Do Not Trust in Numbers: Practical Distributed Cryptography With General Trust

Orestis Alpos
University of Bern
orestis.alpos@unibe.ch

Christian Cachin
University of Bern
christian.cachin@unibe.ch

Abstract

In *distributed cryptography* independent parties jointly perform some cryptographic task. In the last decade distributed cryptography has been receiving more attention than ever. Distributed systems power almost all applications, blockchains are becoming prominent, and, consequently, numerous practical and efficient distributed cryptographic primitives are being deployed.

The failure models of current distributed cryptographic systems, however, lack expressibility. Assumptions are only stated through numbers of parties, thus reducing this to *threshold cryptography*, where all parties are treated as identical and correlations cannot be described. Distributed cryptography does not have to be threshold-based. With *general distributed cryptography* the *authorized sets*, the sets of parties that are sufficient to perform some task, can be arbitrary, and are usually modeled by the abstract notion of a general *access structure*.

Although the necessity for general distributed cryptography has been recognized long ago and many schemes have been explored in theory, relevant practical aspects remain opaque. It is unclear how the user specifies a trust structure efficiently or how this is encoded within a scheme, for example. More importantly, implementations and benchmarks do not exist, hence the efficiency of the schemes is not known.

Our work fills this gap. We show how an administrator can intuitively describe the access structure as a Boolean formula. This is then converted into encodings suitable for cryptographic primitives, specifically, into a tree data structure and a monotone span program. We focus on three general distributed cryptographic schemes: *verifiable secret sharing*, *common coin*, and *distributed signatures*. For each one we give the appropriate formalization and security definition in the general-trust setting. We implement the schemes and assess their efficiency against their threshold counterparts. Our results suggest that the general distributed schemes can offer richer expressibility at no or insignificant extra cost. Thus, they are appropriate and ready for practical deployment.

1 Introduction

1.1 Motivation

Throughout the last decade, largely due to the advent of blockchains, there has been an ever-increasing interest in distributed systems and practical cryptographic primitives. Naturally, the type of cryptography most suitable for distributed systems is distributed cryptography: independent parties jointly hold a secret key and perform some cryptographic task.

Many deployments of distributed cryptography exist today. Threshold signature schemes [24, 9] distribute the signing power among a set of parties. They have been used in state-machine replication (SMR) protocols, where they serve as unique and constant-size vote certificates [65, 45]. Furthermore, random-beacon and common-coin schemes [19, 13] provide a source of reliable and distributed randomness. In SMR protocols they facilitate, among other tasks, leader election [14, 21, 52] and sharding [66, 37]. As a third example, multiparty computation (MPC) is a cryptographic tool that enables a group of parties to compute a function of their private inputs. It finds applications in protecting digital assets[1], or pri-

[1] *Fireblocks*: https://www.fireblocks.com, *Sepior*: https://sepior.com

vate keys and cryptocurrency wallets[2], often worth millions of dollars. Applications also include highly sensitive and private data[3], related, for example, to DNA[4] or efforts against human trafficking [20]. Security is, hence, of paramount importance. MPC has been combined with blockchains to enable private computations [8] and fairness [41, 3].

One can thus say that we are in the era of distributed cryptography. However, all currently deployed distributed-cryptographic schemes express their trust assumptions through a number, with a threshold, hence reducing to the setting of *threshold cryptography*, where all parties may misbehave with the same probability. In other words, parties are considered identical, leading to a monoculture-type view of the system. On the other hand, distributed cryptography does not have to be threshold-based. In *general distributed cryptography* the *authorized sets*, the sets of parties sufficient to perform the task, can be arbitrary, and are specified through a general, non-threshold *access structure* (AS). Our position is that general distributed cryptography is essential for distributed systems.

Increasing systems resilience and security. First, general distributed cryptography has the capacity to increase the resilience of a system, as failures are, in practice, always correlated [64]. Cyberattacks, exploitation of specific implementation vulnerabilities, zero-day attacks, and so on very seldom affect all parties in an identical way — they often target a specific operating systems or flavor of it, a specific hardware vendor, or a specific software version. Similarly, attackers may compromise specific parties more easily, due to different administrator policies or different levels of cyber and physical security. In another example, blockchain nodes are typically hosted by cloud providers or mining farms, hence failures are correlated there as well. Such failure correlations are known and have been observed; they can be expressed in a system that supports general trust, significantly increasing resilience and security.

Let us now see a concrete example of how such correlations can be captured. Cachin [12] describes an AS where parties are differentiated in two dimensions, based on their location and operating system (OS). In an instantiation with 16, possibly Byzantine, parties, organized in four locations and four OS, the AS tolerates the *simultaneous* failure of all parties in one location and all parties with a specific OS. Hence, it encodes specific knowledge and correlation patterns, and can even tolerate executions with up to seven failed parties, something not possible in the threshold setting, where only five out of the 16 may fail. Once general distributed cryptography is deployed, this example can be generalized to any number of parties and dimensions.

Facilitating personal assumptions and Sybil resistance. Some works in the area of distributed systems generalize trust assumptions in yet another dimension: they allow each party to specify its own. The consensus protocol of Stellar [40], implemented in the Stellar blockchain[5], allows each party to specify the access structure of its choice, which can consist of arbitrary sets and nested thresholds. Similarly, the consensus protocol implemented by Ripple [57] in the XRP ledger[6] also allows each party to choose who it trusts and communicates with. In both networks, the resulting representation of trust in the system, obtained when the trust assumptions of all parties are considered together, can only be expressed as a generalized structure. Hence, current threshold-cryptographic schemes cannot be integrated or used on top of these networks. For example, a common coin scheme — necessary for achieving consensus in asynchronous networks — would need to support general trust. In addition to that, practical and easy to deploy general distributed cryptographic schemes can function as a catalyst for more applications built on top of these blockchains.

Another feature of both Stellar and Ripple is that they achieve open membership without employing a proof-of-work or proof-of-stake mechanism. That is, they achieve Sybil resistance by allowing a party to selectively trust or ignore other parties. This approach can lead to more efficient, less energy consuming, and arguably more open and inclusive blockchains. As described earlier, however, this results in trust assumptions where parties are not treated as identical. Departing from a threshold mindset towards

[2] *DFNS*: https://www.dfns.co/, *Keyless*: https://keyless.io, *Zengo*: https://zengo.com, *Unbound security*: https://github.com/unboundsecurity [3] *Sharemind*: https://sharemind.cyber.ee, *Partisia*: https://partisia.com
[4] https://partisia.com/better-data-solutions/surveys [5] https://www.stellar.org/ [6] https://ripple.com/

general access structures is, thus, a prerequisite for wider adoption.

1.2 State of the art

In this work we focus on three important distributed-cryptographic primitives for distributed protocols.

Verifiable secret sharing. *Secret sharing* [58] allows a dealer to share a secret in a way that only authorized sets can later reconstruct it. *Verifiable Secret Sharing (VSS)* [31, 51] additionally allows the parties to verify their shares against a malicious dealer.

Common coin. A *common coin* [54, 13] scheme allows a set of parties to calculate a pseudorandom function $\mathcal{U}$, mapping coin names C to uniformly random bits $\mathcal{U}(C) \in \{0, 1\}$ in a distributed way.

Distributed signatures. A *distributed signature* [24, 9] scheme allows a set of parties to collectively sign a message. The parties hold key shares of an unknown private key and create signature shares on individual messages. Once sufficient signature shares are available, they are combined into a unique distributed signature, which can be verified with the standard algorithm of the underlying signature scheme.

The generalization of threshold-cryptographic schemes to any linear access structure is known and typically employs Monotone Span Programs (MSP) [36], a linear-algebraic model of computation. General schemes using the MSP have already been described in theory [17, 48, 44, 29]. However, no implementations or deployments exist yet, despite all the merit of general distributed cryptography. In our point of view, the reasons are the following.

- Essential implementation details are missing, and questions related to the usability of general schemes have never been answered in a real system. How can the trust assumptions, initially only in the mind of an administrator, be encoded in a cryptographic scheme? How does the system administrator efficiently do this? Previous general distributed schemes assume the MSP is given to all algorithms, but how is this built from the trust assumptions? Usability is a necessary ingredient for the adoption of a new technological setting, and usability in turn leads to increased security.
- Some distributed cryptographic schemes, such as VSS, have been described in models weaker than the MSP. Others, such as common coins, have, to the best of our knowledge, not been formulated with general trust assumptions. Can we describe and prove all schemes of interest in a unified language? This would facilitate the understanding of general distributed cryptography and pave the way for easier standardization and implementation.
- Most importantly, implementations and benchmarks do not exist, hence the efficiency of general schemes is not known. What is the concrete efficiency of the MSP? How does a generalized scheme compare to its threshold counterpart? How much efficiency needs to be "sacrificed" in order to support general trust?

1.3 Contributions

The goal of this work is to bridge the gap between theory and practice by answering the aforementioned questions, so as to pave the way for the adoption of general distributed cryptography.

- We explore intuitive ways for an administrator to specify the trust assumption, starting from a collection of sets or a Boolean formula, described in a JSON file. This is then converted into two different encodings, a tree data structure and an MSP, the former used for checking whether a set of parties is authorized and the latter for all algebraic operations. An algorithm and its efficiency are shown for building the MSP from the user input. Finally, the practicality of these encodings is validated through examples, among which an access structure used in the live Stellar blockchain.
- In this work, we first recall a general VSS scheme. We then extend the common-coin construction of Cachin, Kursawe, and Shoup [13] into the general-trust model. Moreover, we present a general distributed-signature scheme based on BLS signatures [10], which extends the threshold scheme

of Boldyreva [9]. All schemes are in the MSP model, and we provide security definitions and proofs that are appropriate for the general-trust setting.

- We implement and benchmark the aforementioned schemes, both threshold and general versions. We assess the efficiency of the general schemes and provide detailed explanation of the observed behavior, insights, and possible optimizations. The benchmarks include multiple trust assumptions, thereby exploring how they affect the efficiency of the schemes.

1.4 Related work

General distributed cryptography. Secret sharing over arbitrary access structures has been extensively studied in theory. The first scheme is presented by Ito, Saito, and Nishizeki [35], where the secret is shared independently for every authorized set. Benaloh and Leichter [7] use monotone Boolean formulas to express the access structure and introduce a recursive secret-sharing construction. Gennaro presents a general VSS scheme [29], where trust is specified as Boolean formulas in disjunctive normal form. As a result, a party receives as many shares as the number of conjunctions it appears in. Choudhury presents general asynchronous VSS and common-coin schemes secure against a computationally-unbounded adversary [16].

Later, the *Monotone Span Program (MSP)* is introduced [36] as a linear-algebraic model of computation. Since then, VSS schemes with general access structures have been formulated in terms of an MSP. In the information-theoretic setting, Cramer, Damgård, and Maurer [17] construct a VSS scheme for any monotone access structure. Nikov *et al.* [48] extend this work to add proactive resharing. A general VSS scheme is also presented by Mashhadi, Dehkordi, and Kiamari [44], which requires multiparty computation for share verification.

A different line of work encodes the access structure using a *vector-space secret-sharing scheme* [11], a special case of an MSP.[7] Specifically, Herranz and Sáez [33] construct a VSS scheme based on Pedersen's VSS [51]. Herranz, Padró, and Sáez [32] construct general distributed RSA signatures based on the threshold RSA scheme of Shoup [60]. Distributed key generation schemes have also been described based on vector-space secret sharing [22, 23].

Attribute-based signatures. ABS schemes [42] are related to distributed signatures. In ABS a signer possesses a number of attributes and can only produce a valid signature if they satisfy a certain predicate on the set of all attributes. ABS schemes are similar to distributed signatures in that they usually encode the attribute predicate as an MSP, but differ from distributed signatures in terms of security requirements (they have to consider attribute privacy and adaptive attribute selection), and hence result in more complicated schemes [49, 39].

Common coin schemes. *Common coin* schemes (also called *shared coins*, *coin tossing* schemes, or *random beacons* [25, 19]) model randomness produced in a distributed way. Multiple threshold schemes have been proposed in the literature [19, 13, 54] and are used in practice [25]. Raikwar and Gligoroski [55] present an overview and classification. Our work extends the common-coin scheme of Cachin, Kursawe, and Shoup [13]. The same threshold construction appears in DiSE [1, Figure 6], where it is modeled as a DPRF [46]. The scheme outputs an unbiased value.

Lower bounds for general secret sharing. Superpolynomial lower bounds are known for MSPs [56, 4] and general secret sharing [38]. As the focus of this work is on practical aspects, we assume that the administrator can, in the first place, efficiently describe the trust assumptions, either as a collection of sets or as a Boolean formula. Arguably, access structures of practical interest fall in this category. Moreover, it is known that MSPs are more powerful than Boolean formulas and circuits. Babai, Gál, and Wigderson [4] prove the existence of monotone Boolean functions that can be computed by a linear-size

[7] A vector-space secret-sharing scheme can be seen as an MSP where each party owns exactly one row. The MSP is, hence, a stronger model as it can encode any access structure [36, 6].

MSP but only by exponential-size monotone Boolean formulas. In those cases the MSP can be directly plugged into a generalized scheme.

