are expanded in the code before compilation, procedures provide a flexible way to structure protocols. In contrast, functionalities can be likened to "binaries", which can be utilized without needing to understand their internal workings.

2.2 The Arithmetic Black-Box Model

MPC protocols are typically designed using secret-sharing techniques. Secret-sharing is a method that enables a dealer to distribute a secret $x \in \mathbb{F}_q$ among parties in such a way that no subset of $\leq t$ parties learns anything about the secret, while any subset of $\geq t+1$ parties can completely reconstruct it. This is commonly denoted with $[\![x]\!]$. The specific choice of secret-sharing schemes is typically influenced by the concrete security setting (*e.g.* passive vs active security, or honest vs dishonest majority). To remain agnostic to any choice, we use a standard abstraction from the literature known as the *arithmetic black box* (ABB), which precisely models the ability of parties to distribute secrets, perform arithmetic operations on them, and reconstruct secrets. The allowed arithmetic operations include both affine operations such as additions, subtractions, or multiplication by constants—all of which can be typically performed *locally* in an actual instantiation thanks to the linear properties of secret-sharing schemes—and also the multiplication of secrets, which requires an interactive protocol in an actual instantiation.

Remark 2 (Sampling both public and secret random values). For our protocols, we describe a slightly modified version of ABB model which allows for a few extra operations that will be useful for our constructions. These are: (1) the ability to sample random strings known to all parties, and (2) the ability to sample *secret* random values that are unknown to any party. Both of these are common in the literature and can be realized using standard techniques. For instance, sampling secret random values can be generated by having each party distribute a random share and then summing these shares to produce the secret value (for honest majority this can be optimized by using super-invertible matrices [DN07]). For public randomness, the parties can reconstruct a secret random value, and if longer strings are needed, a PRG can be used to expand it.

Given this, our ABB functionality ($\mathcal{F}_{\mathsf{ABB}}$) is given below. We set all of our schemes in this $\mathcal{F}_{\mathsf{ABB}}$-hybrid model.

Functionality 1: $\mathcal{F}_{\mathsf{ABB}}$

The functionality keeps track of an internal dictionary of pairs $(x, \mathtt{id})$ mapping field elements to IDs. Although we do not write this explicitly, the functionality aborts in obvious cases such as if the provided parties' IDs are inconsistent, if the IDs are not populated when the functionality is asked to read them, or if the IDs are populated when the functionality is asked to write them. The functionality supports the following commands:

- On input $(\mathsf{input}, \mathtt{id}, P_i)$ from all parties and $(\mathsf{input}, \mathtt{id}, P_i, x)$ from P_i, the functionality stores $(x, \mathtt{id})$.
- On input $(\mathsf{add}, \mathtt{id}_1, \mathtt{id}_2, \mathtt{id}_3)$ from all parties, the functionality fetches $(x, \mathtt{id}_1)$ and $(y, \mathtt{id}_2)$, and stores $(x + y, \mathtt{id}_3)$.
- On input $(\mathsf{addmult\text{-}cons}, \mathtt{id}_1, \mathtt{id}_2, (a, b))$ from all parties, the functionality fetches $(x, \mathtt{id}_1)$ and stores $(a \cdot x + b, \mathtt{id}_2)$.
- On input $(\mathsf{mult}, \mathtt{id}_1, \mathtt{id}_2, \mathtt{id}_3)$ from all parties, the functionality fetches $(x, \mathtt{id}_1)$ and $(y, \mathtt{id}_2)$, and stores $(x \cdot y, \mathtt{id}_3)$.
- On input $(\mathsf{output}, \mathtt{id}, P_i)$ from all parties, the functionality fetches $(x, \mathtt{id})$ and sends x to P_i.
- On input $(\mathsf{rand}, \mathtt{id})$ from all parties, the functionality samples $r \leftarrow \mathbb{F}_q$ uniformly at random and stores $(r, \mathtt{id})$.
- On input (coin) from all parties, the functionality samples $r \leftarrow \mathbb{F}_q$ uniformly at random and sends r to all parties.

Remark 3 (On optimizations of concrete instantiations). The primary advantage of the ABB model is its generality, which allows for a cleaner description that accommodates multiple adversarial settings. However, actual implementations of our protocols can be significantly *optimized* by leveraging specific characteristics of the underlying adversarial model. This is especially beneficial in scenarios with passive security or an honest majority setting, where multiple efficiency improvements are possible. We discuss some of these specific optimizations in Section 7.

Notation and Simplification of the ABB. When applying the ABB model in our constructions, we omit low-level details as noting IDs for simplicity. We denote a value x that is stored $(x, \mathtt{id})$ by $[\![x]\!]$, where, as noted, we ignore the specific ID from the notation. For operations, we write $[\![x+y]\!] \leftarrow [\![x]\!] + [\![y]\!]$ to represent a call to the add command, and similarly for other operations. We also use $\alpha \leftarrow \mathsf{coin}$ and $[\![x]\!] \leftarrow \mathsf{rand}$ for when the parties call the commands coin and rand, respectively. This notation is typically reserved for secret-sharing schemes and it is deliberate: in a concrete instantiation of the ABB model, $[\![x]\!]$ would mean that x is secret-shared among the parties. Furthermore, it will not be uncommon that we refer to $[\![x]\!]$ as the parties having "shares" of x.

The majority of our constructions work with matrices, and we let notation reflect this by using $[\![\mathbf{X}]\!]$ for matrices $\mathbf{X}$, meaning that each individual entry is stored in the functionality. Sometimes we also use $[\![\mathbf{X}]\!] \leftarrow \mathsf{rand}(\mathbb{F}_q^{a\times b})$ to emphasize the dimensions of the sampled matrix. Matrix operations such as addition and multiplication can be instantiated with the underlying field arithmetic, Note that matrix addition and multiplication can be instantiated with the underlying field arithmetic, though matrix multiplication can be optimized by leveraging its structure rather than naively applying field operations. In some cases, multiplication of $n \times n$ matrices can be reduced from n^3 products to n^2. This is discussed in Appendix A. Additionally, other optimizations can be considered when concretely instantiating the model by exploiting several intricacies of the adversarial model at hand. We discuss some of these specific optimizations in Section 7.

2.3 Digital Signature Schemes

We start by introducing a formal definition of signature schemes in the standard model. A signature scheme is a tuple of algorithms $(\mathsf{SetUp}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, defined as follows (henceforth, pp will be implicitly given to all functionality):

$\mathsf{SetUp}(1^\lambda) \to \mathsf{pp}$**.** Given the security parameter as input 1^λ, the procedure outputs the public parameters pp.

$\mathsf{KeyGen}_{\mathsf{pp}}() \to (\mathsf{pk}, \mathsf{sk})$**.** The key generation procedure outputs a verification key pk and a private signing key sk.

$\mathsf{Sign}_{\mathsf{pp}}(\mathsf{pk}, \mathsf{sk}, \mathsf{msg}) \to \mathsf{Sig}$**.** The signing procedure takes as input the public and private keys, as well as a message to sign (msg), and it outputs a signature Sig.

$\mathsf{Verify}_{\mathsf{pp}}(\mathsf{pk}, \mathsf{msg}, \mathsf{Sig}) \to \{0/1\}$**.** The verification procedure takes as input a verification key pk, a message msg, and a signature Sig, and it outputs 1 if Sig is valid for msg under the verification key pk, and 0 otherwise.

We define threshold signatures as protocols that realize the different features of the signature: key generation, signing, and verification. In this work, we ensure the security of our protocols by proving that they UC-realize ideal functionalities that perform the different algorithms $(\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. This is a common way to study the soundness of distributed protocols [Doe+18; Cas+19; Ara+21]. In particular, we do not attempt to define threshold signatures in the UC framework as in previous works [ADN06; BKP13; BMP22; Mak22].

3 OV-based schemes: UOV and MAYO

In this section, we introduce OV-based schemes, and propose a unified description to represent UOV and MAYO as instances of it. OV-based schemes are are type of multivariate-based cryptography, which is based on the Polynomial System Solving (*PoSSo*) problem of solving systems of m multivariate non-linear polynomial equations in n variables over finite fields. The *PoSSo problem* is proven to be NP-hard even for the simplest case of quadratic equations over the field with two elements [GJ90] (in its decisional variant). When all polynomials are of the quadratic form, the *PoSSo problem* is called the *MQ-Problem*, and known attacks against the problem are exponential. The first attempt to build a cryptographic scheme based on multivariate quadratic polynomials was done by Ong, Schnorr and Shamir [OSS84], but its security is still based on the difficulty of factoring. Modern multivariate-based schemes are based on 1988's C^* scheme of Matsumoto and Imai [MI88] (broken in [Pat00]), which can be used both for encryption and signatures. More concretely, modern multivariate-based schemes are built upon the *Oil and Vinegar* (OV) scheme [Pat97], a simple and well-studied scheme. While broken in polynomial time by [KS98] for a specific set of parameters ($n = 2m$), a variant of the scheme, called the *Unbalanced Oil and Vinegar scheme* (UOV) has withstood all cryptanalysis since 1999 [KPG99].

Oil and Vinegar-based (OV-based) schemes rely on a trapdoor function constructed using multivariate quadratic equations. Specifically, the function $\mathcal{P} : \mathbb{F}_q^n \to \mathbb{F}_q^m$ consists of m homogeneous quadratic polynomials in n variables over a small finite field $\mathbb{F}_q$. The assumption underlying these schemes is that finding preimages (solving the system of quadratic equations) is computationally hard. However, if one possesses additional structural information, known as the trapdoor, it becomes efficient to find preimages for any output. The trapdoor, or secret key, in these schemes is a linear subspace $\mathcal{O} \subset \mathbb{F}_q^n$ of dimension o, such that for all vectors $\mathbf{o} \in O$, the function $\mathcal{P}$ vanishes: $\mathcal{P}(\mathbf{o}) = 0$. In the case of the Oil and Vinegar (OV) scheme, this subspace $\mathcal{O}$ is structured such that $o = m$, and the quadratic map $\mathcal{P}$ can be used to verify the validity of a signature for a message msg using the public key $\mathcal{P}$. This structural property of the secret subspace $\mathcal{O}$ enables the signer, who knows $\mathcal{O}$, to efficiently solve a system of m linear equations to generate valid signatures. In contrast, for anyone without knowledge of this trapdoor structure, generating such signatures remains infeasible. To mitigate known attacks, the *Unbalanced Oil and Vinegar* (UOV) scheme suggests using an unbalanced parameter set where $n > 2m$. This adjustment helps ensure the security of the scheme by increasing the complexity of solving the associated system of equations without the trapdoor.

3.1 Unbalanced Oil and Vinegar (UOV)

The UOV signature scheme (submitted to the "on-ramp" post-quantum NIST process [Beu+23b]) is a OV-based scheme, created from the described trapdoor function with the *Full Domain Hash* approach (specifically, the *Hash-and-Sign with Retry* approach): a signature on a message msg is simply sampled given

a target input $\mathbf{t}$ such that $\mathcal{P}(\mathbf{t}) = \mathcal{H}(\mathsf{msg}||\mathsf{salt})$, where $\mathcal{H}$ is as a cryptographic hash function that outputs elements in the range of $\mathcal{P}$, and salt is a fixed-length bit string chosen uniformly at random for every signature. As noted the linear subspace $\mathcal{O} \subset \mathbb{F}_q^n$ of dimension o ($o = m$) is the trapdoor. To generate the trapdoor function, one picks this subspace $\mathcal{O}$ uniformly at random and then picks $\mathcal{P}$ uniformly at random from the set of multivariate quadratic maps with m components in n variables that vanish on $\mathcal{O}$. Note that on top of the q^m artificial zeros in the subspace $\mathcal{O}$ (guaranteed by the structure of the trapdoor function), we expect roughly q^{n-m} natural zeros that do not lie in $\mathcal{O}$: since $\mathcal{P}$ consists of m equations in n variables, the total number of solutions to $\mathcal{P}(s) = 0$ is roughly q^{n-m}.

Following the description of [Beu20], concretely, when given a target $\mathbf{t}$, one picks a random vector $\mathbf{v} \in \mathbb{F}_q^n$ (a *vinegar* vector) and solves the system $\mathcal{P}(\mathbf{v}+\mathbf{o}) = \mathbf{t}$ for a vector $\mathbf{o} \in \mathcal{O}$ (the *oil* vector). For public quadratic maps $\mathcal{P}$, we can define its differential (or "polar form" [Beu21]) $\mathcal{P}'$ as $\mathcal{P}'(\mathbf{x}, \mathbf{y}) := \mathcal{P}(\mathbf{x}+\mathbf{y}) - \mathcal{P}(\mathbf{x}) - \mathcal{P}(\mathbf{y})$, which is a symmetric bilinear map in $\mathbf{x}$ and $\mathbf{y}$. One can then solve the linear system for $\mathbf{o}$, as:

$$\mathcal{P}(\mathbf{v}+\mathbf{o}) = \underbrace{\mathcal{P}'(\mathbf{v}, \mathbf{o})}_{\text{Linear in } \mathbf{o}} + \underbrace{\cancel{\mathcal{P}(\mathbf{o})}}_{=0} + \underbrace{\mathcal{P}(\mathbf{v})}_{\text{fixed}} = \mathbf{t}$$

(Rearranging gives:)

$$\mathcal{P}'(\mathbf{v}, \mathbf{o}) = \mathbf{t} - \mathcal{P}(\mathbf{v})$$

With success average probability of roughly $1 - 1/q$ over the choice of $\mathbf{v}$ the linear map $\mathcal{P}'(\mathbf{v}, \cdot)$ will be non-singular, which means that the linear system $\mathcal{P}(\mathbf{v}+\mathbf{o}) = \mathbf{t}$ has a unique solution. If it is not the case, one restarts for a new value of $\mathbf{v}$. It should be noted that there is no formal security proof of the scheme which concretely reduces it to a "hard" mathematical problem(s) (though, there has been efforts on this front [SSH11; CDP22; Cog+24; KX24]). Instead, the security analysis of UOV relies on looking at all the known attacks and analyzing how they influence its concrete hardness.

3.2 MAYO

MAYO is a OV-based signature scheme that has been submitted to the "on-ramp" NIST PQC standardization process [Beu+23a]. As it is OV-based, $\mathcal{P}$ has the same structure as in OV-based schemes with the exception that the dimension of the linear subspace $\mathcal{O}$ on which the trapdoor $\mathcal{P}$ evaluates to zero is "too small", i.e., $\dim(\mathcal{O}) = o$, with o less than m. As the oil subspace is hidden in $\mathbb{F}_q^n$, if its size is smaller, it follows that it is harder to search for it. This, in turn, means that other parameters of the scheme can be reduced without an impact, and that key sizes can be reduced. As in UOV, to generate the trapdoor function, one first picks the subspace $\mathcal{O}$ uniformly at random and then one picks $\mathcal{P}$ uniformly at random from the set of multivariate quadratic maps with m components in n

variables that vanish in $\mathcal{O}$. Reducing the dimension of $\mathcal{O}$ drastically shrinks the key sizes, but it also means that the signing algorithm will not work anymore as there are not enough degrees of freedom to always create a signature with high probability. To solve this problem, in MAYO, $\mathcal{P}$ is not used as is in the signature and verification procedures, and instead relies on a "whipping technique"[5]. Both the signer and verifier "whip-up" $\mathcal{P}$ into a k-fold larger map $\mathcal{P}^\star : \mathbb{F}_q^{kn} \to \mathbb{F}_q^m$, with m polynomials in k sets of n variables, where k is a fixed parameter of the scheme. This, in turn, means that if $\mathcal{P}$ vanishes on $\mathcal{O}$, then it follows that $\mathcal{P}^\star$ vanishes on $\mathcal{O}^k$. Formally, $\mathcal{P}^\star$ is defined as:

$$\mathcal{P}^\star(\mathbf{x}_1, \ldots, \mathbf{x}_k) := \sum_{i=1}^{k} \mathbf{E}_{ii}\mathcal{P}(\mathbf{x}_i) + \sum_{i=1}^{k}\sum_{j=i+1}^{k} \mathbf{E}_{ij}\mathcal{P}'(\mathbf{x}_i, \mathbf{x}_j)\,,$$

For all $i \in \{1, \ldots, k\}$ and all $j \in \{i{+}1, \ldots, k\}$, the matrix $\mathbf{E}_{ij} \in \mathbb{F}_q^{m\times m}$ is fixed and public. These matrices are chosen such that, under the correspondence between vectors in $\mathbb{F}_q^m$ and polynomials in $\mathbb{F}_q[X]$ of degree at most m, multiplication by $\mathbf{E}_{ij}$ corresponds to multiplication by powers of X modulo an irreducible polynomial $f(X) \in \mathbb{F}_q[X]$ of degree m. A MAYO signature, $\mathbf{S} = (\mathbf{s}_1, \ldots, \mathbf{s}_k) \in \mathbb{F}_q^{nk}$, is considered valid if $\mathcal{P}^\star(\mathbf{s}_1, \ldots, \mathbf{s}_k) = \mathcal{H}(\mathsf{msg})$, where $\mathcal{H}(\mathsf{msg})$ is the hash of a message.

