Quorum-based Secure Multi-Party Computation

Donald Beaver[1] and Avishai Wool[2]

[1] IBM/Transarc; beaver@transarc.com
[2] Bell Laboratories, Lucent Technologies; yash@research.bell-labs.com

Abstract. This paper describes efficient protocols for multi-party computations that are information-theoretically secure against passive attacks. The results presented here apply to access structures based on quorum systems, which are collections of sets enjoying a naturally-motivated self-intersection property. Quorum-based access structures include threshold systems but are far richer and more general, and they have specific applicability to several problems in distributed control and management. The achievable limits of security in quorum-based multi-party computation are shown to be equivalent to those determined by Hirt and Maurer in [HM97], drawing a natural but non-obvious connection between quorum systems and the extremes of secure multi-party computation. Moreover, for both the general case and for specific applications, the protocols presented here are simpler and more efficient.

1 Introduction

1.1 Overview

Multi-party computation – a means for groups to engage in joint computation as though an absolutely trusted third party were available to help them – has enjoyed a great deal of attention for several years, with much effort spent on exploring the limits of robustness and efficiency [GMW86, GMW87, GHY88, BGW88, CCD88, RB89, Bea91b, BG89, BH92, DDFY94, CFGN96]. Virtually without exception, all solutions have been based on threshold secret sharing, and hence they are themselves threshold-oriented. Recently, Hirt and Maurer characterized tolerable adversary sets in general multi-party computations without restriction to purely threshold-based sets [HM97]. Our work explores and expands on a non-threshold-based approach to multi-party computation, investigating quorum-based multi-party protocols and providing more efficient solutions for general and specific cases.

Whom to Trust. The general motivation for multi-party computation is simple: apply a decentralized approach to ensure robustness and security in joint computations, increasing reliability by dissipating trust among many individuals. Formally, a collection of n players wish to compute some function $f(x_1, \ldots, x_n)$ of their respective, private inputs, revealing the final result but nothing more. Ideally, they could resort to a trusted third party to collect the inputs and return

[0] Part of this work was done at the DIMACS Research & Education Institute Cryptography and Network Security Workshop (DREI'97), August 1997.

EUROCRYPT'98, Helsinki, Finland, June 1998.

only the result; the goal of secure multi-party computation is to achieve the same task without the existence of any such helper.

Threshold Schemes. Multi-party protocols are typically based on secret sharing. In the past virtually all of them relied on some form of Shamir's polynomial-based sharing scheme [Sha79, Bla79]. Because these sharing schemes are threshold based – *i.e.*, reconstruction of a secret depends on having some minimal number of shares, rather than some particular set of players – there is a natural limit on the types of dishonest coalitions that can be tolerated by multi-party computations. In particular, an honest majority is necessary and sufficient for verifiable sharing and multi-party computation [RB89, Bea91b] – *for threshold-based protocols*. Conversely, as long as no dishonest coalition exceeds the threshold of size $\lfloor \frac{n-1}{2} \rfloor$, secure multi-party computation is achievable. This characterization of tolerable adversaries is natural, elegant, and seemingly all that needs be said.

Weighted Threshold Schemes. The simplest generalization involves a weighted approach, in which important players (such as CEO's and VP's) are essentially granted more shares of the secret than the peons. The characterization of tolerable adversaries remains simple; the weighted collection of shares held by honest players (resp. dishonest players) must be a majority (resp. minority) of all shares. Thus the characterization of what is achievable needs only slight modification. A disadvantage of such a weighted approach is that the important players have more shares so they need to perform more of the computational work.

Quorum Systems. The approach we take here is motivated by distributed control and management problems such as *mutual exclusion* (cf. [GB85]), *data replication protocols* (cf. [DGS85, Her84]) and *name servers* (cf. [MV88]). In these applications certain operations are allowed only if they are authorized by a set of members which is defined to be a *quorum*. The requirement from the quorum sets is that every two quorums should have at least one member in common. The existence of a common member helps prevent uncoordinated action between two different groups, whether that action be committing to a newly-written value or returning the latest version. Clearly, any threshold system (weighted or not) whose threshold exceeds half the total population (or weight) qualifies as a quorum system: the pigeonhole principle ensures that any two threshold-passing sets will intersect nontrivially.

Adding Security. Secret sharing schemes for arbitrary access structures have been developed by Ito, Saito, and Nishizeki [ISN87]. Schemes for structures represented by monotone formulas were proposed by Benaloh and Leichter [BL88]. More recently, Naor and Wool have designed schemes for structures which represent specific quorum systems [NW96]. The natural question to ask is whether these sharing schemes are usable for multi-party *computations* as well. We answer this question in the affirmative, and present several new protocols that use quorum-based secret sharing schemes. In the process we also present some new quorum secret-sharing schemes which we utilize in the protocols. Thus we are able to support non-threshold access structures without resorting to weighted threshold schemes. This allows us to escape the "honest majority" barrier to

some degree — our protocols *can* tolerate some dishonest majorities of players. However, this comes at a price: the protocols cannot tolerate certain dishonest minorities.

Attack Model. In this work we focus on passive attacks, in which all the players follow the protocol. However, dishonest players may pool their information in order to learn some of the secret inputs of other players (this is sometimes called the "honest but curious" model). The adversary is only permitted to choose a dishonest coalition from the collection of coalitions that the protocol tolerates (these will be the complements of quorum sets). On the other hand, our security requirement is of the strongest type: that a dishonest coalition can learn nothing about the other's secret inputs, in the information-theoretic sense.

Efficiency. Once the security of a protocol has been established, we are interested in two measures of its quality: the number of messages sent per multiplication, which captures the communication complexity of the protocol; and the size of the shares each player holds, which captures the amount of local computation each player needs to perform.

1.2 Connecting multi-party computation and quorum systems

Multi-party computation protocols have traditionally been defined and analyzed in terms of the collection $\mathcal{B}$ of bad coalitions that they tolerate. For instance, for a threshold-based protocol this collection is of the form "all the sets of cardinality less than t." Early on it was discovered that there is a simple combinatorial condition which precludes the existence of *any* protocol that tolerates certain types of $\mathcal{B}$'s. The following lemma has been proved in several places [BGW88, BG89, CK91] and follows from a direct reduction to two-party protocols.