2 Background and model

Notation. A bold symbol $\boldsymbol{a}$ denotes a vector of some dimension in $\mathbb{N}^+$. However, we avoid distinguishing between $\boldsymbol{a}$ and $\boldsymbol{a}^\intercal$, that is, $\boldsymbol{a}$ denotes both a row and a column vector. Moreover, for vectors $\boldsymbol{a} \in \mathcal{K}^{|a|}$ and $\boldsymbol{b} \in \mathcal{K}^{|b|}$, where $\mathcal{K}$ is a field, $\boldsymbol{a}||\boldsymbol{b} \in \mathcal{K}^{|a|+|b|}$ denotes their concatenation, and a_i is short for $\boldsymbol{a}[i]$. The notation $x \overset{\$}{\leftarrow} S$ means that x is chosen uniformly at random from set S. The set of all parties is $\mathcal{P} = \{p_1, \ldots, p_n\}$.

Adversary structures [34] and access structures [7]. An *adversary structure* $\mathcal{F}$ is a collection of all *unauthorized* subsets of $\mathcal{P}$, and an *access structure (AS)* $\mathcal{A}$ is a collection of all *authorized* subsets of $\mathcal{P}$. Both are monotone. Any subset of an unauthorized set is unauthorized, i.e., if $F \in \mathcal{F}$ and $B \subset F$, then $B \in \mathcal{F}$, and any superset of an authorized set is authorized, i.e., if $A \in \mathcal{A}$ and $C \supset A$, then $C \in \mathcal{A}$. As in the most general case [34] we assume that any set not in the access structure can be corrupted by the adversary, that is, the adversary structure and the access structure are the complement of each other. We say that $\mathcal{F}$ is a Q^2 *adversary structure* if no two sets in $\mathcal{F}$ cover the whole $\mathcal{P}$.

Corruption model. In all schemes we assume that the adversary structure $\mathcal{F}$, implied by the access structure $\mathcal{A}$, is a Q^2 adversary structure. The adversary is Byzantine and static, and corrupts a set $F \in \mathcal{F}$ which is, w.l.o.g, maximally unauthorized, i.e., there is no $F' \in \mathcal{F}$ such that $F' \supset F$.

Monotone span programs [36]. *Monotone span programs* (MSP) have been introduced as a linear-algebraic model of computation. Given a finite field $\mathcal{K}$ and a set of parties $\mathcal{P}$, an MSP is a tuple (M, ρ), where M is an $m \times d$ matrix over $\mathcal{K}$ and ρ is a surjective function $\{1, \ldots, m\} \to \mathcal{P}$ that labels each row of M with a party. We say that party p_i *owns* row $j \in \{1, \ldots, m\}$ if $\rho(j) = p_i$. The *size* of the MSP is m, the number of its rows. Finally, the fixed vector $\boldsymbol{e}_1 = [1, 0, \ldots, 0] \in \mathcal{K}^d$ is called the *target vector*.

For any set $A \subseteq \mathcal{P}$ we define M_A to be the $m_A \times d$ matrix obtained from M by keeping only the rows owned by parties in A, i.e., rows j with $\rho(j) \in A$. Let $M_A^\intercal$ denote the transpose of M_A and $\mathit{Im}(M_A^\intercal)$ the span of the rows of M_A. We say that the MSP *accepts* A if the rows of M_A span $\boldsymbol{e}_1$, i.e., $\boldsymbol{e}_1 \in \mathit{Im}(M_A^\intercal)$. Equivalently, there is a *recombination vector* $\boldsymbol{\lambda}_A$ such that $\boldsymbol{\lambda}_A M_A = \boldsymbol{e}_1$. Otherwise, the MSP *rejects* A.

For any access structure $\mathcal{A}$, we say that an MSP *accepts* $\mathcal{A}$ if it accepts exactly the authorized sets $A \in \mathcal{A}$. It has been proven that each MSP accepts exactly one monotone access structure and each monotone access structure can be expressed in terms of an MSP [6, 36]. Hence, an MSP uniquely defines an access structure, which in turn implies an adversary structure.

Algorithm 1 (Linear secret-sharing scheme). A *linear secret-sharing scheme* (LSSS) over a finite field $\mathcal{K}$ shares a secret $x \in \mathcal{K}$ using a *coefficient vector* $\boldsymbol{r}$, in such a way that every share is a linear combination of x and the entries of $\boldsymbol{r}$. Linear secret-sharing schemes are equivalent to monotone span programs [6, 36]. We formalize an LSSS as two algorithms, *Share()* and *Reconstruct()*.

1. *Share*(x). Choose uniformly at random $d-1$ elements $r_2, \ldots, r_d$ from $\mathcal{K}$ and define the *coefficient vector* $\boldsymbol{r} = (x, r_2, \ldots, r_d)$. Calculate the secret shares $\boldsymbol{x} = (x_1, \ldots, x_m) = M\boldsymbol{r}$. Each x_j, with $j \in [1, m]$, belongs to party $p_i = \rho(j)$. Hence, p_i receives in total m_j shares, where m_j is the number of MSP rows owned by p_i.
2. *Reconstruct*$(A, \boldsymbol{x}_A)$. To reconstruct the secret given an authorized set A and the shares $\boldsymbol{x}_A$ of parties in A, find the recombination vector $\boldsymbol{\lambda}_A$ and compute the secret as $\boldsymbol{\lambda}_A \boldsymbol{x}_A$.

A secret-sharing schemes satisfies two properties. The first is *correctness*, which demands that any authorized set $A \in \mathcal{A}$ can reconstruct the secret. It is satisfied by construction of the MSP, which accepts the access structure $\mathcal{A}$. The second is *privacy*, stating any unauthorized set $F \in \mathcal{F}$ obtains no information about the secret. This is formalized by the following lemma.

Lemma 1 (Privacy of linear secret-sharing schemes [36]). *Let $\mathcal{M} = (M, \rho)$ be an MSP over finite field $\mathcal{K}$, which accepts the access structure $\mathcal{A}$, and F an unauthorized set, i.e. $F \notin \mathcal{A}$, with shares $\boldsymbol{x}_F = M_F \boldsymbol{r}$. Then, for every secret $\tilde{x} \in \mathcal{K}$ there exists a coefficient vector $\tilde{\boldsymbol{r}}$ which shares the secret $\tilde{x}$, i.e., $\tilde{r}_1 = \tilde{x}$, and satisfies $\boldsymbol{x}_F = M_F \tilde{\boldsymbol{r}}$.*

Computational assumptions. Let $G = \langle g \rangle$ be a group of prime order q and $x_0 \xleftarrow{\$} \{0, \ldots, q-1\}$. The *Discrete Logarithm (DL)* assumption is that no efficient probabilistic algorithm, given $g_0 = g^{x_0} \in G$, can compute x_0, except with negligible probability. The *Computational Diffie-Hellman (CDH)* assumption is that no efficient probabilistic algorithm, given $g, \hat{g}, g_0 \in G$, where $\hat{g} \xleftarrow{\$} G$ and $g_0 = g^{x_0}$, can compute $\hat{g}_0 = \hat{g}^{x_0}$, except with negligible probability.

Definition 1 (Gap Diffie-Hellman group [10]). Let $G_1 = \langle g_1 \rangle$ and $G_2 = \langle g_2 \rangle$ be two groups of prime order q, and $h \xleftarrow{\$} G_1$. Let $\alpha, \beta \xleftarrow{\$} \{0, \ldots, q-1\}$.

- The *computational co-Diffie-Hellman (co-CDH)* problem on (G_1, G_2) asks, on input $g_2, g_2^\alpha \in G_2$ and $h \in G_1$, to compute $h^\alpha \in G_1$.
- The *decisional co-Diffie-Hellman (co-DDH)* problem on (G_1, G_2) asks, on input $g_2, g_2^\alpha \in G_2$ and $h, h^\beta \in G_1$, to output TRUE if $\alpha = \beta$ and FALSE otherwise. In the first case we say that $(g_2, g_2^\alpha, h, h^\alpha)$ is a *co-Diffie-Hellman tuple*.
- We say that (G_1, G_2) is a *Gap co-Diffie-Hellman (co-GDH)* group pair if co-DDH is easy but co-CDH is hard to solve on (G_1, G_2). For a more formal definition we refer the reader to [10].

3 Specifying and encoding the trust assumptions

An important aspect concerning the implementation and deployment of general distributed cryptography is specifying the Access Structure (AS). We require a solution that is intuitive, so that users or administrators can easily specify it, that facilitates the necessary algebraic operations, such as computing and recombining secret shares, and in the same time offers an efficient way to check whether a given set is authorized.

The administrator first specifies the access structure as a monotone Boolean formula, which consists of *and*, *or*, and *threshold* operators. A *threshold* operator $\Theta_k^K(q_1, \ldots, q_K)$ specifies that any subset of $\{q_1, \ldots, q_K\}$ with cardinality at least k is authorized, where each q_i can be a party identifier or a nested operator. Observe that the *and* and *or* operators are special cases of this, but we allow them as well for better usability. We remark that the representation as a monotone Boolean formula also includes the case where the access structure is initially given as a collection of sets. That is, if $A_1, A_2, \ldots, A_m$ are the authorized sets, and $A_i = \{p_{i_1}, p_{i_2}, \ldots, p_{i_{m_i}}\}$, then this can be seen as Boolean formula in disjunctive normal form, $f = A_1 \vee A_2 \vee \ldots \vee A_m$, where $A_i = \{p_{i_1} \wedge p_{i_2} \wedge \ldots \wedge p_{i_{m_i}}\}$. Hence, we assume the AS can be described as a monotone Boolean formula. This is an intuitive format and can be easily specified in JSON format, as shown in the examples that follow.

The next step is to internally encode the access structure within a scheme. For this we use two different encodings. First, the Boolean formula is encoded as a tree, where a node represents an operator and its children are the operands. The size of the tree is linear in the size of the Boolean formula. Checking whether a set is authorized consists in a depth-first traversal of the tree, and hence takes time linear in the size of the tree. This data structure allows for efficient evaluation of the Boolean formula, The second is Monotone Span Program (MSP), which is the basis for all our general distributed cryptographic primitives. The MSP is directly constructed from the JSON-encoded Boolean formula. Both are made available to all parties.

Building the MSP from a monotone Boolean formula [47, 2]. We now describe how an MSP can be constructed given a monotone Boolean formula. The details of the algorithm can be found in Appendix A. We use a recursive insertion-based algorithm. The main observation is that the t-of-n threshold access structure is encoded by an MSP $\mathcal{M} = (M, \rho)$ over finite field $\mathcal{K}$, with M being the $n \times t$ Vandermonde matrix

$$V(n,t) = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{t-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{t-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{t-1} \end{pmatrix},$$

for $x_i \in \mathcal{K}$ pairwise different and nonzero. The algorithm parses the Boolean formula as a sequence of nested *threshold* operators (*and* and *or* are special cases of *threshold*). Starting from the outermost operator, it constructs the Vandermonde matrix that implements it and then recursively performs *insertions* for the nested threshold operators. In a high level, an *insertion* replaces a row of M with a second MSP M' (which encodes the nested operator) and pads with 0 the initial matrix M, in case M' is wider than M.

If the Boolean formula includes in total c operators in the form $\Theta_{d_i}^{m_i}$, then the final matrix M of the MSP that encodes it has $m = \sum_1^c m_i - c + 1$ rows and $d = \sum_1^c d_i - c + 1$ columns, hence size linear in the size of the formula.

Example 1. Recent work [27] presents the example of an *unbalanced-AS*[8], where n parties in $\mathcal{P}$ are distributed into two organizations $\mathcal{P}_1$ and $\mathcal{P}_2$, and the adversary is expected to be within one of the organizations, making it easier to corrupt parties from that organization. They specify this with two thresholds, t and k, and allow the adversary to corrupt at most t parties from $\mathcal{P}$ and in the same time at most k parties from $\mathcal{P}_1$ or $\mathcal{P}_2$. For example, we can set $t = \lfloor n/2 \rfloor$ and $k = \lfloor t/2 \rfloor - 1$. let $n = 9$, $\mathcal{P}_1 = \{p_1, \ldots, p_5\}, \mathcal{P}_2 = \{p_6, \ldots, p_9\}, t = 4$, and $k = 1$. The access structure (taken as the complement of the adversary structure) is $\mathcal{A} = \{A \subset \mathcal{P} : |A| > 4 \vee (|A \cap \mathcal{P}_1| > 1 \wedge |A \cap \mathcal{P}_2| > 1)\}$. In terms of a monotone Boolean formula, this can be written as $F_{\mathcal{A}} = \Theta_5^9(\mathcal{P}) \vee \left(\Theta_2^5(\mathcal{P}_1) \wedge \Theta_2^4(\mathcal{P}_2)\right)$. The MSP constructed with the given algorithm has $m = 2n$ rows and $d = t + 2k + 2 = n - 1$ columns.

Example 2. Another classical general AS from the field of distributed systems is the *M-Grid* [43]. Here $n = k^2$ parties are arranged in a $k \times k$ grid and up to $b = k - 1$ Byzantine parties are tolerated. An authorized set consists of any t rows and t columns, where $t = \lceil \sqrt{b/2 + 1} \rceil$.[9] Let us set $n = 16$ and, hence, $k = 4$, $b = 3$, and $t = 2$. This means that and any two rows and two columns (twelve parties in total) make an authorized set. The Boolean formula that describes this AS is $F_{\mathcal{A}} = \Theta_2^4 \left(\Theta_4^4(R_1), \Theta_4^4(R_2), \Theta_4^4(R_3), \Theta_4^4(R_4)\right) \wedge \Theta_2^4 \left(\Theta_4^4(C_1), \Theta_4^4(C_2), \Theta_4^4(C_3), \Theta_4^4(C_4)\right)$, where R_ℓ and C_ℓ denote the sets of parties at row and column ℓ, respectively. We call this access structure the *grid-AS*.