To compute $\mathcal{P}^\star(\mathbf{S})$, the verifier (see algorithm 5) begins by calculating $\mathcal{P}(\mathbf{s}_i)$ and $\mathcal{P}'(\mathbf{s}_i, \mathbf{s}_j)$ for all $i \in \{1, \ldots, k\}$ and all $j \in \{i+1, \ldots, k\}$, and then combines them to obtain $\mathcal{P}^\star(\mathbf{S})$. Given that matrices $\mathbf{E}_{ij}$ represent multiplication by powers of $X \pmod{f(X)}$, the verifier can apply the appropriate powers of X to the polynomials for $\mathcal{P}(\mathbf{s}_i)$ and $\mathcal{P}'(\mathbf{s}_i, \mathbf{s}_j)$ and perform a single reduction modulo $f(X)$. Similarly, to sign a message (see algorithm 4), the signer partially evaluates $\mathcal{P}$ and $\mathcal{P}'$ on k vectors $(\mathbf{v}_1, \ldots, \mathbf{v}_k) \in \mathbb{F}_q^{n-o}$, and combines these results to determine the coefficients of a linear system, $\mathbf{A}\mathbf{x} = \mathbf{y}$, whose solution yields the signature. Just as in UOV, MAYO has a restart probability in case that the linear system does not have a solution. It is defined as a probability average-bounded by $\frac{q^{k-(n-o)}}{q-1} + \frac{q^{m-ko}}{q-1}$. As in UOV, the security analysis of MAYO mostly relies on looking at all the known attacks and analyzing how they influence its hardness.

3.3 OV-based Schemes Description

In this section, we present a unified representation for both UOV and MAYO (as OV-based schemes) that allows us to simultaneously thresholdize them. We will refer to them as OV-based henceforth.

Parameters. OV-based schemes are parameterized by the following:

- q the size of a finite field $\mathbb{F}_q$. In MAYO, in all current proposed parameter sets, $q = 16$; in UOV, it varies depending on the security level.

[5] Similar to the "whipping techniques" of the SNOVA algorithm [Wan+23; Beu24; Cab+24].

- m, the number of multivariate quadratic polynomials in the public key.
- n, the number of variables in the multivariate quadratic polynomials in the public key.
- o, the dimension of the oil subspace $\mathcal{O}$.
- k, the whipping parameter, satisfying $(k < n - o)$. This parameter is only present in MAYO.
- $\mathbf{E}_{i,j} \in \mathbb{F}_q^{m \times m}$ for $(0 \leq i \leq j < k)$, fixed public matrices.

These parameters cover both UOV and MAYO by adopting the following constraints:

- UOV is instantiated with $o = m$, $k = 1$, and $\mathbf{E}_{0,0} = \mathbf{I}_m$.
- MAYO fixes the public matrices $(\mathbf{E}_{i,j})_{i,j}$ such that

$$\begin{bmatrix} \mathbf{E}_{1,1} & \mathbf{E}_{1,2} & \dots & \mathbf{E}_{1,k} \\ \mathbf{E}_{1,2} & \mathbf{E}_{2,2} & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{E}_{1,k} & \mathbf{E}_{k-1,1} & \dots & \mathbf{E}_{k,k} \end{bmatrix}$$

  is non-singular. They correspond to multiplication by $z \mod f(Z)$

Description. In the following, we formally describe the algorithms that constitute an OV-based signature scheme. Contrary to the specifications of both MAYO and UOV, we generalize their description such that operations are only performed over matrices. Note that, for practicality in thresholdizing, we replace the private seed expansion for generating key material with direct sampling of values. Although this modifies the concrete execution of the algorithms from their specifications, it does not impact security, and signature verification remains unchanged. In practice, we can easily revert to seed sampling in the final protocol for the expansion of pseudo-random matrices in the public key.

Key Generation functionality. The functionality is described in Algorithm 1. In the following, we note some details of the functionality. To create the trapdoor, one chooses at random a sequence of m multivariate quadratic polynomials $(\mathcal{P})$ that vanish on the space spanned by rows of $\begin{pmatrix} \mathbf{O}^\mathsf{T} & \mathbf{I}_o \end{pmatrix}$, where $\mathbf{O}$ is sampled at random and $\mathbf{I}$ is the identity matrix. Each polynomial $p_i(\mathbf{x})_{i \in m}$ can be uniquely represented by an upper triangular matrix:

$$x^\top \begin{pmatrix} \mathbf{P}_i^{(1)} & \mathbf{P}_i^{(2)} \\ \mathbf{0} & \mathbf{P}_i^{(3)} \end{pmatrix} x$$

so that $p_i(\mathbf{x}) = \mathbf{x}^\top \mathbf{P}_i \mathbf{x}$. Each polynomial p_i vanishes in $\mathbf{O}$ if:

$$\mathbf{O}^\mathsf{T} \mathbf{P}_i^{(1)} \mathbf{O} + \mathbf{O}^\mathsf{T} \mathbf{P}_i^{(2)} + \mathbf{P}_i^{(3)}$$

is skew-symmetric. Hence, one picks $\mathbf{P}_i^{(1)} \in \mathbb{F}_q^{(n-o)\times(n-o)}$ and $\mathbf{P}_i^{(2)} \in \mathbb{F}_q^{(n-o)\times o}$ uniformly at random, and derives $\mathbf{P}_i^{(3)} = \mathsf{Upper}(-\mathbf{O}^\top \mathbf{P}_i^{(1)}\mathbf{O} - \mathbf{O}^\top \mathbf{P}_i^{(2)})$. As part of the key generation process, we also generate a value, denoted by $\mathbf{L}_i$, designed to improve signing efficiency. We'll delve into its purpose and functionality in the following section.

Algorithm 1 OV-based.KeyGen()

Output: A key pair $(\mathsf{pk}, \mathsf{sk})$.

1: //`Derive` $\mathbf{O}$ `and` $(\mathbf{P}^{(1)}, \mathbf{P}^{(2)})$ `randomly. Note that this differs from the specification, as we are not deriving them from a fixed seed.`
2: $\mathbf{O} \xleftarrow{\$} \mathbb{F}_q^{(n-o)\times o}$
3: $\{\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}\}_{i\in[m]} \xleftarrow{\$} (\mathbb{F}_q^{(n-o)\times(n-o)}, \mathbb{F}_q^{(n-o)\times o})$
4: //`Compute` $\mathbf{P}_i^{(3)} \in \mathbb{F}_q^{o\times o}$.
5: **for** i from 0 to $(m-1)$ **do**
6: $\quad \mathbf{P}_i^{(3)} \leftarrow \mathsf{Upper}(-\mathbf{O}^\mathsf{T}(\mathbf{P}_i^{(1)}\mathbf{O} - \mathbf{P}_i^{(2)}))$
7: $\quad \mathbf{L}_i \leftarrow ((\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)\mathsf{T}})\mathbf{O} + \mathbf{P}_i^{(2)})$
8: **return** $(\mathsf{pk} = (\{\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}, \mathbf{P}_i^{(3)}\}_{\{i\in[m]\}}), \mathsf{sk} = (\mathbf{O}, \{\mathbf{L}_i\}_{\{i\in[m]\}}))$.

Signing. The functionality is described in Algorithm 4. To sign a message, $\mathsf{msg} \in \{0,1\}^\star$, the signer first hashes msg together with a salt, $\mathsf{salt} \in \{0,1\}^{\mathsf{saltlen}}$ (whose purpose is to protect against side-channel and fault injection attacks). The result is a target vector $\mathbf{t} = \mathcal{H}(\mathsf{msg}||\mathsf{salt}) \in \mathbb{F}_q^m$. The signature is comprised of salt and a preimage $\mathbf{s} \in \mathbb{F}_q^{kn}$ for $\mathbf{t}$. To compute this preimage $\mathbf{s}$, the signer deterministically generates a "vinegar" vector $\mathbf{v} = \mathsf{Expand}_v(m||\mathsf{salt}||\mathbf{O}||\mathsf{ctr}) \in \mathbb{F}_q^{n-o}$, where ctr is a byte-sized counter, initialized at $\mathsf{0x0}$. Then, the signer solves a system of linear equations to find a vector $\mathbf{x} \in \mathbf{F}_q^{ko}$ so that $\mathbf{s} = (\mathbf{v}+\mathbf{O}\mathbf{x})||\mathbf{x} \in \mathbf{F}_q^{kn}$ is the preimage of $\mathbf{t}$. If the linear system is singular, ctr is incremented by one, and the signing starts again with the new $\mathbf{v} = \mathsf{Expand}_v(\mathsf{msg}||\mathsf{salt}||\mathbf{O}||\mathsf{ctr})$. It is noteworthy that the signer does not have to keep secret how many attempts were required to find a solution, and it is, thus, not an issue to leak that the attempt failed. After d failed attempts, the signer aborts: in honest executions, this happens with an extremely small probability ($\leq 2^{-786}$ for UOV, for example). For this functionality, we use the values $\mathbf{L}_i$, and, as this value is independent of msg, we pre-compute it once during key generation, and store it as part of the secret key.

Note that for the signing functionality, we call internally the functions: OV-based.Compute_A, OV-based.Compute_y and $\mathbf{x} \leftarrow \mathsf{SampleSolution}(\mathbf{A}, \mathbf{y})$. The first and second are algorithms to compute intermediate values, while the latter is used to find a solution to a system of linear equations of the form $\mathbf{A}\mathbf{x} = \mathbf{y}$ for an unknown $\mathbf{x}$. This can be implemented in two ways as noted by [Beu+23c; Beu+23d]: (i) directly computing the solution using constant-time Gaussian elimination, or (ii) first computing the right-inverse and multiplying it by the right side of the equation. The MAYO specification uses the first approach

with a random vector $\mathbf{r} \in \mathbb{F}_q^{ko}$ that is used to pick each of the q^{ko-m} solutions with equal probability. If the input matrix $\mathbf{A}$ does not have rank m, then SampleSolution$(\mathbf{A}, \mathbf{y}, \mathbf{r})$ outputs $\perp$.

Algorithm 2 OV-based.Compute_A$(\{\mathbf{M}_i\}_{i\in[m]})$

Input: Private matrices $\{\mathbf{M}_i\}_{i\in[m]}$, each of dimension $\mathbb{F}_q^{k\times o}$.
Output: Private matrix $\mathbf{A} \in \mathbb{F}_q^{m\times ko}$.

$\mathbf{A} \leftarrow 0_{m\times ko}, \ell \leftarrow 0$
$(\widehat{\mathbf{M}}_i)_{i\in[k]} = \mathbf{0}$ ▷ $\widehat{\mathbf{M}}_i$ is in $\mathbb{F}_q^{m\times o}$
for $t = 0$ to $k-1$ **do**
  for $j = 0$ to $m-1$ **do**
    $\widehat{\mathbf{M}}_t[j,:] = \mathbf{M}_j[t,:]$
for $t = 0$ to $k-1$ **do**
  for $j = t$ to $k-1$ **do**
    $\mathbf{A}[:, t*o : (t+1)*o] \mathrel{+}= \mathbf{E}^\ell \widehat{\mathbf{M}}_j$
    if $i \neq j$ **then**
      $\mathbf{A}[:, j*o : (j+1)*o] \mathrel{+}= \mathbf{E}^\ell \widehat{\mathbf{M}}_t$
    $\ell \leftarrow \ell + 1$
return $\mathbf{A}$

Algorithm 3 OV-based.Compute_y$(\{\mathbf{Y}_i\}_{i\in[m]})$

Input: Matrices $\{\mathbf{Y}_i\}_{i\in[m]}$, each of dimension $\mathbb{F}_q^{k\times k}$.
Output: Vector $\mathbf{y} \in \mathbb{F}_q^m$.

$\mathbf{y} \leftarrow 0_m \in \mathbb{F}_q^m$
$\ell \leftarrow 0$
for j from 0 to $k-1$ **do**
  $\mathbf{U}_j \leftarrow \mathbf{Y}_j$
  for t from $k-1$ to j **do**
    $\mathbf{u} = \begin{cases} [(\mathbf{Y}_a)_{j,j}]_{a\in[m]} & \text{if } j = t \\ [(\mathbf{Y}_a)_{j,t} + (\mathbf{Y}_a)_{t,j}]_{a\in[m]} & \text{if } j \neq t \end{cases}$
    $\mathbf{y} \leftarrow \mathbf{y} + \mathbf{E}^\ell \cdot \mathbf{u}$
    $\ell \leftarrow \ell + 1$
return $\mathbf{y}$

Verification. The functionality can be seen in Algorithm 5. The verifier accepts a signature $\mathbf{s} = \mathcal{H}(\mathsf{msg}||\mathsf{salt})$ by first recomputing $\mathbf{t} = \mathcal{H}(\mathsf{msg}||\mathsf{salt})$ and checking that $\mathcal{P}^\star(\mathbf{s}_i) = \mathbf{t}$.

Algorithm 4 OV-based.Sign(sk, pk, msg)

Input: Secret key (sk), public key (pk).
Input: Message msg.
Output: Signature $\mathsf{Sig} = (\mathbf{S}, \mathsf{salt})$.

1: //Parse $(\mathbf{O}, \{\mathbf{L}_i\}_{\{i\in[m]\}})$ and $(\{\mathbf{P}_i^{(1)}\}_{\{i\in[m]\}}, \{\mathbf{P}_i^{(2)}\}_{\{i\in[m]\}})$ from sk and pk.
2: $(\mathbf{O}, \{\mathbf{L}\}_{\{i\in[m]\}}) \leftarrow \mathsf{sk}$
3: $\{\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}\}_{i\in[m]} \leftarrow \mathsf{pk}$
4: //Hash salted message.
5: $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{\lambda+64}$
6: $\mathbf{t} \leftarrow \mathcal{H}(\mathsf{msg}\|\mathsf{salt})$ ▷ $\mathbf{t} \in \mathbb{F}_q^m$
7:
8: $\mathbf{V} \xleftarrow{\$} \mathbb{F}_q^{k\times(n-o)}$
9: **for** i from 1 to m **do**
10: $\quad \mathbf{M}_i \leftarrow \mathbf{V} \cdot \mathbf{L}_i$ ▷ $\mathbf{M}_i \in \mathbb{F}_q^{k\times o}$
11: $\quad \mathbf{Y}_i \leftarrow \mathbf{V} \cdot \mathbf{P}_i^{(1)} \cdot \mathbf{V}^\mathsf{T}$ ▷ $\mathbf{Y}_i \in \mathbb{F}_q^{k\times k}$
12: //Build the linear system $\mathbf{Ax} = \mathbf{y}$.
13: $\mathbf{A} \leftarrow$ OV-based.Compute_A$(\{\mathbf{M}_i\}_{i\in[m]})$
14: $\mathbf{y} \leftarrow \mathbf{t} -$ OV-based.Compute_y$(\{\mathbf{Y}_i\}_{i\in[m]})$
15:
16: //Try to sample a random solution $\mathbf{x}$ to $\mathbf{Ax} = \mathbf{y}$.
17: $\mathbf{x} \leftarrow$ SampleSolution$(\mathbf{A}, \mathbf{y})$ ▷ $\mathbf{x} \in \mathbb{F}_q^{ko} \cup \{\perp\}$
18: **if** $\mathbf{x} = \perp$ **then**
19: $\quad$ **go to** 7
20: //Output the signature.
21: $\mathbf{X} \leftarrow$ Matrixify$(\mathbf{x})$ ▷ $\mathbf{X} \in \mathbf{F}_q^{k\times o}$, s.t. $\mathbf{x}$ is concatenation of rows of $\mathbf{X}$
22: $\mathbf{S} \leftarrow (\mathbf{V} + (\mathbf{OX}^\mathsf{T})^\mathsf{T}, \mathbf{X})$ ▷ $\mathbf{S} \in \mathbb{F}_q^{k\times n}$
23: **return** $\mathsf{Sig} = (\mathbf{S}, \mathsf{salt})$.

