Error Detection in Monotone Span Programs with Application to Communication-Efficient Multi-Party Computation

Nigel P. Smart[1,2] and Tim Wood[1,2]

[1] University of Bristol, Bristol, UK.
[2] KU Leuven, Leuven, Belgium.
nigel.smart@kuleuven.be,t.wood@kuleuven.be

Abstract. Recent improvements in the state-of-the-art of MPC for non-full-threshold access structures introduced the idea of using a collision-resistant hash functions and redundancy in the secret-sharing scheme to construct a communication-efficient MPC protocol which is computationally-secure against malicious adversaries, with abort. The prior work is based on replicated secret-sharing; in this work we extend this methodology to *any* LSSS implementing a $\mathcal{Q}_2$ access structure. To do so we need to establish a folklore property of error detection for such LSSS and their associated Monotone Span Programs. In doing so we obtain communication-efficient online and offline protocols for MPC in the pre-processing model.

1 Introduction

Secure multi-party computation (MPC) allows a set of parties to compute a function on their combined secret inputs so that all parties learn the output of the function and no party can learn anything that cannot be inferred from the output and their own inputs alone. As a field it has recently received a lot of attention and has been explored in a variety of contexts: for example, private auctions [12], secure statistical analysis of personal information [10] and protection against side-channel attacks in hardware [8, 33, 34].

Most MPC protocols fall into one of two broad categories: garbled circuits, and linear-secret-sharing-scheme-based (LSSS-based) MPC. The garbled-circuit approach, which began with the work of Yao [36], involves some collection of parties "garbling" a circuit to conceal the internal circuit evaluations, and then later a single party or a collection of parties jointly evaluating the garbled circuit. By contrast, the LSSS-based approach involves using a so-called linear secret-sharing scheme, in which the parties: "share" a secret into several *shares* which are distributed to different parties, perform computations on the shares, and then reconstruct the secret at the end by combining the shares to determine the output. LSSS-based MPC is traditionally presented in the context of information-theoretic security, although many modern practical protocols that realise LSSS-based MPC often make use of computationally-secure primitives such as somewhat-homomorphic encryption (SHE) [23] or oblivious transfer (OT) [30]. In this paper, we focus on computationally-secure LSSS-based MPC.

An access structure for a set of parties defines which subsets of parties are allowed to discover the secret if they pool their information. Such quorums of parties are often called *qualified* sets of parties. An access structure is called $\mathcal{Q}_\ell$ (for $\ell \in \mathbb{N}$) if the union of any set of ℓ unqualified sets of parties is missing at least one party. We discuss this in some detail later, but for now the reader can think of an (n, t)-threshold scheme where $t < n/\ell$ which is where a subset of parties is qualified if and only if it is of size at least $t + 1$. Computationally-secure LSSS-based MPC has recently seen significant, efficient instantiations for full-threshold access structures [7, 22, 23, 30], which is where the protocol is secure if at least one party is honest, even if the adversary causes the corrupt parties to run arbitrary code (though this behaviour may cause the protocol to abort rather than provide output to the parties). In the threshold case similar efficient instantiations are known, such as the older VIFF protocol [20] which uses (essentially) information-theoretic primitives only.

While protocols providing full-threshold security are an important research goal, in the real world such guarantees of security do not always match the use-cases that appear. Different applications call for different access structures, and not necessarily the usual threshold examples. For example, a company may have four directors (CEO, CTO, CSO and CFO) and access may be granted in the two combinations (CEO and CFO) or (CTO and CSO and CFO). In such a situation it may be more efficient to tailor the protocol to this structure, rather than try to shoe-horn the application into a more standard (i.e. full-threshold) structure. Indeed, while it is possible that a computation can be performed in a full-threshold setting and then the outputs distributed in accordance with the access structure, such a process requires all parties to participate equally in the computation, which may not be feasible in the real world, especially if the computing parties are distributed over a wide network, and susceptible to outages if the total number of parties is large.

Most LSSS-based MPC protocols split the computation into two parts: an *offline phase*, in which parties interact using "expensive" public-key cryptography to lay the groundwork for an *online phase* in which only "cheap" information-theoretic primitives are required. The online phase is where the actual circuit evaluation takes place. For the access structures considered in this work, namely $\mathcal{Q}_2$ structures, the offline phase is almost as fast as the online phase. Thus the goal here is to minimize the cost of communication in both phases.

Realising MPC for different access structures has been well studied: shortly following the advent of Shamir's secret-sharing scheme [9, 35], the first formal MPC – as opposed to 2PC – protocols [5, 15, 26] were constructed, with varying correctness guarantees for different threshold structures. These works were developed by Hirt and Maurer [27], and then Beaver and Wool [3] for general access structures, culminating in Maurer's relatively more recent work [32]. In this last work it is shown that passively-secure information-theoretic MPC is possible if the access structure is $\mathcal{Q}_2$, and full active security (without requiring abort) is possible if the access structure is $\mathcal{Q}_3$. The latter has seen various optimisations in the literature, for example [21], making use of packed secret sharing to obtain a bandwidth-efficient perfectly-secure protocol.

In recent work [31], Keller et al. show that by generalising a method of Araki et al [1, 25] communication-efficient computationally-secure MPC with abort can be realised for $\mathcal{Q}_2$ access structures, if replicated secret-sharing is used. The methodology in [1,25,31] uses the explicit properties of replicated secret-sharing so as to authenticate various shares. This enables active security with abort to be achieved relatively cheaply, albeit at the expense in general of the pre-deployment of a large number, depending on the access structure, of symmetric keys to enable the generation of pseudo-random secret sharings (PRSSs) in a non-interactive manner. A disadvantage of replicated sharing is the potentially larger (than average) memory footprint needed for each party per secret, and consequently there is still a relatively large communication cost involved when the parties need to send shares across the network. In this work we extend this prior work to produce a protocol for *any* LSSS that supports the $\mathcal{Q}_2$ access structure.

1.1 Authentication of Shares

Many of practical MPC protocols begin with a basic passively-secure (a.k.a. *semi-honest* or *honest-but-curious*) protocol, in which corrupt parties execute the protocol honestly but try to deduce anything they can about other parties' data from their own data and the communication tapes. Such passively-secure protocols for $\mathcal{Q}_2$ access structures are highly efficient, and are information-theoretically secure. The passively secure protocols are then augmented to obtain active security with abort by using some form of "share authentication"; in this security setting, corrupt parties may deviate arbitrarily from the protocol description but if they do so the honest parties will abort the protocol.

At a high level, modern actively-secure LSSS-based MPC protocols combine:

1. A linear (i.e. additively homomorphic) secret sharing scheme;
2. A passive multiplication protocol; and
3. An authentication protocol.

The communication efficiency of the computation (usually an arithmetic or Boolean circuit) depends heavily on how authentication is performed.

In the full-threshold SPDZ protocol [23] and its successors, e.g. [22, 30], authentication is achieved with additively homomorphic message authentication codes (MACs). For each secret that is shared amongst the parties, the parties also share a MAC on that secret. Since the authentication is additively homomorphic and the sharing scheme is linear, this means that the sum (and consequently scalar multiple) of authenticated shares is authenticated "for free" by performing the addition (or scalar multiplication) on the associated MACs. More work is required for multiplication of secrets, but the general methodology for doing these operations on shared secrets is now generally considered "standard" for MPC in this setting.

One important branch of this authentication methodology contributing significantly to their practical performance is the amortisation of verification costs by batch-checking MACs, a technique developed in [6,23], amongst other works. A different approach to batch verification for authentication of shares, in the case of $\mathcal{Q}_2$ access structures, was introduced by Furakawa et al. [25], in the context of the three-party honest-majority setting, i.e. a $(3, 1)$-threshold access structure. This work extended a passively-secure protocol of Araki et al. [1] in the same threshold setting. This approach dispenses with the MACs and instead achieves authentication of shares using a collision-resistant hash function when authenticating an open-to-all operation, and uses redundancy of the underlying secret sharing scheme in an open-to-one operation. Their protocol can be viewed as a bootstrapping of the passively-secure protocol of Beaver and Wool [3], with an optimised sharing procedure (highly tailored to the $(3, 1)$-threshold access structure), to provide a communication-efficient actively-secure protocol (with abort). By using a hash function they sacrifice the information-theoretic security of Beaver-Wool for computational security, and also use computationally-secure share generation operations to improve the offline phase.