Lemma 1. *A set system $\mathcal{B}$ is said to* 2-cover *the universe U if two sets $B_1, B_2 \in \mathcal{B}$ exist such that $B_1 \cup B_2 = U$. If $\mathcal{B}$ 2-covers U, then no multi-party protocol that tolerates $\mathcal{B}$ exists for computing the AND (or OR, or finite-field multiplication) function.*

Recently Hirt and Maurer showed that the converse of Lemma 1 is also true (for honest-but-curious attacks), by describing a general protocol which tolerates any collection $\mathcal{B}$ that does not 2-cover U. Their protocol involves a recursive decomposition of the bad collection $\mathcal{B}$ into 3 sub-collections, and simulating the 2-of-3 threshold-based protocol of [BGW88] at each level of the recursion. Thus they proved the following characterization.

Theorem 2. [HM97] *A multi-party protocol that tolerates $\mathcal{B}$ exists iff U is not 2-covered by $\mathcal{B}$.*

The connection between this characterization and our work is captured by the following observation.

Lemma 3. *Let $\mathcal{B}$ be a collection of bad coalitions over U. Let $\mathcal{G} = \{G : U \setminus G \in \mathcal{B}\}$ be the collection of complements of the bad coalitions. Then $\mathcal{B}$ does not 2-cover U iff $\mathcal{G}$ is a quorum system.*

Proof: Consider some $B_1, B_2 \in \mathcal{B}$, and let $G_1, G_2 \in \mathcal{G}$ be the complements of B_1, B_2, respectively. If $u \notin B_1 \cup B_2$ then clearly $u \in G_1 \cap G_2$, and vice versa. Therefore every two sets of $\mathcal{B}$ do not cover U iff every two sets of $\mathcal{G}$ have a non-empty intersection. ∎

Therefore multi-party protocols are characterized equivalently by the quorum systems that they *respect* and by the bad coalitions they *tolerate*. However, focusing on the positive, quorum-based view leads naturally to the use of quorum secret sharing schemes, which form the foundation for our new protocols.

1.3 Contributions

Our first result is a new secure multi-party protocol which respects any arbitrary quorum system (and hence tolerates any collection of bad coalitions that does not 2-cover U). By this we show an alternative proof to the characterization of Hirt and Maurer [HM97]. However our protocol is much more efficient; for a system of n players and m minimal quorums our protocol sends $O(n^2 m)$ messages per multiplication, as compared to the $\Omega(m^{2.709})$ messages sent by Hirt and Maurer's protocol.[3] Note that in general $m = 2^{\Omega(n)}$, so it is of far greater importance to reduce the dependence on m. Moreover, our protocol is much simpler to describe and to analyze, since it involves neither recursion nor simulation of subprotocols.

We also describe several multi-party protocols which are based on specific quorum systems. The quorum systems that we consider were devised for various distributed control mechanisms and are shown (in the referenced papers) to have favorable properties such as high availability and low load [NW94]. The quorum systems we consider are the crumbling wall quorum systems of [PW97], the finite projective plane (FPP) quorum system of [Mae85], and the 2-of-3 majority-tree systems of [AE91] and [Kum91]. The protocols we provide for these systems are all fully polynomial in n, with a complexity of $O(n^2)$ messages per multiplication. Thus to our knowledge they are the first non-threshold protocols which are as efficient as the original protocol of [BGW88, CCD88].

For the crumbling walls and tree-based systems we use the secret sharing schemes proposed in [NW96]. In the former scheme the shares are twice the size of the secrets, while the latter is optimal (*i.e.*, the size of the shares equals the size of the secret). For the FPP system we present a new optimal secret sharing scheme, which may be interesting in its own right. Thus the local computations that the players perform in all these protocols are extremely efficient, typically consisting of a constant number of multiplications and a linear number of additions and random bit generations.

Organization: In §2 we define quorum systems, secret scharing schemes, and secure multi-party computation. In §3 we present our general quorum-based protocol for arbitrary quorum systems. §4 describes our efficient (polynomial in n) protocols for special quorum systems.

---

[3] In their paper Hirt and Maurer only claim that their protocol is polynomial in m. We provide a more detailed complexity analysis for their algorithm in the sequel.

2 Definitions

2.1 Quorum systems and access specifications

Definition 4. Let U denote a universe of players. A *set system* $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ is a collection of subsets $Q_i \subseteq U$. A *quorum system* is a set system $\mathcal{Q}$ that has the *intersection property:* $Q_i \cap Q_j \neq \emptyset$ for all $Q_i, Q_j \in \mathcal{Q}$. The sets of the system are called *quorums*.

We use $n = |U|$ to denote the number of players. Unless otherwise noted, the quorums in all the quorum systems mentioned in this paper are *minimal:* $Q_i \not\subseteq Q_j$ for every two quorums $Q_i, Q_j \in \mathcal{Q}$. We use $m = |\mathcal{Q}|$ to denote the number of (minimal) quorums.

Definition 5. An *access specification* is a disjoint pair $(\mathcal{G}, \mathcal{B})$ of collections of subsets, such that $\mathcal{G}$ is monotone increasing (that is, $(\forall G_1 \in \mathcal{G})(G_1 \subseteq G_2 \Rightarrow G_2 \in \mathcal{G})$), and $\mathcal{B}$ is monotone decreasing.

Remark: The first collection, $\mathcal{G}$, describes sets of players that are permitted to access secrets; the second collection, $\mathcal{B}$, describes coalitions who should not learn anything about such secrets. Sets in $\mathcal{G}$ are called *good*, while those in $\mathcal{B}$ are called *bad*. Note that $(\mathcal{G}, \mathcal{B})$ need not be a partition; there may be sets that are neither good nor bad. If, however, $(\mathcal{G}, \mathcal{B})$ is a partition, namely $\mathcal{G} \cup \mathcal{B} = 2^U$, then we say it is *unambiguous*.

