Scalable and Unconditionally Secure Multiparty Computation

Ivan Damgård and Jesper Buus Nielsen*

Dept. of Computer Science, BRICS, Aarhus University

Abstract. We present a multiparty computation protocol that is unconditionally secure against adaptive and active adversaries, with communication complexity $\mathcal{O}(\mathcal{C}n)k + \mathcal{O}(Dn^2)k + \text{poly}(n\kappa)$, where $\mathcal{C}$ is the number of gates in the circuit, n is the number of parties, k is the bit-length of the elements of the field over which the computation is carried out, D is the multiplicative depth of the circuit, and κ is the security parameter. The corruption threshold is $t < n/3$. For passive security the corruption threshold is $t < n/2$ and the communication complexity is $\mathcal{O}(n\mathcal{C})k$. These are the first unconditionally secure protocols where the part of the communication complexity that depends on the circuit size is linear in n. We also present a protocol with threshold $t < n/2$ and complexity $\mathcal{O}(\mathcal{C}n)k + \text{poly}(n\kappa)$ based on a complexity assumption which, however, only has to hold *during* the execution of the protocol – that is, the protocol has so called everlasting security.

1 Introduction

In secure multiparty computation a set of n parties, $\mathcal{P} = \{P_1, \ldots, P_n\}$, want to compute a function of some secret inputs held locally by some of the parties. The desired functionality is typically specified by a function $f : (\{0,1\}^*)^n \rightarrow (\{0,1\}^*)^n$. Party P_i has input $x_i \in \{0,1\}^*$ and output $y_i \in \{0,1\}^*$, where $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$. During the protocol a subset $C \subset \mathcal{P}$ of the parties can be *corrupted*. Security means that all parties receive correct outputs and that the messages seen by the corrupted parties $P_i \in C$ during the protocol contain no information about the inputs and outputs of the honest parties $(\mathcal{P} \setminus C)$, other than what can be computed efficiently from the inputs and outputs of the corrupted parties. *Passive security* means that the above condition holds when all parties follow the protocol. *Active security* means that the above condition holds even when the corrupted parties in C might deviate from the protocol in an arbitrary coordinated manner. When a protocol is secure against all subsets of size at most t we say that it is secure against an adversary with *corruption threshold* t.

In the *cryptographic model* each pair of parties is assumed to share an authenticated channel and the corrupted parties are assumed to be poly-time bounded, to allow the use of cryptography. In the *information theoretic model* it is assumed

* Funded by the Danish Agency for Science, Technology and Innovation.

A. Menezes (Ed.): CRYPTO 2007, LNCS 4622, pp. 572–590, 2007.

that each pair of parties share a perfectly secure channel and that the parties have an authenticated broadcast channel, and it is assumed that the corrupted parties are computationally unbounded. We talk about *cryptographic security* versus *unconditional security*.

The MPC problem dates back to Yao [Yao82]. The first generic solutions with cryptographic security were presented in [GMW87, CDG87], and later generic solutions with unconditional security were presented in [BGW88, CCD88, RB89, Bea89]. In both cases, security against an active adversary with threshold $t < n/2$ is obtained, and $t < n/2$ is known to be optimal.

Thresh.	**Adv.**	**Communication**	**Reference**
$t < n/2$	passive	$\mathcal{O}(\mathcal{C}n^2)k$	[BGW88]
$t < n/2$	active	$\mathcal{O}(\mathcal{C}n^5)k + \text{poly}(n\kappa)$	[CDD+99]
$t < n/3$	active	$\mathcal{O}(\mathcal{C}n^2)k + \text{poly}(n\kappa)$	[HM01]
$t < n/2$	active	$\mathcal{O}(\mathcal{C}n^2)k + \text{poly}(n\kappa)$	[BH06]
$t < n/2$	passive	$\mathcal{O}(\mathcal{C}n)k$	this paper
$t < n/3$	active	$\mathcal{O}(\mathcal{C}n)k + \mathcal{O}(Dn^2)k + \text{poly}(n\kappa)$	this paper
$t < n/2$	active, limited during protocol	$\mathcal{O}(\mathcal{C}n)k + \text{poly}(n\kappa)$	this paper