The above protocols for replicated sharing in a $(3, 1)$-threshold access structure of [1,25] simultaneously reduce the number of secure communication channels needed *and* the total number of bits sent per multiplication. Recent work [31] has shown that these techniques can be generalised from $(3, 1)$-threshold to *any* $\mathcal{Q}_2$ access structure, using replicated secret-sharing. Both [25] and [31] make use of the fact that replicated sharing provides a trivial method to authenticate a full set of shares; i.e. it somehow offers a form of error-detection.

A recent protocol due to Chida et al. [16] also considers actively-secure honest-majority MPC and makes use of MACs. In their work, the communication cost is a constant number of elements per multiplication, but the messages are broadcast, so this cost is linear in the number of parties. Our protocol also has linear overhead, but following the methodology of [31], the total number of uni-directional channels is reduced and so the asymptotic cost is lower (for threshold access structures). The benefit of the Chida protocol is that there is no offline processing, and the total cost of active computation is less than ours (they achieve roughly twice the cost of passive multiplication as opposed to our roughly threefold cost). However, if one is interested purely in online times then our protocol is more efficient than that of Chida et al.

1.2 Our contribution

While the replicated secret-sharing of [31] offers flexibility in being able to realise *any* access structure, unfortunately it *can* require an exponentially-large number of shares to be held by each party for each shared secret. As threshold access structures illustrate, using a general MSP may enable the same access structure to be realised in a more efficient manner, which motivates our work in this area.

The two main contributions of this work are as follows:

- Showing we can get authentication of shares almost for free for any MSP realising $\mathcal{Q}_2$ access structures. Assuming an offline phase which produces

Beaver triples, this gives us active security with abort, at the cost of replacing information-theoretic with computational security.
- We also provide, in the full version, a generic way to reduce the amount of communication required for the passive multiplication subprotocol in the offline phase for multiplicative MSPs.

Thus we generalise the online phase of [31] to arbitrary MSPs, hence allowing the benefits of that work to be achieved without necessarily requiring the cost of replicated secret sharing. Whereas many of the previous protocols are optimised for access structures on specific numbers of parties, or use specific secret-sharing schemes, our optimisation of the passive online multiplication is generic in the sense that it only uses the $\mathcal{Q}_2$ nature of the access structure for authentication: [25] and [31] are special cases of our optimisation.

Our contribution, then, is not so much our full MPC protocol as it is the mechanism for an actively-secure multiplication in the $\mathcal{Q}_2$ setting. Viewing the protocol in this more modular sense allows us to separate the LSSS from the actual multiplication and thus allows us to reduce the search for finding an efficient MPC protocol for a given $\mathcal{Q}_2$ access structure to finding an LSSS with a small total number of shares.

To conclude this section, we briefly remark how our work relates to the correspondence between LSSSs and linear codes. Cramer et al. [19] showed how the correspondence between linear secret-sharing schemes and linear codes reveals an efficient method by which qualified parties can *correct* any errors in a set of shares for some secret. The ability to do so requires the access structure to be $\mathcal{Q}_3$, since if this holds then a strongly-multiplicative LSSS realising it allows honest parties to correct any errors introduced by the adversary. This is not a direct connection to error-correction codes since such LSSSs do not necessarily allow unique decoding of the entire share vector: it is only the component of the share vector corresponding to the *secret* that is guaranteed to be correct. In our work we show that if the access structure is $\mathcal{Q}_2$ then any LSSS realising it allows honest parties to agree on whether or not the secret is correct: thus we obtain a form of error-detection. This reveals why the protocols above (viz., [25, 31]) are able to perform the error-detection causing abort. This result seems to be folklore – but we could find no statement or proof in the literature to this effect, and so we prove the required properties here.

2 Preliminaries

2.1 Notation

Let $\mathbb{F}$ denote a finite field; we write $\mathbb{F} = \mathbb{F}_q$ for q some prime power if $\mathbb{F}$ is the field of q elements. We write $r \xleftarrow{\$} \mathbb{F}$ to mean that r is sampled uniformly at random from $\mathbb{F}$. Vectors are written in bold and are taken to be column vectors. We denote by $\mathbf{0}$ a vector consisting entirely of zeros of appropriate dimension, determined by the context, and similarly by $\mathbf{1}$ a vector consisting entirely of ones. For a vector $\mathbf{x}$ we write the i^{th} component as $\mathbf{x}_i$, whereas $\mathbf{x}^i$ denotes the

i^{th} vector from a sequence of vectors. We use the notation $\mathbf{e}^i$ for the i^{th} standard basis vector (defined by $\mathbf{e}^i_j := \delta_{ij}$ where δ_{ij} is the Kronecker delta). We denote by $[n]$ the set $\cup_{i=1}^{n}\{i\}$, and by $\mathcal{P}$ the complete set of parties, which we take to be $\{P_i\}_{i\in[n]}$. Given some set S, a subset of some larger set S', we write $a \notin S$ to indicate that element a is in $S' \setminus S$; in general, S' will be implicit, according to context. We define the function $\mathit{supp} : \mathbb{F}^m \to 2^{\mathcal{P}}$ via $\mathbf{s} \mapsto \{i \in [m] : \mathbf{s}_i \neq 0\}$. We use the notation $A \subseteq B$ to mean that A is a (not necessarily proper) subset of B, contrasted with $A \subsetneq B$ where A is a proper subset of B. We write λ and κ for the statistical and computational security parameters respectively.

Given a vector space $V \subseteq \mathbb{F}^d$, we denote by $V^\perp$ the orthogonal complement; that is, $V^\perp = \{\mathbf{w} \in \mathbb{F}^d : \langle \mathbf{v}, \mathbf{w}\rangle = 0\}$, where $\langle \mathbf{v}, \mathbf{w}\rangle = \mathbf{v}^\top \cdot \mathbf{w}$ is the standard inner product. From basic linear algebra, $(V^\perp)^\perp = V$. For a matrix $M \in \mathbb{F}^{m\times d}$, we write $M^\top$ for the transpose. If M is a matrix representing a linear map $\mathbb{F}^d \to \mathbb{F}^m$, then $\mathrm{im}(M^\top) = \ker(M)^\perp$ by the fundamental theorem of linear algebra.

2.2 Access Structures, MSPs, LSSSs and Linear Codes

Access structures: Fix $\mathcal{P} = \{P_i\}_{i\in[n]}$ and let $\Gamma \subseteq 2^{\mathcal{P}}$ be a monotonically increasing set, i.e. Γ is closed under taking supersets: if $Q \in \Gamma$ and $Q' \supseteq Q$ then $Q' \in \Gamma$. Similarly, let $\Delta \subseteq 2^{\mathcal{P}}$ be a monotonically decreasing set, i.e. Δ is closed under taking subsets: if $U \in \Delta$ and $U' \subseteq U$ then $U' \in \Delta$. We call the pair (Γ, Δ) a *monotone access structure* if $\Gamma \cap \Delta = \varnothing$. If $\Delta = 2^{\mathcal{P}} \setminus \Gamma$, then we say the access structure is *complete*. In this paper, we will only be concerned with complete monotone access structures and so this is assumed throughout without qualification. The sets in Γ, usually denoted by Q, are called *qualified*, and the sets in Δ, usually denoted by U, are called *unqualified*. Partial ordering is induced on Γ and Δ by the standard subset relation denoted by "$\subseteq$": we write Γ^- for the set of *minimally qualified sets* where minimality is with respect to "$\subseteq$": $\Gamma^- = \{Q \in \Gamma : \text{ if } Q' \in \Gamma \text{ and } Q' \subseteq Q \text{ then } Q' = Q\}$; similarly, Δ^+ denotes the set of *maximally unqualified sets* where maximality is with respect to "$\subseteq$": $\Delta^+ = \{U \in \Delta : \text{ if } U' \in \Delta \text{ and } U \subseteq U' \text{ then } U' = U\}$

An access structure is said to be $\mathcal{Q}_2$ (resp. $\mathcal{Q}_3$) if the union of no two (resp. three) sets in Δ is the whole of $\mathcal{P}$. A consequence of this is that in a $\mathcal{Q}_2$ access structure, the complement of a qualified set is unqualified, and *vice versa*.