Example 3. The Stellar blockchain supports general trust assumptions for consensus [40]. Each party can specify its own access structure, which is composed of nested threshold operators. We extract[10] the AS of one Stellar validator and show in Figure 1 a JSON file that can be used in our general schemes. It can be directly translated into an MSP, enabling general distributed cryptography in or on top of the blockchain of Stellar. The MSP constructed with the presented algorithm has $m = 25$ rows and $d = 15$ columns.

4 Verifiable secret sharing

In this section we recall a general Verifiable Secret Sharing (VSS) scheme [33]. It generalizes Pedersen's VSS [31, 51] to the general setting.

[8] This is a special case of bipartite AS [50]. [9] The conditions on b and t of an M-Grid for the so-called *dissemination Byzantine quorum systems* have been stated by Alpos and Cachin [2]. [10] https://www.stellarbeat.io/, https://api.stellarbeat.io/docs/

```
{  "select": 6,
   "out-of": [
     {"select": 2, "out-of": ["Blockdaemon1", "Blockdaemon2", "Blockdaemon3"]},
     {"select": 2, "out-of": ["SDF1", "SDF2", "SDF3"]},
     {"select": 2, "out-of": ["WirexSingapore", "WirexUK", "WirexUS"]},
     {"select": 2, "out-of": ["CoinqvestFinland", "CoinqvestHongKong", "CoinqvestGermany"]},
     {"select": 2, "out-of": ["SatoshiPayUS", "SatoshiPaySG", "SatoshiPayDE"]},
     {"select": 2, "out-of": ["FranklinTempleton1", "FranklinTempleton2", "FranklinTempleton3"]},
     {"select": 3, "out-of": ["LOBSTR1", "LOBSTR2", "LOBSTR3", "LOBSTR4", "LOBSTR5"]},
     {"select": 2, "out-of": ["Hercules", "Lyra", "Boötes"]}
]}
```

Figure 1. A JSON file that specifies the access structure of the SDF1 validator in the live Stellar blockchain (we use the literals returned by Stellar as party identifiers).

Security. The security of a general VSS scheme is formalized by the following properties (in analogy with the threshold setting [31, 51]).

1. Completeness. If the dealer is not disqualified, then all honest parties complete the sharing phase and can then reconstruct the secret.
2. Correctness. For any authorized sets A_1 and A_2 that have accepted their shares and reconstruct secrets z_1 and z_2, respectively, with overwhelming probability it holds that $z_1 = z_2$. Moreover, if the dealer is honest, then $z_1 = z_2 = s$.
3. Privacy. Any unauthorized set F has no information about the secret.

The scheme. The scheme is synchronous and uses the same communication pattern as the standard VSS protocols [31, 51]. Hence complaints are delivered by all honest parties within a known time bound, and we assume a broadcast channel, to which all parties have access. Let $G = \langle g \rangle$ be a group of large prime order q and $h \xleftarrow{\$} G$.

1. *Share(x)*. The dealer uses Algorithm 1 to compute the *secret-shares* $\boldsymbol{x} = (x_1, \dots, x_m) = LSSS.$*Share($x$)*. The dealer also chooses a random value $x' \in \mathbb{Z}_q$ and computes the *random-shares* $\boldsymbol{x}' = (x'_1, \dots, x'_m) = LSSS.$*Share($x'$)*. Let $\boldsymbol{r} = (x, r_2, \dots, r_d)$ and $\boldsymbol{r}' = (x', r'_2, \dots, r'_d)$ be the corresponding coefficient vectors. The dealer computes commitments to the coefficients $C_1 = g^x h^{x'} \in G$ and $C_\ell = g^{r_\ell} h^{r'_\ell} \in G$, for $\ell = 2, \dots d$, and broadcasts them. The *indexed share* (j, x_j, x'_j) is given to party $p_i = \rho(j)$. Index j is included because each p_i may receive more than one such tuples, if it owns more than one row in the MSP. We call a *sharing* the set of all indexed shares $X_i = \{(j, x_j, x'_j) \mid \rho(j) = p_i\}$ received by party p_i.
2. *Verify(j, x_j, x'_j)*. For each indexed share $(j, x_j, x'_j) \in X_i$, party p_i verifies that
$$g^{x_j} h^{x'_j} = \prod_{\ell=1}^{d} C_\ell^{M_{j\ell}}, \tag{1}$$
where $\boldsymbol{M_j}$ is the j-th row-vector of M and $M_{j\ell}$, for $\ell \in \{1, \dots d\}$, are its entries.
3. *Complain()*. Complaints are handled exactly as in the standard version [31]. Party p_i broadcasts a *complaint* against the dealer for every invalid share. The dealer is disqualified if a complaint is delivered, for which the dealer fails to reveal valid shares.
4. *Reconstruct(A, X_A)*. Given the sharings $X_A = \{(j, x_j, x'_j) \mid \rho(j) \in A\}$ of an authorized set A, a combiner party first verifies the correctness of each share. If a share is found to be invalid, reconstruction is aborted. The combiner constructs the vector $\boldsymbol{x}_A = [x_{j_1}, \dots, x_{j_{m_A}}]$, consisting of the m_A secret-shares of parties in A, and, using Algorithm 1, returns $LSSS.$*Reconstruct($A, \boldsymbol{x}_A$)*.

Theorem 2. *Under the discrete logarithm assumption for group G, the above* general VSS *scheme is secure (satisfies completeness, correctness, and privacy).*

A proof can be found in Appendix C. Completeness holds by construction of the scheme, while correctness reduces to the discrete-log assumption. For the privacy property, we pick arbitrary secrets x and $\tilde{x}$ and show that the adversary cannot distinguish between two executions with secret x and $\tilde{x}$.

5 Common coin

The scheme extends the threshold coin scheme of Cachin, Kursawe, and Shoup [13] to accept any general access structure. It works on a group $G = \langle g \rangle$ of prime order q and uses the following cryptographic hash functions: $H : \{0,1\}^* \to G$, $H' : G^6 \to \mathbb{Z}_q$, and $H'' : G \to \{0,1\}$. The first two, H and H', are modeled as random oracles. The idea is that a secret value $x \in \mathbb{Z}_q$ uniquely defines the value $\mathcal{U}(C)$ of a publicly-known coin name C as follows: hash C to get an element $\tilde{g} = H(C) \in G$, let $\tilde{g}_0 = \tilde{g}^x \in G$, and define $\mathcal{U}(C) = H''(\tilde{g}_0)$. The value x is secret-shared among $\mathcal{P}$ and unknown to any party. Parties can create coin shares using their secret shares. Any party that receives enough coin shares can then obtain $\tilde{g}_0$ by interpolating x in the exponent.

Security. The security of a general common-coin scheme is captured by the following properties (analogous to threshold common coins [13]).

1. Robustness. Except with negligible probability, the adversary cannot produce a coin C and valid coin shares for an authorized set, such that and their combination outputs a value different than $\mathcal{U}(C)$.
2. Unpredictability. Unpredictability is defined through the following game. The adversary corrupts, w.l.o.g, a maximally unauthorized set F. It interacts with honest parties according to the scheme and in the end outputs a coin name C, which was not submitted for coin-share generation to *any* honest party, as well as a coin-value prediction $b \in \{0,1\}$. Then, the probability that $\mathcal{U}(C) = b$ should not be significantly different from $1/2$.

The scheme. It consists of the following algorithms.

1. *KeyGen()*. A dealer chooses uniformly an $x \in \mathbb{Z}_q$ and shares it among $\mathcal{P}$ using the MSP-based LSSS from Algorithm 1, i.e., $\boldsymbol{x} = (x_1, \ldots, x_m) = LSSS.\mathit{Share}(x)$. The secret key x is destroyed after it is shared. We call a *sharing* the set of all key shares $X_i = \{(j, x_j) \mid \rho(j) = p_i\}$ received by party p_i. The verification keys $g_0 = g^x$ and $g_j = g^{x_j}$, for $1 \le j \le m$, are made public.
2. *CoinShareGenerate(C)*. For coin C, party p_i calculates $\tilde{g} = H(C)$ and generates a coin share $\tilde{g}_j = \tilde{g}^{x_j}$ for each key share $(j, x_j) \in X_i$. Party p_i also generates a *proof of correctness* for each coin share, i.e., a proof that $\log_{\tilde{g}} \tilde{g}_j = \log_g g_j$. This is the Chaum-Perdersen proof of equality of discrete logarithms [15] collapsed into a non-interactive proof using the Fiat-Shamir heuristic [28]. For every coin share $\tilde{g}_j$ a valid proof is a pair $(c_j, z_j) \in \mathbb{Z}_q \times \mathbb{Z}_q$, such that

$$c_j = H'(g, g_j, h_j, \tilde{g}, \tilde{g}_j, \tilde{h}_j), \text{ where } h_j = g^{z_j}/g_j^{c_j} \text{ and } \tilde{h}_j = \tilde{g}^{z_j}/\tilde{g}_j^{c_j}. \tag{2}$$

   Party p_i computes such a proof for coin share $\tilde{g}_j$ by choosing s_j at random, computing $h_j = g^{s_j}, \tilde{h}_j = \tilde{g}^{s_j}$, obtaining c_j as in (2), and setting $z_j = s_j + x_j c_j$.
3. *CoinShareVerify*$(C, \tilde{g}_j, (c_j, z_j))$. Verify the proof above.
4. *CoinShareCombine()*. Each party sends its coin sharing $\{(j, \tilde{g}_j, c_j, z_j) \mid \rho(j) = p_i\}$ to a designated combiner. Once valid coin shares from an authorized set A have been received, find the recombination vector $\boldsymbol{\lambda}_A$ for set A and calculate $\tilde{g}_0 = \tilde{g}^x$ as

$$\tilde{g}_0 = \prod_{j \mid \rho(j) \in A} \tilde{g}_j^{\boldsymbol{\lambda}_A[j]}, \tag{3}$$

   where the set $\{j \mid \rho(j) \in A\}$ denotes the MSP indexes owned by parties in A. The combiner outputs $H''(\tilde{g}_0)$.

Theorem 3. *In the random oracle model, the above* general common coin *scheme is secure (robust and unpredictable) under the assumption that CDH is hard in G.*

The proof is presented in Appendix D. In a high level, we assume an adversary that can predict the value of a coin with non-negligible probability and show how to use this adversary to solve the CDH problem in G. The simulator, which is given g, the public key $g_0 = g^x$, and some $\hat{g}$ as a CDH instance, programs the random oracle H to output $\hat{g}$ for some hash query $\hat{C}$ of the adversary. If the adversary succeeds in predicting the value of $\hat{C}$, then the simulator can extract $\hat{g}_0 = \hat{g}^x$, the solution to its CDH input, from the hash query $H''(\hat{g}_0)$ made by the adversary. The proof has to handle specific issues that arise from the general access structures. Specifically, the simulator, given the shares of F, has to create valid shares (sometimes 'hidden' in the exponent) for other parties. As opposed to the threshold case, it can be the case that the shares of F, together with the secret x, do not fully determine all other shares. In that case, the simulator chooses specific shares and assign them values with appropriate distribution. We describe the details for this in Appendix B.

6 Distributed signatures

In a distributed signature scheme parties hold *key shares* of an unknown private key, created with a *KeyGen()* algorithm, run either by a trusted party or in a distributed manner. Using these, they create *signature shares* on individual messages, using algorithm *Sign()*. Once sufficient signature shares are available, they can be combined into a unique *distributed signature*, using algorithm *SigShareCombine()*. Both signature shares and the distributed signature can be verified as a standard signature of the underlying signature scheme, using *SigShareVerify()* and *Verify()*, respectively.

We now show a general distributed-signature scheme based on the BLS signature scheme [10], which extends the threshold scheme of Boldyreva [9] in the general-trust setting. It works with a co-GDH group pair $G_1, G_2 = \langle g_2 \rangle$ with $|G_1| = |G_2| = q$, for q prime.

Security. In accordance with threshold distributed signatures [60], we demand two basic requirements from general distributed signatures, robustness and unforgeability.

1. Robustness. We say that the scheme is *robust* if the adversary cannot prevent the successful termination (creation of a valid general distributed signature).
2. Unforgeability. It is defined through the following game. The adversary corrupts an adversary set $F \in \mathcal{F}$ of its choice. In the dealing phase the adversary receives all the private-key shares owned by parties in F, as well as the public key and all verification keys. After the dealing phase the adversary submits signing requests for messages of its choice to the honest parties. We say that the adversary *forges* a signature if at the end of the game it outputs a valid signature on a message that was not submitted as a signing request to *any* honest party (together with F this would have given the adversary enough signature shares to reconstruct the distributed signature). The scheme is *unforgeable* if it is infeasible for the adversary to forge a signature.

The scheme. It consists of the following algorithms.

1. *KeyGen()*. A trusted dealer chooses random $x \in \mathbb{Z}_q$ as the global and unknown to all parties private key and shares it among $\mathcal{P}$ using the MSP-based LSSS from Algorithm 1, i.e., $\boldsymbol{x} = (x_1, \ldots, x_m) = LSSS.\mathit{Share}(x)$. The public key is $v = g_2^x \in G_2$ and the verification keys are $v_j = g_2^{x_j} \in G_2$, for $1 \leq j \leq m$, and they are published. The *sharing* $X_i = \{(j, x_j) \mid \rho(j) = p_i\}$ is given to p_i.
2. *Sign*(μ, X_j). For each indexed share $(j, x_j) \in X_i$, the owner party p_i calculates an indexed share of the signature (j, σ_j), where $\sigma_j = H(\mu)^{x_j} \in G_1$.
3. *SigShareVerify*(μ, σ_j, v, v_j). Verify that $(g_2, v_j, H(\mu), \sigma_j)$ is a co-Diffie-Hellman tuple.