Fig. 1. Comparison of some unconditionally secure MPC protocols

The communication complexity of a MPC protocol is taken to be the total number of bits sent and received by the honest parties in the protocol. Over the years a lot of research have been focused on bringing down the communication complexity of active secure MPC [GV87, BB89, BMR90, BFKR90, Bea91, GRR98] [CDM00, CDD00, HMP00, HM01, HN05, DI06, HN06]. Building on a lot of previous work, it was recently shown in [DI06, HN06]that there exist cryptographic secure MPC protocols with communication complexity $\mathcal{O}(\mathcal{C}n)k+\text{poly}(n\kappa)$, where $\mathcal{C}$ is the size of a Boolean circuit computing f, k is the bit length of elements from the field (or ring) over which the computation takes place (at least $k = \log_2(n)$) and $\text{poly}(n\kappa)$ is some complexity independent of the circuit size, typically covering the cost of some fixed number of broadcasts or a setup phase. All earlier protocols had a first term of at least $\mathcal{O}(\mathcal{C}n^2)k$, meaning that the communication complexity depending on $\mathcal{C}$ was quadratic in the number of parties. Having linear communication complexity is interesting as it means that the work done by a single party is independent of the number of parties participating in the computation, giving a fully scalable protocol. Until the work presented in this paper it was, however, not known whether there existed an *unconditionally secure* protocol with linear communication complexity. In fact, this was an open problem even for passive secure protocols. We show that indeed it is possible to construct an unconditional secure protocol where the part of the communication complexity which depends on the circuit size is linear in n. For active security, however, we get a quadratic dependency on the multiplicative depth of the circuit, denoted by D. Our results are compared to some previous unconditionally secure protocol in Fig. 1.

Note that our active secure protocol is more efficient than even the previous most efficient passive secure protocol, as clearly $D \leq \mathcal{C}$. Note also that our active secure protocol obtains threshold $t < n/3$. When no broadcast channel is given, this is optimal. If a broadcast channel is given, however, it is possible to get active security for $t < n/2$ [RB89]. By using an unconditionally hiding commitment scheme to commit parties to their shares, it is possible to obtain a protocol with resilience $t < n/2$ and with communication complexity $\mathcal{O}(\mathcal{C}n)\kappa + \text{poly}(n\kappa)$. This protocol is not unconditional secure, but at least has everlasting security in the sense that if the computational assumptions are not broken *during the run of the protocol*, then the protocol is secure against a later computationally unbounded adversary.

In comparison to the earlier work achieving linear complexity, we emphasize that achieving unconditional security is not only of theoretical interest: using "information theoretic" techniques for multiparty computation is *computationally* much more efficient than using homomorphic public-key encryption as in [HN06], for instance. We can therefore transplant our protocols to the computational setting, using cryptography only as a transport mechanism to implement the channels, and obtain a protocol that is computationally more efficient than the one from [HN06], but incomparable to what can be obtained from [DI06]: we get a better security threshold but a non-constant number of rounds.

On the technical side, we make use of a technique presented in [HN06] that was used there to get computational security. It is based on the fact that the standard method for secure multiplication based on homomorphic encryption uses only public, broadcasted messages. The idea is now to select a "king", to whom everyone sends what they would otherwise broadcast. The king does whatever computation is required and returns results to the parties. This can save communication, provided we can verify the king's work in a way that is cheap when amortized over many instances.

This idea is not immediately applicable to unconditionally secure protocols, because the standard method for multiplication in this scenario involves private communication where players send different messages to different players. This can of course not all be sent to the king without violating privacy. We therefore introduce a new multiplication protocol, involving only public communication that is compatible with the "king-paradigm" (the main idea is used in the Triples protocol in Fig. 4). This, together with adaptation of known techniques, is sufficient for passive security.

For active security, we further have to solve the problem that each player frequently has to make a secret shared value public, based on shares he has received in private. Here, we need to handle errors introduced by corrupt players. Standard VSS-based techniques would be quadratic in n, but we show how to use error correction based on Van der Monde matrices to correct the errors, while keeping the overall complexity linear in n (this idea is used in the OpenRobust protocol in Fig. 9).