In an (n, t)-threshold access structure, any set of $t + 1$ parties is qualified, whilst any set of t or fewer parties is unqualified. Thus Γ^- contains $\binom{n}{t+1}$ sets in total. The term *full threshold* refers to an $(n, n-1)$-threshold access structure. For an arbitrary complete monotone access structure, the set of minimally qualified sets together with the set of maximally unqualified sets uniquely determine the entire structure. The dual access structure Γ^* of an access structure Γ is defined by $\Gamma^* := \{Q \in 2^{\mathcal{P}} : 2^{\mathcal{P}} \setminus Q \notin \Gamma\}$. Cramer et al. [19] showed that an access structure Γ is $\mathcal{Q}_2$ if and only if $\Gamma^* \subseteq \Gamma$.

Linear Secret Sharing Schemes: An LSSS is a method of sharing secret data amongst parties. It consists of three multi-party algorithms: Input, Open, and ALF (affine linear function), allowing parties to provide secret inputs, reveal (or *open*) secrets, and compute an affine linear function on shared secrets. In a practical sense, this means that the parties can add secrets, multiply by scalars, and add public constants to a shared secret, all by local computations. In this work we consider, as examples, the three most well-known secret-sharing schemes: Shamir; replicated, also known as CNF-based (conjunctive-normal-form-based); and DNF-based (disjunctive-normal-form-based). We will use the term *additive sharing* to mean that a secret s takes the value $s = \sum_{i=1}^{n} s_i$ where s_i is held by P_i and the s_i's are uniformly random subject to the constraint that they sum to s.

An LSSS is called *multiplicative* if the whole set of parties $\mathcal{P}$ can compute an additive sharing of the product of two secrets by performing only local computations. If the product is to be kept as a secret and used further in the computation, it is usually necessary for the parties to engage in one or more rounds of communication to convert the additive sharing into a sharing in the LSSS being used. A secret-sharing scheme is called *strongly multiplicative* if, for any $U \in \Delta$, the parties in $\mathcal{P} \backslash U$ can compute an additive sharing of the product of two secrets by local computations. Such schemes offer robustness, since the adversary, corrupting an unqualified set of parties, cannot prevent the honest parties from reconstructing the desired secret. Cramer et al. [18] showed that any (non-multiplicative) LSSS realising a $\mathcal{Q}_2$ access structure can be converted to a multiplicative LSSS for the same access structure so that each party holds at most twice the number of shares it held originally. There is currently no known construction to convert an arbitrary $\mathcal{Q}_3$ LSSS to a strongly multiplicative LSSS with only polynomial blow-up in the number of shares each party must hold [18, 19].

Monotone Span Programs: Span programs, and monotone span programs specifically, were introduced by Karchmer and Wigderson [29] as a model of computation. It has been shown that MSPs have a close relationship to secret-sharing schemes, as discussed informally below.

Definition 1. *A* Monotone Span Program *(MSP), denoted by $\mathcal{M}$, is a quadruple $(\mathbb{F}, M, \boldsymbol{\varepsilon}, \psi)$ where $\mathbb{F}$ is a field, $M \in \mathbb{F}^{m \times d}$ is a full-rank matrix for some m and $d \leq m$, $\boldsymbol{\varepsilon} \in \mathbb{F}^d$ is an arbitrary non-zero vector called the* target vector*, and $\psi : [m] \twoheadrightarrow \mathcal{P}$ is a surjective "labelling" map of rows to parties. The size of $\mathcal{M}$ is defined to be m, the number of rows of the matrix M.*

Typically, $\boldsymbol{\varepsilon} = \mathbf{e}^1$ or $\boldsymbol{\varepsilon} = \mathbf{1}$, but it can be an arbitrary non-zero vector: changing it simply changes how the vector $\mathbf{x}$ is selected, and corresponds to performing column operations on the columns of M, which does not change the access structure the MSP realises by results of Beimel et al. [4]. Some definitions of MSP do not require that M have full rank, since if this is not the case, one can iteratively remove any columns which are linearly dependent on preceding columns without changing the access structure $\mathcal{M}$ computes. We make this assumption for the sake of simplicity later on.

We say that the row-map ψ defines which rows are "owned" by each party. Given a set $S \subseteq \mathcal{P}$, we denote by M_S the submatrix of M whose rows are indexed by the set $\{i \in [m] : \psi(i) \in S\}$, and similarly $\mathbf{s}_S$ is the subvector of $\mathbf{s}$

whose entries are indexed by the same. Later, we will somewhat abuse notation by denoting again by M_S, where now $S \subseteq [m]$, the submatrix whose rows are indexed by S. Context will determine which matrix we mean since the indexing set is either a set of parties, or a set of row indices. If $\mathbf{s} \in \mathbb{F}^m$, then we call $\mathbf{s}_Q$ a *qualified subvector* of $\mathbf{s}$ if $Q \in \Gamma$, and an *unqualified subvector* otherwise. An MSP $\mathcal{M}$ is said to compute an access structure Γ if it holds that $Q \in \Gamma$ if and only if $\exists\, \boldsymbol{\lambda}^Q \in \mathbb{F}^m$ (i.e. depending on Q) such that $M^\top \cdot \boldsymbol{\lambda}^Q = \boldsymbol{\varepsilon}$ and $\psi(supp(\boldsymbol{\lambda}^Q)) \subseteq Q$. In other words, $\boldsymbol{\varepsilon} \in \mathrm{Im}(M_Q^\top)$ if and only if Q is qualified. Note that we write $\boldsymbol{\lambda}^Q$ to show that this vector is associated to the set Q; compare with $\boldsymbol{\lambda}^Q_Q$, which is the subvector of $\boldsymbol{\lambda}^Q$ whose co-ordinates are indexed by Q, to be consistent with the notation above. This means that the parties in the set Q "own" rows of the matrix M which can be combined in a public, known linear combination encoded as the vector $\boldsymbol{\lambda}^Q$, to obtain the target vector $\boldsymbol{\varepsilon}$.

Monotone Span Programs induce LSSSs in the following way: Sample $\mathbf{x} \xleftarrow{\$} \mathbb{F}^d$ subject to $\langle \mathbf{x}, \boldsymbol{\varepsilon} \rangle = s$, the secret. Now let $\mathbf{s} = M \cdot \mathbf{x}$ and for each $i \in [m]$, give $\mathbf{s}_i$ (that is, the i^{th} co-ordinate) to party $\psi(i)$. Thus party P_i has the vector $\mathbf{s}_{\{P_i\}}$. We call $\mathbf{x}$ the *randomness vector* since $\mathbf{x}$ is chosen uniformly at random, subject to $\langle \mathbf{x}, \boldsymbol{\varepsilon} \rangle = s$, to generate $\mathbf{s} := M \cdot \mathbf{x}$, the *share vector*. The co-ordinates of $\mathbf{s}$ are precisely the shares of the secret which are distributed to parties according to the mapping ψ. We say that a share vector $\mathbf{s}$ *encodes* a secret s if $\mathbf{s} = M \cdot \mathbf{x}$ for some $\mathbf{x}$ where $\langle \mathbf{x}, \boldsymbol{\varepsilon} \rangle = s$. An MSP is called *ideal* if ψ is injective; since it is surjective by definition, an ideal MSP is an MSP for which ψ is bijective – i.e. each party receives exactly one share.

The associated access structure for an MSP is such that $\boldsymbol{\varepsilon}$ is contained in the linear span of the rows of M owned by any qualified set of parties, and also so that $\boldsymbol{\varepsilon}$ is *not* in the linear span of the rows owned by any unqualified set of parties. It is well known that, given a monotone access structure (Γ, Δ), there exists an MSP $\mathcal{M}$ computing it [24, 28, 29].