Algorithm 5 OV-based.Verify(pk, msg, Sig)

Input: Public key (pk).
Input: Message msg.
Input: Signature $\mathsf{Sig} = (\mathbf{S}, \mathsf{salt})$.
Output: A boolean indicating if the signature is valid.

1: //Parse $(\{\mathbf{P}_i^{(1)}\}_{\{i\in[m]\}}, \{\mathbf{P}_i^{(2)}\}_{\{i\in[m]\}}, \{\mathbf{P}_i^{(3)}\}_{\{i\in[m]\}})$ from pk.
2: $\{\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}, \mathbf{P}_i^{3}\}_{i\in[m]} \leftarrow \mathsf{pk}$
3: //Hash salted message.
4: $\mathbf{t} \leftarrow \mathcal{H}(\mathsf{msg}\|\mathsf{salt})$ ▷ $\mathbf{t} \in \mathbf{F}_q^m$
5: **for** i from 1 to m **do**
6: $\quad \mathbf{Y}_i \leftarrow \mathbf{S} \begin{pmatrix} \mathbf{P}_i^{(1)} & \mathbf{P}_i^{(2)} \\ \mathbf{0} & \mathbf{P}_i^{(3)} \end{pmatrix} \mathbf{S}^\mathsf{T}$
7: $\mathbf{y} \leftarrow \mathsf{Compute\_y}(\{\mathbf{Y}_i\}_{i\in[m]})$ ▷ $\mathbf{y} = \mathcal{P}^*(\mathbf{s})$
8: **return** $\mathbf{y} == \mathbf{t}$ ▷ Accept signature if $\mathbf{y} = \mathbf{t}$.

4 Solving Systems of Linear Equations Obliviously

To enable our threshold constructions, we require a core operation: solving a system of linear equations in which both the input matrix and the target vector remain secret (hence, solving in an oblivious manner). This operation, which is the SampleSolution algorithm (which is used in Line 17 of Algorithm 4), must sample a solution uniformly at random from the set of all possible solutions.

A simple protocol for this task was proposed by Cozzo and Smart [CS19] (which we refer to as "One-Sided Masking") in the case where the matrix defining the system is *square*. This simple protocol outputs a solution or $\perp$: the latter occurs in the case that the input matrix has a determinant zero (hence, the matrix is singular) or the "one-side" masking has a determinant zero (which happens with probability $1/q$). However, as we discuss in Appendix B, their protocol leaks sensitive information about the input matrix structure if it is rank-deficient, disclosing far more than mere non-invertibility.

In this work, we address this limitation by proposing a novel approach that, first, works for general non-square matrices, and, secondly, (and crucial for the security of our threshold schemes), reveals only the rank of the matrix in cases of rank deficiency: this significantly reduces the leakage compared to [CS19].

In Section 5.5, we provide explanations as to why this reduced leakage appears secure for our specific threshold setting. Still, we leave as an open problem the design of a protocol for linear system solving that, in cases of rank deficiency, reveals only that the matrix is not full rank without additional leakage. In the following, we begin by defining the target functionality in Section 4.1, and, in Section 4.2, we follow with the presentation of our concrete protocol.

4.1 Functionality for Solving Systems of Linear Equations in the ABB Model

Recall that our work is set in the ABB model to ensure general applicability. Consequently, we model the task of sampling solutions to linear systems of equations as an extended functionality, denoted $\mathcal{F}_{\mathsf{ABB+Solve}}$, which augments the ABB model by adding one extra instruction that, precisely, samples solutions to systems defined by matrix and target vectors that are *stored in the ABB's dictionary* (representing values that are "secret-shared"). To ensure completeness, we provide a full, detailed description of the functionality, including necessary specifics regarding indexes of the dictionary. However, as mentioned in Section 2.2, afterward (and in particular for our protocol) we simplify the notation substantially by working at the level of matrices instead of individual values, and omitting indexes where possible, among other notational simplifications.

The functionality $\mathcal{F}_{\mathsf{ABB+Solve}}$ takes as a parameter a distribution $L(r)$ which models leakage as a function of the rank r of $\mathbf{A}$. In the concrete protocol, when $\mathbf{A}$ is rank-deficient, the difference $r - r^+$ is revealed (where $r^+ \leftarrow L(r)$) rather than r itself, which potentially provides an extra layer of security.

Functionality 2: $\mathcal{F}_{\mathsf{ABB+Solve}}(L)$

The functionality has the *exact same* operations as $\mathcal{F}_{\mathsf{ABB}}$, with the addition of: On input $(\mathsf{solve}, \{\mathtt{id}_{ij}\}_{i,j=1,1}^{s,t}, \{\mathtt{id}_i\}_{i=1}^{s}, \{\mathtt{id}'_j\}_{j=1}^{t})$ from the parties, where $s \leq t$,

the functionality fetches $\{(a_{ij}, \mathtt{id}_{ij})\}_{i,j=1,1}^{s,t}$ and $\{(b_i, \mathtt{id}_i)\}_{i=1}^{s}$ and proceeds as follows:

1. Let $\mathbf{A} \in \mathbb{F}_q^{s\times t}$: the (i,j) entry of $\mathbf{A}$ is a_{ij}. Let $\boldsymbol{b} \in \mathbb{F}_q^s$ so that $\boldsymbol{b}[i] = b_i$. Let $r = \mathsf{rank}(\mathbf{A})$.
2. Sample $r^+ \leftarrow L(r)$. If $r - r^+ < s$, then send $(\mathsf{rank\text{-}defect}, r - r^+)$ to all parties.
3. Else, sample a uniformly random element $\mathbf{x} \in \mathbb{F}_q^t$ constrained to $\mathbf{A}\cdot\mathbf{x} = \mathbf{b}$, and store $(\mathbf{x}[j], \mathtt{id}'_j)$ for $j \in [t]$.

4.2 Concrete Protocol for Solving Systems of Linear Equations

Our protocol to instantiate $\mathcal{F}_{\mathsf{ABB+Solve}}$ is given as Protocol Π_{Solve} below. Let us recall that we write $[\![\mathbf{A}\cdot\mathbf{B}]\!] \leftarrow [\![\mathbf{A}]\!]\cdot[\![\mathbf{B}]\!]$ for when parties use the command mult in $\mathcal{F}_{\mathsf{ABB}}$, and similarly we write $[\![\mathbf{A}+\mathbf{B}]\!] \leftarrow [\![\mathbf{A}]\!] + [\![\mathbf{B}]\!]$ for when they use the command add. Importantly, we remark that, in an actual instantiation of $\mathcal{F}_{\mathsf{ABB}}$, additions will come *for free* in terms of communication costs while multiplications require some form of interaction. To emphasize this, in the functionality, we write "parties compute locally" for additions and other local operations, even though in the ABB model there is no such thing as "local computation".

Protocol 1: Π_{Solve}

The protocol is set in the $\mathcal{F}_{\mathsf{ABB}}$-hybrid. All the commands except for (solve) are forwarded directly to $\mathcal{F}_{\mathsf{ABB}}$.

On input $(\mathsf{solve}, [\![\mathbf{A}]\!], [\![\mathbf{b}]\!])$, where $\mathbf{A}$ has dimensions $s\times t$ and $[\![\mathbf{b}]\!]$ has dimension s, the parties proceed as follows:

1. Parties call $[\![\mathbf{R}]\!] \leftarrow \mathsf{rand}(\mathbb{F}_q^{s\times s})$ and $[\![\mathbf{S}]\!] \leftarrow \mathsf{rand}(\mathbb{F}_q^{t\times t})$.
2. Parties call $[\![\mathbf{A}\cdot\mathbf{S}]\!] \leftarrow [\![\mathbf{A}]\!]\cdot[\![\mathbf{S}]\!]$.
3. Parties call $[\![\mathbf{T}]\!] \leftarrow [\![\mathbf{R}]\!]\cdot[\![\mathbf{A}\cdot\mathbf{S}]\!]$.
4. Parties open $\mathbf{T} \leftarrow [\![\mathbf{T}]\!]$. If $r = \mathsf{rank}(\mathbf{T}) < s$ then the parties output $(\mathsf{rank\text{-}defect}, r)$.
5. Otherwise, let $\mathbf{T}^{-1} \in \mathbb{F}_q^{t\times s}$ be a right inverse of $\mathbf{T}$, that is, $\mathbf{T}\mathbf{T}^{-1} = \mathbf{I}_{s\times s}$. The parties call $[\![\mathbf{A}^{-1}]\!] \leftarrow [\![\mathbf{S}]\!]\cdot\mathbf{T}^{-1}\cdot[\![\mathbf{R}]\!]$. It can be checked that $\mathbf{A}^{-1} \in \mathbb{F}_q^{t\times s}$ satisfies $\mathbf{A}\cdot\mathbf{A}^{-1} = \mathbf{I}_{s\times s}$.
6. Let $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_{t-s} \in \mathbb{F}_q^{t-s}$ be a basis for $\ker(\mathbf{T})$. The parties call $([\![z_1]\!], \ldots, [\![z_{t-s}]\!]) \leftarrow \mathsf{rand}(\mathbb{F}_q^{t-s})$.
7. Parties compute locally $[\![\mathbf{z}]\!] \leftarrow \sum_{i=1}^{t-s} [\![z_i]\!]\cdot\boldsymbol{\beta}_i$.
8. Parties call $[\![\mathbf{x}]\!] \leftarrow [\![\mathbf{A}^{-1}]\!]\cdot[\![\mathbf{b}]\!] + [\![\mathbf{S}]\!]\cdot[\![\mathbf{z}]\!]$
9. Output $[\![\mathbf{x}]\!]$.

Theorem 1. *Protocol Π_{Solve} instantiates functionality $\mathcal{F}_{\mathsf{ABB+Solve}}(L)$ in the $\mathcal{F}_{\mathsf{ABB}}$-hybrid model, where $L(r)$ is such that:*

$$L(r) = \left\{ r - \mathsf{rank}\left(\mathbf{R} \cdot \begin{bmatrix} \mathbf{I}_r & \\ \mathbf{0}_{(s-r)\times r} & \mathbf{0}_{s\times(t-r)} \end{bmatrix} \cdot \mathbf{S} \right) \mid \mathbf{R} \leftarrow \mathbb{F}_q^{s\times s}, \mathbf{S} \leftarrow \mathbb{F}_q^{t\times t} \right\}$$

Proof. To prove Theorem 1, we describe a simulator $\mathcal{S}$ that interacts with the ideal functionality $\mathcal{F}_{\mathsf{ABB+Solve}}(L)$ and with the adversary, emulating internally the arithmetic and coin-sampling functionalities, and emulating virtual honest parties.

Initially, $\mathcal{S}$ calls $\mathcal{F}_{\mathsf{ABB+Solve}}(L)$, receiving either (as output) some sharings $[\![\mathbf{w}]\!]$ of a solution,[6] or $(\mathsf{rankdef}, r - r^+)$ with $r - r^+ < s$ and $r^+ \leftarrow L(r)$. $\mathcal{S}$ proceeds by emulating rand in Item 1 by distributing random shares: this is possible because the adversary's shares are independent of the underlying secret. $\mathcal{S}$ also emulates the shared multiplications in Item 2 and Item 3. To match the real-world distribution in Item 4, $\mathcal{S}$ samples invertible matrices $\mathbf{U}' \leftarrow \mathbb{F}_q^{s\times s}$ and $\mathbf{V}' \leftarrow \mathbb{F}_q^{t\times t}$, adjusting the shares of the honest parties so that the reconstruction yields $\mathbf{T}' = \mathbf{U}' \cdot \mathbf{J} \cdot \mathbf{V}'$, where $\mathbf{J}$ is a matrix with $r - r^+$ ones in its diagonal and zeros elsewhere. We claim this has the same distribution as the real world, where parties reconstruct $\mathbf{T} = \mathbf{R} \cdot \mathbf{A} \cdot \mathbf{S}$ with $\mathbf{R} \in \mathbb{F}_q^{s\times s}$ and $\mathbf{S} \in \mathbb{F}_q^{t\times t}$ as uniform matrices that are unknown to the adversary, conditioned on the event $\mathsf{rank}(\mathbf{T}) = r - r^+$. To see this, as an intermediate step, consider the distribution of matrices $\mathbf{T}'' = \mathbf{X} \cdot (\mathbf{R} \cdot \mathbf{A} \cdot \mathbf{S}) \cdot \mathbf{Y}$, with $(\mathbf{X} \in \mathbb{F}_q^{s\times s}, \mathbf{Y} \in \mathbb{F}_q^{t\times t})$ as random invertible matrices and $(\mathbf{R} \in \mathbb{F}_q^{s\times s}, \mathbf{S} \in \mathbb{F}_q^{t\times t})$ as uniformly distributed matrices, conditioned on $\mathsf{rank}(\mathbf{T}'') = r - r^+$. We can easily see that $\mathbf{X} \cdot \mathbf{R}$ and $\mathbf{S} \cdot \mathbf{Y}$ are uniformly distributed since $\mathbf{X}$ and $\mathbf{Y}$ are invertible. Hence, $\mathbf{T}''$ has the same distribution as $\mathbf{T}$.

Now, let us study in more detail the distribution of $\mathbf{T}''$: remember that we condition on $\mathsf{rank}(\mathbf{T}'') = r - r^+$, and we observe that $\mathsf{rank}(\mathbf{T}'') = \mathsf{rank}(\mathbf{R} \cdot \mathbf{A} \cdot \mathbf{S})$ as $(\mathbf{X}, \mathbf{Y})$ are invertible matrices. We can, hence, write $\mathbf{R} \cdot \mathbf{A} \cdot \mathbf{S} = \mathbf{U} \cdot \mathbf{J} \cdot \mathbf{V}$ for some invertible matrices $(\mathbf{U}, \mathbf{V})$. Thus, it holds that $\mathbf{T}' = \mathbf{R}' \cdot (\mathbf{R} \cdot \mathbf{A} \cdot \mathbf{S}) \cdot \mathbf{S}'$ with $\mathbf{R}' := \mathbf{U}'\mathbf{U}^{-1}$ and $\mathbf{S}' := \mathbf{V}^{-1}\mathbf{V}'$. Since $\mathbf{U}$ and $\mathbf{V}$ are invertible, and $\mathbf{U}'$ and $\mathbf{V}'$ are uniform and invertible matrices unknown to the adversary, it follows that $\mathbf{R}'$ and $\mathbf{S}'$ are also uniform and invertible matrices that are unknown to the adversary. Therefore, $\mathbf{T}'$ follows the same distribution as $\mathbf{T}''$ conditioned on the value of its rank, and consequently holds the same distribution as $\mathbf{T}$. We additionally note that the distribution of $r - r^+$ follows exactly the distribution of ranks of $\mathbf{T}$ by choice of $L(r)$.