2 Preliminaries

Model. We use $\mathcal{P} = \{P_1, \ldots, P_n\}$ to denote a set of n parties which are to do the secure evaluation, and we assume that each pair of parties share a perfectly secure channel. Furthermore, we assume that the parties have access to an authenticated broadcast channel. We allow some subset of corrupted parties $C \subset \mathcal{P}$, of size at most t, to behave in some arbitrary coordinated manner. We call $H = \mathcal{P} \setminus C$ the honest parties. We consider secure circuit evaluation. All input gates are labeled by a party from $\mathcal{P}$. That party provides a secret input for the gate, and the goal of the secure circuit evaluation is to make the output of the circuit public to each party in $\mathcal{P}$. The protocol takes an extra input $\kappa \in \mathbb{N}$, the security parameter, which is given to all parties. The protocol should run in time $\text{poly}(\kappa)$ and the "insecurity" of the protocol should be bounded by $\text{poly}(\kappa)2^{-\kappa}$.

The Ground Field and the Extension Field. For the rest of the paper we fix a finite field $\mathbb{F}$ over which most of our computations will be done. We call $\mathbb{F}$ the ground field. We let $k = \log_2(|\mathbb{F}|)$ denote the bit-length of elements from $\mathbb{F}$. We also fix an extension field $\mathbb{G} \supset \mathbb{F}$ to be the smallest extension for which $|\mathbb{G}| \geq 2^{\kappa}$. Since $|\mathbb{G}| < 2^{2\kappa}$, a field element from $\mathbb{G}$ can be written down using $\mathcal{O}(\kappa)$ bits. We call $\mathbb{G}$ the extension field.

Van der Monde Matrices. We write a matrix $M \in \mathbb{F}^{(r,c)}$ with r rows and c columns as $M = \{m_{i,j}\}_{i=1,\ldots,r}^{j=1,\ldots,c}$. For $C \subseteq \{1, \ldots, c\}$ we let $M^C = \{m_{i,j}\}_{i=1,\ldots,r}^{j \in C}$ denote the matrix consisting of the columns from M indexed by $j \in C$. We use $M^{\top}$ to denote the transpose of a matrix, and for $R \subseteq \{1, \ldots, r\}$ we let $M_R = ((M^{\top})^R)^{\top}$. For distinct elements $\alpha_1, \ldots, \alpha_r \in \mathbb{F}$ we use $\text{Van}^{(r,c)}(\alpha_1, \ldots, \alpha_r) \in \mathbb{F}^{(r,c)}$ to denote the Van der Monde matrix $\{\alpha_i^j\}_{i=1,\ldots,r}^{j=0,\ldots,c-1}$, and we use $\text{Van}^{(r,c)} \in \mathbb{F}^{(r,c)}$ to denote *some* Van der Monde matrix of the form $\text{Van}^{(r,c)}(\alpha_1, \ldots, \alpha_r)$ when the elements $\alpha_1, \ldots, \alpha_r$ are inconsequential. It is a well-known fact that all $\text{Van}^{(c,c)}$ are invertible. Consider then $V = \text{Van}^{(r,c)}$ with $r > c$, and let $R \subset \{1, \ldots, r\}$ with $|R| = c$. Since V_R is a Van der Monde matrix $\text{Van}^{(c,c)}$ it follows that V_R is invertible. So, any c rows of a Van der Monde matrix form an invertible matrix. In the following we say that Van der Monde matrices are super-invertible.

Secret Sharing. For the rest of the paper a subset $I \subseteq \mathbb{F}^*$ of $|\mathcal{P}|$ non-zero elements is chosen. Each party in $\mathcal{P}$ is then assigned a unique element i from I, and we index the parties in $\mathcal{P}$ by P_i for $i \in I$. For notational convenience we will assume that $I = \{1, \ldots, |\mathcal{P}|\}$, which is possible when $\mathbb{F}$ has characteristic at least n. All our techniques apply also to the case where $\mathbb{F}$ has smaller characteristic, as long as $|\mathbb{F}| > n$.