4. *SigShareCombine*($(j_1, \sigma_{j_1}), \ldots, (j_{m_A}, \sigma_{j_{m_A}})$). Once the indexed signature shares $\sigma_{j_1}, \ldots, \sigma_{j_{m_A}}$ from an authorized group A have been received, recover the distributed signature as $\sigma = \prod_{j \in A} \sigma_j^{\boldsymbol{\lambda}_A[j]}$, where $\boldsymbol{\lambda}_A[j]$ are the entries of the recombination vector that corresponds to A.
5. *Verify*(μ, σ, v). Verify that $(g_2, v, H(\mu), \sigma)$ is a co-Diffie-Hellman tuple.

Theorem 4. *Assuming that standard BLS signatures are secure, the above general distributed signature scheme is secure (robust and unforgeable).*

The proof is presented in Appendix E. We show that the general distributed signature scheme is simulatable. This, together with the unforgeability of the standard BLS scheme, implies the unforgeability property [30, Definition 3].

7 Evaluation

In this section we compare the polynomial-based and MSP-based encodings for trust assumptions, and benchmark the presented schemes on multiple general trust assumptions. To this goal, we benchmark each scheme on four configurations, resulting from different combinations of encoding and access structure (AS), as seen in Table 1. Notice that the first two describe the same threshold AS, encoded once by a polynomial and once an MSP. With the first two configurations we investigate the practical difference between polynomial-based and MSP-based encoding of the same access structure. The last three configurations measure the efficiency we sacrifice for more powerful and expressive AS.

Table 1. Evaluated configurations and corresponding MSP dimensions. Configurations with general AS encode it as an MSP (necessary for algebraic operations, such as sharing and reconstruction) and as a tree (for checking whether a set of parties is authorized).

Configuration	Encoding	Access Structure	MSP dimensions m	d
polynomial $(n+1)/2$	polynomial	$\lceil \frac{n+1}{2} \rceil$-of-$n$	-	-
MSP $(n+1)/2$	MSP+tree	$\lceil \frac{n+1}{2} \rceil$-of-$n$	n	$\lceil \frac{n+1}{2} \rceil$
MSP Unbalanced	MSP+tree	*unbalanced-AS*, Example 1	$2n$	$n-1$
MSP Grid	MSP+tree	*grid-AS*, Example 2	$2n$	$2(n+t-k) \approx 2n$

We implement all presented schemes in C++. The benchmarks only consider CPU complexity, by measuring the time it takes a party to execute each algorithm. Network latency, parallel share verification, and communication-level optimizations are not considered, as they are independent to the encoding of the AS. All benchmarks are made on a virtual machine running Ubuntu 22.04, with 16 GB memory and 8 dedicated CPUs of an AMD EPYC-Rome Processor at 2.3GHz and 4500 bogomips. The number of parties n is always a square, for *grid-AS* to be well-defined, and we report mean values and standard deviation over 100 runs with different inputs.

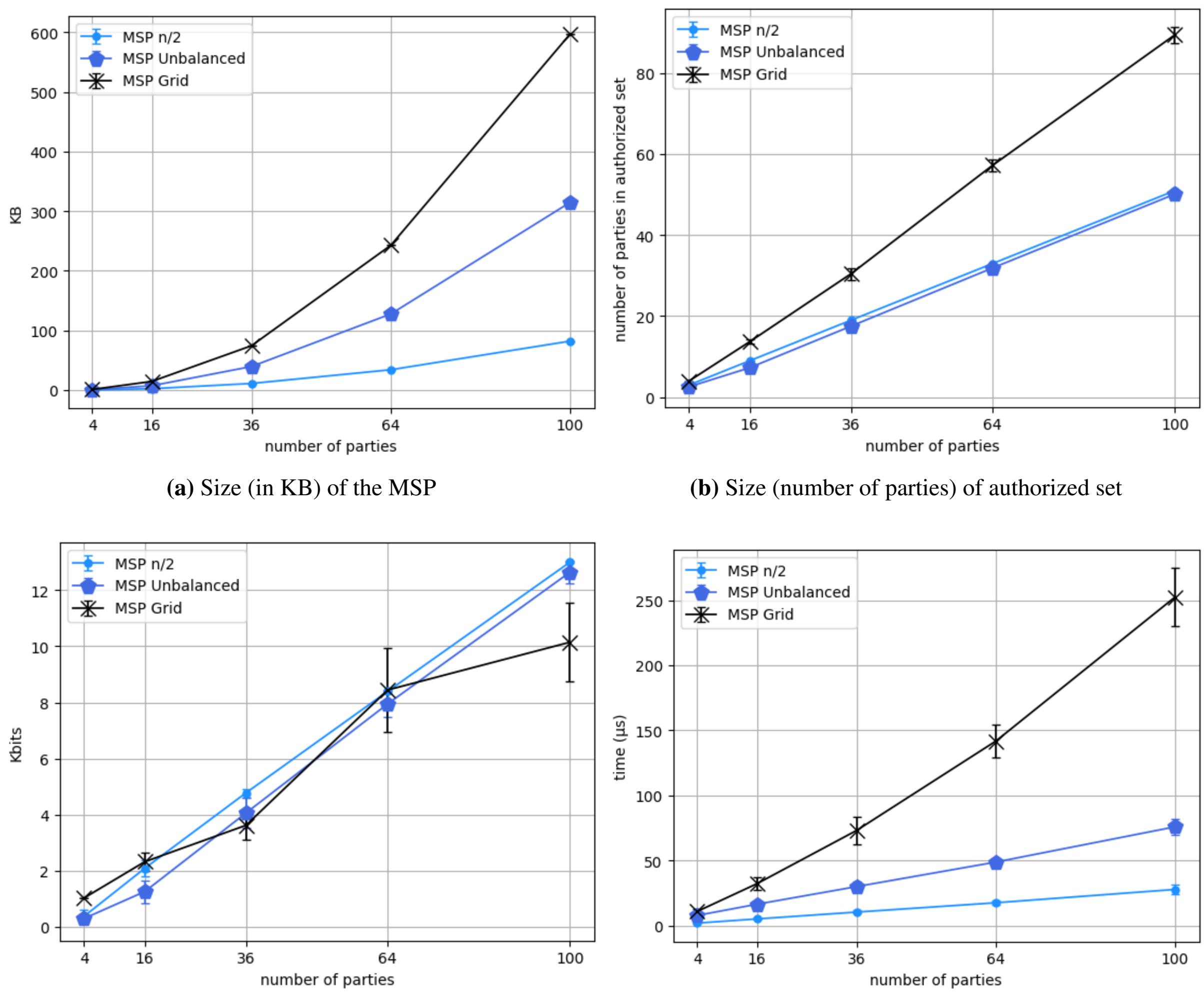
7.1 Benchmarking basic properties of the MSP

We first measure the space (size in KB) needed to store the MSP that describes each general AS. The MSP needs to be stored by every party, as it used to compute the recombination vector. We remark that, by construction of Algorithm 3, an AS described with a large number of nested operators results in a spare MSP matrix. The result for different values of n is shown in Figure 2a.

We next measure the size (as number of parties) of authorized sets for each AS. Authorized sets are obtained in the following way. Starting from an empty set, add a party chosen uniformly from the set of all parties, until the set becomes authorized. This simulates an execution where shares arrive in an arbitrary order, and may result in authorized sets that are not minimal, in the sense that they are supersets of smaller authorized sets, but contain redundant parties. We repeat this experiment 1000 times and report the average size in Figure 2b. For the $\lceil \frac{n+1}{2} \rceil$-of-$n$ AS, of course, authorized sets are always of size $\lceil \frac{n+1}{2} \rceil$. For the *unbalanced-AS* they slightly smaller, and for the *grid-AS* they are significantly larger, as they contain full rows and columns of the grid.

We next measure the bit length of the recombination vector. This is relevant because the schemes involve interpolation in the exponent, exponentiation is an expensive operation, and a shorter recombination vector results in fewer exponentiations. We observe in Figure 2c that the complexity of the AS (in terms of the size of the Boolean formula or the JSON file that describes it) does not necessarily affect the bit length of the recombination vector. There are two important observations to explain Figure 2c. First, each entry of the recombination vector that corresponds to a redundant party is 0, as that share does not contribute to reconstruction. Second, we observe through our benchmarks that, when the MSP is sparse and has entries with short bit length, then the recombination vector also has a short bit length.

Finally, in Figure 2d we report the time it takes to check whether a given set is authorized. This set is chosen uniformly at random among all subsets of $\mathcal{P}$ and an average is taken over 1000 sets. As explained in Section 3, the algorithm that checks for authorized sets uses the tree representation of the AS, as it is more efficient than using the MSP.



(a) Size (in KB) of the MSP

(b) Size (number of parties) of authorized set

(c) Bit length of recombination vector

(d) Time to check for authorized set

Figure 2. Benchmarking basic properties of the MSP, for a varying number of parties. In 2d, the tree representation of the AS is used and the set is chosen uniformly among all subsets of $\mathcal{P}$.

7.2 Running time of verifiable secret sharing

We implement and compare the MSP-based scheme of Section 4 with Pedersen's VSS[11] [51], which we refer to as *general VSS* and *threshold VSS*, respectively. For the *Share()* algorithm we report the time it takes a dealer to share a random secret $s \in \mathbb{Z}_q$, for *Verify()* the average time it takes a party to verify *one* of its shares (notice that in the general scheme a party may receive more than one shares), and for *Reconstruct()* the time it takes a party to reconstruct the secret from an authorized group. For the latter, the group is assumed authorized, i.e., we do not include the time to check whether it is authorized, as this is efficiently done using the tree encoding. The results are shown in Figure 3.

The first conclusion (comparing the first two configurations in Figures 3a and 3b) is that the MSP-based and polynomial-based operations are equally efficient, when instantiated with the same AS. The only exception is the *Reconstruct()* algorithm, shown in Figure 3c, where general VSS is up to two times slower. This is because computing the recombination vector employs Gaussian elimination, which has cubic time complexity. Nevertheless, the reconstruction of the secret only involves operations in field $\mathcal{K}$, which is relatively fast — *Reconstruct()* is an order of magnitude faster than *Verify()*.

The second conclusion (comparing the last three configurations, i.e., the ones that use general trust) is that general VSS is moderately affected by the complexity of the AS. For *Share()*, shown in Figure 3a, more complex AS incur a slowdown because a larger number of shares and commitments have to be created. *Reconstruct()*, in Figure 3c, is also slower with more complex AS, because it performs Gaussian elimination on a larger matrix. We conclude this is the only part of general VSS that cannot be made as efficient as in threshold VSS. On the other hand, *Verify()*, in Figure 3b, exhibits an interesting behavior: the more complex the AS, the faster it is on average to verify *one* share. This might seem counter-intuitive, but can be explained from the observations of Section 7.1; more complex AS result in an MSP with many 0-entries, hence the exponentiations of (1) are faster.

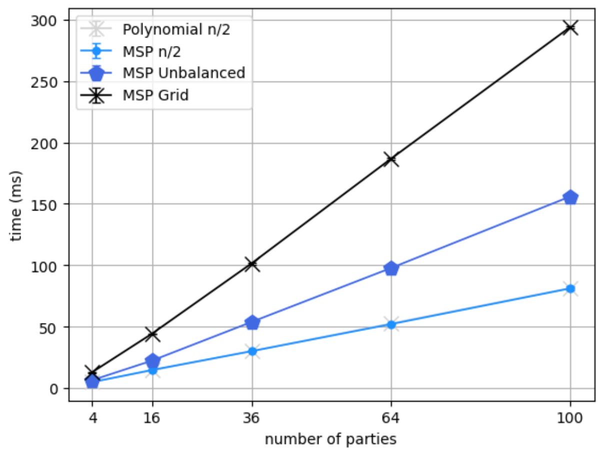
An observation that might be useful for future optimizations is that almost the entire time of *Share()* is spent computing commitments; the dealer computes d commitments, which require $2d$ exponentiations. As shown in Figure 3d, the computation of shares is orders of magnitude faster. Another possible optimization is to parallelize algorithm *Share()*, since the computation of shares and commitments is independent of each other.

7.3 Running time of common coin

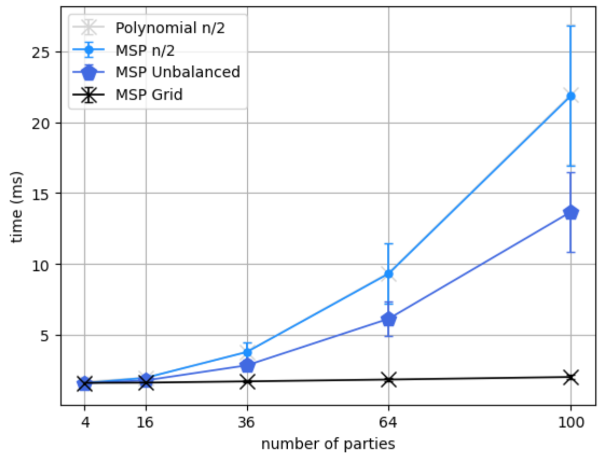
We implement the general scheme of Section 5 and the threshold coin scheme from [13]. For both schemes G is instantiated as an order-q subgroup of $\mathbb{Z}_p$, where $p = qm + 1$, for q a 256-bit prime, p a 3072-bit prime, and $m \in \mathbb{N}$. These lengths offer 128-bit security and are chosen according to current recommendations for discrete logarithm prime fields [26, Chapter 4.5.2] [12]. The arithmetic is done with NTL [62]. The hash functions H, H', H'' use the openSSL implementation of SHA-512 (so that it's not required to expand the digest before reducing modulo the 256-bit q [61, Section 9.2]).