In case $\mathsf{rank}(\mathbf{T}') < s$, then $\mathcal{S}$ returns its rank as leakage, as in the real world. Now, in the case in which $\mathsf{rank}(\mathbf{T}') = s$ and a solution $[\![\mathbf{w}]\!]$ was distributed by $\mathcal{F}_{\mathsf{ABB+Solve}}(L)$, we proceed as following Let $\mathbf{x}$ be the output in the real world,

[6] We use $\mathbf{w}$ instead of $\mathbf{x}$ to distinguish between the output in the ideal and real worlds.

and note that:

$$\begin{aligned}\mathbf{A}\cdot\mathbf{x} &= \mathbf{A}\cdot(\mathbf{A}^{-1}\mathbf{b}+\mathbf{S}\mathbf{z})\\ &= \mathbf{A}\cdot(\mathbf{S}\mathbf{T}^{-1}\mathbf{R}\mathbf{b})+\mathbf{A}\mathbf{S}\mathbf{z}\\ &= \mathbf{R}^{-1}\cdot\left(\mathbf{R}\mathbf{A}\mathbf{S}\cdot\mathbf{T}^{-1}\cdot\mathbf{R}\mathbf{b}+\mathbf{R}\mathbf{A}\mathbf{S}\cdot\mathbf{z}\right)\\ &= \mathbf{R}^{-1}\cdot\left(\mathbf{T}\cdot\mathbf{T}^{-1}\cdot\mathbf{R}\mathbf{b}+\mathbf{T}\cdot\mathbf{z}\right)\\ &= \mathbf{R}^{-1}\cdot(\mathbf{I}_{s\times s}\cdot\mathbf{R}\mathbf{b}+\mathbf{0})\\ &= \mathbf{b}.\end{aligned}$$

Hence, the output $\mathbf{x}$ produced by Π_{Solve} is indeed a valid solution to the system. Next, we need to establish that $\mathbf{x}$ is uniformly distributed. For this, note that $\mathbf{z}$ is a uniformly random element—and unknown to the adversary—in $\mathsf{ker}(\mathbf{T})$ and hence $\mathbf{S}\cdot\mathbf{z}$ is a uniformly random element in $\mathsf{ker}(\mathbf{R}\cdot\mathbf{A}) = \mathsf{ker}(\mathbf{A})$. This implies that $\mathbf{A}^{-1}\cdot\mathbf{b}+\mathbf{S}\cdot\mathbf{z}$ is a uniformly random element that maps to $\mathbf{b}$ under $\mathbf{A}$, as required.

□

5 Threshold OV-based: UOV and MAYO Signatures

Now we describe our main contribution, which consists of threshold variants of OV-based signatures in the ABB model, as specified by Functionality $\mathcal{F}_{\mathsf{ABB+Solve}}$ in page 16. Recall that this functionality extends the standard ABB model ($\mathcal{F}_{\mathsf{ABB}}$) from Section 2.2 by enabling sampling solutions to "secret-shared" systems of equations: an essential component in the OV-based signing algorithm. We begin in Section 5.2 by describing our approach to Distributed Key Generation (DKG), a straightforward process due to the inherent "arithmetic-friendliness" of these signature schemes Next, in Section 5.3, we introduce our threshold signing protocols that generate signatures for a given public message using a "secret-shared" private key. Before detailing these constructions, we provide an overview of the functionality we aim to implement in Section 5.1.

5.1 Threshold OV-based Functionality

The functionality $\mathcal{F}_{\mathsf{ThrSign}}$ described below models the threshold version of OV-based signatures. In an initial "setup phase", the functionality samples a secret key, stores it internally, and announces the corresponding public key. Following this, any public message can be provided jointly by the parties and the functionality will compute a signature for it, returning it to all parties.

As discussed in Section 3, the matrix $\mathbf{A}$ may be rank deficient with non-negligible probability. In a local signature computation, a signer can simply sample a new $\mathbf{V}$ and try again (in Line 7 of Algorithm 4). However, in an MPC context, the runtime of the protocol must be public and, hence, as a minimum, we leak the number of attempts until a full rank $\mathbf{A}$ is obtained and a solution can be found. Our protocol, nevertheless, leaks slightly more than just the number of

attempts: for each failed attempt, we leak a random variable depending on the *rank* of the corresponding rank-deficient matrix $\mathbf{A}$. This behavior is captured by the set Leaks and by distributions $L(r)$ for $0 \leq r \leq s$. We provide a deeper discussion on this matter in Section 5.5.

Functionality 3: $\mathcal{F}_{\mathsf{ThrSign}}(L)$

Setup phase: On input (sample-key) from all the parties for the first time (future calls to this command are ignored), the functionality proceeds as follows:

1. Run $(\mathsf{pk}, \mathsf{sk}) \leftarrow$ OV-based.KeyGen().
2. Internally store the secret key $\mathsf{sk} = (\mathbf{O}, (\mathbf{L}_i)_{i\in[m]})$.
3. Output to all parties the public key $\mathsf{pk} = (\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}, \mathbf{P}_i^{(3)})_{i\in[m]}$.

Signing: On input $(\mathsf{sign}, \mathsf{msg})$, where $\mathsf{msg} \in \{0,1\}^*$ is the message to be signed, the functionality proceeds as follows:

1. Sample $\mathsf{salt} \xleftarrow{\$} \{0,1\}^{\lambda+64}$ and let $\mathbf{t} \leftarrow \mathcal{H}(\mathsf{msg}\|\mathsf{salt}) \in \mathbb{F}_q^m$.
2. Initialize $\mathsf{Leaks} = [\quad]$.
3. Sample a matrix $\mathbf{V} \leftarrow \mathbb{F}_q^{k\times(n-o)}$.
4. For $i \in [m]$: Compute $\mathbf{M}_i = \mathbf{V} \cdot \mathbf{L}_i$, and $\mathbf{Y}_i = \mathbf{V} \cdot \mathbf{P}_i^{(1)} \cdot \mathbf{V}^\intercal$.
5. Compute $\mathbf{A} \leftarrow$ OV-based.Compute_A$(\{\mathbf{M}_i\}_{i\in[m]})$ and $\mathbf{y} \leftarrow \mathbf{t} -$ OV-based.Compute_y$(\{\mathbf{Y}_i\}_{i\in[m]})$. Let $r = \mathsf{rank}(\mathbf{A})$ and $r^+ \leftarrow L(r)$.
6. If $r - r^+ < m$, append $r - r^+$ to Leaks and **go to** 3.
7. Sample $\mathbf{x} \in \mathbb{F}_q^{ko}$ uniformly at random constrained to $\mathbf{A}\mathbf{x} = \mathbf{y}$ (by calling SampleSolution).
8. Compute the signature $\mathbf{S} \leftarrow (\mathbf{V} + (\mathbf{O}\mathbf{X}^\intercal)^\intercal, \mathbf{X})$, where $\mathbf{X} \leftarrow \mathsf{Matrixify}(\mathbf{x})$, and output $((\mathbf{S}, \mathsf{salt}), \mathsf{Leaks})$ to all parties.

5.2 Procedure for Distributed Key Generation

In certain threshold applications, like key management via MPC (cf. [Lin20]), the secret key is held by a single party. This party utilizes a set of additional parties for threshold signing, effectively safeguarding the key similarly to the functionality of a hardware security module[7]. However, several other applications, like the ones we discuss in Section 8, assume that the full view of the secret key is unknown to any single party, for which it is paramount to execute a *Distributed Key Generation* (DKG) protocol to produce "shares" of the secret key while only leaking the corresponding public key. In the classical setting, there are some DKGs for Discrete-Logarithm (DLOG)-based threshold signatures though few of them arrive at a round-optimal [Kat23], fully-secure [Wik05; Gen+07] solution. In the classical setting, a simple solution sees the parties generate random shares of a secret key $\langle s \rangle$, locally use those as exponents to compute $\langle g^s \rangle$, and

[7] We note that our construction trivially allows for these use cases, by letting the owner of the secret key call the input command in $\mathcal{F}_{\mathsf{ABB+Solve}}$ (which in practice corresponds to distributing secret-shares of the secret key under certain secret-sharing schemes).

reconstruct g^s. However, for a post-quantum setting, like lattice-based signatures schemes, there is no simple solution: parties must sample obliviously from Gaussian distributions, for example, which is a much harder problem [ENP24].

In this section, we show that, fortunately, the structure of OV-based public keys allows for very simple and efficient DKG protocols. We describe our DKG for OV-based schemes in Procedure π_{KeyGen} below.

Procedure 2: π_{KeyGen}

Input: No inputs. The procedure is set in the $\mathcal{F}_{\mathsf{ABB+Solve}}$-hybrid model.

Output: A public key $\mathsf{pk} = (\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}, \mathbf{P}_i^{(3)})_{i\in[m]}$, and a secret-shared secret key $([\![\mathbf{O}]\!], [\![\mathbf{L}_1]\!], \ldots, [\![\mathbf{L}_m]\!])$, where $\mathbf{O} \in \mathbb{F}_q^{(n-o)\times o}$ and $\mathbf{L}_i \in \mathbb{F}_q^{(n-o)\times o}$.

1. Parties call $[\![\mathbf{O}]\!] \leftarrow \mathsf{rand}(\mathbb{F}_q^{(n-o)\times o})$.
2. For $i \in [m]$, parties call $\mathbf{P}_i^{(1)} \leftarrow \mathsf{coin}(\mathbb{F}_q^{(n-o)\times(n-o)})$ and $\mathbf{P}_i^{(2)} \leftarrow \mathsf{coin}(\mathbb{F}_q^{(n-o)\times o})$.
3. For $i \in [m]$, parties locally compute $[\![\mathbf{P}_i^{(1)} \cdot \mathbf{O}]\!] \leftarrow \mathbf{P}_i^{(1)} \cdot [\![\mathbf{O}]\!]$.
4. For $i \in [m]$, parties call $[\![\mathbf{O}^\mathsf{T} \cdot (\mathbf{P}_i^{(1)} \cdot \mathbf{O} - \mathbf{P}_i^{(2)})]\!] \leftarrow [\![\mathbf{O}^\mathsf{T}]\!] \cdot ([\![\mathbf{P}_i^{(1)} \cdot \mathbf{O}]\!] - \mathbf{P}_i^{(2)})$.
5. Parties compute locally $[\![\mathbf{P}_i^{(3)}]\!] \leftarrow \mathsf{Upper}([\![-\mathbf{O}^\mathsf{T}(\mathbf{P}_i^{(1)}\mathbf{O} - \mathbf{P}_i^{(2)})]\!])$.
6. Parties reveal $\mathbf{P}_i^{(3)}$.
7. For $i \in [m]$, parties compute locally $[\![(\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)\mathsf{T}})\mathbf{O}]\!] \leftarrow (\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)\mathsf{T}}) \cdot [\![\mathbf{O}]\!]$.
8. Parties compute locally $[\![\mathbf{L}_i]\!] \leftarrow [\![(\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)\mathsf{T}})\mathbf{O}]\!] + \mathbf{P}_i^{(2)}$ for $i \in [m]$.
9. All parties output $(\mathsf{pk} = (\{\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}, \mathbf{P}_i^{(3)}\}_{\{i\in[m]\}})$ as the public key, and they store $([\![\mathbf{O}]\!], \{[\![\mathbf{L}_i]\!]\}_{\{i\in[m]\}}))$ as the "shares" of the secret key.

5.3 Procedure for Threshold Signing

We are now concerned with instantiating the threshold signing procedure of Functionality $\mathcal{F}_{\mathsf{ThrSign}}$. Consider a secret key $\mathsf{sk} = (\mathbf{O}, \{\mathbf{L}_i\}_{i\in[m]})$, where $\mathbf{O} \in \mathbb{F}_q^{(n-o)\times o}$ and $\mathbf{L}_i \in \mathbb{F}_q^{(n-o)\times o}$, and assume it is stored in the ABB as $([\![\mathbf{O}]\!], [\![\mathbf{L}_1]\!], \ldots, [\![\mathbf{L}_m]\!])$. The procedure to securely compute a threshold signature on a public message msg is given below as Procedure π_{Sign}.

Procedure 3: π_{Sign}

Input: Public key and secret key stored in $\mathcal{F}_{\mathsf{ABB+Solve}}$: $\mathsf{pk} = (\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}, \mathbf{P}_i^{(3)})_{i\in[m]}, \mathsf{sk} = ([\![\mathbf{O}]\!], [\![\mathbf{L}_1]\!], \ldots, [\![\mathbf{L}_m]\!])$. A message msg to be signed.

Output: A signature $\mathbf{S} \in \mathbb{F}_q^{k\times n}$ on msg.

1. Parties call $\mathsf{salt} \leftarrow \mathsf{coin}(\{0,1\}^{\lambda+64})$ and let $\mathbf{t} \leftarrow \mathcal{H}(\mathsf{msg}\|\mathsf{salt}) \in \mathbb{F}_q^m$. Let $\mathsf{Leaks} = [\ \ ]$.
2. Parties call $[\![\mathbf{V}]\!] \leftarrow \mathsf{rand}(\mathbb{F}_q^{k\times(n-o)})$
3. For $i \in [m]$: Parties call $[\![\mathbf{M}_i]\!] \leftarrow [\![\mathbf{V}]\!] \cdot [\![\mathbf{L}_i]\!]$.

4. For $i \in [m]$: Parties call $[\![\mathbf{Y}_i]\!] \leftarrow [\![\mathbf{V}]\!] \cdot \mathbf{P}_i^{(1)} \cdot [\![\mathbf{V}^\intercal]\!]$.
5. Compute locally $[\![\mathbf{A}]\!] \leftarrow$ OV-based.Compute_A$(\{[\![\mathbf{M}_i]\!]\}_{i\in[m]})$ and $[\![\mathbf{y}]\!] \leftarrow [\![\mathbf{t}]\!] -$ OV-based.Compute_y$(\{[\![\mathbf{Y}_i]\!]\}_{i\in[m]})$. This is possible since both OV-based.Compute_y and OV-based.Compute_A are *linear* functions of their arguments.
6. Parties call the command solve of $\mathcal{F}_{\mathsf{ABB+Solve}}$ on inputs $[\![\mathbf{A}]\!]$ and $[\![\mathbf{y}]\!]$. If the output is $(\mathsf{rank\text{-}deficient}, r)$, parties append r to Leaks and **go to** 2. Else, let $[\![\mathbf{x}]\!]$ be the output.
7. Parties compute locally $[\![\mathbf{X}]\!] \leftarrow \mathsf{Matrixify}([\![\mathbf{x}]\!])$ and call $[\![\mathbf{X} \cdot \mathbf{O}^\intercal]\!] \leftarrow [\![\mathbf{X}]\!] \cdot [\![\mathbf{O}^\intercal]\!]$.
8. Parties compute locally $[\![\mathbf{S}']\!] \leftarrow [\![\mathbf{V} + (\mathbf{O}\mathbf{X}^\intercal)^\intercal]\!]$, and they open $\mathbf{S}' \leftarrow [\![\mathbf{S}']\!]$.
9. Parties return as output $(\mathsf{Sig} = (\mathbf{S}, \mathsf{salt}), \mathsf{Leaks})$, where $\mathbf{S} = (\mathbf{S}' \mid \mathbf{X}) \in \mathbb{F}_q^{k\times n}$.

5.4 Protocol for Instantiating $\mathcal{F}_{\mathsf{ThrSign}}$

Finally, we compose the procedures π_{KeyGen} and π_{Sign} to instantiate the Functionality $\mathcal{F}_{\mathsf{ThrSign}}$. This is detailed in the protocol Π_{ThrSign} below, along with the corresponding simulation-based proof in Theorem 2.

Protocol 4: Π_{ThrSign}

The protocol is set in the $\mathcal{F}_{\mathsf{ABB+Solve}}$-hybrid model.

Setup phase: On input (sample-key), the parties execute π_{KeyGen}. They obtain a stored secret key $[\![\mathsf{sk}]\!]$ and a corresponding public key pk.

Signing: On input (sign, msg) and, if the setup was previously performed, parties call π_{Sign} on input the key pairs and msg, which results in $(\mathsf{Sig} = (\mathbf{S}, \mathsf{salt}), \mathsf{Leaks})$. The parties return this result as the output of Π_{ThrSign}.