Definition 6. Let $\mathcal{Q}$ be a quorum system. Let $\mathcal{G}(\mathcal{Q}) = \{G \supseteq Q : Q \in \mathcal{Q}\}$ be the collection of sets containing some quorum, and let $\mathcal{B}(\mathcal{Q}) = \{B : U \setminus B \in \mathcal{G}(\mathcal{Q})\}$ be the collection of sets whose complement contains a quorum. Then the *quorum access specification* of $\mathcal{Q}$ is $(\mathcal{G}(\mathcal{Q}), \mathcal{B}(\mathcal{Q}))$.

Remark: Quorum systems whose access specification is unambiguous have many interesting properties. Such systems have attracted attention in several contexts, and much is known about them. The terminology associated with these systems differs from discipline to discipline, and a non-comprehensive list of names includes "non-dominated coterie" [GB85]; "simple decisive game" [Owe82]; "self-dual monotone boolean function" [IK93]; "ultrafilter" [BKK94]; and "ν-critical hypergraph" [Für88].

2.2 Secret sharing

Definition 7. Let $U = \{1, \ldots, n\}$ and let S be a finite set of secrets. A *secret-sharing scheme* (SSS) is a mapping $\Pi : S \times R \mapsto S_1 \times \cdots \times S_n$, where R is a set of random strings, and for each $i \in U$, S_i is a set of *secret shares*. Π is said to realize an access specification $(\mathcal{G}, \mathcal{B})$ if it satisfies the following conditions:

1. The secret can be reconstructed by any subset in $\mathcal{G}$. That is, associated with every set $G \in \mathcal{G}$ $(G = \{i_1, \ldots, i_{|G|}\})$ there is a function $h_G : S_{i_1} \times \cdots \times S_{i_{|G|}} \mapsto S$ such that for every $(s, r) \in S \times R$, if $\Pi(s, r) = \{s_1, \ldots, s_n\}$ then $h_G(s_{i_1}, \ldots, s_{i_{|G|}}) = s$.

2. No subset in $\mathcal{B}$ can reveal any partial information about the secret (in the information theoretic sense). Formally, for any subset $B \in \mathcal{B}$, for every two secrets $a, b \in S$, and for every possible collection of shares $\{s_i\}_{i \in B}$: $\Pr(\{s_i\}_{i \in B} | a) = \Pr(\{s_i\}_{i \in B} | b)$, where the probability is taken over the random string r.

2.3 Multi-party computation

The following formalizations are standard and based on commonly accepted approaches [GMR89, BGW88, CCD88, Bea91a, MR91]. Let C be a circuit over logical (or finite arithmetical) gates, having n inputs and one output,[4] and let $f(x_1, \ldots, x_n)$ be the function it computes. A *multi-party protocol* is a set of n interactive, probabilistic Turing machines, called *players*. A protocol *computes* f if each player outputs $f(x_1, \ldots, x_n)$ at the end of execution.

An *adversary class* is a collection of subsets of $U = \{1, \ldots, n\}$. (Because we generally discuss the passive case, we identify adversaries with sets of observed players.) A *partial S-view* of an execution is the set of strings describing internal computations, random tosses, and input-output transcripts for players in S, up to some given round of interaction. We often overload the term *view* to mean a specific string in a given execution, the distribution on strings over all executions, or the partial view at some moment, as convenience dictates. An execution of a protocol is characterized by a *result-vector* of n outputs along with the final view of the adversary.

Let simulator S be given an adversary B as input and have access to an ideal protocol in which a trusted, external party collects the inputs $x_1, \ldots, x_n$ and returns the result, $f(x_1, \ldots, x_n)$. In addition to providing progressive partial B-views, the simulator produces a final result-vector (whose adversary-view is the concatenation of the progressive partial B-views).

Definition 8. A protocol that computes f is said to *tolerate* adversary class $\mathcal{B}$ if there is a simulator S such that for any $B \in \mathcal{B}$ and any $x_1, \ldots, x_n$, the result-vector produced by S is identically distributed to that obtained by executing the protocol. (The protocol is also said to tolerate each set $B \in \mathcal{B}$.)

Our constructions follow the conventional *share-compute-reconstruct* paradigm introduced in [GMW86]. Thus we have additional properties to seek, including whether particular sets are capable of reconstructing the final result.

Definition 9. A multi-party protocol for f is said to *respect* a set G if at the end of the computation phase, the members of G can collectively reconstruct the function value. A protocol is said to *respect* a collection of sets if it respects each set in the collection.

Definition 10. A protocol is said to *securely implement* an access specification $(\mathcal{G}, \mathcal{B})$ if it tolerates $\mathcal{B}$ and respects $\mathcal{G}$.

Definition 11. Let $\mathcal{Q}$ be a quorum system. A protocol is said to be $\mathcal{Q}$-private if it securely implements the quorum access specification $(\mathcal{G}(\mathcal{Q}), \mathcal{B}(\mathcal{Q}))$.

[4] This is easily generalized to multiple and private outputs.

The preceding commentary is easily generalized to families of functions. A finite set of protocols is *complete* for some function family if it enables sharing and reconstruction, and through finite composition can evaluate any finite circuit securely. Our approach is standard [GMW86, BGW88, CCD88]: to provide efficient protocols for sharing, reconstruction, addition (linear combination), and multiplication.

3 A general protocol

In this section we present a general multi-party protocol which is $\mathcal{Q}$-private for any quorum system $\mathcal{Q}$. Let $\mathcal{Q}$ be a quorum system, and let C be a circuit which computes a function f. To describe a $\mathcal{Q}$-private protocol which computes f, we need first to show a secret sharing scheme that realizes $(\mathcal{G}(\mathcal{Q}), \mathcal{B}(\mathcal{Q}))$. Then it suffices to show how the players can compute sums and products $\mathcal{Q}$-privately using the shares. The computation of f is performed by simulating the circuit C gate by gate.