By a d-polynomial we mean a polynomial $f(X) \in \mathbb{F}[X]$ of degree *at most* d. To share a value $x \in \mathbb{F}$ with degree d, a uniformly random d-polynomial $f(X) \in \mathbb{F}[X]$ with $f(0) = x$ is chosen, and P_i is given the share $x_i = f(i)$. This is the same as letting $x_0 = x$, choosing $x_1, \ldots, x_d \in \mathbb{F}$ uniformly at random and letting $(y_1, \ldots, y_n) = M^{(d)}(x_0, x_1, \ldots, x_d)$, where $M^{(d)} = \text{Van}^{(n,d+1)}(1, \ldots, n)$.

In the following we call a vector $\boldsymbol{y} = (y_1, \ldots, y_n) \in \mathbb{F}^n$ a d-sharing (of x) if there exists $(x_1, \ldots, x_d) \in \mathbb{F}^d$ such that $M^{(d)}(x, x_1, \ldots, x_d)$ and $\boldsymbol{y}$ agree on the

share of all honest parties; I.e., $\boldsymbol{y}_H = M_H^{(d)}(x, x_1, \ldots, x_d)$. In the following we typically use $[x]$ to denote a d-sharing of x with $d = t$, and we always use $\langle x \rangle$ to denote a d-sharing of x with $d = 2t$.

Whenever we talk about a sharing $[x]$ we implicitly assume that party P_i is holding x_i such that $[x] = (x_1, \ldots, x_n)$. We introduce a shorthand for specifying computations on these local shares. If sharings $[x^{(1)}], \ldots, [x^{(\ell)}]$ have been dealt, then each P_i is holding a share $x_i^{(l)}$ of each $[x^{(l)}]$. Consider any function $f : \mathbb{F}^\ell \to \mathbb{F}^m$. By $([y^{(1)}], \ldots, [y^{(m)}]) = f([x^{(1)}], \ldots, [x^{(\ell)}])$, we mean that each P_i computes $(y_i^{(1)}, \ldots, y_i^{(m)}) = f(x_i^{(1)}, \ldots, x_i^{(\ell)})$, defining sharings $[y^{(k)}] = (y_1^{(k)}, \ldots, y_n^{(k)})$ for $k = 1, \ldots, m$. It is well-known that if f is an affine function and each $[x^{(l)}]$ is a consistent d-sharing, then the $[y^{(k)}]$ are consistent d-sharings of $(y^{(1)}, \ldots, y^{(m)}) = f(x^{(1)}, \ldots, x^{(\ell)})$. Furthermore, if $[x_1]$ and $[x_2]$ are consistent d-sharings, then $[y] = [x_1][x_2]$ is a consistent $2d$-sharing of $y = x_1 x_2$. When $d = t$ we therefore use the notation $\langle y \rangle = [x_1][x_2]$.

Error Correction. It is well-known that Van der Monde matrices can be used for error correction. Let $M = \mathrm{Van}^{(r,c)}$ be any Van der Monde matrix with $r > c$. Let $\boldsymbol{x} \in \mathbb{F}^c$ and let $\boldsymbol{y} = M\boldsymbol{x}$. Consider any $R \subset \{1, \ldots, r\}$ with $|R| = c$. By Van der Monde matrices being super-invertible it follows that M_R is invertible. Since $\boldsymbol{y}_R = M_R \boldsymbol{x}$ it follows that $\boldsymbol{x} = M_R^{-1} \boldsymbol{y}_R$, so that $\boldsymbol{x}$ can be computed from any r entries of $\boldsymbol{y}$. It follows that if $\boldsymbol{x}^{(1)} \neq \boldsymbol{x}^{(2)}$ and $\boldsymbol{y}^{(1)} = M\boldsymbol{x}^{(1)}$ and $\boldsymbol{y}^{(2)} = M\boldsymbol{x}^{(2)}$, then $\mathrm{ham}(\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}) \geq r - c + 1$, where ham denotes the Hamming distance. Assume now that $r \geq c + 2t$ for some positive integer t, such that $\mathrm{ham}(\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}) \geq 2t + 1$. This allows to correct up to t errors, as described now. Let $\boldsymbol{y} = M\boldsymbol{x}$ and let $\boldsymbol{y}'$ be any vector with $\mathrm{ham}(\boldsymbol{y}, \boldsymbol{y}') \leq t$. It follows that $\mathrm{ham}(\boldsymbol{y}', M\boldsymbol{x}') \geq t + 1$ for all $\boldsymbol{x}' \neq \boldsymbol{x}$. Therefore $\boldsymbol{y}$ can be computed uniquely from $\boldsymbol{y}'$ as the vector $\boldsymbol{y}$ with $\mathrm{ham}(\boldsymbol{y}, \boldsymbol{y}') \leq t$. Then $\boldsymbol{x}$ can be compute from $\boldsymbol{y}$ as $\boldsymbol{x} = M_R^{-1} \boldsymbol{y}_R$ with e.g. $R = \{1, \ldots, c\}$. The Berlekamp-Welch algorithm allows to compute $\boldsymbol{y}$ from $\boldsymbol{y}'$ at a price in the order of performing Gaussian elimination on a matrix from $\mathbb{F}^{(r,r)}$.