The results are shown in Figure 4. We only show the benchmark of *CoinShareCombine()*, because *KeyGen()* behaves very similar to *Share()* in the VSS, and *CoinShareGenerate()* and *CoinShareVerify()* are identical in the general and threshold scheme (the average time to create and verify, respectively, one coin share was always approximately 4.5ms). In Section 7.2 we observed that *Reconstruct()* was slower for the general scheme, because it involved no exponentiations and the cost of matrix manipulations dominated the running time. Here, however, *CoinShareCombine()* runs similarly in all cases, as the exponentiations in (3) become dominant. As a matter of fact, the general scheme is sometimes faster. This is because complex AS often result in recombination vectors with shorter bit length, as shown in Section 7.1, hence exponentiations are faster.

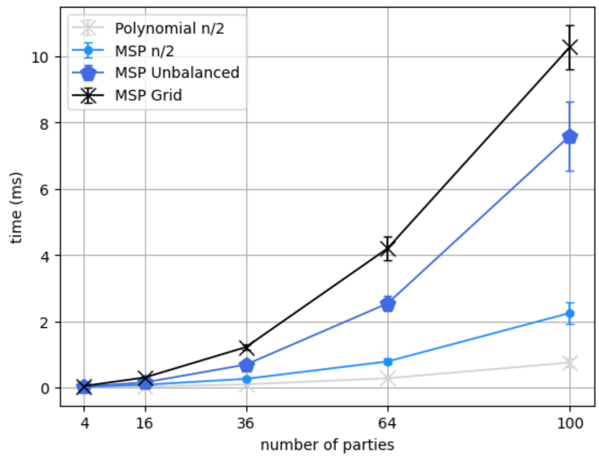
[11] Polynomial evaluation is done without the DFT optimization. [12] Summary of recommendations from multiple organizations: https://www.keylength.com/en/3



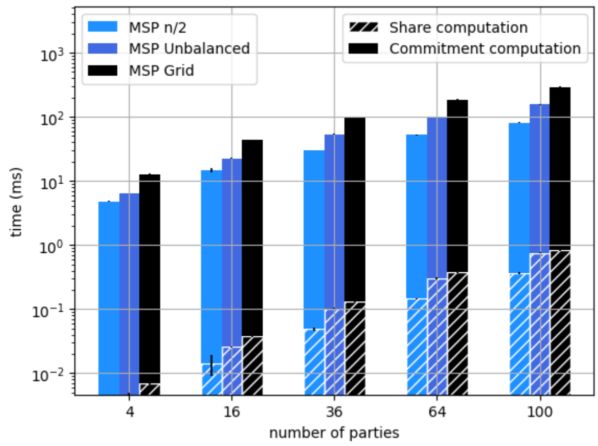
(a) Time taken for *Share()*



(b) Time taken for *Verify()*



(c) Time taken for *Reconstruct()*



(d) Time taken by *Share()* calculating shares and commitments

Figure 3. Time taken by each algorithm in the threshold and general VSS for a varying number of parties. Figure 3a measures the time for a dealer to share a secret, 3b the time for a party to verify *one* of its shares, and 3c the time for a party to reconstruct the secret. Figure 3d compares the time (in logarithmic scale) needed by *Share()* to compute the shares against the time to compute commitments to the shares.

7.4 Running time of distributed signatures

We have implemented the general distributed signature scheme from Section 6. Our extension for generalized operations are made on the *bls* library, which in turn uses *mcl*[13] for pairing operations. The security of these libraries has been reviewed [53] on behalf of the Ethereum Foundation. The benchmarks are done over BLS12-381[5], a widely used pairing-friendly curve offering 128 bits of security [18, Section 4.1].

The observations are similar to those for the previous schemes. Creating and verifying a single signature share, as shown in Figure 5a, does not depend on the scheme or the complexity of the AS, hence the corresponding algorithms run in constant time. On the other hand, *SigShareCombine()*, as shown in Figure 5b, is moderately affected by the complexity of the AS: similar to *Reconstruct()* in the VSS and different from *CoinShareGenerate()* in the common-coin scheme, *SigShareCombine()* does not involve exponentiations, but only calculation of the recombination vector and multiplication of elliptic curve points by constants. For this reason the computation of the recombination vector dominates running

[13] https://github.com/herumi/bls, commit 64d13b9, https://github.com/herumi/mcl, version 1.40.

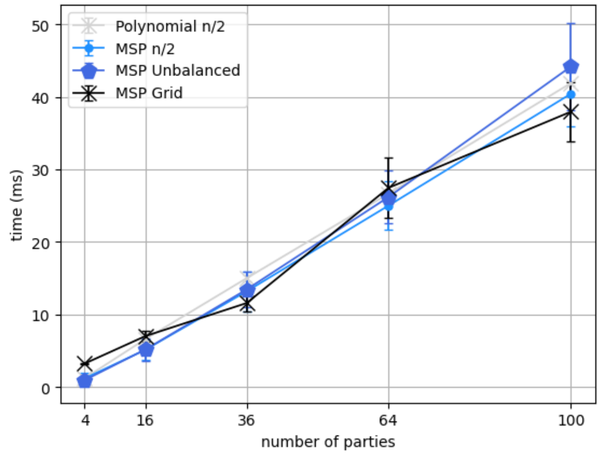
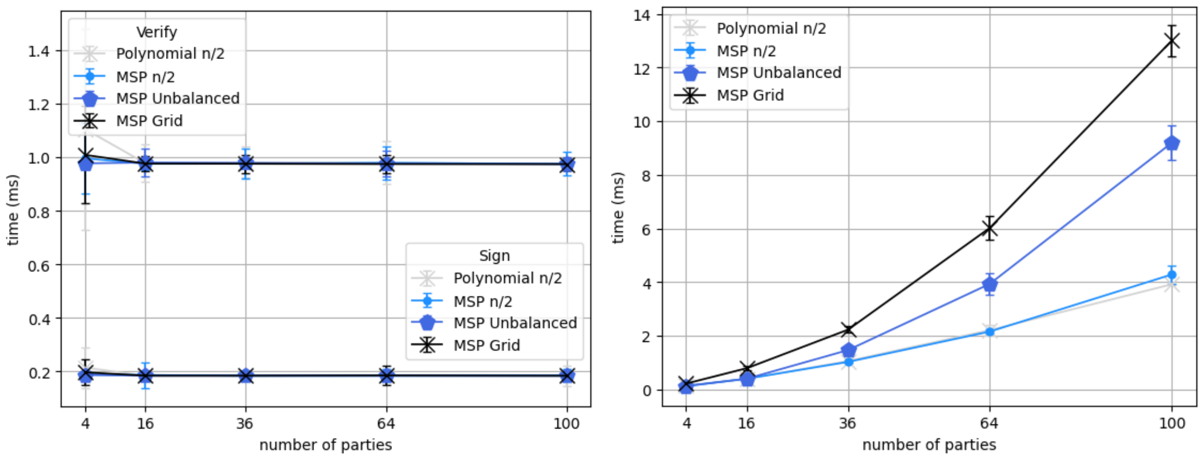


Figure 4. Time taken by *CoinShareCombine()* in the threshold and general coin for a varying number of parties.



(a) Time taken for *Sign()* and *Verify()*

(b) Time taken for *SigShareCombine()*

Figure 5. Time taken by each algorithm in the threshold and general distributed signature scheme for a varying number of parties. Figure 5a measures the time for a party to create and verify *one* signature share and 5b the time to combine an authorized set of signature shares.

time, and *SigShareCombine()* becomes slower on more complex AS.

We finally remark that the general distributed signature scheme is considerably more efficient than the state-of-the-art solution: assuming we have m signatures from an authorized set, the state-of-the-art would require each party to verify all of them. When a scheme with general trust is available, the signatures can first be combined. The cost of combining them remains in all cases much lower than the cost of verifying each one individually.

8 Discussion

Conclusion. In this work we provide the first implementation and practical assessment of distributed cryptography with general trust. We fill all gaps on implementation details and show how a system can be engineered to support general distributed cryptography. We describe, implement, and benchmark distributed cryptographic schemes, specifically, a verifiable secret-sharing scheme, a common-coin scheme, and a distributed signature scheme (as a generalization of threshold signatures), all supporting general trust assumptions. For completeness, we also present the security proofs for all general schemes and handle specific cases that arise from the general trust assumptions (see Theorem 3). Our results suggest that practical access structures can be used with no significant efficiency loss. It can even be the case (VSS share verification, Figure 3b) that operations are on average faster with complex trust structures encoded as Monotone Span Programs (MSP). We nevertheless expect future optimizations, orthogonal to our work, to make MSP operations even faster. Similar optimizations have already been discovered for polynomial evaluation and interpolation [63]. We expect that our work will improve the understanding and facilitate the wider adoption of general distributed cryptography.

Future work. Distributed key generation (DKG) is a significant component in distributed cryptographic schemes. It eliminates the strong assumption of a trusted dealer by distributing this task among the parties. The basic idea is that each party runs an instance of VSS in parallel, sharing a random secret, and then locally adds the shares of the instances that successfully terminated (i.e., their dealer did not get disqualified). The shared secret, which never becomes known to any party, is uniquely determined as the sum of the random secrets of the instances that terminated. This technique can be used in MSP-based DKG protocols, as well, although we leave the formal description of an MSP-based DKG scheme as future work. This boils down to the linearity of MSPs: adding two share vectors $\boldsymbol{z}_1 = M\boldsymbol{r}_1$ and $\boldsymbol{z}_2 = M\boldsymbol{r}_2$, where $\boldsymbol{r}_1[1] = x_1$ and $\boldsymbol{r}_2[1] = x_2$, and then interpolating from some authorized set A will always result in the sum of the two shared secrets, i.e., $\boldsymbol{\lambda}_A(\boldsymbol{z}_1 + \boldsymbol{z}_2) = \boldsymbol{\lambda}_A M(\boldsymbol{r}_1 + \boldsymbol{r}_2) = x_1 + x_2$.

Acknowledgments

This work has been funded by the Swiss National Science Foundation (SNSF) under grant agreement Nr. 200021_188443 (Advanced Consensus Protocols).

References

[1] S. Agrawal, P. Mohassel, P. Mukherjee, and P. Rindal, "Dise: Distributed symmetric-key encryption," in *CCS*, pp. 1993–2010, ACM, 2018.

[2] O. Alpos and C. Cachin, "Consensus beyond thresholds: Generalized byzantine quorums made live," in *SRDS*, pp. 21–30, IEEE, 2020.

[3] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on bitcoin," *Commun. ACM*, vol. 59, no. 4, pp. 76–84, 2016.

[4] L. Babai, A. Gál, and A. Wigderson, "Superpolynomial lower bounds for monotone span programs," *Comb.*, vol. 19, no. 3, pp. 301–319, 1999.

[5] P. S. L. M. Barreto, B. Lynn, and M. Scott, "Constructing elliptic curves with prescribed embedding degrees," in *SCN*, vol. 2576 of *Lecture Notes in Computer Science*, pp. 257–267, Springer, 2002.

[6] A. Beimel, *Secure Schemes for Secret Sharing and Key Distribution*. PhD thesis, Technion, 1996.

[7] J. C. Benaloh and J. Leichter, "Generalized secret sharing and monotone functions," in *CRYPTO*, vol. 403 of *Lecture Notes in Computer Science*, pp. 27–35, Springer, 1988.

[8] F. Benhamouda, C. Gentry, S. Gorbunov, S. Halevi, H. Krawczyk, C. Lin, T. Rabin, and L. Reyzin, "Can a public blockchain keep a secret?," in *TCC (1)*, vol. 12550 of *Lecture Notes in Computer Science*, pp. 260–290, Springer, 2020.

[9] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *Public Key Cryptography*, vol. 2567 of *Lecture Notes in Computer Science*, pp. 31–46, Springer, 2003.

[10] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," *J. Cryptol.*, vol. 17, no. 4, pp. 297–319, 2004.

[11] E. F. Brickell, "Some ideal secret sharing schemes," in *EUROCRYPT*, vol. 434 of *Lecture Notes in Computer Science*, pp. 468–475, Springer, 1989.

[12] C. Cachin, "Distributing trust on the internet," in *DSN*, pp. 183–192, IEEE Computer Society, 2001.

[13] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography," *J. Cryptol.*, vol. 18, no. 3, pp. 219–246, 2005.

[14] J. Camenisch, M. Drijvers, T. Hanke, Y. Pignolet, V. Shoup, and D. Williams, "Internet computer consensus," *IACR Cryptol. ePrint Arch.*, p. 632, 2021.

[15] D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *CRYPTO*, vol. 740 of *Lecture Notes in Computer Science*, pp. 89–105, Springer, 1992.

[16] A. Choudhury, "Almost-surely terminating asynchronous byzantine agreement against general adversaries with optimal resilience," in *ICDCN*, pp. 167–176, ACM, 2023.

[17] R. Cramer, I. Damgård, and U. M. Maurer, "General secure multi-party computation from any linear secret-sharing scheme," in *EUROCRYPT*, vol. 1807 of *Lecture Notes in Computer Science*, pp. 316–334, Springer, 2000.