3.1 The Gen-SSS secret sharing scheme

Consider a quorum system $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ of minimal quorums, and let x be the value to be shared. First x is represented randomly as a sum-share $x = \sum_{j=1}^{m} x_j$. This can be done by assigning uniformly chosen random values to x_j for $j = 1, \ldots, m-1$ and assigning $x_m = x - \sum_{j=1}^{m-1} x_j$. We call the values $x_1, \ldots, x_m$ the *parts* of x to differentiate them from the shares of the scheme. The share $s_u(x)$ of player u is the set of parts corresponding to the quorums that contain u:

$$s_u(x) \leftarrow \{x_j\}_{Q_j \ni u}.$$

Proposition12. *Gen-SSS is a secret sharing scheme realizing the quorum access specification* $(\mathcal{G}(\mathcal{Q}), \mathcal{B}(\mathcal{Q}))$.

Proof: Consider some set $G \in \mathcal{G}(\mathcal{Q})$ which contains the quorum $Q_i \in \mathcal{Q}$. By the intersection property, $Q_i \cap Q_j \neq \emptyset$ for all $j = 1, \ldots, m$, so for every quorum Q_j there exists a player $u \in Q_i$ which has the corresponding part x_j. Therefore the players in Q_i collectively have all the parts $\{x_j\}_{j=1}^{m}$ and can reconstruct the secret x.

Now consider a set $B \in \mathcal{B}(\mathcal{Q})$. Then by definition there exists some quorum $Q_i \in \mathcal{Q}$ such that $Q_i \subseteq U \setminus B$. Only members of Q_i receive the part x_i, and therefore no player in B has x_i.

Let ξ^x be a specific assignment of Gen-SSS shares which encodes a secret value x. Then for any other possible value y there exists a secret-sharing ξ^y, which encodes y, such that the projections of ξ^x and ξ^y on the set B are identical. ξ^y is constructed from ξ^x by replacing the part x_i by $x_i + (y - x)$, and thus the change only affects the shares held by members of Q_i. Therefore

$$\Pr\left(\{s_u(x)\}_{u \in B} | x\right) = \Pr\left(\{s_u(y)\}_{u \in B} | y\right),$$

so the union of the shares of all the players in B gives no information about x. ∎

3.2 The Gen-MP protocol

The Gen-MP protocol for evaluating a circuit obliviously is the direct composition of protocols for linear combination and multiplication, described below.

Computing a linear combination Assume that two secrets $x = \sum_{i=j}^{m} x_j$ and $y = \sum_{i=j}^{m} y_j$ are shared among the players according to the Gen-SSS scheme. Let α and β be fixed constants. The players represent $z = \alpha x + \beta y$ as $z = \sum_{j=1}^{m} z_j$ as follows. Each player u locally computes

$$s_u(z) \leftarrow \{z_j \leftarrow \alpha x_j + \beta y_j\}_{Q_j \ni u}.$$

It is easy to see that the new $s_u(z)$ shares are valid Gen-SSS shares for z, and since the computation did not involve the exchange of any messages the computation is $\mathcal{Q}$-private.

Computing a product As before let $x = \sum_{j=1}^{m} x_j$ and $y = \sum_{j=1}^{m} y_j$ be shared among the players according to Gen-SSS. The players aim to compute the product $z = x \cdot y = \sum_{i=1}^{m} \sum_{j=1}^{m} x_i y_j$.

The protocol depends on a mapping $\rho : [1 \ldots m] \times [1 \ldots m] \mapsto U$, for which $\rho(i,j) \in (Q_i \cap Q_j)$. Such a mapping ρ exists since by the intersection property we have $(Q_i \cap Q_j) \neq \emptyset$. We say that a player u is *in charge of computing* the term $x_i y_j$ if $\rho(i,j) = u$. In order to compute the product, each player u performs the following steps:

1. Player u locally computes the sum of all the terms she is in charge of: $w_u \leftarrow \sum_{i,j:\rho(i,j)=u} x_i y_j$.
2. Player u secret-shares the value w_u among all the players using the Gen-SSS scheme. Namely, u computes a randomized sum-share version of w_u by $w_u = \sum_{j=1}^{m} w_{uj}$, and sends the part w_{uj} to every player $v \in Q_j$.
3. After player u receives the parts $\{w_{vj}\}_{v \in U, Q_j \ni u}$, she computes her share: $s_u(z) \leftarrow \left\{z_j \leftarrow \sum_{v \in U} w_{vj}\right\}_{Q_j \ni u}$.

Lemma 13. *Protocol Gen-MP computes correct Gen-SSS shares of $z = x \cdot y$.*

Proof: Consider some quorum Q_k. As before, by the intersection property the members of Q_k hold all the parts $\{z_j\}_{j=1}^{m}$, so they can compute the reconstruction function $\sum_{j=1}^{m} z_j$. Plugging in the expressions for w_u and w_{uj} we have that

$$\sum_{j=1}^{m} z_j = \sum_{j=1}^{m} \sum_{u \in U} w_{uj} = \sum_{u \in U} \sum_{j=1}^{m} w_{uj} = \sum_{u \in U} w_u = \sum_{u \in U} \sum_{i,j:\rho(i,j)=u} x_i y_j,$$

and since by the definition of ρ exactly one player is in charge of computing every term $x_i y_j$, the last sum is equal to

$$\sum_{i,j=1}^{m} x_i y_j = x \cdot y. \qquad \blacksquare$$

Theorem 14. *Protocol Gen-MP computes $x \cdot y$ $\mathcal{Q}$-privately.*

Proof: We wish to show that the information gained by B is no different than if the input pieces were given to a trusted party who reconstructs x and y then shares xy. It suffices to provide a simulator that can construct a perfectly accurate view for B using only the z-shares returned to B by such a trusted party.

By definition there exists some quorum $Q \in \mathcal{Q}$ that is *untainted*, namely $Q \cap B = \emptyset$. Note, incidentally, that the trivial case in which Q is of size 1 is easily dealt with — the single, uncorrupted party behaves as a trusted monarch and the protocol collapses, securely.