Randomness Extraction. We also use Van der Monde matrices for randomness extraction. Let $M = \mathrm{Van}^{(r,c)^\top}$ be the transpose of any Van der Monde matrix with $r > c$. We use the computation $(y_1, \ldots, y_c) = M(x_1, \ldots, x_r)$ to extract randomness from $(x_1, \ldots, x_r)$. Assume that $(x_1, \ldots, x_r)$ is generated as follows: First $R \subset \{1, \ldots, r\}$ with $|R| = c$ is picked and a uniformly random $x_i \in_R \mathbb{F}$ is generated for $i \in R$. Then for $j \in T$, with $T = \{1, \ldots, r\} \setminus R$, the values x_j are generated with an arbitrary distribution independent of $\{x_i\}_{i \in R}$. For any such distribution of $(x_1, \ldots, x_r)$ the vector $(y_1, \ldots, y_c)$ is uniformly random in $\mathbb{F}^c$. To see this, note that $(y_1, \ldots, y_c) = M(x_1, \ldots, x_r)$ can be written as $(y_1, \ldots, y_c) = M^R(x_1, \ldots, x_r)_R + M^T(x_1, \ldots, x_r)_T$. Since $M^R = \mathrm{Van}_R^{(r,c)^\top}$ we have that M^R is invertible. By definition of the input distribution, the vector $(x_1, \ldots, x_r)_R$ is uniformly random in $\mathbb{F}^c$. Therefore $M^R(x_1, \ldots, x_r)_R$ is uniformly random in $\mathbb{F}^c$. Since $(x_1, \ldots, x_r)_T$ was sampled independent of $(x_1, \ldots, x_r)_R$, it follows that $M^R(x_1, \ldots, x_r)_R + M^T(x_1, \ldots, x_r)_T$ is uniformly random in $\mathbb{F}^c$, as desired.

3 Private, $t < n/2$

We first present a passive secure circuit-evaluation protocol. Later we show how to add robustness. Throughout this section we assume that there are at most $t = \lfloor (n-1)/2 \rfloor$ corrupted parties. To prepare for adding robustness, some parts of the passive protocol are slightly more involved than necessary. The extra complications come from the fact that we want to place all dealings of sharings in a preprocessing phase, where the inputs have not been used yet. This will later allow a particularly efficient way of detecting cheating parties, and will furthermore give a circuit-evaluation phase which consists of only opening sharings, limiting the types of errors that can be encountered after the parties entered their inputs into the computation.

3.1 Random Double Sharings

We first present a protocol, DOUBLE-RANDOM(ℓ), which allows the parties to generate sharings $[r_1], \ldots, [r_\ell]$ and $\langle R_1 \rangle, \ldots, \langle R_\ell \rangle$, where each $[r_l]$ is a uniformly random t-sharing of a uniformly random value $r_l \in \mathbb{F}$ and each $\langle R_l \rangle$ is a uniformly random $2t$-sharing of $R_l = r_l$. We consider the case $\ell = n - t$. For larger ℓ, the protocol is simply run in parallel a number of times. As part of the protocol the parties use a fixed matrix $M = \mathrm{Van}^{(n,n-t)^\top}$ for randomness extraction.