Theorem 2. *Protocol Π_{ThrSign} instantiates $\mathcal{F}_{\mathsf{ThrSign}}(L)$ in the $\mathcal{F}_{\mathsf{ABB+Solve}}(L)$-hybrid model.*

Proof. To prove Theorem 2, we note first that the proof is rather straightforward as the protocol simply translates the signature functionality with functionalities in the $\mathcal{F}_{\mathsf{ABB}}$ model, and leverages $\mathcal{F}_{\mathsf{ABB+Solve}}$. Formally, we describe a simulator $\mathcal{S}$ that interacts with the ideal functionality $\mathcal{F}_{\mathsf{ThrSign}}(L)$ and with the adversary, emulating internally the arithmetic and coin sampling functionalities, and emulating virtual honest parties.

Let us start with the simulation of the Distributed Key Generation procedure π_{KeyGen}. When $\mathcal{S}$ is called on input sample-key, it calls the functionality $\mathcal{F}_{\mathsf{ThrSign}}(L)$ on input sample-key. It obtains a public key $\mathsf{pk} = (\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}, \mathbf{P}_i^{(3)})_{i\in[m]}$ as output. Then, it simulates rand in Item 1 by distributing random shares of $[\![\mathbf{O}]\!]$. As in our proof of $\mathcal{F}_{\mathsf{ABB+Solve}}$, this is possible because the shares of the

adversary are independent of the underlying secret. Next, for Item 2, $\mathcal{S}$ simulates coin by returning the matrices $(\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)})_{i\in[m]}$ from pk. $\mathcal{S}$ is now also able to simulate the adversary's shares of $[\![\mathbf{P}_i^{(1)} \cdot \mathbf{O}]\!]$ in Item 3 using $\mathbf{P}_i^{(1)}$ and the previously sampled shares of $[\![\mathbf{O}]\!]$. Then, Item 4 is again simulated by sampling shares of $[\![\mathbf{O}^\mathsf{T} \cdot (\mathbf{P}_i^{(1)} \cdot \mathbf{O} - \mathbf{P}_i^{(2)})]\!]$ at random for the adversary as they are independent of the underlying secret. Since parties are required to reveal $\mathbf{P}_i^{(3)}$ in Item 6, $\mathcal{S}$ also selects honest shares of the above value. To ensure consistency with the adversary's shares and $\mathbf{P}_i^{(3)}$, $\mathcal{S}$ simply extends the sharing $[\![\mathbf{O}^\mathsf{T} \cdot (\mathbf{P}_i^{(1)} \cdot \mathbf{O} - \mathbf{P}_i^{(2)})]\!]$, which is always possible since at most $t-1$ shares are given to the adversary. Simulation of Item 7 and Item 8 follows the same approach, and $\mathcal{S}$ samples the shares of $\mathbf{L}_i$ at random for the adversary, as they are again independent of the underlying secret.

Given this simulated procedure, we turn to the simulation of threshold signing procedure π_{Sign}. Whenever $\mathcal{S}$ is called on $(\mathsf{sign}, \mathsf{msg})$, it calls the functionality $\mathcal{F}_{\mathsf{ThrSign}}(L)$ on $(\mathsf{sign}, \mathsf{msg})$. It obtains as output $((\mathsf{Sig}, \mathsf{salt}), \mathsf{Leaks})$. It simulates the call to coin in Item 1 by returning salt. Note, however, that there is a loop structure due to the restarts in Item 2. For each iteration of the loop structure, $\mathcal{S}$ emulates (Item 3, Item 4, Item 5) by sampling random shares of $[\![\mathbf{V}]\!], [\![\mathbf{M}_i]\!], [\![\mathbf{Y}_i]\!]$ (respectively for each item) for the adversary. As these are always fresh shares and less than t shares are given to the adversary, we ensure that they are independent of the underlying secret. For each iteration of the loop structure, $\mathcal{S}$ emulates Item 6 as well, by using the table Leak: given a rank $r \in \mathsf{Leak}$, we simulate the solve protocol by returning $(\mathsf{rank\text{-}defect}, r)$. After going through all of Leaks, $\mathcal{S}$ simulates solve by sampling random shares of $[\![x]\!]$. Finally, for Item 8, $\mathcal{S}$ samples random shares of $\mathbf{S}'$ for the adversary and chooses honest shares of $\mathbf{S}'$ and $\mathbf{X}$ to be consistent with the final signature $\mathsf{Sig} = (\mathbf{S}'|\mathbf{X})$.

□

5.5 On the Leakage in $\mathcal{F}_{\mathsf{ABB+Solve}}$

Recall that our functionality $\mathcal{F}_{\mathsf{ABB+Solve}}$ (see Section 4.2) not only produces a signature, but also a set of positive integers Leaks representing the *rank* of each matrix $\mathbf{A}$ that turned out to be rank-deficient. This arises from the difficulty of determining whether a given secret matrix is full rank without leaking the rank itself. Now, the "most ideal" functionality that models OV-based signing should produce the signature, and *nothing else*. Since our functionality technically leaks more than such an ideal setting, we discuss the potential implications of this in terms of the unforgeability of the underlying scheme.

We first note here that many OV-based schemes (including MAYO and UOV) follow the *Hash and Sign with Retry* paradigm [KX22]. For a given selection of random coins in the inverse trapdoor function, and a given message hash, a preimage of the hash may not necessarily exist. At a high level, the signature algorithm addresses this challenge by resampling the coins used in the inverse of the trapdoor function until a valid preimage is found. The term "retry" reflects this resampling process. Once a preimage is successfully obtained, it is output

Scheme	λ	Interval rank($\mathbf{A}$)
UOV-Ip	128	$[41, 44]$
UOV-Is	128	$[59, 64]$
UOV-III	192	$[68, 72]$
UOV-V	256	$[91, 96]$
MAYO_1	128	$[62, 64]$
MAYO_2	128	$[62, 64]$
MAYO_3	192	$[94, 96]$
MAYO_5	256	$[127, 128]$

Table 1: Overwhelming bounds on the rank of the matrices $\mathbf{A}$ for the different parameter sets of UOV and MAYO, as reported in [Cog+24].

as the signature for the message, along with the salt if one is used. These retries, in turn, imply a bias in the sampling and that an amount of information leakage is present in these schemes, but there does not seem to be any attack that can take advantage of the information leakage[8]. In the ROM-based security proof for MAYO, for example, (see section 5.2 of [Beu+23a]), an adversary can make at most Q_s signing queries to the random oracle. Given the probability average-bounded by $B = \frac{q^{k-(n-o)}}{q-1} + \frac{q^{m-ko}}{q-1}$ if $Q_s \cdot B \leq 1/2$, this results in only a constant factor reduction in advantage, indicating that security of MAYO is not compromised by much. If $Q_s \cdot B > 1$, the security proof no longer provides guarantees; but, as pointed out in the specification, there does not seem to be any attack that can take advantage of the leakage. The same analysis applies to UOV, but with a bigger leakage due to the larger restarting probability of approximately $B = 1/q$. After decades of cryptanalysis, no attacks are known that can efficiently make use of this leakage. While this leakage is noted in the specifications of OV-based schemes, it should be further formalized in their security proofs, for instance with a Renyi divergence argument.

In our threshold cases, note that nothing else besides the rank of the rank-deficient matrix $\mathbf{A}$ is revealed, and furthermore, such a matrix is entirely discarded, and a freshly new matrix $\mathbf{V}$ is sampled for a new attempt (leading to a new matrix $\mathbf{A}$). While leaking the rank $\mathbf{A}$ during signing appears like non-trivial information on the private material, the security of this tweaked scheme follows rather naturally from the assumptions described and made in OV-based schemes. Leaking the rank of $\mathbf{A}$ slightly increases the information available to an attacker over just leaking the rank deficiency of $\mathbf{A}$. However, as shown in [Cog+24] and in Table 1 , the rank of $\mathbf{A}$ is concentrated in a small interval with overwhelming probability, and thus we conjecture that leaking it is roughly equivalent to leaking that $\mathbf{A}$ is rank deficient. In the continuity of previous works on multivariate

[8] In fact, for many OV-based schemes, including UOV and MAYO, leaking that the matrix is not invertible or how many attempts were tried via a timing side-channel is not an issue as the matrix is discarded anyway [Beu+23d]. Many novel side-channel analyses of these schemes do not take advantage of this information [Aul+23a; Aul+23b].

cryptography, we thus conjecture that the rank leakage of our threshold solution does not significantly increase the probability that an adversary could forge a signature in UOV or MAYO.

6 Verifying OV-based Signatures Obliviously

In Section 5, we described the procedures and protocols that allows us to perform threshold signing. In this section, we will focus on the procedure to verify said threshold signatures, but we will go beyond that and explore other types of verification in threshold settings. Normally, the task of threshold signatures schemes is concerned with signing and verifying a public message using secret-shared keys. However, there are some practical applications that involve *verifying* the validity of a *secret-shared signature* on a *secret-shared message*. For instance, in [Ara+21; BJ18], it is noted that general MPC does not put any restriction on what kind of inputs are allowed: one can ensure, for example, that a given secret-shared input provided to some MPC computation is valid in regards to a verification procedure. For example, in the classical millionaire's problem [Yao82] where Alice and Bob want to determine who has the highest net worth without revealing anything else (their input), it may be important for the parties to ensure that the inputs provided are not fabricated but rather that they correspond to the actual net worth. To achieve this, parties supply not only their net worth but also a signature from an authority (like certified banks) as part of the MPC input[9]. Prior to executing the actual protocol, parties run the signature verification algorithm in MPC to confirm that their inputs are correctly signed.

In this section, we will first introduce the functionality (see Section 6.1) and protocols needed for threshold verification (see Section 6.2), and, later, expand them for secret-shared signatures and secret-shared messages.

6.1 Threshold OV-based Functionality: Verification

The functionality $\mathcal{F}_{\mathsf{ThrVerif}}$ described below models the verification of threshold OV-based signature schemes. The functionality is straightforward as the verification procedure of OV-based schemes simply involves a matrix product.

Functionality 4: $\mathcal{F}_{\mathsf{ThrVerif}}$

The functionality has all the commands as $\mathcal{F}_{\mathsf{ABB}}$.

On input $(\mathsf{verify}, \mathsf{msg}, \mathsf{pk}, \mathsf{Sig})$, where $\mathsf{msg} \in \{0,1\}^*$ is the message to be verified, the functionality proceeds as follows:

[9] It is also important to note that the signature must also be in secret-shared form to maintain confidentiality. Traditional signature schemes do not inherently ensure the privacy of the message: appending a message to a given signature results in a signature scheme that lacks privacy.

1. Parse $(\mathbf{S}, \mathsf{salt})$ from Sig and $(\{\mathbf{P}_i^{(1)}\}_{\{i\in[m]\}}, \{\mathbf{P}_i^{(2)})\}_{\{i\in[m]\}}, \{\mathbf{P}_i^{(3)})\}_{\{i\in[m]\}}$ from pk. Let $\mathbf{t} \leftarrow \mathcal{H}(\mathsf{msg}\|\mathsf{salt}) \in \mathbb{F}_q^m$.
2. For i from 1 to m, set: $\mathbf{Y}_i \leftarrow \mathbf{S} \begin{pmatrix} \mathbf{P}_i^{(1)} & \mathbf{P}_i^{(2)} \\ \mathbf{0} & \mathbf{P}_i^{(3)} \end{pmatrix} \mathbf{S}^\mathsf{T}$
3. Set: $\mathbf{y} \leftarrow \mathsf{OV\text{-}based.Compute\_y}(\{\mathbf{Y}_i\}_{i\in[m]})$
4. Send $(\mathsf{accept}, \mathsf{Sig})$ to all parties if $\mathbf{y} == \mathbf{t}$. Else, send $(\mathsf{reject}, \mathsf{Sig})$.

6.2 Protocol for Threshold Verification

To securely instantiate $\mathcal{F}_{\mathsf{ThrVerif}}$, the protocol leverages the simplicity of the verification algorithm in OV-based schemes, which reduces to a matrix product. The protocol Π_{ThrVerif} is essentially the same as Algorithm 5, where the last step (the check) leaks no information in case of failure.

6.3 Certifying Inputs in a Threshold Manner

Given the functionality $\mathcal{F}_{\mathsf{ThrVerif}}$ and its instantiation, Π_{ThrVerif}, we consider the possibility of expanding its scope to verify not only a complete view of the signature and message but also their secret-shared representations. This extension would enable the certification of both public values and secret-shared ones, broadening the functionality's applicability. We call this ability "certifying inputs in a threshold manner".

In a classical setting, in [Ara+21], the authors make use of PS signatures [PS15], based on the observation that PS signatures are somewhat "MPC-friendly" and will allow for this certification. OV-based schemes exhibit a limited degree of this MPC-friendliness (in the verification procedure) as they require messages to be hashed prior to signing/verifying, and hashing of secret-shared data is quite costly. However, it is worth mentioning that the verification procedure of OV-based schemes offers opportunities for enhancing other MPC functionalities (as we see in $\mathcal{F}_{\mathsf{ThrVerif}}$). Here, we present a verification procedure that removes the use of the hash function (and of the salt) and assumes that the target $\mathbf{t}$ is already the result of hashing of secret-shared $(\mathsf{msg}||\mathsf{salt})$: $\mathbf{t}$ is, hence, defined over the message space $\mathbb{F}_q^m$ directly. The assumption made here is that a trusted party performs and distributes the hashing of the secret-shared data. Below we define a functionality $\mathcal{F}_{\mathsf{ABB+CertInp}}$ which has the same features as the ABB model from Section 2.2 and expands it with the ability for parties to prove that a given (secret) input is "correct". Concretely, we model this by allowing the parties to provide as additional (secret) input a OV-based signature, which the functionality verifies using a public key provided by all parties. Crucially, $\mathcal{F}_{\mathsf{ABB+CertInp}}$ differs from $\mathcal{F}_{\mathsf{ThrVerif}}$ in that fetches the already distributed $\mathbf{t}$, contrary to the hashing performed in Item 1.

Functionality 5: $\mathcal{F}_{\mathsf{ABB+CertInp}}$

The functionality has all the commands as $\mathcal{F}_{\mathsf{ABB}}$.

On input $(\mathsf{cert}-\mathsf{verify}, \mathtt{IDs}, \mathsf{pk})$, where $\mathtt{IDs}$ is a set of IDs, the functionality proceeds as follows:

1. Parse $(\{\mathbf{P}_i^{(1)}\}_{\{i\in[m]\}}, \{\mathbf{P}_i^{(2)})\}_{\{i\in[m]\}}, \{\mathbf{P}_i^{(3)})\}_{\{i\in[m]\}}$ from pk.
2. Use $\mathtt{IDs}$ to fetch a stored value $\mathbf{t} \in \mathbb{F}_q^m$ and a stored signature $\mathbf{S} \in \mathbb{F}_q^{k\times n}$ (without the salt).
3. For i from 1 to m, set: $\mathbf{Y}_i \leftarrow \mathbf{S}\begin{pmatrix}\mathbf{P}_i^{(1)} & \mathbf{P}_i^{(2)}\\ \mathbf{0} & \mathbf{P}_i^{(3)}\end{pmatrix}\mathbf{S}^{\mathsf{T}}$
4. Set: $\mathbf{y} \leftarrow \mathsf{OV\text{-}based.Compute\_y}(\{\mathbf{Y}_i\}_{i\in[m]})$
5. Send (accept, S) to all parties if $\mathbf{y} == \mathbf{t}$. Else, send (reject, S).