Let $\beta(B) = \{k : Q_k \cap B \neq \emptyset\}$ be the collection of (indices of) tainted quorums. Consider an $n \times m$ table of all the w_{uj} values appearing in the protocol, whose row sums correspond to the players' w_u values and whose column sums correspond to the final z_j parts. Define the *compromised region* in this table to be

$$W_B = \{(u,j) | u \in B\} \cup \{(u,j) | j \in \beta(B)\}.$$

Along with the compromised values $\{w_u\}_{u \in B}$, W_B indexes the values that B is permitted to observe (within dishonest players or as received from good ones) in the multiplication protocol. The remaining region of $\{w_{uj}\}$, namely the values indexed by $\overline{W}_B$, is not included in B's view, nor are $\{w_u\}_{u \notin B}$.

We describe how to simulate an execution and then show that the fake view can be modified as needed when constraints on x, y or xy are made.

(1) Using the partial inputs $\{x_j, y_j\}_{j \in \beta(B)}$, we first calculate $\{w_u\}_{u \in B}$ directly. We run "bad" players in B honestly on these values, deriving a perfectly accurate distribution for values $\{w_{uj}\}$ for all $u \in B$ and all j. The remaining compromised region lies in rows corresponding to uncompromised players. When the $\{z_j\}_{j \in \beta(B)}$ values are obtained from the imaginary trusted host, this compromised region is generated uniformly at random, subject to column constraints: $\sum_{u \in U} w_{uj} = z_j$.

(2) In an actual execution, the values of $\{w_u\}_{u \notin B}$ are determined by the input shares, then give rise to the sum-shares $\{w_{uj}\}$, which B then partially observes (*i.e.*, those in W_B). The values of $\{w_u\}_{u \in B}$ are likewise determined by input shares, but also revealed to B. For $j \in \beta(B)$, the column sums are determined by values revealed to B; for $j \notin \beta(B)$, the column sums include the uncompromised region, which is chosen uniformly at random (subject to row sums matching the uncompromised $\{w_u\}_{u \notin B}$).

Observe ("notably") that the second stage of experiment (1) is identical to setting arbitrary values in $\{w_u\}_{u \notin B}$ and then choosing the compromised region uniformly at random subject to row and column constraints. By the properties of sum-sharing, this in turn is equivalent to setting parts x_i and y_i for $i \notin \beta(B)$ to arbitrary values, executing honest programs on the results, and reporting only the compromised portion $\{w_{uj}\}_{u \notin B, j \in \beta(B)}$. (The B-view for rows in B is clearly identical in both cases.)

Thus, experiments (1) and (2) provide identical distributions. As long as there exist appropriate settings to the uncompromised values, the "notable" observation (previous paragraph) that shows the distributions are identical continues to hold. Further, burdensome inspection shows that such settings always remain

possible, even when B learns further information, as when the final output value is revealed or when gates are tied together in composition. (Indeed, B may be able to calculate some secrets and uncompromised entries along the way, even though it does not see them directly. The important point is that such entries *can* always be found when facing the simulated views.) ∎

3.3 Communication complexity

Proposition 15. *Denote the average quorum size by* $q = \frac{1}{m}\sum_{j=1}^{m} |Q_j|$. *Then protocol Gen-MP sends* nmq *messages.*

This and several other propositions are proved in the Appendix.

Corollary 16. *The communication complexity of protocol Gen-MP is* $O(n^2 m)$.

Our protocol performs quite favorably in comparison to the general construction of Hirt and Maurer [HM97], whose communication complexity appears to be $\Omega(m^{2.709})$.

To see this, observe that [HM97] involves a recursive construction in which a set of players is replaced by three overlapping sets of players, each with size 2/3 of the original. The depth h of the recursion is thus determined by $(2/3)^h m \leq 3$, which gives $h \approx -\log_3 m/(log_3 2/3) > 2.709 \log_3 m$. The size (and hence communication complexity) thus exceeds $3^h = \Omega(m^{2.709})$.

Because m grows with the number of coalitions and is therefore generally exponential in the number of players, it is of far greater importance to reduce dependence on m. Our protocol incurs only a linear factor of m, with a small polynomial term in n.

4 Efficient protocols for particular sharing schemes

4.1 The crumbling wall protocol

The Crumbling Walls (CW) are a family of quorum systems due to [PW97]. This family includes, among others, the CWlog system (see Figure 1). The players in a wall are logically arranged in *rows* of varying *widths*. A quorum in a wall is the union of one full row and a representative from every row below the full row. The best crumbling walls are those in which the top row has width $n_1 = 1$ and every other row has width $n_i \geq 2$. In [PW97] it is shown that such walls are non-dominated coteries, *i.e.*, their quorum access specification is unambiguous (recall Definition 6). Note that many of the quorums in a crumbling wall are small minorities — in the wall depicted in Figure 1 the smallest quorums are of size $O(\log n)$.

The CW-SSS secret sharing scheme Our multi-party protocol is based on the following secret sharing scheme, called CW-SSS, due to [NW96]. Consider a wall CW of d rows, with row 1 having width $n_1 = 1$ and $n_i \geq 2$ for all $i \geq 2$. The basic secret unit s is a single bit, therefore all the arithmetic in this section is over $GF(2)$. This secret s is first randomly split into d bits such that $a_1 + \cdots + a_d = s$. Using these a_i bits we can define their partial parities, $t_i \leftarrow a_1 + \cdots + a_{i-1}$, and

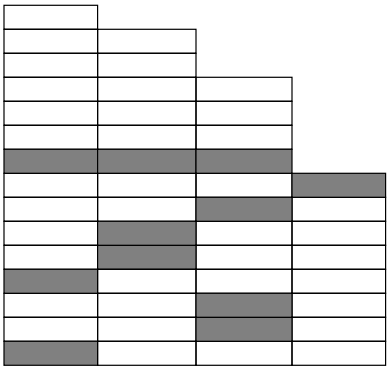

Fig. 1. A CWlog with $n = 49$ elements and $d = 15$ rows, with one quorum shaded. In this system row i has width $\lfloor \log 2i \rfloor$.