In more detail: A qualified set of parties $Q \in \Gamma$ can compute the secret from the qualified subvector $\mathbf{s}_Q$ because by construction of M there is a publicly-known recombination vector $\boldsymbol{\lambda}$ associated to this set Q such that $\psi(supp(\boldsymbol{\lambda})) \subseteq Q$ and $M^\top \boldsymbol{\lambda} = \boldsymbol{\varepsilon}$. Note that while $\psi(supp(\boldsymbol{\lambda})) \subseteq Q$, this subset of Q must still be qualified – it just may be the case that not all of the parties' shares are required to reconstruct the secret (for example, if multiple parties hold the same share). Since $\psi(supp(\boldsymbol{\lambda})) \subseteq Q$, we have $\langle \boldsymbol{\lambda}, \mathbf{s} \rangle = \langle \boldsymbol{\lambda}_Q, \mathbf{s}_Q \rangle$, so given $\mathbf{s}_Q$ the parties can compute $\langle \boldsymbol{\lambda}_Q, \mathbf{s}_Q \rangle$, and since $\langle \boldsymbol{\lambda}_Q, \mathbf{s}_Q \rangle = \langle \boldsymbol{\lambda}, \mathbf{s} \rangle = \langle \boldsymbol{\lambda}, M \cdot \mathbf{x} \rangle = \langle M^\top \boldsymbol{\lambda}, \mathbf{x} \rangle = \langle \boldsymbol{\varepsilon}, \mathbf{x} \rangle = s$, they can thus determine the secret.

Conversely, for any unqualified set of parties $U \in \Delta$, again by construction of M we have that $\boldsymbol{\varepsilon} \notin \mathrm{im}(M_U^\top)$, which is equivalent to each of the following three statements:

- $\boldsymbol{\varepsilon} \notin \ker(M_U)^\perp$
- $\exists\, \mathbf{k} \in \ker(M_U)$ such that $\langle \boldsymbol{\varepsilon}, \mathbf{k} \rangle \neq 0$
- $\exists\, \mathbf{k} \in \mathbb{F}^d$ such that $M_U \cdot \mathbf{k} = \mathbf{0}$ with $\langle \boldsymbol{\varepsilon}, \mathbf{k} \rangle = 1$

From the last statement, we can see that for any secret s, for any randomness vector $\mathbf{x} \in \mathbb{F}^d$ encoding it – i.e. where $\langle \mathbf{x}, \boldsymbol{\varepsilon} \rangle = s$ – for any other secret $s' \in \mathbb{F}$ we have $M_U\mathbf{x} = M_U\mathbf{x} + \mathbf{0} = M_U\mathbf{x} + M_U((s' - s) \cdot \mathbf{k}) = M_U(\mathbf{x} + (s' - s) \cdot \mathbf{k})$. Thus if $\mathbf{x}$ encodes the secret s, then the randomness vector $\mathbf{x} + (s' - s) \cdot \mathbf{k}$ encodes s' by linearity of the inner product, but the share vectors held by parties in U are the same. Thus the set of shares received by an unqualified set of parties provides no information about the secret.

In this work we show that for any MSP computing any $\mathcal{Q}_2$ access structure, there exists a matrix N such that for any vector $\mathbf{e} \neq \mathbf{0}$ for which $\psi(\mathit{supp}(\mathbf{e})) \notin \mathit{\Gamma}$, we either have $N \cdot \mathbf{e} \neq \mathbf{0}$, or $N \cdot \mathbf{e} = \mathbf{0}$ and $\langle \mathbf{e}, \boldsymbol{\varepsilon} \rangle = 0$. The matrix N is essentially the parity-check matrix of the code generated by the matrix M of the MSP and turns out to be very useful for efficiently detecting cheating behaviour.

2.3 MPC

Network: We assume secure point-to-point channels. When broadcasting shares but we do not assume broadcast channels: in this context we mean an honest party sends the same element to each other party over the given secure channel.

Security Model: Our protocols are modelled and proved secure in the Universal Composability (UC) framework introduced by Canetti [13] and we assume the reader is familiar with it. We assume static corruptions by the adversary, meaning that the adversary corrupts some set of parties once at the beginning of the protocol. We will usually denote the set of parties the adversary corrupts by $A \subseteq \mathcal{P}$. We assume the adversary is active, meaning that the corrupted parties may execute arbitrary code determined by the adversary, and additionally we allow the protocol to abort prematurely – i.e. the protocols are *actively-secure with abort*. The protocol is secure against a computationally bounded adversary, who must find a collision of the hash function to cheat without causing the protocol to abort.

Pre-processing: Many modern MPC protocols split computation into two phases, the *offline* or *pre-processing phase* and the *online phase*. In the offline phase, the parties engage in several rounds of communication to produce data which can then be used in the online phase. The purpose of doing this is that the pre-processing can be done at any time prior to the execution of the online phase, can be made independent of the function to be computed, and may use expensive public-key primitives, in order to allow the online phase to use only fast information-theoretic primitives. In our protocol design, we follow this model, although we only require symmetric-key primitives throughout since the access structure is $\mathcal{Q}_2$.

Hash authentication: The work of Furakawa et al. [25] is in the three-party honest majority case. A secret is additively split into three parts, and each party is given a different set of two of them. To open a secret, each party sends to one other party the share that party is missing, symmetrically. This suffices for all parties to obtain all shares, but does not ensure that the one corrupt party sent the correct share. This is where the hash evaluation comes in: after a secret is

Hash API

The hash API implemented via the hash function $H : \mathbb{F}^* \to \{0,1\}^\lambda$ consists of the following three algorithms:

- $H.\mathsf{Initialise}()$: the hash function is initialised.
- $H.\mathsf{Update}(\mathbf{s})$: the hash function is updated with the vector $\mathbf{s}$.
- $H.\mathsf{Output}()$: the hash function is evaluated and output provided.

Figure 1. Hash API

opened, all parties update their hash function locally with all three shares (the two they held and the one they received); after possibly many secrets are opened, the parties broadcast (here meaning each party sends to the other two parties over a secure channel) the outputs of their hash evaluations and compare what they received with what they computed themselves. If any hashes differ, they abort. This process ensures that the shares held by all parties are consistent, even though each party need only send one share to one party per opening. If many shares are opened in the execution of the protocol (as is the case in SPDZ-like protocols, since every multiplication requires two secrets to be opened), this significantly reduces communication overhead, at the cost of cryptographic assumptions for the existence of a collision-resistant hash function. This was generalised to any replicated scheme $\mathcal{Q}_2$ LSSS by Keller et al. [31].

In our work, we apply similar techniques to Furukawa et al. and Keller et al. to the problem of opening values to parties, but in a significantly more general case. We achieve this by proving the folklore results that say an LSSS is error-detecting if and only if it is $\mathcal{Q}_2$. Our protocol will use the "standard" hash function API given in Figure 1; in brief, our methods are as follows:

- If single party P_i is required to learn a secret, all the other parties send all of their shares to P_i, and then P_i performs an error-detection check on the shares received, telling all parties to abort if errors are detected.
- If all parties are required to learn a secret, the parties engage in a round of communication in which not all parties need to communicate with each other. The parties reconstruct a view of what they think other parties have received, even if they have not communicated with all other parties. After opening possibly many secrets, each party calls Output on the hash function, broadcasts their output, and checks every other party's hash value against their own; we will see that this process authenticates the secrets.

In the next two sections we outline why the methodologies for the two cases are correct. The proof of security of our protocol can be found in the full version.

3 Opening a Value to One Party

In this section, we show that for an LSSS realising a $\mathcal{Q}_2$ access structure, if the share vector $\mathbf{s}$ for some secret s is modified with an error vector $\mathbf{e}$ with unqualified support then $\mathbf{s} + \mathbf{e}$ is either no longer a valid share vector (i.e. is not in $\mathrm{im}(M)$), or the error vector encodes 0, and so by linearity $\mathbf{s}+\mathbf{e}$ also encodes s. In our MPC protocol, this will provide an efficient method by which a party to whom a secret is opened (by all other parties sending that party all of their shares) can check whether or not the adversary has introduced an error. The procedure of opening to a single party is necessary in order for the parties to provide input and obtain output in an actively-secure manner.