[18] Crypto Forum Research Group (CFRG), Internet Research Task Force (IRTF), "Pairing-Friendly Curves." https://www.ietf.org/archive/id/draft-irtf-cfrg-pairing-friendly-curves-10.html, 2021.

[19] S. Das, V. Krishnan, I. M. Isaac, and L. Ren, "Spurt: Scalable distributed randomness beacon with transparent setup," in *IEEE Symposium on Security and Privacy*, pp. 2502–2517, IEEE, 2022.

[20] Data Sharing Coalition, "Developing a safe and trusted collaboration environment to monitor and combat human trafficking," 2021. https://datasharingcoalition.eu/2021/developing-a-safe-and-trusted-collaboration-environment-to-monitor-and-combat-human-trafficking.

[21] B. David, P. Gazi, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *EUROCRYPT (2)*, vol. 10821 of *Lecture Notes in Computer Science*, pp. 66–98, Springer, 2018.

[22] V. Daza, J. Herranz, and G. Sáez, "Constructing general dynamic group key distribution schemes with decentralized user join," in *ACISP*, vol. 2727 of *Lecture Notes in Computer Science*, pp. 464–475, Springer, 2003.

[23] V. Daza, J. Herranz, and G. Sáez, "On the computational security of a distributed key distribution scheme," *IEEE Trans. Computers*, vol. 57, no. 8, pp. 1087–1097, 2008.

[24] Y. Desmedt, "Society and group oriented cryptography: A new concept," in *CRYPTO*, vol. 293 of *Lecture Notes in Computer Science*, pp. 120–127, Springer, 1987.

[25] Drand, "A distributed randomness beacon daemon," 2022. https://drand.love.

[26] ECRYPT-CSA, "Algorithms, key size and protocols report," *H2020-ICT-2014 – Project 645421*, 2018. https://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf.

[27] R. Eriguchi and K. Nuida, "Homomorphic secret sharing for multipartite and general adversary structures supporting parallel evaluation of low-degree polynomials," in *ASIACRYPT (2)*, vol. 13091 of *Lecture Notes in Computer Science*, pp. 191–221, Springer, 2021.

[28] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO*, vol. 263 of *Lecture Notes in Computer Science*, pp. 186–194, Springer, 1986.

[29] R. Gennaro, *Theory and practice of verifiable secret sharing*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1996.

[30] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust threshold DSS signatures," in *EUROCRYPT*, vol. 1070 of *Lecture Notes in Computer Science*, pp. 354–371, Springer, 1996.

[31] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *J. Cryptol.*, vol. 20, no. 1, pp. 51–83, 2007.

[32] J. Herranz, C. Padró, and G. Sáez, "Distributed RSA signature schemes for general access structures," in *ISC*, vol. 2851 of *Lecture Notes in Computer Science*, pp. 122–136, Springer, 2003.

[33] J. Herranz and G. Sáez, "Verifiable secret sharing for general access structures, with application to fully distributed proxy signatures," in *Financial Cryptography*, vol. 2742 of *Lecture Notes in Computer Science*, pp. 286–302, Springer, 2003.

[34] M. Hirt and U. M. Maurer, "Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract)," in *PODC*, pp. 25–34, ACM, 1997.

[35] M. Ito, A. Saito, and T. Nishizeki, "Secret sharing scheme realizing general access structure," *Electronics and Communications in Japan*, vol. 72, pp. 56–64, 1989.

[36] M. Karchmer and A. Wigderson, "On span programs," in *Computational Complexity Conference*, pp. 102–111, IEEE Computer Society, 1993.

[37] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *IEEE Symposium on Security and Privacy*, pp. 583–598, IEEE Computer Society, 2018.

[38] K. G. Larsen and M. Simkin, "Secret sharing lower bound: Either reconstruction is hard or shares are long," in *SCN*, vol. 12238 of *Lecture Notes in Computer Science*, pp. 566–578, Springer, 2020.

[39] J. Li, M. H. Au, W. Susilo, D. Xie, and K. Ren, "Attribute-based signature and its applications," in *AsiaCCS*, pp. 60–69, ACM, 2010.

[40] M. Lokhava, G. Losa, D. Mazières, G. Hoare, N. Barry, E. Gafni, J. Jove, R. Malinowsky, and J. McCaleb, "Fast and secure global payments with stellar," in *SOSP*, pp. 80–96, ACM, 2019.

[41] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. K. Miller, "Honeybadgermpc and asynchromix: Practical asynchronous MPC and its application to anonymous communication," in *CCS*, pp. 887–903, ACM, 2019.

[42] H. K. Maji, M. Prabhakaran, and M. Rosulek, "Attribute-based signatures," in *CT-RSA*, vol. 6558 of *Lecture Notes in Computer Science*, pp. 376–392, Springer, 2011.

[43] D. Malkhi, M. K. Reiter, and A. Wool, "The load and availability of byzantine quorum systems," *SIAM J. Comput.*, vol. 29, no. 6, pp. 1889–1906, 2000.

[44] S. Mashhadi, M. H. Dehkordi, and N. Kiamari, "Provably secure verifiable multi-stage secret sharing scheme based on monotone span program," *IET Inf. Secur.*, vol. 11, no. 6, pp. 326–331, 2017.

[45] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT protocols," in *CCS*, pp. 31–42, ACM, 2016.

[46] M. Naor, B. Pinkas, and O. Reingold, "Distributed pseudo-random functions and kdcs," in *EUROCRYPT*, vol. 1592 of *Lecture Notes in Computer Science*, pp. 327–346, Springer, 1999.

[47] V. Nikov and S. Nikova, "New monotone span programs from old," *IACR Cryptol. ePrint Arch.*, p. 282, 2004.

[48] V. Nikov, S. Nikova, B. Preneel, and J. Vandewalle, "On distributed key distribution centers and unconditionally secure proactive verifiable secret sharing schemes based on general access structure," in *INDOCRYPT*, vol. 2551 of *Lecture Notes in Computer Science*, pp. 422–436, Springer, 2002.

[49] T. Okamoto and K. Takashima, "Decentralized attribute-based encryption and signatures," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. 103-A, no. 1, pp. 41–73, 2020.

[50] C. Padró and G. Sáez, "Secret sharing schemes with bipartite access structure," *IEEE Trans. Inf. Theory*, vol. 46, no. 7, pp. 2596–2604, 2000.

[51] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *CRYPTO*, vol. 576 of *Lecture Notes in Computer Science*, pp. 129–140, Springer, 1991.

[52] Protocol Labs, "Filecoin: A decentralized storage network." https://filecoin.io/filecoin.pdf, 2017.

[53] Quarkslab SAS, "Technical assessment of herumi libraries." https://blog.quarkslab.com/resources/2020-12-17-technical-assessment-of-herumi-libraries/20-07-732-REP.pdf, 2020.

[54] M. O. Rabin, "Randomized byzantine generals," in *FOCS*, pp. 403–409, IEEE Computer Society, 1983.

[55] M. Raikwar and D. Gligoroski, "Sok: Decentralized randomness beacon protocols," in *ACISP*, vol. 13494 of *Lecture Notes in Computer Science*, pp. 420–446, Springer, 2022.

[56] R. Robere, T. Pitassi, B. Rossman, and S. A. Cook, "Exponential lower bounds for monotone span programs," in *FOCS*, pp. 406–415, IEEE Computer Society, 2016.

[57] D. Schwartz, N. Youngs, and A. Britto, "The Ripple protocol consensus algorithm." Ripple Labs, available online, https://ripple.com/files/ripple_consensus_whitepaper.pdf, 2014.

[58] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[59] V. Shoup, "Lower bounds for discrete logarithms and related problems," in *EUROCRYPT*, vol. 1233 of *Lecture Notes in Computer Science*, pp. 256–266, Springer, 1997.

[60] V. Shoup, "Practical threshold signatures," in *EUROCRYPT*, vol. 1807 of *Lecture Notes in Computer Science*, pp. 207–220, Springer, 2000.

[61] V. Shoup, *A Computational Introduction to Number Theory and Algebra Version 2*. Cambridge University Press, 2009.

[62] V. Shoup, "Number Theory Library for C++ version 11.5.1," 2020. https://shoup.net/ntl.

[63] A. Tomescu, R. Chen, Y. Zheng, I. Abraham, B. Pinkas, G. Golan-Gueta, and S. Devadas, "Towards scalable threshold cryptosystems," in *IEEE Symposium on Security and Privacy*, pp. 877–893, IEEE, 2020.

[64] W. Vogels, "Life is not a State-Machine." https://www.allthingsdistributed.com/2006/08/life_is_not_a_statemachine.html, 2006.

[65] M. Yin, D. Malkhi, M. K. Reiter, G. Golan-Gueta, and I. Abraham, "Hotstuff: BFT consensus with linearity and responsiveness," in *PODC*, pp. 347–356, ACM, 2019.

[66] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *CCS*, pp. 931–948, ACM, 2018.

A Building the MSP from a monotone Boolean formula

In this section we present the algorithm used in our model to construct the monotone span program (MSP) $\mathcal{M} = (M, \rho)$ for a given monotone Boolean formula (MBF) F. The algorithm is also used in the works of Nikov and Nikova [47], and Alpos and Cachin [2].

The algorithm parses F as a sequence of nested *threshold* operators (*and* and *or* are special cases of *threshold*). Starting from the outermost operator, it constructs the *Vandermonde* matrix that implements it and then recursively performs *insertions* for the nested threshold operators.

Definition 2 (The MSP for a threshold access structure [6]). The $n \times t$ Vandermonde matrix is the matrix

$$V(n,t) = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{t-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{t-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{t-1} \end{pmatrix},$$

for $x_i \in \mathcal{K}$ pairwise different and nonzero. An MSP $\mathcal{M} = (M, \rho)$, with $M = V(n,t)$ and ρ a function that maps each row r_i of M to party $p_i \in \mathcal{P}$ encodes the t-of-n threshold access structure over the set of parties $\mathcal{P}$.

Definition 3 (Insertion). Let $\mathcal{M}^{(k)} = (M^{(k)}, \rho^{(k)})$, for $k \in \{1,2,3\}$, be MSPs over a finite field $\mathcal{K}$, where $M^{(k)}$ has dimensions $m_k \times d_k$. Denote by $\boldsymbol{r}_i^{(k)}$ the rows of each $M^{(k)}$, for $1 \le i \le m_k$, by $\boldsymbol{r}[j]$ the j^{th} column of a row $\boldsymbol{r}$, by $\boldsymbol{r}[j_1 : j_2]$ a range of columns j_1 to j_2, by $\boldsymbol{0}^{\ell}$ a row with ℓ zero elements, and by $\boldsymbol{r} \| \boldsymbol{r}'$ the concatenation of two rows $\boldsymbol{r}$ and $\boldsymbol{r}'$. Let $\boldsymbol{r}_z$ be the unique w.l.o.g. row of $M^{(1)}$ owned by $p_z \in \mathcal{P}^{(1)}$. The *insertion* of $M^{(2)}$ in row $\boldsymbol{r}_z$ of $M^{(1)}$, written as $\mathcal{M}^{(1)}(\boldsymbol{r}_z \to \mathcal{M}^{(2)})$, is an MSP $\mathcal{M}^{(3)}$, which has rows identical to $M^{(1)}$, except for $\boldsymbol{r}_z$, which is repeated m_2 times in $M^{(3)}$, each time multiplied by the first column of $M^{(2)}$, and with the rest of the columns 2 to d_2 of $M^{(2)}$ appended in the end. The function $\rho^{(3)}$ labels the rows of $M^{(3)}$ with the same owners as $\rho^{(1)}$, except for $\boldsymbol{r}_z$, as the newly inserted rows are labeled according to $\rho^{(2)}$.

Formally, $M^{(3)}$ is an $(m_1 + m_2 - 1) \times (d_1 + d_2 - 1)$ matrix with rows

$$\boldsymbol{r}_i^{(3)} = \begin{cases} \boldsymbol{r}_i^{(1)} \mid\mid \boldsymbol{0}^{d_2 - 1} & 1 \le i \le z-1 \\ \boldsymbol{r}_z * \boldsymbol{r}_{i-z+1}^{(2)}[1] \mid\mid \boldsymbol{r}_{i-z+1}^{(2)}[2 : d_2] & z \le i \le z + m_2 - 1 \\ \boldsymbol{r}_{i-m_2+1}^{(1)} \mid\mid \boldsymbol{0}^{d_2-1} & z + m_2 \le i \le m_1 + m_2 - 1 \end{cases} \tag{4}$$

and $\rho^{(3)}$ is a surjective function $\{1, \dots, m_1 + m_2 - 1\} \to (\mathcal{P}^{(1)} \setminus \{p_z\}) \cup \mathcal{P}^{(2)}$ defined as

$$\rho^{(3)}(i) = \begin{cases} \rho^{(1)}(i) & 1 \le i \le z-1 \\ \rho^{(2)}(i - z + 1) & z \le i \le z + m_2 - 1 \\ \rho^{(1)}(i - m_2 + 1) & z + m_2 \le i \le m_1 + m_2 - 1 \end{cases}$$

The pseudocode is shown in Algorithm 1. If $F = \Theta_d^m(F_1, \dots, F_m)$ is an MBF, where each F_i can be a party or a nested threshold operator, the algorithm first extracts the values m, d and $F_1, \dots, F_m$ from F (line 2) and creates the MSP for F (lines 1–13). For each F_i, if it is a nested operator, a fresh *virtual party* v_i is created and associated with F_i (the map V_{map} is used to keep track of this association). A virtual party is treated exactly as an actual party, except it is used only during this construction. The MSP for F is a Vandermonde matrix (line 13), created using both actual and virtual parties as the set $\mathcal{P}$. In the second part of the algorithm (lines 14–17) the MSPs for the nested operators (virtual parties v_i) are recursively created (line 15) and inserted in $\mathcal{M}$, according to Definition 3. The mapping ρ^{-1}, that maps a party to the rows they own in $\mathcal{M}$, is used to get the row $\boldsymbol{r}_i$ of M that was labeled with v_i. Notice that in line 10, a fresh variable is created for each nested operator, so v_i owns a single row.