$t_1 = 0$. For a row i, split t_i randomly into n_i bits A_i^j such that $A_i^1 + \cdots + A_i^{n_i} = t_i$. The share s_i^j of the j'th element in row i contains two bits: a_i and A_i^j.

A quorum Q, which contains a full row i and a representative in each row $k > i$ can reconstruct the secret bit s from the shares $\{s_i^j\}$ generated by $s = \left(\sum_{j=1}^{n_i} A_i^j\right) + \left(\sum_{k \geq i} a_k\right)$.

The CW-MP protocol Let secret x be shared using pieces (a_i, A_i^j) and let secret y be shared using (b_i, B_i^j). Linear combinations ($z = \alpha x + \beta y$) are trivial: each player (i, j) performs the operation locally on the corresponding pieces: $(\alpha a_i + \beta b_i, \alpha A_i^j + \beta B_i^j)$.

Multiplication is somewhat more complex, but it can be achieved through appropriate application of sum-sharing. We use (i, j) to denote the j'th player in row i.

1. Local multiplication: Player (i, j) locally computes $e_i^j \leftarrow a_i B_i^j + b_i A_i^j$.
2. Randomization: Player (i, j) in row $i \geq 2$ flips a random value r_i^j. He then
   2a. sends r_i^j to the top player $(1, 1)$, and
   2b. sends $e_i^j + r_i^j$ to all other players (i, k) in his row.
3. Partial reconstruction:
   3a. The top player $(1, 1)$ sets $c_1 \leftarrow a_1 b_1 - \sum_{i \geq 2} \sum_{j=1}^{n_i} r_i^j$.
   3b. Player (i, j) in row $i \geq 2$ sets $c_i \leftarrow a_i b_i + \sum_{j=1}^{n_i} (e_i^j + r_i^j)$.
   It is not hard to verify that all the members of each row i compute the same value c_i, and that $\sum_{i=1}^{d} c_i = xy$. We now want to share the progressive parities of the c_i's in the successive rows.
4. Sharing across rows below: In each row $i = 1, \ldots, d-1$ player $(i, 1)$ sum-shares c_i into $c_i = \sum_{j=1}^{n_k} t(i, k, j)$ for every row $k = i + 1, \ldots, d$ and sends the share $t(i, k, j)$ to player (k, j).

5. Computing parity of rows above: Player (i,j) in row $i \geq 2$ sets $C_i^j \leftarrow \sum_{k=1}^{i-1} t(i,k,j)$. Thereby each row i has secretly computed (without reconstruction) the sum-share of the partial parity $c_1 + \cdots + c_{i-1}$.

The communication complexity of this protocol is clearly $O(n^2)$ bits and $O(1)$ rounds of message exchange.

Proposition17. *The preceding protocols securely compute shares of $z = \alpha x + \beta y$ (resp., $z = x \cdot y$) $\mathcal{Q}$-privately according to the CW-SSS scheme.*

4.2 The finite projective plane protocol

In this section we describe an efficient $O(n^2)$ protocol for the case where the underlying quorum system is a finite projective plane (FPP) [Mae85]. For a prime r let $t = r^k$ for some integer k. Then the finite projective plane of order t is a quorum system with $n = t^2 + t + 1$ players and $m = t^2 + t + 1$ quorums of size $t + 1$. Moreover, each player is a member of $t + 1$ quorums, and the intersection of every two quorums consists of a single player.

Note that since $n = m$ and the average quorum size $q \approx \sqrt{n}$ the general protocol of section 3 already has a polynomial message complexity of $O(n^{2.5})$ for the FPP system. However, here we can use a much more efficient secret sharing scheme: The size of the share held by each player is 1 or 2, in comparison to the $O(m)$-sized shares used in Gen-SSS. Thus the local computations in the FPP-MP protocol take constant time.

The FPP-SSS secret sharing scheme In this section all the arithmetic is performed in the ring $\mathbf{Z}_t$, where t is the order of the finite projective plane. Let x be the secret to be shared. As in Gen-SSS, x is represented randomly as a sum-share $x = \sum_{j=1}^{m} x_j \pmod t$. Then the share of player u is the sum of the parts corresponding to the quorums that contain u:

$$a_u(x) \leftarrow \sum_{Q_j \ni u} x_j \pmod t.$$

Proposition18. *FPP-SSS is a secret sharing scheme realizing the quorum access specification $(\mathcal{G}(FPP), \mathcal{B}(FPP))$.*

The FPP-MP protocol Assume that x and y are shared among the players according to the FPP-SSS scheme. In order to compute $z = \alpha x + \beta y$ $\mathcal{Q}$-privately the players locally compute $a_u(z) \leftarrow \alpha a_u(x) + \beta a_u(y)$. It is clear that the resultant shares are valid and no information was revealed by this computation.

To compute the product xy, each player u performs the following steps:

1. Player u locally computes the product $w_u \leftarrow a_u(x) \cdot a_u(y)$.
2. Player u secret-shares the value w_u among all the players using the FPP-SSS scheme. Namely, u computes a randomized sum-share version of w_u by $w_u = \sum_{j=1}^{m} w_{uj}$. Then for every $v \in U$ u computes $\alpha_{uv} = \sum_{Q_j \ni v} w_{uj}$ and sends α_{uv} to v.

3. After player u receives all the values $\{\alpha_{vu}\}_{v \in U}$, she computes her share $a_u(z)$ as $a_u(z) \leftarrow \sum_{v \in U} \alpha_{vu}$.

Lemma 19. *Protocol FPP-MP computes correct FPP-SSS shares of $z = x \cdot y$.*

Proposition 20. *The preceding protocols securely compute shares of $z = \alpha x + \beta y$ (resp., $z = x \cdot y$) $\mathcal{Q}$-privately according to the FPP-SSS scheme.*

Remark: In the above scheme the arithmetic modulus t is tied to the system size n since $n = t^2 + t + 1$. To disconnect this tie, we devised a variant of the FPP-SSS scheme in which the modulus can be arbitrary (e.g., we can use $GF(2)$). The price we pay for this extra freedom is that the share held by every player will include an additional correcting term. We omit the details in this abstract.