Lemma 1. *For any MSP $\mathcal{M} = (\mathbb{F}, M, \boldsymbol{\varepsilon}, \psi)$ computing a $\mathcal{Q}_2$ access structure Γ, for any vector $\mathbf{s} \in \mathbb{F}^m$,*

$$\psi(\mathit{supp}(\mathbf{s})) \notin \Gamma \implies \begin{cases} \mathbf{s} \notin \mathit{im}(M), \text{ or} \\ \mathbf{s} \in \mathit{im}(M) \text{ and } \mathbf{s} = M\mathbf{x} \text{ for some } \mathbf{x} \in \mathbb{F}^d \text{ where } \langle \mathbf{x}, \boldsymbol{\varepsilon} \rangle = 0. \end{cases}$$

Proof. If $\psi(\mathit{supp}(\mathbf{s})) \notin \Gamma$ then $\mathcal{P} \setminus \psi(\mathit{supp}(\mathbf{s})) \in \Gamma$ since the access structure is $\mathcal{Q}_2$. Thus there is at least one set $Q \in \Gamma$ where $Q \subseteq \mathcal{P} \setminus \psi(\mathit{supp}(\mathbf{s}))$ for which $\mathbf{s}_i = 0$ for all $i \in [m]$ where $\psi(i) \in Q$ (i.e. $\mathbf{s}_Q = \mathbf{0}$), by definition of *supp*.

Recall that for a qualified set Q of parties to reconstruct the secret, they take the appropriate recombination vector $\boldsymbol{\lambda}$ (which has the property that $\psi(\mathit{supp}(\boldsymbol{\lambda})) \subseteq Q$) and compute $s = \langle \boldsymbol{\lambda}, \mathbf{s} \rangle$. For this particular Q and corresponding recombination vector $\boldsymbol{\lambda}$, we have $\langle \boldsymbol{\lambda}, \mathbf{s} \rangle = \langle \boldsymbol{\lambda}_Q, \mathbf{s}_Q \rangle$ since $\psi(\mathit{supp}(\boldsymbol{\lambda})) \subseteq Q$, and $\langle \boldsymbol{\lambda}_Q, \mathbf{s}_Q \rangle = \langle \boldsymbol{\lambda}_Q, \mathbf{0} \rangle = 0$ by the above, so the secret is 0.

If $\mathbf{s} \in \mathrm{im}(M)$ then every set $Q \in \Gamma$ must compute the secret as 0 by the definition of MSP (though note that it is not necessarily the case that $\mathbf{s}_Q = \mathbf{0}$ for all $Q \in \Gamma$). Thus the share vector $\mathbf{s}$ is in $\mathrm{im}(M)$ and encodes the secret $s = 0$.

Otherwise, $\mathbf{s} \notin \mathrm{im}(M)$, and we are done. □

We now show that if the adversary (controlling an unqualified set of parties) adds an error vector $\mathbf{e}$ to a share vector $\mathbf{s}$, the resulting vector $\mathbf{c} := \mathbf{s}+\mathbf{e}$ will either not be a valid share vector, or will encode the same secret as $\mathbf{s}$ (by linearity). Adding in an error $\mathbf{e}$ that does not change the value of the secret can be viewed as the adversary re-randomising the shares he holds for corrupt parties.

Lemma 2. *Let $\mathcal{M} = (\mathbb{F}, M, \boldsymbol{\varepsilon}, \psi)$ be an MSP computing $\mathcal{Q}_2$ access structure Γ and $\mathbf{c} = \mathbf{s} + \mathbf{e}$ be the observed set of shares, given as a valid share vector $\mathbf{s}$ encoding secret s, with error $\mathbf{e}$. Then there exists a matrix N such that*

$$\psi(\mathit{supp}(\mathbf{e})) \notin \Gamma \implies \text{either } \mathbf{e} \text{ encodes the error } e = 0, \text{ or } N \cdot \mathbf{c} \neq \mathbf{0}$$

Proof. Let N be any matrix whose rows form a basis of $\ker(M^\top)$ and suppose $\mathbf{e} \in \mathbb{F}^m$. By the fundamental theorem of linear algebra, $\ker(M^\top) = \mathrm{im}(M)^\perp$, so $\mathbf{s} \in \mathrm{im}(M)$ if and only if $N \cdot \mathbf{s} = \mathbf{0}$. Since $\psi(\mathit{supp}(\mathbf{e})) \notin \Gamma$, then by Lemma 1 we have that either $\mathbf{e} \notin \mathrm{im}(M)$, or $\mathbf{e} \in \mathrm{im}(M)$ and $e = 0$.

If $\mathbf{e} \in \mathrm{im}(M)$ then $e = 0$ and we are done, whilst if $\mathbf{e} \notin \mathrm{im}(M)$ then $N \cdot \mathbf{e} \neq \mathbf{0}$. In the latter case, since $\mathbf{s} \in \mathrm{im}(M)$ we have $N \cdot \mathbf{s} = \mathbf{0}$ and hence $N \cdot \mathbf{c} = N \cdot (\mathbf{s}+\mathbf{e}) = N \cdot \mathbf{s} + N \cdot \mathbf{e} = \mathbf{0} + N \cdot \mathbf{e} \neq \mathbf{0}$. □

The matrix N is usually called the *cokernel* of M, and can be viewed as the parity-check matrix of the code defined by generator matrix M. The method to open a secret to a single party P_i is then immediate: all parties send their shares to P_i, who then concatenates the shares into a share vector $\mathbf{s}$ and computes $N \cdot \mathbf{s}$. Since the adversary controls an unqualified set of parties, if $N \cdot \mathbf{s} = 0$ then by Lemma 2 the share vector $\mathbf{s}$ encodes the correct secret. In this case, P_i recalls any recombination vector $\boldsymbol{\lambda}$ and computes the secret as $s = \langle \boldsymbol{\lambda}, \mathbf{s} \rangle$, and otherwise tells the parties to abort.

4 Opening a Value to All Parties

To motivate our procedure for opening to all parties and to show that it is correct, we first discuss the naïve method of opening shares in a semi-honest protocol, then show how to reduce the communication, and then explain how to obtain a version which is actively-secure (with abort).

To open a secret in a passively-secure protocol, all parties can broadcast all of their shares so that all parties can reconstruct the secret. This method contains redundancy if the access structure is not full-threshold since proper subsets of parties can reconstruct the secret by definition of the access structure. This implies the existence of "minimal" communication patterns for each access structure and LSSS, in which parties only communicate sufficiently for every party to have all shares corresponding to a qualified set of parties.

When bootstrapping to active security, we see that the redundancy allows verification of opened secrets: honest parties can check *all* other parties' broadcasted shares for correctness. When reducing communication with the aim of avoiding the redundancy of broadcasting, honest parties must still be able to detect when the adversary sends inconsistent or erroneous shares. In particular, parties *not* receiving shares from the adversary must also be able to detect that cheating has occurred in spite of not directly being sent erroneous shares.

To achieve this, in our protocol each party will receive enough shares from other parties to determine "optimistically" all shares held by all parties – that is, reconstruct the entire share vector – and then all parties will compare their reconstructed share vectors. To amortise the cost of comparison, the parties will actually update a local collision-resistant hash function each time they reconstruct a new share vector and will then compare the final output of the hash function at the end of the computation, when output is required. This, in essence, is the idea behind the protocols of Furakawa et al. [25] and Keller et al. [31] that are tailored to replicated secret-sharing.

To fix ideas, consider the case of Shamir's scheme: a set of $t+1$ distinct points determines a unique polynomial of degree at most t that passes through them. This fact not only enables the secret to be computed using $t+1$ shares, but additionally enables determining the entire polynomial (the coefficients of which are the share vector for the scheme) and consequently all other shares.

For some LSSSs it is not the case that *any* qualified set of parties have enough information to reconstruct all shares[3].

To allow the parties to perform reconstruction, each party is assigned a set of shares that it will receive, which we encode as a map $\mathsf{q} : \mathcal{P} \to 2^{[m]}$ defined as follows: for each $P_i \in \mathcal{P}$, define $\mathsf{q}(P_i)$ to be a set $S_i \subseteq [m]$ such that:

- $\ker(M_{S_i}) = \{\mathbf{0}\}$; that is, the kernel of the submatrix M restricted to the rows indexed by S_i, is trivial; and
- $\psi^{-1}(\{P_i\}) \subseteq S_i$, where ψ^{-1} denotes the preimage of the row-map ψ; that is, each party includes all of their own shares in the set S_i.