If F includes in total c threshold operators in the form $\Theta_{d_i}^{m_i}$, the resulting matrix M has $m = \sum_1^c m_i - c + 1$ rows and $d = \sum_1^c d_i - c + 1$ columns.

Algorithm 1 Construction of an MSP from a monotone Boolean formula F.

buildMSP(F)
  let $\Theta_d^m(F_1, \ldots, F_m)$ be the formula F
  $R \leftarrow \emptyset$
  $V \leftarrow \emptyset$
  $V_{\text{map}} \leftarrow \emptyset$
  for each F_i **do**
    if F_i is a literal p **then**
      $R \leftarrow R \cup \{p\}$
    else
      declare v_i a new virtual party
      $V \leftarrow V \cup \{v_i\}$
      $V_{\text{map}} \leftarrow V_{\text{map}} \cup \{(v_i, F_i)\}$
  $\mathcal{M} \leftarrow$ *Vandermonde-MSP*$(m, d, R \cup V)$
  for each $v_i \in V$ **do**
    $\mathcal{M}_2 \leftarrow$ **buildMSP**$(V_{\text{map}}(v_i))$
    $r_i \leftarrow \rho^{-1}(v_i)$
    $\mathcal{M} \leftarrow \mathcal{M}(r_i \rightarrow \mathcal{M}_2)$
  return $\mathcal{M}$

B Interpolation on general access structures

Let $\mathcal{M} = (M, \rho)$ be an MSP over $\mathcal{K}$, with M an $m \times d$ matrix. We have seen in the definition of LSSS that an authorized set A can reconstruct the secret through the equation $\boldsymbol{\lambda}_A \boldsymbol{x}_A = x$. Besides that, in the following sections we will sometimes have to perform a different kind of interpolation: given the secret shares of parties in a set $F \in \mathcal{F}$ that is maximally unauthorized, i.e., there exists no $F' \in \mathcal{F}$ such that $F' \supset F$, and the secret x, we want to compute valid secret shares for parties not in F, where *valid* means that the reconstruction of the secret from any authorized set will result in the same value. In this section we explain why this interpolation is not trivial and present an algorithm that achieves it.

In threshold secret sharing this is done using polynomial interpolation: the secret shares of F and the secret x uniquely determine every other share. In general secret sharing it can be the case that F, even though maximally unauthorized, and the secret x do not uniquely determine the secret shares for the rest of the parties. For example, this can be because a party $p_i \notin F$ can now own more than one secret shares, in a way that adding all the shares of p_i to F makes it authorized, while adding *some* shares of p_i to F keeps in unauthorized. More specifically, if the rank of M_F is $d-1$ then all secret shares are uniquely defined. If the rank of M_F is $d-1-k$, for $k \in \mathbb{N}$, then there exist k secret shares (each corresponding to an MSP row), that do not belong to parties in F and are linearly independent from the shares of parties in F. The values of these secret shares can be chosen arbitrarily from $\mathcal{K}$ in the interpolation we wish to perform. These *extra rows* are given to the interpolation algorithm in the form of a set $R \subset \{1, \ldots, m\}$.

Formally, the algorithm has the following inputs and outputs.

- Inputs: (1) A maximally unauthorized set of parties $F \subset \mathcal{P}$ and their secret shares $\boldsymbol{x}_F \in \mathcal{K}^{m_F}$, (where m_F is the number of MSP rows owned by parties in F, and might be greater than $|F|$). (2) A set of extra MSP-row indexes $R \subset \{1, \ldots, m\}$, with $\rho(j) \notin F$, for all $j \in R$, and the corresponding secret shares $\boldsymbol{x}_R \in \mathcal{K}^{m_R}$ (where $m_R = |R|$). The sets F and R are such that the rank of the matrix $\binom{M_F}{M_R}$, that consists of the MSP rows either owned by parties in F or corresponding to indexes in R, is $d-1$. Notice that the rows indexed by R can all be chosen to be linearly independent from each other and from the rows owned by parties in F, hence the shares $\boldsymbol{x}_R$ can be chosen uniformly from the underlying field. (3) The secret x that corresponds to the secret shares $\boldsymbol{x}_F$ and $\boldsymbol{x}_R$. (4) An index $j \in [1, \ldots m]$.
- Output: Coefficients $\Lambda_j^{(1)} \in \mathcal{K}$ and $\boldsymbol{\Lambda}_j^{(2)} \in \mathcal{K}^{m_F + m_R}$, such that the secret share x_j can be calculated as a linear combination of these coefficients and the input values, that is, $x_j = \Lambda_j^{(1)} x +$

$\boldsymbol{\Lambda}_j^{(2)}(\boldsymbol{x}_F \| \boldsymbol{x}_R)$.

The algorithm works as follows. The given secret shares $\boldsymbol{x}_F \| \boldsymbol{x}_R$ have been computed as

$$\begin{pmatrix} \boldsymbol{x}_F \\ \boldsymbol{x}_R \end{pmatrix} = \begin{pmatrix} M_F \\ M_R \end{pmatrix} \boldsymbol{r} \tag{5}$$

where $\boldsymbol{r} = (x, r_2, \ldots, r_d)$ is unknown, except for the secret x. Since we know x, we can rewrite the previous equation as

$$\begin{pmatrix} x \\ \boldsymbol{x}_F \\ \boldsymbol{x}_R \end{pmatrix} = \begin{pmatrix} \boldsymbol{e}_1 \\ M_F \\ M_R \end{pmatrix} \boldsymbol{r}.$$

We define

$$\overline{M} = \begin{pmatrix} \boldsymbol{e}_1 \\ M_F \\ M_R \end{pmatrix}.$$

Observe that the MSP rows determined by F and R together are still unauthorized, and thus $\boldsymbol{e}_1$ is linearly independent from the rows in $\binom{M_F}{M_R}$. Moreover, by construction of F and R, the rank of $\binom{M_F}{M_R}$ is $d-1$. From these facts we get that $\overline{M}$ has full rank d. Moreover, let $\overline{m}$ be the number of rows in $\overline{M}$.

We now make use of d recombination vectors $\boldsymbol{\lambda}_\ell$, for $\ell \in [1, \ldots d]$. Each recombination vector $\boldsymbol{\lambda}_\ell$ is defined as an $\overline{m}$-vector such that $\boldsymbol{\lambda}_\ell \overline{M} = \boldsymbol{e}_\ell$, where $\boldsymbol{e}_\ell$ is the ℓ-th unit vector (i.e., consists of 0s, except for a 1 in position ℓ) of dimension d. In other words, $\boldsymbol{\lambda}_\ell$ expresses a linear combination of rows of $\overline{M}$ that gives the vector $\boldsymbol{e}_\ell$. Since the rank of $\overline{M}$ is d, all these recombination vectors exist. Additionally, define Λ as the $(d, \overline{m})$ matrix with the d recombination vectors as rows, i.e.,

$$\Lambda = \begin{pmatrix} \boldsymbol{\lambda}_1 \\ \boldsymbol{\lambda}_2 \\ \cdots \\ \boldsymbol{\lambda}_d \end{pmatrix}.$$

Notice that

$$\Lambda \cdot \overline{M} = I_d,$$

where I_d is the (d,d) identity matrix, and, by multiplying both members with $\boldsymbol{r}$,

$$\Lambda \cdot \begin{pmatrix} x \\ \boldsymbol{x}_F \\ \boldsymbol{x}_R \end{pmatrix} = \boldsymbol{r}.$$

By defining as $\boldsymbol{\Lambda}^{(1)}$ the first column of Λ and as $\Lambda^{(2)}$ the last $\overline{m}-1$ columns, the last equation can be rewritten as

$$\left(\boldsymbol{\Lambda}^{(1)} \Lambda^{(2)}\right) \cdot \begin{pmatrix} x \\ \boldsymbol{x}_F \\ \boldsymbol{x}_R \end{pmatrix} = \boldsymbol{r},$$

or

$$\boldsymbol{\Lambda}^{(1)} x + \Lambda^{(2)}(\boldsymbol{x}_F \| \boldsymbol{x}_R) = \boldsymbol{r}.$$

From the last equation we get

$$x_j = \boldsymbol{M}_j \boldsymbol{r} = \boldsymbol{M}_j \boldsymbol{\Lambda}^{(1)} x + \boldsymbol{M}_j \Lambda^{(2)}(\boldsymbol{x}_F \| \boldsymbol{x}_R),$$

or, by setting $\Lambda_j^{(1)} = \boldsymbol{M}_j \boldsymbol{\Lambda}^{(1)}$ and $\boldsymbol{\Lambda}_j^{(2)} = \boldsymbol{M}_j \Lambda^{(2)}$,

$$x_j = \Lambda_j^{(1)} x + \boldsymbol{\Lambda}_j^{(2)}(\boldsymbol{x}_F \| \boldsymbol{x}_R).$$

C Proof of Theorem 2 for the VSS scheme

We first repeat from [36] the related proof of Lemma 1.

Proof. Let the dimensions of M be $m \times d$, and let the secret shared by $\boldsymbol{r}$ be x, i.e., $r_1 = x$. By definition of an unauthorized set, the rows of M_F do not span $\boldsymbol{e}_1$. That means, $\mathit{rank}(M_F) < \mathit{rank}\binom{M_F}{\boldsymbol{e}_1}$ and, from linear algebra, we know that $|\mathit{kernel}(M_F)| > |\mathit{kernel}\binom{M_F}{\boldsymbol{e}_1}|$. This implies the existence of a vector $\boldsymbol{w} \in \mathcal{K}^d$, $\boldsymbol{w} \neq \boldsymbol{0}$, such that $M_F\boldsymbol{w} = \boldsymbol{0}$ (i.e., $\boldsymbol{w} \in \mathit{kernel}(M_F)$), and $w_1 = 1$ (i.e., $\boldsymbol{w} \notin \mathit{kernel}\binom{M_F}{\boldsymbol{e}_1}$). Define $\tilde{\boldsymbol{r}} = \boldsymbol{r} + (\tilde{x} - x)\boldsymbol{w}$. Notice that $\tilde{r}_1 = \tilde{x}$, so $\tilde{\boldsymbol{r}}$ shares the secret $\tilde{x}$. Moreover, $M_F\tilde{\boldsymbol{r}} = M_F\boldsymbol{r} + (\tilde{x} - x)M_F\boldsymbol{w} = M_F\tilde{\boldsymbol{r}}$. □

We now prove Theorem 2.

Completeness. By inspection of the scheme, honest parties accept their shares. Equation (1) will hold because

$$\prod_{\ell=1}^{d} C_\ell^{M_{j\ell}} = g^{\sum_{\ell=1}^{d} r_\ell M_{j\ell}} h^{\sum_{\ell=1}^{d} r'_\ell M_{j\ell}} = g^{\boldsymbol{M_j r}} h^{\boldsymbol{M_j r'}} = g^{x_j} h^{x'_j}$$

Furthermore, by definition of a Q^2 adversary structure, an authorized set A made of honest parties always exists, and, by definition of the MSP, the recombination vector $\boldsymbol{\lambda}_A$ of A always exists. Thus, a party can always reconstruct the secret from the shares of A. □

Correctness. For the first part assume, towards a contradiction, that $z_1 \neq z_2$. Also, let z'_1 and z'_2 be the reconstruction from the random-shares of the two sets. Since the shares are correct, it must hold that $g^{z_1}h^{z'_1} = C_1 = g^{z_2}h^{z'_2}$. Here we show that $g^{z_1}h^{z'_1} = C_1$. For $k = 1, 2$ the secret shares and random shares of parties in the two sets are

$$x_{A_k} = \{x_j^{(k)} \mid \rho(j) \in A_k\} \quad , \quad x'_{A_k} = \{x_j'^{(k)} \mid \rho(j) \in A_k\}.$$

Moreover, z_1, z'_1, z_2, z'_2 are calculated by honest parties as

$$z_k = \boldsymbol{\lambda}_{A_k}\boldsymbol{x}_{A_k} \quad , \quad z'_k = \boldsymbol{\lambda}_{A_k}\boldsymbol{x}'_{A_k} \tag{6}$$

Written as vectors, where m_k is the number of shares in A_k, for $k = 1, 2$, we have

$$\begin{aligned} \boldsymbol{x}_{A_k} &= (x_{j_1}, \dots, x_{j_{m_k}}) \\ \boldsymbol{x'}_{A_k} &= (x'_{j_1}, \dots, x'_{j_{m_k}}) \\ \boldsymbol{\lambda}_{A_k} &= (\lambda_{j_1}, \dots, \lambda_{j_{m_k}}). \end{aligned} \tag{7}$$

We have that

$$
\begin{aligned}
g^{z_1}h^{z'_1} &\overset{(6)}{=} g^{\boldsymbol{\lambda}_{A_1}\boldsymbol{x}_{A_1}} h^{\boldsymbol{\lambda}_{A_1}\boldsymbol{x}'_{A_1}} \\
&\overset{(7)}{=} g^{\sum_{j:\rho(j)\in A_1}\lambda_j x_j^{(1)}} h^{\sum_{j:\rho(j)\in A_1}\lambda_j x_j'^{(1)}} \\
&= \prod_{j:\rho(j)\in A_1} \left(g^{x_j^{(1)}} h^{x_j'^{(1)}}\right)^{\lambda_j} \\
&\overset{(1)}{=} \prod_{j:\rho(j)\in A_1} \Big(\prod_{\ell=1}^{d} C_\ell^{M_{j\ell}}\Big)^{\lambda_j} \\
&= \prod_{\ell=1}^{d} \prod_{j:\rho(j)\in A_1} C_\ell^{M_{j\ell}\lambda_j} \\
&= \prod_{\ell=1}^{d} C_\ell^{\sum_{j:\rho(j)\in A_1} M_{j\ell}\lambda_j} \\
&= \prod_{\ell=1}^{d} C_\ell^{\boldsymbol{\lambda}_A \boldsymbol{M}_{A\ell}}, \text{ where } \boldsymbol{M}_{A\ell} \text{ is the } \ell\text{-th row of } M_A \\
&\overset{\boldsymbol{\lambda}_A M_A=\boldsymbol{e}_1}{=} \prod_{\ell=1}^{d} C_\ell^{e_{1\ell}}, \text{ where } e_{1\ell} \text{ is the } \ell\text{-th entry of } \boldsymbol{e}_1 \\
&\overset{\boldsymbol{e}_1=[1,0,\ldots,0]}{=} C_1
\end{aligned}
$$

In the same way we get that $g^{z_2}h^{z'_2} = C_1$.