4.3 Hierarchical quorum protocols

In the hierarchical quorum system (HQS), due to [Kum91], the individuals are the leaves of a complete ternary tree in which internal nodes are 2-of-3 majority gates. The related Tree quorum system [AE91] employs full ternary trees, but the center child of each tree is a leaf. The HQS and Tree systems enjoy high availability and low load.

The HQS-SSS secret sharing scheme A secret sharing scheme for HQS appears in [NW96]; here we give only a brief description of the construction, which is essentially recursive Shamir sharing.

If p is a node in a ternary tree, let $p.1$, $p.2$ and $p.3$ be its children (if any). This gives a natural labeling for nodes, *e.g.* "root.1.3.1.2." Let $\texttt{split}(x)$ be a distribution over $\mathrm{GF}(4)^3$ obtained by selecting a random a and computing $(f(01), f(10), f(11))$ where $f(u) = au + x$.

Sharing value v for secret x is straightforward recursion from the root: $\texttt{Share}(x, v, p)$ sets $x[p] \leftarrow v$ if p is a leaf; otherwise set $(v_1, v_2, v_3) \leftarrow \texttt{split}(v)$, then recurse on $\texttt{Share}(x, v_i, p.i)$ $(i = 1, 2, 3)$.

The HQ-MP protocol Let secret x be shared using pieces $x[p]$ and let secret y be shared using $y[p]$. To perform a linear combination $(z = \alpha x + \beta y)$, players simply perform the operation locally: $z[p] \leftarrow \alpha x[p] + \beta y[p]$.

Although intuitively obvious, a recursive construction for multiplication using [BGW88, CCD88] requires some care. In particular, the internal nodes are "virtual," namely represented by sets of players; thus otherwise obvious steps such as "p sends m to q" are ill-defined when p and q are not leaf nodes.

The steps needed to apply [BGW88, CCD88] include linear combination and multiplication of local values, and sending messages. (Sharing, reconstruction, selecting random values, and degree reduction can be built with these primitives.) It remains to be seen, therefore, how messages are sent from p to q.

There are four cases. If p and q are leaves, m is sent directly. If p is a leaf while q is internal, p runs $\texttt{Share}(m_{pq}, m, q)$; m_{pq} is the label for the secret message, held by players under q. If p is internal while q is a leaf, players under p send their

shares of m to q for reconstruction. If p and q are internal, then $p.i$ recursively "sends" its share of m to $q.i$ (for $i = 1, 2, 3$).

The communication cost of the overall construction is $O(n^2)$ per gate. Because [BGW88, CCD88] can be executed with one multiplication per virtual player, the primary concern is to measure the cost of "sending" messages. Straightforward arguments show that the worst case trees are those that are least balanced (*e.g.*, all the subtrees lie under left children). In this extreme, the cost of sending a message at height H from the bottom is $\Omega(Hn)$. Multiplication requires $O(1)$ simulated messages for any particular troika; thus the net cost of multiplication is $O(n^2)$.

Remark: This protocol can be seen as a particular instance of the [HM97] construction, if certain unconstrained choices are cleverly made; thus, a security argument can be derived from [HM97, BGW88, CCD88]. Note, however, that the constructive argument in [HM97] permits several degrees of freedom, most of which lead to an *exponential* algorithm for these quorum systems.

Acknowledgments

We are very grateful to Daniel Bleichenbacher for many highly useful discussions, and for introducing the result of Hirt and Maurer to us. We also thank Martin Hirt and Ueli Maurer for discussions and comparisons of their work.

References

[AE91] D. Agrawal and A. El-Abbadi. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Trans. Comp. Sys.*, 9(1):1–20, 1991.

[Bea91a] D. Beaver. Foundations of secure interactive computing. In *Advances in Cryptology – CRYPTO'91, LNCS 576*, pages 377–391. Springer-Verlag, 1991.

[Bea91b] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *J. Cryptology*, 4(2):75–122, 1991.

[BG89] D. Beaver and S. Goldwasser. Multiparty computation with faulty majority. In *Proc. 30th IEEE Symp. Foundations of Comp. Sci. (FOCS)*, pages 468–473, 1989.

[BH92] D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In *Advances in Cryptology – EUROCRYPT'92, LNCS 658*, pages 307–323. Springer-Verlag, 1992.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th ACM Symp. Theory of Computing (STOC)*, pages 1–10, Chicago, Illinois, 2–4 May 1988.

[BKK94] S. Ben-David, M. Karchmer, and E. Kushilevitz. On ultrafilters and NP. In *Proceedings of the 9th Annual Conference on Structure in Complexity Theory*, pages 97–105. IEEE Computer Society Press, 1994.

[BL88] J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *Advances in Cryptology – CRYPTO'88, LNCS 403*, pages 27–36. Springer-Verlag, 1988.

[Bla79] G. R. Blakely. Safeguarding cryptographic keys. *Proc. AFIPS, NCC*, 48:313–317, 1979.

[CCD88] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In *Proc. 20th ACM Symp. Theory of Computing (STOC)*, pages 11–19, Chicago, Illinois, 2–4 May 1988.

[CFGN96] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure multiparty computation. In *Proc. 28th ACM Symp. Theory of Computing (STOC)*, pages 639–648, 1996.

[CK91] B. Chor and E. Kushilevitz. A zero-one law for Boolean privacy. *SIAM J. Discrete Math.*, 4:36–47, 1991.

[DDFY94] A. De Santis, Y. Desmet, Y. Frankel, and M. Yung. How to share a function securely. In *Proc. 26th ACM Symp. Theory of Computing (STOC)*, pages 522–533, 1994.

[DGS85] S. B. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in partitioned networks. *ACM Computing Surveys*, 17(3):341–370, 1985.