These sets are used as follows. Each P_i receives a set of shares, denoted by $\mathbf{s}^i_{\mathsf{q}(P_i)}$, for a given secret. Then in order to reconstruct all shares, P_i tries to find $\mathbf{x}^i$ such that $\mathbf{s}^i_{\mathsf{q}(P_i)} = M_{\mathsf{q}(P_i)} \cdot \mathbf{x}^i$ and then computes $\mathbf{s}^i = M \cdot \mathbf{x}^i$ as the reconstructed share vector, which is then used to update the hash function (locally). Trivially, we can take $\mathsf{q}(P_i) = [m]$ for all $P_i \in \mathcal{P}$, which corresponds to broadcasting all shares; however, better choices of q result in better communication efficiency. In the full version we give a somewhat-optimised algorithm for finding a "good" map q for a given MSP. We remark that finding the map q is not always as straightforward as it is for replicated secret-sharing in which each party must obtain precisely all the shares it does not have; for many LSSSs, this is overkill: for example, Shamir sharing only requires receiving t shares from other parties, not all $n-1$ other shares it does not possess.

If such an $\mathbf{x}^i$ does not exist then it must be because the adversary sent one or more incorrect shares, because $\mathbf{s}^i_{\mathsf{q}(P_i)}$ should be a subvector of *some* share vector. In this case, the party or parties unable to reconstruct tell all parties to abort.

If such an $\mathbf{x}^i$ does exist for each party then the adversary could still cause different parties to reconstruct different share vectors (and thus output different secrets), but then the hashes would differ and the honest parties would abort. The first condition, $\ker(M_{S_i}) = \{\mathbf{0}\}$, ensures that if all parties follow the protocol, they all reconstruct the *same* share vector, since there are multiple possible share vectors for a given secret, otherwise an honest execution may lead to an abort.

Indeed, the only thing the adversary can do without causing abort – either immediately or later on when hashes are compared – is to change his shares so that his shares combined with the honest parties' shares form a valid share vector. Intuitively, one can think of this as the adversary re-randomising the shares owned only by corrupt parties, which is not possible in Shamir or replicated secret-sharing, but is in DNF-based sharing, and in general is possible if and only if the LSSS admits non-trivial share vectors with unqualified support.

More formally, we have the following lemma that shows that if all parties *can* reconstruct share vectors and the share vectors are consistent, then the adversary cannot have introduced an error.

[3] In the full version we provide a formal description of MSPs in which all qualified sets of parties can reconstruct the entire share vector and explain how such MSPs are "good" for our protocol.

Protocol Π_{Opening}

For each $P_i \in \mathcal{P}$, the parties decide on some $\boldsymbol{\lambda}^i$, which is any recombination vector such that $supp(\boldsymbol{\lambda}^i) \subseteq \mathsf{q}(P_i)$. See Section 4 for the definition of q. We denote by H^i the hash function updated locally by P_i which will be initialised as in Figure 5, at the start of the MPC protocol. If at any point a party receives the message Abort, it runs the subprotocol **Abort**.

OpenTo(i):
If $i = 0$, the secret s encoded via share vector $\mathbf{s}$, is to be opened to all, otherwise it is to be opened only to player P_i.

If $i = 0$ then each $P_j \in \mathcal{P}$ does the following:

1. Retrieve from memory the recombination vector $\boldsymbol{\lambda}^j$.
2. For each $P_\ell \in \mathcal{P}$, for each $k \in \mathsf{q}(P_\ell)$, if $\psi(k) = P_j$ then send $\mathbf{s}_k$ to P_ℓ.
3. For each $k \in \mathsf{q}(P_j)$, wait to receive $\mathbf{s}_k$ from party $\psi(k)$.
4. Concatenate local and received shares into a vector denoted by $\mathbf{s}^j_{\mathsf{q}(P_j)} \in \mathbb{F}^{|\mathsf{q}(P_j)|}$.
5. (Locally) output $s = \langle \boldsymbol{\lambda}^j_{\mathsf{q}(P_j)}, \mathbf{s}^j_{\mathsf{q}(P_j)} \rangle$.
6. Solve $M_{\mathsf{q}(P_j)} \cdot \mathbf{x}^j = \mathbf{s}^j_{\mathsf{q}(P_j)}$ for $\mathbf{x}^j$. If there are no solutions, run **Abort**.
7. Execute $H^j.\mathsf{Update}(M\mathbf{x}^j)$.

If $i \neq 0$, the secret encoded via share vector $\mathbf{s}$ is to be opened to party P_i. The parties do the following:

1. Each $P_j \in \mathcal{P} \setminus \{P_i\}$ sends $\mathbf{s}_{\{P_j\}}$ to P_i, who concatenates local and received shares into a vector $\mathbf{s}$.
2. Party P_i computes $N \cdot \mathbf{s}$; if it is equal to $\mathbf{0}$, P_i (locally) outputs $s = \langle \boldsymbol{\lambda}^i, \mathbf{s} \rangle$, and otherwise runs **Abort**.

Verify: Each $P_i \in \mathcal{P}$ does the following:

1. Compute $h^i := H^i.\mathsf{Output}()$.
2. Send h^i to all other parties $P_j \in \mathcal{P} \setminus \{P_i\}$ over pair-wise secure channels.
3. Wait for h^j from all other parties $P_j \in \mathcal{P} \setminus \{P_i\}$.
4. If $h^j \neq h^i$ for any j, run **Abort**.

Abort: If a party calls this subroutine, it sends a message Abort to all parties and aborts. If a party receives a message Abort, it aborts.

Broadcast: When P_i calls this procedure to broadcast a value s,

1. Party P_i sends the secret s to all other players over pair-wise secure channels.
2. When party P_j receives the share, it executes $H^j.\mathsf{Update}(s)$.

Figure 2. Protocol Π_{Opening}

Lemma 3. *Let $q : \mathcal{P} \to 2^{[m]}$ be defined as above and let $\mathbf{s}^i_{q(i)}$ denote the subvector of shares received by party P_i for a given secret. Suppose it is possible for each party $P_i \in \mathcal{P}$ to find a vector $\mathbf{x}^i$ such that $\mathbf{s}^i_{q(P_i)} = M_{q(P_i)}\mathbf{x}^i$; let $\mathbf{s}^i := M \cdot \mathbf{x}^i$*

for each $i \in [n]$. If $\mathbf{s}^i = \mathbf{s}^j$ for all honest parties P_i and P_j, then the adversary did not introduce an error on the secret.

Proof. The existence of q follows from the fact that "at worst" we can take $\mathsf{q}(P_i) = [m]$ for all $P_i \in \mathcal{P}$. There is a unique $\mathbf{x}^i$ solving $\mathbf{s}^i_{\mathsf{q}(P_i)} = M_{\mathsf{q}(P_i)} \cdot \mathbf{x}^i$ (not *a priori* necessarily the same for all parties) because $\ker(M_{\mathsf{q}(P_i)}) = \{\mathbf{0}\}$ for all $P_i \in \mathcal{P}$ by the first requirement in the definition of q.

Let A denote the set of corrupt parties. Since A is unqualified, the honest parties form a qualified set $Q = \mathcal{P} \setminus A$ since the access structure is $\mathcal{Q}_2$.

Each honest party uses their own shares in the reconstruction process by the second requirement in the definition of q, so if $\mathbf{s}^i = \mathbf{s}^j$ for all honest parties P_i and P_j, then in particular they all agree on a qualified subvector defined by honest shares – i.e. $\mathbf{s}^i_Q = \mathbf{s}^j_Q$ for all honest parties P_i and P_j. Thus some qualified subvector of the share vector is well defined, which uniquely defines the secret by definition of MSP. □

As mentioned in the introduction, our results in the last two sections are somewhat analogous to the result of Cramer et al. [19, Thm 1] which roughly shows that for a strongly multiplicative LSSS implementing a $\mathcal{Q}_3$ access structure, honest parties can always agree on the correct secret (when all parties broadcast their shares). In Figure 2 we present the methods we use to open secret shared data in different situations.

Functionality $\mathcal{F}_{\mathsf{Prep}}$

The functionality maintains a list Value of secrets that it stores. The set A indexes the corrupt parties (unknown to the honest parties).