1. Each $P_i \in \mathcal{P}$: Pick a uniformly random value $s^{(i)} \in_R \mathbb{F}$ and deal a t-sharing $[s^{(i)}]$ and a $2t$-sharing $\langle s^{(i)} \rangle$.
2. Compute
$$([r_1], \ldots, [r_{n-t}]) = M([s^{(1)}], \ldots, [s^{(n)}])$$
$$(\langle R_1 \rangle, \ldots, \langle R_{n-t} \rangle) = M(\langle s^{(1)} \rangle, \ldots, \langle s^{(n)} \rangle) \ ,$$
and output $(([r_1], \langle R_1 \rangle), \ldots, ([r_{n-t}], \langle R_{n-t} \rangle))$.

Fig. 2. Double-Random$(n - t)$

The protocol is given in Fig. 2. Assume that t parties are corrupted, leaving exactly $m = n - t$ honest parties. The m sharings $[s^{(i)}]$ dealt by the honest parties are independent uniformly random sharings of independent, uniformly random values unknown by the corrupted parties. The matrix M being a super-invertible matrix with m rows then implies that the sharings $([r_1], \ldots, [r_m])$ are independent uniformly random t-sharings of uniformly random values unknown by the corrupted parties. In the same way $(\langle R_1 \rangle, \ldots, \langle R_m \rangle)$ are seen to be independent, uniformly random $2t$-sharings of uniformly random elements unknown by the corrupted parties, and $R_l = r_l$. We also use a protocol RANDOM(ℓ) which runs as DOUBLE-RANDOM(ℓ) except that the $2t$-sharings $\langle R \rangle$ are not generated.

Each of the $2n$ dealings communicate $\mathcal{O}(n)$ field elements from $\mathbb{F}$, giving a total communication complexity of $\mathcal{O}(n^2 k)$. Since $n - t = \Theta(n)$ pairs are generated, the communication complexity per generated pair is $\mathcal{O}(nk)$. A general number ℓ of pairs can thus be generated with communication complexity $\mathcal{O}(n\ell k + n^2 k)$.

3.2 Opening Sharings

The next protocol, $\textsc{Open}(d, [x])$, is used for reconstructing a d-sharing efficiently. For this purpose a designated party $P_{\texttt{king}} \in \mathcal{P}$ will do the reconstruction and send the result to the rest of the parties.

1. Each $P_i \in \mathcal{P}$: Let $P_{\texttt{king}} \in \mathcal{P}$ be some agreed-upon party and send the share x_i of $[x]$ to $P_{\texttt{king}}$.
2. $P_{\texttt{king}}$: Compute a d-polynomial $f(X) \in \mathbb{F}[X]$ with $f(i) = x_i$ for all $P_i \in \mathcal{P}$, and send $x = f(0)$ to all parties.
3. Each $P_i \in \mathcal{P}$: Output x.

Fig. 3. Open$(d, [x])$

It is clear that if $[x]$ is a d-sharing of x and there are no active corruptions, then all honest parties output x. The communication complexity is $2(n-1) = \mathcal{O}(n)$ field elements from $\mathbb{F}$.

3.3 Multiplication Triples

We then present a protocol, $\textsc{Triples}(\ell)$ which allows the parties to generate ℓ multiplication triples, which are just triples $([a], [b], [c])$ of uniformly random t-sharings with $c = ab$.

1. All parties: Run $\textsc{Random}(2\ell)$ and $\textsc{Double-Random}(\ell)$ and group the outputs in ℓ triples $([a], [b], ([r], \langle R \rangle))$. For each triple in parallel, proceed a follows:
   (a) All parties: Compute $\langle D \rangle = [a][b] + \langle R \rangle$.
   (b) All parties: Run $D \leftarrow \textsc{Open}(2t, \langle D \rangle)$.
   (c) All parties: Compute $[c] = D - [r]$, and output $([a], [b], [c])$.

Fig. 4. Triples(ℓ)

The sharings $[a]$ and $[b]$ are t-sharings, so $[a][b]$ is a $2t$-sharing of ab. Therefore $\langle D \rangle = [a][b] + \langle R \rangle$ is a uniformly random $2t$-sharing of $D = ab + R$. The revealing of D leaks no information on a or b, as R is uniformly random. Therefore the protocol is private. Then $[c] = D - [r]$ is computed. Since $[r]$ is a t-sharing, $[c]$ will be a t-sharing, of $D - r = ab + R - r = ab$. The communication complexity of generating ℓ triples is seen to be $\mathcal{O}(n\ell k + n^2 k)$.