[Für88] Z. Füredi. Matchings and covers in hypergraphs. *Graphs and Combinatorics*, 4:115–206, 1988.

[GB85] H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *J. ACM*, 32(4):841–860, 1985.

[GHY88] Z. Galil, S. Haber, and M. Yung. Cryptographic computation: Secure fault-tolerant protocols and the public-key model. In *Advances in Cryptology – CRYPTO'87, LNCS 293*, pages 135–155. Springer-Verlag, 1988.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Computing*, 18(1):186–208, 1989.

[GMW86] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *Proc. 27th IEEE Symp. Foundations of Comp. Sci. (FOCS)*, pages 174–187. IEEE, 1986.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Proc. 19th ACM Symp. Theory of Computing (STOC)*, pages 218–229, 1987.

[Her84] M. P. Herlihy. *Replication Methods for Abstract Data Types*. PhD thesis, Massachusetts Institute of Technology, MIT/LCS/TR-319, 1984.

[HM97] M. Hirt and U. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation. In *Proc. 16th ACM Symp. Princip. of Dist. Comp. (PODC)*, pages 25–34, August 1997.

[IK93] T. Ibaraki and T. Kameda. A theory of coteries: Mutual exclusion in distributed systems. *IEEE Trans. Par. Dist. Sys.*, 4(7):779–794, 1993.

[ISN87] M. Ito, A. Saito, and T. Nishizeki. Secret sharing schemes realizing general access structure. In *Proc. IEEE Global Telecommunication Conf. (Globecomm 87)*, pages 99–102, 1987.

[Kum91] A. Kumar. Hierarchical quorum consensus: A new algorithm for managing replicated data. *IEEE Trans. Comput.*, 40(9):996–1004, 1991.

[Mae85] M. Maekawa. A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comp. Sys.*, 3(2):145–159, 1985.

[MR91] S. Micali and P. Rogaway. Secure computation. In *Advances in Cryptology – CRYPTO'91, LNCS 576*, pages 392–404. Springer-Verlag, 1991.

[MV88] S. J. Mullender and P. M. B. Vitányi. Distributed match-making. *Algorithmica*, 3:367–391, 1988.

[NW94] M. Naor and A. Wool. The load, capacity and availability of quorum systems. In *Proc. 35th IEEE Symp. Foundations of Comp. Sci. (FOCS)*, pages 214–225, 1994. To appear in SIAM J. Computing 1998.

[NW96] M. Naor and A. Wool. Access control and signatures via quorum secret sharing. In *Proc. 3rd ACM Conf. Comp. and Comm. Security*, pages 157–168, New Delhi, India, 1996. Also available as Theory of Cryptography Library record 96-08, `http://theory.lcs.mit.edu/~tcryptol/1996.html`.

[Owe82] G. Owen. *Game Theory*. Academic Press, second edition, 1982.

[PW97] D. Peleg and A. Wool. Crumbling walls: A class of practical and efficient quorum systems. *Distributed Computing*, 10(2):87–98, 1997.

[RB89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. 21st ACM Symp. Theory of Computing (STOC)*, pages 73–85, 1989.

[Sha79] A. Shamir. How to share a secret. *Comm. ACM*, 22(11):612–613, 1979.

Appendix

Proof of Proposition 15: The only messages sent by the protocol are the shares w_{uj} sent in step 2. Each player u sends the part w_{uj} to every player $v \in Q_j$, giving: $n \sum_{j=1}^{m} |Q_j| = nmq$. ∎

Proof of Proposition 18: The reconstruction function for a quorum Q_j is the sum of all the shares held be the members of Q_j. From the definition of the shares, we know: $\sum_{u \in Q_j} a_u(x) = \sum_{u \in Q_j} \sum_{Q_i \ni u} x_i$. Since every two quorums intersect in a single player, the part x_i appears once in the double summation for every $i \neq j$. The part x_j itself appears $t+1$ times, but since the arithmetic is mod t, the last sum equals x. Hence every quorum Q_j can reconstruct x.

Now consider a set $B \in \mathcal{B}(\mathrm{FPP})$. Then by definition there exists some quorum $Q_i \in \mathrm{FPP}$ such that $Q_i \subseteq U \setminus B$. The part x_i appears in the sum $a_u(x)$ only for $u \in Q_i$. Therefore x_i is independent of the shares held by members of B. From this it is easy to see that the union of the shares of all players in B gives no information about x. ∎

Proof of Lemma 19: First we show that the w_u's sum up to z.

$$\sum_{u \in U} a_u(x) a_u(y) = \sum_{u \in U} \left(\sum_{Q_i \ni u} x_i \right) \left(\sum_{Q_j \ni u} y_j \right) = \sum_{u \in U} \sum_{\substack{Q_i \ni u \\ Q_j \ni u}} x_i y_j .$$

The term $x_i y_j$ appears in the sum once for every $u \in Q_i \cap Q_j$. Therefore $x_i y_j$ appears once when $i \neq j$ and $t+1$ times when $i = j$. Since we are working mod t we conclude that $\sum_{u \in U} a_u(x) a_u(y) = \sum_{i,j=1}^{m} x_i y_j = xy$. Now consider some quorum Q_i. Plugging $a_v(z)$ into the reconstruction function for Q_i we get

$$\sum_{v \in Q_i} a_v(z) = \sum_{v \in Q_i} \sum_{u \in U} \alpha_{uv} = \sum_{v \in Q_i} \sum_{u \in U} \sum_{Q_j \ni v} w_{uj} = \sum_{u \in U} \sum_{v \in Q_i} \sum_{Q_j \ni v} w_{uj}.$$

Now we use a similar trick (this time holding Q_i fixed). For any u, the part w_{uj} appears in the sum once for every $v \in Q_i \cap Q_j$. Since we are working mod t,

$$\sum_{u \in U} \sum_{j=1}^{m} w_{uj} = \sum_{u \in U} w_u = \sum_{u \in U} a_u(x) a_u(y) = xy. \qquad ∎$$

This article was processed using the LaTeX macro package with LLNCS style