Triples: On input (Triple, N_T) from all parties, the functionality does the following:

1. For i from 1 to N_T:
   (a) Sample $a^i, b^i \xleftarrow{\$} \mathbb{F}$ and compute share vectors $\mathbf{a}^i$ and $\mathbf{b}^i$.
   (b) Send $(\mathbf{a}^i_A, \mathbf{b}^i_A)$ to the adversary.
   (c) Receive a subvector of shares $\tilde{\mathbf{c}}^i_A$ from the adversary.
   (d) Compute a vector $\mathbf{c}^i = M \cdot \mathbf{x}^i_c$ such that $\langle \mathbf{x}^i_c, \boldsymbol{\varepsilon} \rangle = a^i \cdot b^i$ and $\mathbf{c}^i_A = \tilde{\mathbf{c}}^i_A$. If no such vector $\mathbf{c}^i$ exists, set an internal flag `Abort` to true and continue.
2. Wait for a message OK or Abort from the adversary.
3. If the response is OK and the internal flag `Abort` has not been set to true, for each honest $P_i \in \mathcal{P}$, send $(\mathbf{a}^i_{\{P_i\}}, \mathbf{b}^i_{\{P_i\}}, \mathbf{c}^i_{\{P_i\}})_{i=1}^{N_T}$ to each honest party P_i, and otherwise output the message Abort to all honest parties and abort.

Figure 3. Functionality $\mathcal{F}_{\mathsf{Prep}}$

Functionality $\mathcal{F}_{\mathsf{MPC}}$

Initialise: On input Init from all parties, the functionality initialises the array $\mathsf{Value}[]$. Accept a message OK or Abort from the adversary; if the message is OK then continue, and otherwise send the message Abort to all parties and abort.

Input: On input $(\mathsf{Input}, \mathsf{id}, x)$ from party P_i and $(\mathsf{Input}, \mathsf{id}, \perp)$ from all other parties, where id is a fresh identifer, the functionality sets $\mathsf{Value}[\mathsf{id}] := x$.

Add: On input $(\mathsf{Add}, \mathsf{id}_1, \mathsf{id}_2, \mathsf{id}_3)$ from all parties, if id_3 is a fresh identifier and $\mathsf{Value}[\mathsf{id}_1]$ and $\mathsf{Value}[\mathsf{id}_2]$ have been defined, the functionality sets $\mathsf{Value}[\mathsf{id}_3] := \mathsf{Value}[\mathsf{id}_1] + \mathsf{Value}[\mathsf{id}_2]$.

Multiply: On input $(\mathsf{Multiply}, \mathsf{id}_1, \mathsf{id}_2, \mathsf{id}_3)$ from all parties, if id_3 is a fresh identifier and $\mathsf{Value}[\mathsf{id}_1]$ and $\mathsf{Value}[\mathsf{id}_2]$ have been defined, the functionality waits for a message OK or Abort from the adversary. If the adversary sends the message Abort, send the message Abort to all parties and the adversary and abort, and otherwise set $\mathsf{Value}[\mathsf{id}_3] := \mathsf{Value}[\mathsf{id}_1] \cdot \mathsf{Value}[\mathsf{id}_2]$.

Output: On input $(\mathsf{Output}, \mathsf{id}, i)$ from all parties, if $\mathsf{Value}[\mathsf{id}]$ has been defined, the functionality does the following:

- If $i = 0$, send $\mathsf{Value}[\mathsf{id}]$ to the adversary and wait for a signal OK or Abort in return. If it signals Abort, send the message Abort to all parties and the adversary and abort, and otherwise send $\mathsf{Value}[\mathsf{id}]$ to all parties. If not aborted, wait for another signal OK or Abort. If the adversary signals Abort, send the message Abort to all parties and the adversary and abort.
- If $i \neq 0$ and P_i is corrupt, then the functionality sends $\mathsf{Value}[\mathsf{id}]$ to the adversary and waits for the adversary to signal OK or Abort. If it signals Abort, send the message Abort to all parties and abort.
- If $i \neq 0$ and P_i is honest, the functionality waits for the adversary to signal OK or Abort. If it signals Abort, send the message Abort to all parties and the adversary and abort, and otherwise send $\mathsf{Value}[\mathsf{id}]$ to P_i.

Figure 4. Functionality $\mathcal{F}_{\mathsf{MPC}}$

5 MPC Protocol

We are now ready to present our protocol to implement the MPC functionality offering active security with abort as given in Figure 4. We present the online method here, leaving the offline method for the full version. Our offline method is much more scalable than [31] since the dependence on replicated secret-sharing is removed. The offline method implements the functionality given in Figure 3. Our online protocol, in Figure 5, makes use of the opening protocol Π_{Opening} given in Figure 2 earlier. The majority of our protocol uses standard MPC techniques for secret-sharing. In particular, the equation the parties compute for the multiplication is a standard application of Beaver's circuit randomisation technique [2], albeit for a general LSSS.

Correctness of our input procedure follows from the input method given in the non-interactive pseudo-random secret-sharing protocol of [17]. In particular for party P_i to provide an input s in a secret-shared form $\mathbf{s}$, the parties will first take a secret-sharing $\mathbf{r}$ of a uniformly random secret r – which is some a or b from a Beaver triple – and open it by calling **OpenTo**(i). Then P_i determines the encoded secret (using any recombination vector) and broadcasts $\epsilon := s - r$. The parties compute the share vector as $\mathbf{s} := \epsilon \cdot \mathbf{u} + \mathbf{r}$ where $\mathbf{u}$ is a pre-agreed sharing of 1, which may be the same vector used to compute all inputs, by which we mean that for $i \in [m]$, party $\psi(i)$ computes $\mathbf{s}_i := \epsilon \cdot \mathbf{u}_i + \mathbf{r}_i$. Since this r is uniformly random by assumption, it hides the input s in the broadcast of ϵ. This is proved formally in our simulation proof.

We have the following proposition, which we prove in the full version under the UC framework of Canetti [14]. Here we use $(\Pi_{\mathsf{MPC}} \| \Pi_{\mathsf{Opening}})$ to mean simply that the union of the procedures from both protocols are used.

Proposition 1. *The protocol* $(\Pi_{\mathsf{MPC}} \| \Pi_{\mathsf{Opening}})$ *securely realises* $\mathcal{F}_{\mathsf{MPC}}$ *for a* $\mathcal{Q}_2$ *access structure in the presence of a computationally-bounded active adversary, corrupting any unqualified set of parties, in the* $\mathcal{F}_{\mathsf{Prep}}$*-hybrid model, assuming the existence of a collision-resistant hash function and point-to-point secure channels.*

We note that since we do not use MACs, we can also instantiate our protocol over small finite fields[4], or indeed using a LSSS over a ring. The latter will hold as long as the reconstruction vectors can be defined over the said ring. By taking a ring such as $\mathbb{Z}/2^{32}\mathbb{Z}$ we thus generalise the Sharemind methodology [11] to an arbitrary $\mathcal{Q}_2$ structure. Also note that we can extend $\mathcal{F}_{\mathsf{Prep}}$ in a trivial way so as to obtain other forms of pre-processing such shares of bits etc. as in [22].

Acknowledgements

We thank for the anonymous reviewers for their helpful comments and remarks. This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT, by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract No. N66001-15-C-4070, and by EPSRC via grant EP/N021940/1.

References

1. Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 16. pp. 805–817. ACM Press (Oct 2016)
2. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (Aug 1992)

[4] If using a small ring/finite field we simply need to modify the sacrificing stage in the triple production process; no changes are needed for the online phase at all.