The protocol ($\Pi_{\mathsf{ABB+CertInp}}$) for securely instantiating $\mathcal{F}_{\mathsf{ABB+CertInp}}$ is relatively straightforward given that the verification algorithm in OV-based schemes simply involves a matrix product. However, a subtlety in this protocol is that, at the end, the functionality performs an equality check, leaking nothing in case the check fails. A standard approach in MPC for such equality checks is to compute the difference between the two values ($\mathbf{y}$ and $\mathbf{t}$), multiply this difference by a secret random value, and then reconstruct the result. If the two original values are equal, the result will be zero; otherwise, it will appear uniformly random due to the non-zero product with a random factor, ensuring no information is leaked. This approach is effective provided the underlying field has size $> 2^\kappa$, where κ is a statistical security parameter. In smaller fields, the secret random value could unintentionally be zero, compromising the reliability of the check.

We adopt a similar approach, tailored to accommodate the relatively small size of the field $\mathbb{F}_q$ (for instance, in MAYO, $\mathbb{F}_q = 16$ for all security levels). First, recall that in $\mathcal{F}_{\mathsf{ABB+CertInp}}$, the final equality check (see Item 5) is performed between two vectors of dimension m over $\mathbb{F}_q$. After subtraction, this becomes a check for equality to $\mathbf{0}$ on a vector $\mathbf{z} \in \mathbb{F}_q^m$, where each entry is "secret-shared." Assume that $q^m \gg 2^\kappa$, which holds in the OV-based setting[10] (in MAYO, for example, $q^m = 2^{256}$ for security level 1). Let τ be chosen so that $q^\tau \approx 2^\kappa$. We first compress the equality check by taking τ linear combinations of the entries in $\mathbf{z}$ with public random coefficients from $\{0, 1\}$. With probability at least $1 - 2^{-\kappa}$, if $\mathbf{z} \neq 0$, then at least one of these linear combinations will be non-zero. Therefore, it suffices to check whether each of these τ combinations is zero. Let $\mathbf{w} \in \mathbb{F}_q^\tau$ represent these τ linear combinations. We then treat $\mathbf{w}$ not as a vector of dimension τ over $\mathbb{F}_q$ but as a single element W in the extension field $\mathbb{F}_{q^\tau}$. This allows us to perform a single zero-check on W, which is feasible because W now belongs to a sufficiently large field.

Note that for this approach to work, the ABB model must support arithmetic not only over $\mathbb{F}_q$, but also over the extension field $\mathbb{F}_{q^\tau}$. While operations over $\mathbb{F}_{q^\tau}$ can be implemented using arithmetic over $\mathbb{F}_q$, a straightforward approach may

[10] If $q^m \approx 2^\kappa$ or $q^m \ll 2^\kappa$, the protocol step involving "subset sums" can be omitted.

lead to an overhead of approximately m^2 for multiplications, as each product in $\mathbb{F}_{q^\tau}$ would require about m^2 multiplications in $\mathbb{F}_q$. Therefore, we assume the ABB model can natively support arithmetic over $\mathbb{F}_{q^\tau}$, which can be instantiated directly for better efficiency, avoiding the overhead associated with "emulating" $\mathbb{F}_{q^\tau}$ arithmetic via $\mathbb{F}_q$.

Remark 4. If $q^\tau \ll 2^\kappa$, we can simply append zeros. Conversely, if $q^\tau \gg 2^\kappa$, we can improve the zero-check by taking $\log(\kappa)$ linear combinations of $\boldsymbol{z}$ with random public 0/1 vectors (sampled using coin). This produces a vector in $\mathbb{F}_q^{\log(\kappa)}$ that is zero if and only if $\boldsymbol{z} = 0$, with overwhelming probability.

Protocol 5: $\Pi_{\mathsf{ABB+CertInp}}$

The protocol is set in the $\mathcal{F}_{\mathsf{ABB}}$-hybrid. All the commands except for (solve) are forwarded directly to $\mathcal{F}_{\mathsf{ABB}}$.
On input ($\mathsf{verify}, [\![\mathbf{t}]\!], [\![\mathbf{S}]\!], \mathsf{pk}$), the parties execute:

1. Parse $(\{\mathbf{P}_i^{(1)}\}_{\{i\in[m]\}}, \{\mathbf{P}_i^{(2)}\}_{\{i\in[m]\}}, \{\mathbf{P}_i^{(3)}\}_{\{i\in[m]\}})$ from pk.
2. For i from 1 to m, call: $[\![\mathbf{Y}_i]\!] \leftarrow [\![\mathbf{S}]\!] \begin{pmatrix} \mathbf{P}_i^{(1)} & \mathbf{P}_i^{(2)} \\ \mathbf{0} & \mathbf{P}_i^{(3)} \end{pmatrix} [\![\mathbf{S}^\mathsf{T}]\!]$
3. Locally compute: $[\![\mathbf{y}]\!] \leftarrow [\![\mathsf{OV\text{-}based.Compute\_y}(\{\mathbf{Y}_i\}_{i\in[m]})]\!]$, followed by $[\![\mathbf{z}]\!] \leftarrow [\![\mathbf{y}]\!] - [\![\mathbf{t}]\!]$.
4. For $i \in [\tau]$, call $(\alpha_{1i}, \ldots, \alpha_{mi}) \leftarrow \mathsf{coin}(\{0,1\})$.
5. For $i \in [\tau]$, locally compute: $[\![w_i]\!] \leftarrow \sum_{j=1}^m \alpha_{ji} \cdot [\![z_j]\!]$, where $\mathbf{z} = (z_1, \ldots, z_m)$. Let $\boldsymbol{w} = (w_1, \ldots, w_m)$.
6. Interpret $[\![\mathbf{w}]\!]$ as $[\![W]\!]$, where $W \in \mathbb{F}_{q^\tau}$ (a local operation). Call $[\![R]\!] \leftarrow \mathsf{rand}(\mathbb{F}_{q^\tau})$.
7. Call $[\![R \cdot W]\!] \leftarrow [\![R]\!] \cdot [\![W]\!]$.
8. Open $U \leftarrow [\![R \cdot W]\!]$. If $U = 0$, output (accept); else, output (reject).

Theorem 3. *Protocol $\Pi_{\mathsf{ABB+CertInp}}$ instantiates $\mathcal{F}_{\mathsf{ABB+CertInp}}$ in the $\mathcal{F}_{\mathsf{ABB}}$-hybrid model.*

Proof (Sketch). We do not provide a full simulation-based proof but instead, outline the main security arguments. The protocol mirrors the steps of $\mathcal{F}_{\mathsf{ABB+CertInp}}$ within the ABB model, with the main subtlety arising in the final equality check. Security follows from the following observations:

- If there exists any $z_j \neq 0$, then with probability at least $1 - q^{-\tau} \approx 1 - 2^{-\kappa}$, at least one of the linear combinations w_j will also be non-zero.
- If $W = 0$, then $R \cdot W = 0$. Otherwise, when $W \neq 0$, $R \cdot W$ appears uniformly random over $\mathbb{F}_{q^\tau}$ (which can be simulated). This value is zero only if $R = 0$, which occurs with negligible probability, as $q^\tau \approx 2^\kappa$.

7 Instantiation, Costs and Optimizations

Our different protocols and procedures have been described in the ABB model from Section 2.2, with the goal of making the presentation general and applicable

to many different security settings and concrete constructions. However, once a concrete security scenario is considered, instantiating the ABB and applying it to our protocols lends itself to several optimizations that can noticeably boost the efficiency of the final result. We discuss optimizations for the ABB model in Appendix A. In this section, we discuss concrete instantiations and several optimizations that can be done to our protocols (see Section 7.1). Note that our scheme supports adaptive corruption of parties, as long as the instantiation of the ABB model does. We also discuss the approximate costs of our protocols (see Section 7.2).

7.1 Optimizations to our Threshold Signature Protocol

Given a concrete security setting (a dishonest or honest majority setting, for example), we can apply several optimizations to our generic protocol Π_{ThrSign} that exploits certain specific properties. We discuss them in the following.

Dishonest Majority for Π_{ThrSign}. For a dishonest majority setting, the following optimizations/observations can be applied to Π_{Solve}, which is used inside Π_{ThrSign}.

Matrix triples. As observed in [Che+20; MZ17], Beaver's approach [Bea92] can be optimised in the case in which two matrices are being multiplied. In a naive implementation, multiplying two matrices $(\mathbf{A}, \mathbf{B})$ of dimensions $n \times m$ and $m \times \ell$, respectively, requires a total of $nm\ell$ individual multiplications. Since the ABB model provides only field multiplications by default, our protocols and procedures effectively translate matrix products into $nm\ell$ secure multiplications, leading to significant computational overhead. The key insight from the aforementioned works is that by preparing *matrix triples* instead of individual triples, the cost of the online phase can be reduced from $O(nm\ell)$ to $O(nm + m\ell)$.

The technique can be summarized as follows. Let $[\![\cdot]\!]$ denote the secret-sharing scheme used (additive secret-sharing with MACs, for example). To multiply two secret-shared matrices $[\![\mathbf{X}]\!]$ and $[\![\mathbf{Y}]\!]$, the parties first preprocess a triple $([\![\mathbf{A}]\!], [\![\mathbf{B}]\!], [\![\mathbf{C}]\!])$, where $\mathbf{A}$ and $\mathbf{B}$ are random n-by-m and m-by-ℓ matrices, respectively, and $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$. In the online phase, the parties then locally compute and open the values $\mathbf{D} \leftarrow [\![\mathbf{X}]\!] - [\![\mathbf{A}]\!]$ and $\mathbf{E} \leftarrow [\![\mathbf{Y}]\!] - [\![\mathbf{B}]\!]$. Then, they locally compute the product and output: $[\![\mathbf{X} \cdot \mathbf{Y}]\!] \leftarrow \mathbf{D} \cdot [\![\mathbf{B}]\!] + [\![\mathbf{A}]\!] \cdot \mathbf{E} + \mathbf{D} \cdot \mathbf{E} + [\![\mathbf{C}]\!]$. The cost of these openings is $O(nm + m\ell)$, which is substantially lower than the naive approach $O(nm\ell)$ for meaningful parameter regimes.

The generation of the matrices $([\![\mathbf{A}]\!], [\![\mathbf{B}]\!], [\![\mathbf{C}]\!])$ can vary in costs depending on the specific instantiation. While a naive approach would still require $O(nm\ell)$ products, [Che+20] presents a more efficient method that leverages the structural properties of these matrix triples, significantly reducing the overall cost compared to a generic set of $nm\ell$ arbitrary products.

Products by a random matrix. In various stages of our protocol and procedures, parties need to compute the product between a given secret-shared matrix $[\![\mathbf{X}]\!]$

and a newly sampled shared matrix $[\![\mathbf{Y}]\!]$, which is generated using the ABB command rand. Notably, by leveraging the matrix triple concept discussed earlier, we can reduce the communication overhead in the online phase by half. The procedure is as follows. After obtaining $[\![\mathbf{X}]\!]$, parties generate a matrix triple $([\![\mathbf{A}]\!], [\![\mathbf{B}]\!], [\![\mathbf{C}]\!])$ in the offline phase. They then locally compute and open the value $\mathbf{D} \leftarrow [\![\mathbf{X}]\!] - [\![\mathbf{A}]\!]$. Next, they locally compute the product $[\![\mathbf{X}\cdot\mathbf{B}]\!] \leftarrow \mathbf{D}\cdot[\![\mathbf{B}]\!] + [\![\mathbf{C}]\!]$. By setting $[\![\mathbf{Y}]\!] := [\![\mathbf{B}]\!]$, the parties effectively obtain $[\![\mathbf{X}\cdot\mathbf{Y}]\!]$ for a random $\mathbf{Y}$, as required. Importantly, this approach only incurs the cost of *one* matrix opening instead of the two that would be necessary for an implementation.

Honest Majority for Π_{ThrSign}. For an honest majority setting, the following optimizations/observations can be applied to Π_{ThrSign}.

Late Degree-Reduction for Matrix Products. While matrix triples can be effectively utilized in the honest majority context, an alternative approach that circumvents the need for preprocessing triples may be more advantageous. As previously mentioned, in an honest majority scenario, it is common to perform local multiplications that increase the polynomial degree from t to $2t$, followed by degree reduction using double sharings to revert to degree t. A significant benefit of this method is that it allows the parties to compute dot products with the same cost as a single product. Specifically, they can locally multiply the terms, raising the degree to $2t$. Instead of individually reducing the degree of each product, they first sum the degree-$2t$ sharings, yielding degree-$2t$ sharings of the desired dot product, which can subsequently be reduced. By applying this strategy to matrix multiplication, the product of an n-by-m with an m-by-ℓ matrix can be computed with communication complexity proportional to $O(n \times \ell)$, significantly improving upon the naive complexity $O(nm\ell)$ (and, in fact, this is better than the matrix triple approach if m is substantially large).

Active Security with Sublinear Overhead. This observation highlights a crucial aspect of our protocols rather than a straightforward optimization. Our construction is fundamentally based on the ABB model, which inherently requires only secure additions and multiplications. Recent advancements in actively secure honest majority MPC leverage techniques known as *distributed zero-knowledge* [Bon+19], which ensure that the overhead compared to passive security is minimal (specifically, *sublinear* in the number of multiplications performed). These techniques are particularly effective for verifying multiplications, aligning well with our objectives. This is significant, as many other threshold signature schemes, such as those based on elliptic curve cryptography (e.g., ECDSA), necessitate additional operations like group exponentiations. The fact that OV-based signatures schemes require only additions and multiplications is a pivotal advantage: it enables the use of novel methodologies in *generic* MPC.

7.2 Concrete Costs of our Protocols

In the following, we conduct an approximate evaluation of the practicality of our protocols, focusing on both round complexity and computational time. We calculate the number of rounds required for each protocol (specifically, for Π_{ThrSign}), assuming that each depth of multiplication takes one round, coin sampling requires two rounds, and linear and random operations are performed locally. Since the computation time of these protocols is primarily influenced by the secure multiplications, we provide an inventory of the multiplications executed.

Key generation: π_{KeyGen}. Looking at the procedure, we see that Item 1 and Item 2 (sampling of random values) take two rounds. The multiplications needed for Item 7 and Item 4 take two additional rounds. A final round is required for Item 6. This results in a total of 5 rounds for key generation

The multiplications performed in this procedure are: m matrix multiplications of dimension $o \times (n-o)$ and $(n-o) \times o$, and m matrix multiplications of dimension $(n-o) \times (n-o)$ and $(n-o) \times o$.

Signature generation: π_{Sign}. Looking at the procedure, we see that Item 1 (sample of salt) takes two rounds. In parallel, one depth of multiplication is required for Item 3 and Item 4, which results in an extra round. For the procedure solving the system of equations (Π_{Solve}), two depths of multiplications are required for Item 3 and one extra round to open the value in Item 4. In case $\mathbf{T}$ is full-rank, 3 more rounds are required to compute a solution to the system (see from Item 6 onwards). Finally, two rounds are required to transform the solution to the system into a signature and reveal it (see Item 8). In total, it takes 9 rounds to perform signing assuming one finds a solution to the system of equations in the first try. Assuming a probability of success p of solving the system of equations, we can compute the average round complexity as $6+\frac{1}{p}\cdot 3$ (in the current protocol, we would have $p \approx 1 - \frac{2}{q}$).

The multiplications performed in this procedure are:

- m matrix multiplications of dimension $k \times (n-o)$ and $(n-o) \times o$.
- m matrix multiplications of dimension $(n-o) \times (n-o)$ and $(n-o) \times k$.
- 2 matrix multiplication of dimension $m \times ko$ and $ko \times ko$.
- 1 matrix multiplication of dimension $m \times m$ and $m \times ko$.
- 1 multiplication of a matrix of dimension $m \times m$ with a vector $\in m$.
- 1 multiplication of a matrix of dimension $ko \times ko$ with a vector $\in ko$.
- 1 matrix multiplication of dimension $ko \times o$ and $o \times (n-o)$.

Verification of inputs: $\Pi_{\mathsf{ABB+CertInp}}$. Looking at the protocol, we see that it requires a depth of two multiplications (see Item 2 and Item 7) and one reveal (see Item 8). In Item 4 coin sampling is called, which can be started in parallel with the first multiplication. This leads to a 4 round protocol.