Now, since $z_1 \neq z_2$, it is also the case that $z'_1 \neq z'_2$. But from this one can extract the logarithm of h with base g as $\log_g h = (z_1 - z_2)/(z'_2 - z'_1)$, which is, by assumption, not known.

The second part follows immediately from the fact that the dealer is honest and by simple observation that the output of *Reconstruct()* is $\boldsymbol{\lambda}_A \boldsymbol{x}_A = \boldsymbol{\lambda}_A M_A \boldsymbol{r} = \boldsymbol{e}_1 \boldsymbol{r} = x$, for any authorized set A. □

Privacy. Fix wlog a maximally unauthorized set F consisting of parties controlled by the adversary and let m_F the number of shares owned by parties in F. Assume the dealer has shared a secret x using coefficient vectors $\boldsymbol{r} = (x, r_2, \ldots, r_d)$ and $\boldsymbol{r}' = (x', r'_2, \ldots, r'_d)$. The view of the adversary consists of the shares $\boldsymbol{x}_F = (x_{j_1}, \ldots, x_{j_{m_F}})$ and $\boldsymbol{x}'_F = (x_{j_1}, \ldots, x_{j_{m_F}})$, where $\rho(j_k) \in F$, for $k \in \{1, \ldots, m_F\}$, and the commitments $C_1 = g^x h^{x'}$ and $C_\ell = g^{r_\ell} h^{r'_\ell}$, for $\ell \in \{2, \ldots, d\}$, created by the dealer. We then choose arbitrary $\tilde{x} \neq x \in \mathcal{K}$. We want to show that the view of the adversary is consistent with an execution of the VSS where $\tilde{x}$ is the secret shared by the dealer.

Observe that $\tilde{x}$ uniquely defines an $\tilde{x}'$ such that $C_1 = g^{\tilde{x}} h^{\tilde{x}'}$. From Lemma 1, we know there exist coefficient vectors $\tilde{\boldsymbol{r}} = \boldsymbol{r} + (\tilde{x} - x)\boldsymbol{w}$ and $\tilde{\boldsymbol{r}}' = \boldsymbol{r}' + (\tilde{x}' - x')\boldsymbol{w}$, with $\boldsymbol{w} \in \mathcal{K}^d$, that share the secrets $\tilde{x}$ and $\tilde{x}'$, respectively, while the resulting shares $\tilde{\boldsymbol{x}}_F$ and $\tilde{\boldsymbol{x}}'_F$ satisfy $\tilde{\boldsymbol{x}}_F = \boldsymbol{x}_F$ and $\tilde{\boldsymbol{x}}'_F = \boldsymbol{x}'_F$. Notice that the $\boldsymbol{w}$ in the proof of Lemma 1 depends on M_F and not on the coefficient vector, thus it is the same in the equations for $\tilde{\boldsymbol{r}}$ and $\tilde{\boldsymbol{r}}'$.

It remains to show that the commitments $\tilde{C}_\ell = g^{\tilde{r}_\ell} h^{\tilde{r}'_\ell}$, for $\ell \in \{2, \ldots, d\}$, also satisfy $\tilde{C}_\ell = C_\ell$. Let b be the discrete logarithm of h with basis g, i.e., $h = g^b$. Recall that $C_1 = g^x h^{x'} = g^{x+bx'}$ and $C_1 = g^{\tilde{x}} h^{\tilde{x}'} = g^{\tilde{x}+b\tilde{x}'}$. These two equations give

$$x + bx' = \tilde{x} + b\tilde{x}'. \tag{8}$$

We now define the vectors $\boldsymbol{c} = \boldsymbol{r} + b\boldsymbol{r}'$ and $\tilde{\boldsymbol{c}} = \tilde{\boldsymbol{r}} + b\tilde{\boldsymbol{r}}'$ and observe that $C_\ell = g^{c_\ell}$ and $\tilde{C}_\ell = g^{\tilde{c}_\ell}$, where

c_ℓ and $\tilde{c}_\ell$ are the entries of $\boldsymbol{c}$ and $\tilde{\boldsymbol{c}}$, respectively. It is thus enough to show that $\boldsymbol{c} = \tilde{\boldsymbol{c}}$. We have that

$$\begin{aligned} \boldsymbol{c} = \tilde{\boldsymbol{c}} &\Leftrightarrow \boldsymbol{r} + b\boldsymbol{r}' = \boldsymbol{r} + (\tilde{x} - x)\boldsymbol{w} + b\boldsymbol{r}' + b(\tilde{x}' - x')\boldsymbol{w} \\ &\Leftrightarrow (\tilde{x} - x)\boldsymbol{w} + b(\tilde{x}' - x')\boldsymbol{w} = \boldsymbol{0} \\ &\overset{\boldsymbol{w}\neq\boldsymbol{0}}{\Leftrightarrow} \tilde{x} - x + b(\tilde{x}' - x') = 0, \end{aligned}$$

which holds from (8). □

D Proof of Theorem 3 for the common-coin scheme

The proof for the general coin construction follows the lines of the threshold coin scheme [13], but use our method from Section B to handle the interpolation with general access structures.

Proof. Robustness follows from the soundness of the interactive proof of equality of the discrete logarithms. Moreover, the underlying access structure is Q^2, hence there will be enough honest parties to combine the shares and interpolate the coin value.

The rest of this proof concerns unpredictability. We assume an adversary that can predict the value of a coin with non-negligible probability and show how to use this adversary to solve CDH. To successfully attack CDH, it is enough to construct an algorithm that, on input elements $g, \hat{g}, g_0 \in G$, where $\hat{g} \overset{\$}{\leftarrow} G$ and $g_0 = g^{x_0}$, outputs a list that contains $\hat{g}_0 = \hat{g}^{x_0}$ with non-negligible probability [59]. The adversary makes a series of queries for coins $C_1, \ldots, C_t$ for a polynomially large t, and tries to predict the value of the target coin $\hat{C}$. We assume that $\hat{C} = C_s$, for a random $s \in \{1, \ldots, t\}$, which decreases our advantage by a factor of t. For the target coin, let $\hat{g} = H(\hat{C})$ and $\hat{g}_j = \hat{g}^{x_j}$

Only for this part of the proof, we let the adversary corrupt a set $T \supseteq F$, as long as $T \notin \mathcal{A}$, i.e., T is a maximal superset of F that remains unauthorized. This is w.l.o.g: if the adversary cannot predict the coin from T, it cannot predict it from F either. The algorithm simulates the view for the adversary as follows. For party p_i in T we choose its key shares x_j, where $\rho(j) = p_i$, uniformly from $\mathbb{Z}_q$. The verification keys can then be computed as $g_j = g^{x_j}$. For the rest of the verification keys the idea is to use the verification keys we just calculated and g_0, and perform an interpolation in the exponent. However, as explained in Section B, for these to be uniquely determined, the shares of T (called F in Section B) and of some extra indexes R are required. The set of row indexes R is chosen arbitrarily, under the conditions described in Section B. The shares x_j, where $j \in R$, are also chosen uniformly at random, and the corresponding verification keys are again $v_j = g^{x_j}$, where $j \in R$.

We can now use the algorithm described in Section B, with input sets T and R, and with shares $\boldsymbol{x}_T$ and $\boldsymbol{x}_R$ and the secret x raised to g_2:

$$v_j = v^{\Lambda_j^{(1)}} \cdot \prod_{\substack{\ell \text{ such that} \\ \rho(\ell) \in T \vee \ell \in R}} v_\ell^{\Lambda_{j\ell}^{(2)}}. \tag{9}$$

After the verification keys are chosen, we simulate the interaction with the adversary as follows. In the random oracle model, the adversary queries H to obtain $\tilde{g}$ or $\hat{g}$ and the simulator can respond to these queries as it wishes. For coins $C \neq \hat{C}$, the simulator chooses $r \in \mathbb{Z}_q$ at random and sets $\tilde{g} = g^r$ as the value of H at point C. The coin shares for all honest parties can be calculated as $\tilde{g}_j = g_j^r$, where $\rho(j) \notin T$.

The proof of correctness for each coin share can be simulated by invoking the random oracle model for H'. When an honest party is supposed to create a coin share $\tilde{g}_j$, the simulator chooses $c_j, z_j \in \mathbb{Z}_q$ at random, and sets the output of H' at point $(g, g_j, g^{z_j} g_j^{-c_j}, \tilde{g}, \tilde{g}_j, \tilde{g}^{z_j} \tilde{g}_j^{-c_j})$ to be c. Except with negligible probability, the simulator has not already defined the output of H' at this point, so this part of the simulation succeeds.

For the target coin $\hat{C}$ we set $H(\hat{C}) = \hat{g}$. By construction of T, the adversary is not allowed to ask honest parties for coin shares, thus the simulator never has to produce any valid shares. Observe that the adversary, in order to make the prediction $b \in \{0,1\}$ for $\hat{C}$, must query H'' at point $\hat{g}_0$. Hence, when it terminates we output the list of all these queries — by assumption it will contain the solution to CDH with a non-negligible probability. The simulation is perfect, since all the shares and verification keys have the same distribution as in an actual execution of the protocol, except for a negligible probability that our zero-knowledge simulations fail. $\square$

E Proof of Theorem 4 for the general distributed signatures

Robustness. Because $\mathcal{A}$ is Q^2, there exists an authorized set A that consists entirely of honest parties. Moreover, only valid signatures, made with a party's private key share, can pass the verification of algorithm *SigShareVerify()*. Thus, a combiner can verify and use the signature shares of A in algorithm *SigShareCombine()* to create a valid distributed BLS signature. $\square$

Unforgeability. We show that the general distributed signature scheme is simulatable. Simulatability, together with the unforgeability of the standard BLS scheme, imply unforgeability for the general distributed signature scheme [30, Definition 3]. Simulatability means that a simulator, on input the public key v, a message μ with signature σ, and the key shares x_j of parties in F, i.e., $\rho(j) \in F$, can simulate the view for the adversary that is polynomially indistinguishable from an execution of the real protocol that outputs σ as the signature of μ, and where the adversary has key shares x_j, where $\rho(j) \in F$. Intuitively, this shows that an adversary who sees all the private information of parties in F and the signature on a message μ could generate by itself all the public information of the protocol.

The simulator works as follows. First, it has to provide valid verification keys for all parties and all their shares. For parties in F, the simulator can use the given shares x_j, where $\rho(j) \in F$, to compute the verification keys. The rest of the shares are interpolated from the shares x_j. However, as explained in Section B, for these to be uniquely determined, some extra indexes R are required. The set of row indexes R is chosen arbitrarily, under the conditions described in Section B, and the shares that correspond to the indexes in R are chosen uniformly at random. For sets F and R, the simulator computes the verification keys as $v_j = g_2^{x_j}$, where $\rho(j) \in F$ or $j \in R$. For any other p_j the simulator uses the interpolation algorithm described in Section B, with input sets F and R, and with shares $\boldsymbol{x}_F$ and $\boldsymbol{x}_R$ and the secret x raised to g_2, calculating v_j exactly as in (9).

Second, the simulator also has to respond to the adversary's signature queries. Following exactly the same techniques, the simulator can generate all the signature shares given the standard BLS signature σ of message m.

Finally, for any row $j \in \{1, \ldots, m\}$ of the MSP, the verification key $v_j = g_2^{x_j}$ and the signature share $\sigma_j = H(\mu)^{x'_j}$ will satisfy $x_j = x'_j$. For j such that $\rho(j) \in F$ or $j \in R$ this holds because the simulator used a known x_j to calculate these values, while for any other j this holds from the MSP interpolation. Hence, $(g_2, v_i, H(m), \sigma_i)$ is a valid co-Diffie-Hellman tuple and the signature shares will be verified. Moreover, the interpolated key shares have the same distribution as if produced by the real dealer. The view of the adversary is thus statistically indistinguishable from an execution of the real protocol. $\square$