3. Beaver, D., Wool, A.: Quorum-based secure multi-party computation. In: Nyberg, K. (ed.) EUROCRYPT'98. LNCS, vol. 1403, pp. 375–390. Springer, Heidelberg (May / Jun 1998)
4. Beimel, A., Gál, A., Paterson, M.: Lower bounds for monotone span programs. In: 36th FOCS. pp. 674–681. IEEE Computer Society Press (Oct 1995)
5. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th ACM STOC. pp. 1–10. ACM Press (May 1988)
6. Ben-Sasson, E., Fehr, S., Ostrovsky, R.: Near-linear unconditionally-secure multiparty computation with a dishonest minority. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 663–680. Springer, Heidelberg (Aug 2012)
7. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 169–188. Springer, Heidelberg (May 2011)
8. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-order threshold implementations. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 326–343. Springer, Heidelberg (Dec 2014)
9. Blakley, G.R.: Safeguarding cryptographic keys. Proceedings of AFIPS 1979 National Computer Conference 48, 313–317 (1979)
10. Bogdanov, D., Kamm, L., Kubo, B., Rebane, R., Sokk, V., Talviste, R.: Students and taxes: a privacy-preserving social study using secure computation. Cryptology ePrint Archive, Report 2015/1159 (2015), http://eprint.iacr.org/2015/1159
11. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: A framework for fast privacy-preserving computations. In: Jajodia, S., López, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 192–206. Springer, Heidelberg (Oct 2008)
12. Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J., Schwartzbach, M.I., Toft, T.: Secure multiparty computation goes live. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 325–343. Springer, Heidelberg (Feb 2009)
13. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of Cryptology 13(1), 143–202 (2000)
14. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067 (2000), http://eprint.iacr.org/2000/067
15. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th ACM STOC. pp. 11–19. ACM Press (May 1988)
16. Chida, K., Genkin, D., Hamada, K., Ikarashi, D., Kikuchi, R., Lindell, Y., Nof, A.: Fast large-scale honest-majority MPC for malicious adversaries. In: Advances in Cryptology - CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 34–64. Springer (2018)
17. Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 342–362. Springer, Heidelberg (Feb 2005)
18. Cramer, R., Damgård, I., Maurer, U.M.: General secure multi-party computation from any linear secret-sharing scheme. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 316–334. Springer, Heidelberg (May 2000)
19. Cramer, R., Daza, V., Gracia, I., Urroz, J.J., Leander, G., Martí-Farré, J., Padró, C.: On codes, matroids and secure multi-party computation from linear secret sharing schemes. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 327–343. Springer, Heidelberg (Aug 2005)

20. Damgård, I., Geisler, M., Krøigaard, M., Nielsen, J.B.: Asynchronous multiparty computation: Theory and implementation. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 160–179. Springer, Heidelberg (Mar 2009)
21. Damgård, I., Ishai, Y., Krøigaard, M.: Perfectly secure multiparty computation and the computational overhead of cryptography. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 445–465. Springer, Heidelberg (May / Jun 2010)
22. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 1–18. Springer, Heidelberg (Sep 2013)
23. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (Aug 2012)
24. van Dijk, M.: Secret Key Sharing and Secret Key Generation. Ph.D. thesis, Eindhoven University of Technology (1997)
25. Furukawa, J., Lindell, Y., Nof, A., Weinstein, O.: High-throughput secure three-party computation for malicious adversaries and an honest majority. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 225–255. Springer, Heidelberg (Apr / May 2017)
26. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press (May 1987)
27. Hirt, M., Maurer, U.M.: Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In: Burns, J.E., Attiya, H. (eds.) 16th ACM PODC. pp. 25–34. ACM (Aug 1997)
28. Ito, M., Saito, A., Nishizeki, T.: Secret sharing schemes realizing general access structure. In: Proc. IEEE Global Telecommunication Conf. (Globecom'87). pp. 99–102 (1987)
29. Karchmer, M., Wigderson, A.: On span programs. In: Proceedings of Structures in Complexity Theory. pp. 102–111 (1993)
30. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 16. pp. 830–842. ACM Press (Oct 2016)
31. Keller, M., Rotaru, D., P. Smart, N., Wood, T.: Reducing communication channels in MPC. In: International Conference on Security in Communication Networks, SCN 2018. pp. 181–199 (01 2018)
32. Maurer, U.M.: Secure multi-party computation made simple. Discrete Applied Mathematics 154(2), 370–381 (2006)
33. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of nonlinear functions in the presence of glitches. Journal of Cryptology 24(2), 292–321 (Apr 2011)
34. Reparaz, O., Bilgin, B., Nikova, S., Gierlichs, B., Verbauwhede, I.: Consolidating masking schemes. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 764–783. Springer, Heidelberg (Aug 2015)
35. Shamir, A.: How to share a secret. Communications of the Association for Computing Machinery 22(11), 612–613 (Nov 1979)
36. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS. pp. 162–167. IEEE Computer Society Press (Oct 1986)

Protocol $\mathit{\Pi}_{\mathsf{MPC}}$

Note that this protocol calls on procedures from $\mathit{\Pi}_{\mathsf{Opening}}$ in Figure 2. If a party never receives an expected message from the adversary, we assume the receiving party signals Abort to all other parties and aborts.

Initialise: The parties do the following:

1. Each $P_i \in \mathcal{P}$ executes $H^i.\mathsf{Initialise}()$.
2. The parties call $\mathcal{F}_{\mathsf{Prep}}$ with input (Triple, N_T) get N_T triples.
3. The parties agree on a public sharing of the secret 1, denoted by $\mathbf{u}$.
4. Each party has one random secret opened to them for every input they will provide to the protocol: the parties do the following:
   (a) Retrieve from memory a sharing $\mathbf{r}$ of a uniformly random secret r, obtained first or second random secret from a Beaver triple. (The secret used may neither be used again for input nor used in a multiplication.)
   (b) Run **OpenTo**(i) on $\mathbf{r}$ so that P_i obtains r.

Input: For party P_i to input secret s,

1. Party P_i retrieves a secret r from memory, corresponding to a share vector $\mathbf{r}$ established during **Initialise** for inputs, and all parties $P_j \in \mathcal{P}$ retrieve their shares $\mathbf{r}_{\{P_j\}}$.
2. Party P_i executes **Broadcast** to open $\epsilon := s - r$.
3. Each party $P_j \in \mathcal{P}$ computes $\mathbf{s}_{\{P_j\}} := \epsilon \cdot \mathbf{u}_{\{P_j\}} + \mathbf{r}_{\{P_j\}}$.

Add: To add secrets s and s', with corresponding share vectors $\mathbf{s}$ and $\mathbf{s}'$, for each $P_i \in \mathcal{P}$ party P_i computes $\mathbf{s}_{\{P_i\}} + \mathbf{s}'_{\{P_i\}}$.

Multiply: To multiply secrets s and s', with corresponding share vectors $\mathbf{s}$ and $\mathbf{s}'$, each $P_i \in \mathcal{P}$ does the following:

1. Retrieve from memory the shares $(\mathbf{a}_{\{P_i\}}, \mathbf{b}_{\{P_i\}}, \mathbf{c}_{\{P_i\}})$ of a triple $(\mathbf{a}, \mathbf{b}, \mathbf{c})$ obtained in **Initialise**.
2. Compute $\mathbf{s}_{\{P_i\}} - \mathbf{a}_{\{P_i\}}$ and $\mathbf{s}'_{\{P_i\}} - \mathbf{b}_{\{P_i\}}$.
3. Run **OpenTo**(0) on $\mathbf{s} - \mathbf{a}$ and $\mathbf{s}' - \mathbf{b}$ to obtain (publicly) $s - a$ and $s' - b$.
4. If the parties have not aborted, compute the following as the share of the product $\mathbf{c}_{\{P_i\}} + (s-a) \cdot \mathbf{s}'_{\{P_i\}} + (s'-b) \cdot \mathbf{s}_{\{P_i\}} - (s-a) \cdot (s'-b) \cdot \mathbf{u}_{\{P_i\}}$.

OutputTo(i): If $i = 0$, the secret s, encoded via share vector $\mathbf{s}$, is to be output to all parties, so the parties do the following:

1. Run **Verify**.
2. If the parties have not aborted, run **OpenTo**(0) on s.
3. If the parties have not aborted, run **Verify** again.
4. If the parties have not aborted, all parties (locally) output s.

If $P_i \in \mathcal{P}$, the secret s encoded via share vector $\mathbf{s}$ is to be output to party P_i, so the parties do the following:

1. Run **Verify**.
2. If the parties have not aborted, run **OpenTo**(i) on s.
3. If P_i has not aborted it (locally) outputs s.

Figure 5. Protocol $\mathit{\Pi}_{\mathsf{MPC}}$