The multiplications performed in this procedure are: m matrix multiplications of dimension $k \times n$ and $n \times k$. The computational cost of these algorithms is primarily driven by the m multiplications required. Threshold variants of MAYO can be more costly than those of UOV, due to the parameter k, which directly increases the total multiplication cost. For example, at security level 1, MAYO uses $k = 9$, whereas UOV consistently sets $k = 1$.

8 Applications

8.1 TLS and PKI

While the traditional architecture of TLS is based on a client-server infrastructure, most architectures used nowadays deviate from this setting. In a content delivery network (CDN), for example, the aim is to accelerate webpage loading for data-heavy applications by caching data on CDN servers rather than repeatedly fetching it from "original" (or "origin") server for every request. When TLS is used in these cases, CDN servers must terminate TLS connections on behalf of the origin servers, requiring them to sign with the origin server's long-term private key. This setup can be managed in two modes: either the origin server shares its long-term private key material with CDN servers, or it exposes a functionality (given by an auxiliary machine) only used for signing purposes. However, either option introduces security risks as private key material or sensitive operations are shared between different parties and machines.

An alternative approach is to use threshold signing, so that only *shares* of the private key are distributed between parties and machines (hence, the need for a DKG as described in Section 5.2). As noted by [Her23], in the classical setting, what seems to be needed for an actual application of threshold signatures in this TLS setting is both practicality and compatibility with existing, standardized signature schemes, as these schemes are widely used and accessible. If a post-quantum setting is also needed, a similar approach must be taken by looking to incorporate practical post-quantum schemes that are proposed for standardization. As noted in this work, threshold variants of OV-based schemes seem suitable for this application.

Research has suggested [Wes24; Adr24] that schemes like MAYO and UOV are technically feasible to be used in TLS 1.3 connections, though their large public keys present challenges. Cloudflare, for instance, observed a 1.5% increase in median HTTPS handshake latency for every 1 KB of additional server response data which often includes signatures and public keys. Similarly, Chrome reported a 4% increase in TLS handshake latency when using ML-KEM, whose public key signature pair (around 4 KB) increased the ClientHello size by roughly 1 KB. This increase pushed the ClientHello size beyond the standard Internet MTU (around 1400 bytes), causing it to fragment over multiple TCP or UDP packets. These findings suggest that public key and signature sizes significantly impact network performance, underscoring the importance of keeping sizes manageable in threshold versions of these schemes. While increased computational times in threshold variants can be distributed among parties and are less of a bottleneck,

the number of communication rounds might pose practical challenges. Further studies would be valuable future work to see how the applications of the techniques described in Section 7 can help with them.

8.2 The Tor Network and Onion Services

The Tor network serves millions of users a day, with the aim of providing anonymity to its users from the websites they connect to, and concealing what they are connecting to from their Internet service provider and any other intermediary in their path [TTP24]. To maintain security, the network needs to produce a fresh random value every day in such a way that it cannot be predicted in advance or somehow influenced by an attacker. This random number is generated every midnight by the trusted directory authorities[11]. The technique used for this procedure, a distributed random generator scheme, is based on a commit-and-reveal technique. However, it is vulnerable to various attacks as noted in the Tor security analysis [Pro24].

[Hop14] proposed that directory authorities use a threshold signature scheme to sign the value $\mathcal{H}$(time-period) in each consensus document (a document that contains information about all known Tor relays), binding it to a specific time period. In this approach, each authority generates a publicly verifiable share of the signature during the voting process. As long as at least $\lceil n/2 \rceil$ authorities are honest, the full signature can be reconstructed by each client. This threshold signing mechanism ensures that if an adversary does not control at least $\lceil n/2 \rceil$ authorities, they cannot forge or compute the signature, which adds security to the system. This approach has not yet been integrated into the network, primarily due to the lack of a standardized, practical threshold algorithm. OV-based schemes could offer an alternative, as their practical design and the construction make them promising candidates for wider implementation.

Acknowledgments.

We would like to thank Ward Beullens, Lisa Kohl and Yashvanth Kondi for their helpful discussions and insight.

This paper was prepared in part for information purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such

[11] See https://spec.torproject.org/rend-spec/shared-random.html.

solicitation under such jurisdiction or to such person would be unlawful. 2024 JP Morgan Chase & Co.

References

[ADN06] Jesús F. Almansa, Ivan Damgård, and Jesper Buus Nielsen. "Simplified Threshold RSA with Adaptive and Proactive Security". In: 2006, pp. 593–611. DOI: 10.1007/11761679_35.

[Adr24] David Adrian. *Post-quantum cryptography is too damn big*. Personal blogpost. 2024. URL: https://dadrian.io/blog/posts/pqc-signatures-2024/.

[Ala+24] Gorjan Alagic, Maxime Bros, Pierre Ciadoux, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Hamilton Silberg, Daniel Smith-Tone, and Noah Waller. *Status Report on the First Round of the Additional Digital Signature Schemes for the NIST Post-Quantum Cryptography Standardization Process*. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.IR.8528. 2024.

[Ara+21] Diego F. Aranha, Anders P. K. Dalskov, Daniel Escudero, and Claudio Orlandi. "Improved Threshold Signatures, Proactive Secret Sharing, and Input Certification from LSS Isomorphisms". In: 2021, pp. 382–404. DOI: 10.1007/978-3-030-88238-9_19.

[ASY22] Shweta Agrawal, Damien Stehlé, and Anshu Yadav. "Round-Optimal Lattice-Based Threshold Signatures, Revisited". In: 2022, 8:1–8:20. DOI: 10.4230/LIPIcs.ICALP.2022.8.

[Aul+23a] Thomas Aulbach, Fabio Campos, Juliane Krämer, Simona Samardjiska, and Marc Stöttinger. *Separating Oil and Vinegar with a Single Trace*. Cryptology ePrint Archive, Report 2023/335. https://eprint.iacr.org/2023/335. 2023.

[Aul+23b] Thomas Aulbach, Fabio Campos, Juliane Krämer, Simona Samardjiska, and Marc Stöttinger. "Separating Oil and Vinegar with a Single Trace Side-Channel Assisted Kipnis-Shamir Attack on UOV". In: 2023.3 (2023), pp. 221–245. DOI: 10.46586/tches.v2023.i3.221-245.

[BCS19] Carsten Baum, Daniele Cozzo, and Nigel P. Smart. "Using TopGear in Overdrive: A More Efficient ZKPoK for SPDZ". In: 2019, pp. 274–302. DOI: 10.1007/978-3-030-38471-5_12.

[Bea92] Donald Beaver. "Efficient Multiparty Protocols Using Circuit Randomization". In: 1992, pp. 420–432. DOI: 10.1007/3-540-46766-1_34.

[Beu20] Ward Beullens. *Improved Cryptanalysis of UOV and Rainbow*. Cryptology ePrint Archive, Report 2020/1343. https://eprint.iacr.org/2020/1343. 2020.

[Beu21] Ward Beullens. *MAYO: Practical Post-Quantum Signatures from Oil-and-Vinegar Maps*. Cryptology ePrint Archive, Report 2021/1144. https://eprint.iacr.org/2021/1144. 2021.

[Beu+23a] Ward Beullens, Fabio Campos, Sofía Celi, Basil Hess, and Matthias J. Kannwischer. *MAYO*. Tech. rep. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures. National Institute of Standards and Technology, 2023.

[Beu+23b] Ward Beullens, Ming-Shing Chen, Jintai Ding, Boru Gong, Matthias J. Kannwischer, Jacques Patarin, Bo-Yuan Peng, Dieter Schmidt, Cheng-Jhih Shih, Chengdong Tao, and Bo-Yin Yang. *UOV — Unbalanced Oil and Vinegar*. Tech. rep. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures. National Institute of Standards and Technology, 2023.

[Beu+23c] Ward Beullens, Ming-Shing Chen, Shih-Hao Hung, Matthias J. Kannwischer, Bo-Yuan Peng, Cheng-Jhih Shih, and Bo-Yin Yang. *Oil and Vinegar: Modern Parameters and Implementations*. Cryptology ePrint Archive, Report 2023/059. https://eprint.iacr.org/2023/059. 2023.

[Beu+23d] Ward Beullens, Ming-Shing Chen, Shih-Hao Hung, Matthias J. Kannwischer, Bo-Yuan Peng, Cheng-Jhih Shih, and Bo-Yin Yang. "Oil and Vinegar: Modern Parameters and Implementations". In: 2023.3 (2023), pp. 321–365. DOI: 10.46586/tches.v2023.i3.321-365.

[Beu24] Ward Beullens. *Improved Cryptanalysis of SNOVA*. Cryptology ePrint Archive, Paper 2024/1297. 2024. URL: https://eprint.iacr.org/2024/1297.

[BJ18] Marina Blanton and Myoungin Jeong. "Improved Signature Schemes for Secure Multi-party Computation with Certified Inputs". In: 2018, pp. 438–460. DOI: 10.1007/978-3-319-98989-1_22.

[BKP13] Rikke Bendlin, Sara Krehbiel, and Chris Peikert. "How to Share a Lattice Trapdoor: Threshold Protocols for Signatures and (H)IBE". In: 2013, pp. 218–236. DOI: 10.1007/978-3-642-38980-1_14.

[BMP22] Constantin Blokh, Nikolaos Makriyannis, and Udi Peled. *Efficient Asymmetric Threshold ECDSA for MPC-based Cold Storage*. Cryptology ePrint Archive, Paper 2022/1296. 2022. URL: https://eprint.iacr.org/2022/1296.

[Bol03] Alexandra Boldyreva. "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme". In: 2003, pp. 31–46. DOI: 10.1007/3-540-36288-6_3.

[Bon+19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. "Zero-Knowledge Proofs on Secret-Shared Data via Fully Linear PCPs". In: 2019, pp. 67–97. DOI: 10.1007/978-3-030-26954-8_3.

[Bos+24] Cecilia Boschini, Darya Kaviani, Russell W. F. Lai, Giulio Malavolta, Akira Takahashi, and Mehdi Tibouchi. *Ringtail: Practical Two-Round Threshold Signatures from Learning with Errors*. Cryp-

tology ePrint Archive, Paper 2024/1113. 2024. URL: https://eprint.iacr.org/2024/1113.

[Boy+18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. "Compressing Vector OLE". In: 2018, pp. 896–912. DOI: 10.1145/3243734.3243868.

[Boy+22] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. "Correlated Pseudorandomness from Expand-Accumulate Codes". In: 2022, pp. 603–633. DOI: 10.1007/978-3-031-15979-4_21.

[Cab+24] Daniel Cabarcas, Peigen Li, Javier Verbel, and Ricardo Villanueva-Polanco. *Improved Attacks for SNOVA by Exploiting Stability under a Group Action*. Cryptology ePrint Archive, Paper 2024/1770. 2024. URL: https://eprint.iacr.org/2024/1770.

[Can01] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: 2001, pp. 136–145. DOI: 10.1109/SFCS.2001.959888.

[Can+02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. *Universally Composable Two-Party and Multi-Party Secure Computation*. Cryptology ePrint Archive, Report 2002/140. https://eprint.iacr.org/2002/140. 2002.

[Cas+19] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. "Two-Party ECDSA from Hash Proof Systems and Efficient Instantiations". In: 2019, pp. 191–221. DOI: 10.1007/978-3-030-26954-8_7.

[CDP22] Sanjit Chatterjee, M. Prem Laxman Das, and Tapas Pandit. "Revisiting the Security of Salted UOV Signature". In: 2022, pp. 697–719. DOI: 10.1007/978-3-031-22912-1_31.

[Che+20] Hao Chen, Miran Kim, Ilya P. Razenshteyn, Dragos Rotaru, Yongsoo Song, and Sameer Wagh. "Maliciously Secure Matrix Multiplication with Applications to Private Deep Learning". In: 2020, pp. 31–59. DOI: 10.1007/978-3-030-64840-4_2.

[Chi+18] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. "Fast Large-Scale Honest-Majority MPC for Malicious Adversaries". In: 2018, pp. 34–64. DOI: 10.1007/978-3-319-96878-0_2.

[Cog+24] Benoît Cogliati, Pierre-Alain Fouque, Louis Goubin, and Brice Minaud. *New Security Proofs and Techniques for Hash-and-Sign with Retry Signature Schemes*. Cryptology ePrint Archive, Paper 2024/609. 2024. URL: https://eprint.iacr.org/2024/609.

[CS19] Daniele Cozzo and Nigel P. Smart. "Sharing the LUOV: Threshold Post-quantum Signatures". In: 2019, pp. 128–153. DOI: 10.1007/978-3-030-35199-1_7.

[Dam+12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. "Multiparty Computation from Somewhat Homomorphic Encryp-

tion". In: 2012, pp. 643–662. DOI: 10.1007/978-3-642-32009-5_38.

[Dam+13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. "Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits". In: 2013, pp. 1–18. DOI: 10.1007/978-3-642-40203-6_1.

[DE21] Anders P. K. Dalskov and Daniel Escudero. "Honest Majority MPC with Abort with Minimal Online Communication". In: 2021, pp. 453–472. DOI: 10.1007/978-3-030-88238-9_22.

[Des90] Yvo Desmedt. "Abuses in Cryptography and How to Fight Them". In: 1990, pp. 375–389. DOI: 10.1007/0-387-34799-2_29.

[DF90] Yvo Desmedt and Yair Frankel. "Threshold Cryptosystems". In: 1990, pp. 307–315. DOI: 10.1007/0-387-34805-0_28.

[DM20] Luca De Feo and Michael Meyer. "Threshold Schemes from Isogeny Assumptions". In: 2020, pp. 187–212. DOI: 10.1007/978-3-030-45388-6_7.

[DN07] Ivan Damgård and Jesper Buus Nielsen. "Scalable and Unconditionally Secure Multiparty Computation". In: 2007, pp. 572–590. DOI: 10.1007/978-3-540-74143-5_32.

[Doe+18] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. "Secure Two-party Threshold ECDSA from ECDSA Assumptions". In: 2018, pp. 980–997. DOI: 10.1109/SP.2018.00036.

[Doe+19] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. "Threshold ECDSA from ECDSA Assumptions: The Multiparty Case". In: 2019, pp. 1051–1066. DOI: 10.1109/SP.2019.00024.

[Doe+23] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. *Threshold ECDSA in Three Rounds*. Cryptology ePrint Archive, Paper 2023/765. 2023. URL: https://eprint.iacr.org/2023/765.

[ENP24] Thomas Espitau, Guilhem Niot, and Thomas Prest. *Flood and Submerse: Distributed Key Generation and Robust Threshold Signature from Lattices*. Cryptology ePrint Archive, Paper 2024/959. 2024. URL: https://eprint.iacr.org/2024/959.

[Esc+22] Daniel Escudero, Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. "TurboPack: Honest Majority MPC with Constant Online Communication". In: 2022, pp. 951–964. DOI: 10.1145/3548606.3560633.

[Gar+21] François Garillot, Yashvanth Kondi, Payman Mohassel, and Valeria Nikolaenko. "Threshold Schnorr with Stateless Deterministic Signing from Standard Assumptions". In: 2021, pp. 127–156. DOI: 10.1007/978-3-030-84242-0_6.

[Gen+07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". In: 20.1 (Jan. 2007), pp. 51–83. DOI: 10.1007/s00145-006-0347-3.

[GG19] Rosario Gennaro and Steven Goldfeder. *Fast Multiparty Threshold ECDSA with Fast Trustless Setup*. Cryptology ePrint Archive, Report 2019/114. https://eprint.iacr.org/2019/114. 2019.

[GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. *Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security*. Cryptology ePrint Archive, Report 2016/013. https://eprint.iacr.org/2016/013. 2016.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990. ISBN: 0716710455.

[Goy+21] Vipul Goyal, Hanjun Li, Rafail Ostrovsky, Antigoni Polychroniadou, and Yifan Song. "ATLAS: Efficient and Scalable MPC in the Honest Majority Setting". In: 2021, pp. 244–274. DOI: 10.1007/978-3-030-84245-1_9.

[GPS21] Vipul Goyal, Antigoni Polychroniadou, and Yifan Song. "Unconditional Communication-Efficient MPC via Hall's Marriage Theorem". In: 2021, pp. 275–304. DOI: 10.1007/978-3-030-84245-1_10.

[GSZ20] Vipul Goyal, Yifan Song, and Chenzhi Zhu. "Guaranteed Output Delivery Comes Free in Honest Majority MPC". In: 2020, pp. 618–646. DOI: 10.1007/978-3-030-56880-1_22.

[Her23] Armando Faz Hernandez. *Requirements for Threshold TLS*. NIST presentation. 2023. URL: https://csrc.nist.gov/csrc/media/Presentations/2023/mpts2023-day2-talk-threshold-tls-rsa/images-media/mpts2023-2b1-slides--armando--threshold-TLS.pdf.

[Hop14] Nicholas Hopper. *A threshold signature-based proposal for a shared RNG*. Tor Dev Mailing List. 2014. URL: https://lists.torproject.org/pipermail/tor-dev/2014-January/006053.html.

[Kat23] Jonathan Katz. *Round Optimal Fully Secure Distributed Key Generation*. Cryptology ePrint Archive, Paper 2023/1094. 2023. URL: https://eprint.iacr.org/2023/1094.

[KG20] Chelsea Komlo and Ian Goldberg. "FROST: Flexible Round-Optimized Schnorr Threshold Signatures". In: 2020, pp. 34–65. DOI: 10.1007/978-3-030-81652-0_2.

[Kha+22] Irakliy Khaburzaniya, Konstantinos Chalkias, Kevin Lewi, and Harjasleen Malvai. "Aggregating and Thresholdizing Hash-based Signatures using STARKs". In: 2022, pp. 393–407. DOI: 10.1145/3488932.3524128.

[Kon+21] Yashvanth Kondi, Bernardo Magri, Claudio Orlandi, and Omer Shlomovits. "Refresh When You Wake Up: Proactive Threshold Wallets with Offline Devices". In: 2021, pp. 608–625. DOI: 10.1109/SP40001.2021.00067.

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. "MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer". In: 2016, pp. 830–842. DOI: 10.1145/2976749.2978357.

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin. "Unbalanced Oil and Vinegar Signature Schemes". In: 1999, pp. 206–222. DOI: 10.1007/3-540-48910-X_15.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. "Overdrive: Making SPDZ Great Again". In: 2018, pp. 158–189. DOI: 10.1007/978-3-319-78372-7_6.

[KS98] Aviad Kipnis and Adi Shamir. "Cryptanalysis of the Oil & Vinegar Signature Scheme". In: 1998, pp. 257–266. DOI: 10.1007/BFb0055733.

[KX22] Haruhisa Kosuge and Keita Xagawa. *Probabilistic Hash-and-Sign with Retry in the Quantum Random Oracle Model*. Cryptology ePrint Archive, Paper 2022/1359. 2022. URL: https://eprint.iacr.org/2022/1359.

[KX24] Haruhisa Kosuge and Keita Xagawa. "Probabilistic Hash-and-Sign with Retry in the Quantum Random Oracle Model". In: 2024, pp. 259–288. DOI: 10.1007/978-3-031-57718-5_9.

[Lin20] Yehuda Lindell. *Secure Multiparty Computation (MPC)*. Cryptology ePrint Archive, Report 2020/300. https://eprint.iacr.org/2020/300. 2020.

[Lin24] Yehuda Lindell. "Simple Three-Round Multiparty Schnorr Signing with Full Simulatability". In: 1.1 (2024), p. 25. DOI: 10.62056/a36c0l5vt.

[LN17] Yehuda Lindell and Ariel Nof. "A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority". In: 2017, pp. 259–276. DOI: 10.1145/3133956.3133999.

[LNR18] Yehuda Lindell, Ariel Nof, and Samuel Ranellucci. *Fast Secure Multiparty ECDSA with Practical Distributed Key Generation and Applications to Cryptocurrency Custody*. Cryptology ePrint Archive, Report 2018/987. https://eprint.iacr.org/2018/987. 2018.

[Mak22] Nikolaos Makriyannis. *On the Classic Protocol for MPC Schnorr Signatures*. Cryptology ePrint Archive, Paper 2022/1332. 2022. URL: https://eprint.iacr.org/2022/1332.

[MI88] Tsutomu Matsumoto and Hideki Imai. "Public Quadratic Polynominal-Tuples for Efficient Signature-Verification and Message-Encryption". In: 1988, pp. 419–453. DOI: 10.1007/3-540-45961-8_39.

[MZ17] Payman Mohassel and Yupeng Zhang. "SecureML: A System for Scalable Privacy-Preserving Machine Learning". In: 2017, pp. 19–38. DOI: 10.1109/SP.2017.12.

[NIS22] NIST Computer Security Division. *Post-Quantum Cryptography: Digital Signature Schemes*. https://csrc.nist.gov/projects/pqc-dig-sig. 2022.

[NV18] Peter Sebastian Nordholt and Meilof Veeningen. "Minimising Communication in Honest-Majority MPC by Batchwise Multiplication Verification". In: 2018, pp. 321–339. DOI: 10.1007/978-3-319-93387-0_17.

[OSS84] H. Ong, Claus-Peter Schnorr, and Adi Shamir. "Efficient Signature Schemes Based on Polynomial Equations". In: 1984, pp. 37–46. DOI: 10.1007/3-540-39568-7_4.

[Pat00] Jacques Patarin. "Cryptanalysis of the Matsumoto and Imai Public Key Scheme of Eurocrypt'98". In: 20.2 (2000), pp. 175–209. DOI: 10.1023/A:1008341625464.

[Pat97] Jacques Patarin. *The Oil and Vinegar signature scheme*. Dagstuhl Workshop on Cryptography. 1997.

[PB23] René Peralta and Luís T.A.N. Brandão. *NIST first call for multi-party threshold schemes*. National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/ir/2023/NIST.IR.8214C.ipd.pdf. 2023.

[Pin+24] Rafaël Del Pino, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, and Markku-Juhani O. Saarinen. "Threshold Raccoon: Practical Threshold Signatures from Standard Lattice Assumptions". In: 2024, pp. 219–248. DOI: 10.1007/978-3-031-58723-8_8.

[Pro24] The Tor Project. *Security Analysis*. Tor Specifications. 2024. URL: https://spec.torproject.org/srv-spec/security-analysis.html.

[PS15] David Pointcheval and Olivier Sanders. *Short Randomizable Signatures*. Cryptology ePrint Archive, Report 2015/525. https://eprint.iacr.org/2015/525. 2015.

[Ruf+22] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. "ROAST: Robust Asynchronous Schnorr Threshold Signatures". In: 2022, pp. 2551–2564. DOI: 10.1145/3548606.3560583.

[Sha79] Adi Shamir. "How to Share a Secret". In: 22.11 (Nov. 1979), pp. 612–613. DOI: 10.1145/359168.359176.

[Sho99] Victor Shoup. *Practical Threshold Signatures*. Cryptology ePrint Archive, Report 1999/011. https://eprint.iacr.org/1999/011. 1999.

[SSH11] Koichi Sakumoto, Taizo Shirai, and Harunaga Hiwatari. "On Provable Security of UOV and HFE Signature Schemes against Chosen-Message Attack". In: *Post-Quantum Cryptography*. Ed. by Bo-Yin Yang. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 68–82. ISBN: 978-3-642-25405-5.

[TTP24] Inc The Tor Project. *Tor Metrics*. https://metrics.torproject.org/. 2024.

[Wan+23] Lih-Chung Wang, Chun-Yen Chou, Jintai Ding, Yen-Liang Kuan, Ming-Siou Li, Bo-Shu Tseng, Po-En Tseng, and Chia-Chun Wang.

SNOVA. Tech. rep. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures. National Institute of Standards and Technology, 2023.

[Wes24] Bas Westerbaan. *The state of the post-quantum Internet*. Cloudflare blogpost. 2024. URL: https://blog.cloudflare.com/pq-2024/.

[Wik05] Douglas Wikström. "Universally Composable DKG with Linear Number of Exponentiations". In: 2005, pp. 263–277. DOI: 10.1007/978-3-540-30598-9_19.

[Yao82] Andrew Chi-Chih Yao. "Protocols for Secure Computations (Extended Abstract)". In: 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38.

A Concrete Protocols for the ABB Model

Different concrete instantiations of the ABB model depend on the adversarial model. Below, we discuss some concrete protocols that can instantiate the ABB model depending on the security setting.

Dishonest Majority. In settings where all but one of the parties is corrupt (*dishonest majority*), protocols that instantiate the ABB model employ a well-established approach, typically using *additive secret-sharing*. In this scheme, a secret x is shared as $x = x_1 + \cdots + x_n$ across parties. For active security, the most common technique, introduced in [Dam+13], uses *one-time MACs* alongside each shared secret x. This approach includes sharing a global random value Δ and a "MAC" $(\Delta \cdot x)$, which prevents adversarial parties from reconstructing invalid values maliciously.

Protocols in *dishonest majority* often rely on computationally expensive tools, including heavy public-key techniques. A widely adopted strategy to mitigate these costs is to split the MPC protocol into two phases: an *offline phase* (also called a *preprocessing phase*) and an *online phase*. The offline phase is performed before the input (in our case, the message to be signed) is known, while the online phase occurs once the inputs are available. Beaver [Bea92] demonstrated that by generating *multiplication triples*[12] in the offline phase, the online phase can be both information-theoretic and highly efficient, avoiding the need for continuous public-key cryptography. Modern schemes rely on this technique.

Building on this, several protocols have extended the seminal SPDZ works [Dam+13; Dam+12]. These protocols generally follow Beaver's approach and the SPDZ MAC structure, differing primarily in how they instantiate the offline phase (i.e., generate the triples). The original SPDZ protocol utilized *somewhat homomorphic encryption* for the offline phase, ensuring active security via zero-knowledge proofs. MASCOT [KOS16] later improved efficiency by relying on *oblivious transfer* for preprocessing. Further enhancements were made in protocols like Overdrive [KPR18] and TopGear [BCS19], which revisited homomorphic encryption and introduced new zero-knowledge techniques.

A recent development in dishonest-majority MPC protocols is the use of *pseudorandom correlation generators* (PCGs) in the offline phase, allowing parties to non-interactively expand a small initial batch of triples into a larger set. PCGs were introduced by Boyle et al. [Boy+18], with subsequent work providing further refinements. For an overview of current constructions, we refer readers to [Boy+22].

Honest Majority. In scenarios where the adversary corrupts only a minority of the parties, more efficient protocols can be developed that circumvent the need

[12] Given a multiplication triple represented as a tuple $([a], [b], [c])$ where $a, b \in \mathbb{F}$ are random elements that satisfy $c = a \cdot b$, one can perform private multiplication of x and y.

for public-key cryptography. The most widely used tool in this context is Shamir secret-sharing [Sha79]. In this approach, a secret x is distributed by sampling a polynomial f of degree $\leq t$ constrained to $f(0) = x$. Each party i receives a share of the form $f(i)$. Importantly, parties can locally multiply their shares, which allows them to generate new shares represented by a polynomial of degree up to $2t$, which is less than the number of parties. While multiplication triples can be utilized in the honest majority setting, it is also common to employ the concept of *double-sharings*, introduced by Damgård and Nielsen [DN07]. Double-sharings consist of pairs of random values that are secret-shared: one with degree t and the other with degree$2t$. This technique effectively reduces the degree of locally multiplied sharings from $2t$ back to t. For references on concretely efficient protocols in this setting, we refer to the relevant following works [LN17; Chi+18; NV18; GSZ20; Goy+21; DE21; GPS21; Esc+22].

B "One-Sided Masking" in Π_{Solve}.

One can consider a simpler and more efficient variant of Π_{Solve} to return a random solution of the system $\mathbf{A} \cdot \mathbf{x} = \mathbf{y}$ at the expense of leaking more information in case that $\mathbf{A}$ is rank-deficient (where $\mathbf{A} \in \mathcal{F}_q^{s \times t}$) and even more in the case that $\mathbf{A}$ is non-square. This, already proposed by Cozzo and Smart [CS19], consists of "only masking $\mathbf{A}$ on one side". In concrete terms, we compute $\mathbf{T} = \mathbf{A} \cdot \mathbf{S}$ (which we define as "right-masking"), and remove $\mathbf{R}$ from Line 3. This approach results in less calls to $\mathcal{F}_{\mathsf{Rand}}$ (item 1), and two less calls to $\mathcal{F}_{\mathsf{Mult}}$ (lines 3 and 5). However, note that the matrix $\mathbf{T} = \mathbf{A} \cdot \mathbf{S}$ retains the same image space as $\mathbf{A}$ (assuming $\mathbf{S}$ is invertible), and in case that $\mathbf{A}$ is not full rank, this image space is non-trivial.[13] While this could be safely used for some tweaked parameter sets of OV-based schemes that guarantee that, with high probability, $\mathbf{A}$ is full rank, this technique raises concerns in the context of the parameter sets proposed for UOV and MAYO in their specifications for standardization.

Recall that in MAYO and UOV, $\mathbf{A}$ is of dimensions $m \times ko$ with $m \leq ko$. In a "right-masking" approach for $\mathcal{F}_{\mathsf{Solve}}$, if $\mathbf{A}$ is full-rank, then $\mathbf{A} \cdot \mathbf{S}$ (for $\mathbf{S} \leftarrow \mathbb{F}_q^{ko \times ko}$) is uniformly distributed among matrices $\left[\mathbf{I}_m \ \mathbf{0}_{m \times (ko-m)}\right] \cdot \mathcal{U}(\mathbb{F}_q^{ko \times ko})$ and is independent of the matrix $\mathbf{A}$. However, whenever $\mathbf{A}$ is not full-rank, then the matrix $\mathbf{A} \cdot \mathbf{S}$ reveals the image space of $\mathbf{A}$ whenever $\mathbf{S}$ is invertible, which is a non-trivial information. For current parameter sets of MAYO and UOV, $\mathbf{A}$ has a non-negligible probability of being rank deficient (at most 2^{-36} and 2^{-4} respectively, for security level 1).

To understand this leakage in more depth, take the case with $k = 1$ (corresponding to UOV). Here, any vector $\mathbf{x}$ in the kernel of $(\mathbf{A}^\mathsf{T})$ satisfies for all $\sum_{i=1}^{m} \mathbf{L}_i^\mathsf{T} \cdot \mathbf{v} \cdot x_i = 0$. This relationship can be linked to the trapdoor $\mathbf{O}$:

[13] If one masks "on the left", that is, set $\mathbf{T} = \mathbf{R} \cdot \mathbf{A}$, a similar issue is encountered, except this time the kernel of $\mathbf{A}$ is leaked instead of its column space. When $s < t$, the kernel of A contains more information than its column space. For example, even if $\mathbf{A}$ is full rank, its kernel would be non-trivial while its column space would be the whole $\mathbb{F}_q^s$.

$\sum_{i=1}^{m}((\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)\mathsf{T}}) \cdot \mathbf{O} - \mathbf{P}_i^{(2)})^{\mathsf{T}} \cdot \mathbf{v} \cdot x_i = 0$. Although $\mathbf{v}$ remains hidden, this equation reveals structure in the trapdoor $\mathbf{O}$. As a high level intuition for an attack, we could define a system to solve for $\mathbf{O}$. For every basis vector $\mathbf{x}$ of the kernel of $\mathbf{A}^{\mathsf{T}}$, we add ko new equations. We also add $n - o$ variables to guess $\mathbf{v}$. Whenever the kernel of $\mathbf{A}^{\mathsf{T}}$ has a sufficiently high dimension, we thus decrease the difference between the number of variables and the number of equations, making the guessing of $\mathbf{O}$ easier.

We leave the details and practicality of such an attack as future work. However, this appears to be a significant concern: masking on only one side should be avoided when the image space of $\mathbf{A}$ that is leaked may be non-trivial.