3.4 Circuit Evaluation

We are then ready to present the circuit-evaluation protocol. The circuit $\text{Circ} = \{G_{gid}\}$ consists of gates G_{gid} of the following forms.

input: $G_{gid} = (gid, \texttt{inp}, P_j)$, where $P_j \in \mathcal{P}$ provides a secret input $x_{gid} \in \mathbb{F}$.
random input: $G_{gid} = (gid, \texttt{ran})$, where $x_{gid} \in_R \mathbb{F}$ is chosen as a secret, uniformly random element.
affine: $G_{gid} = (gid, \texttt{aff}, a_0, gid_1, a_1, \ldots, gid_\ell, a_\ell)$, where $a_0, a_1, \ldots, a_\ell \in \mathbb{F}$ and $x_{gid} = a_0 + \sum_{l=1}^{\ell} a_l x_{gid_l}$.
multiplication: $G_{gid} = (gid, \texttt{mul}, gid_1, gid_2)$, where $x_{gid} = x_{gid_1} x_{gid_2}$.
output: $G_{gid} = (\texttt{out}, gid_1)$, where all parties are to learn x_{gid_1}.[1]

What it means to securely evaluate Circ can easily be phrased in the UC framework[Can01], and our implementation is UC secure. We will however not prove this with full simulation proofs in the following, as the security of our protocols follow using standard proof techniques.

Preprocessing Phase. First comes a preprocessing phase, where a number of sharings are generated for some of the gates in Circ. The details are given in Fig. 5. The communication complexity is $\mathcal{O}(n\ell k + n^2 k)$, where ℓ is the number of random gates plus the number of input gates plus the number of output gates.

All gates are handled in parallel by all parties running the following:

random: Let r be the number of random gates in Circ, run RANDOM(r) and associate one t-sharing $[x_{gid}]$ to each $(gid, \texttt{ran}) \in$ Circ.
input: Let i be the number of input gates in Circ, run RANDOM(i) and associate one t-sharing $[r_{gid}]$ to each $(gid, \texttt{inp}, P_j) \in$ Circ. Then send all shares of $[r_{gid}]$ to P_j to let P_j compute r_{gid}.
multiplication: Let m be the number of multiplication gates, run TRIPLES(m) and associate one multiplication triple $([a_{gid}], [b_{gid}], [c_{gid}])$ to each $(gid, \texttt{mul}, gid_1, gid_2) \in$ Circ.

Fig. 5. Preprocess(Circ)

Evaluation Phase. Then comes an evaluation phase. During the evaluation phase a t-sharing $[x_{gid}]$ is computed for each gate gid, and we say that gid has been computed when this happens. Note that the random gates are computed already in the preprocessing. A non-output gate a said to be ready when all its input gates have been computed. An output gate is said to be ready when in addition all input gates and random gates in the circuit have been computed.[2] The evaluation proceeds in rounds, where in each round all ready gates are computed in parallel. When several sharings are opened in a round, they are opened in parallel, using one execution of OPEN. The individual gates are handled as detailed in Fig. 6. Note that the evaluation phase consists essentially only of opening sharings and taking affine combinations.

[1] Private outputs can be implemented using a standard masking technique.
[2] This definition will ensure that all inputs have been provided before any outputs are revealed.

The evaluation proceeds in rounds, where in each round all ready gates are computed in parallel, as follows:

input: For $(gid, \texttt{inp}, P_j) \in$ Circ:
1. P_j: Retrieve the input $x_{gid} \in \mathbb{F}$ and send $\delta_{gid} = x_{gid} + r_{gid}$ to all parties.
2. All parties: Compute $[x_{gid}] = \delta_{gid} - [r_{gid}]$.

affine: For $(gid, \texttt{aff}, a_0, gid_1, a_1, \ldots, gid_\ell, a_\ell) \in$ Circ: All parties compute $[x_{gid}] = a_0 + \sum_{l=1}^{\ell} a_l[x_{gid_l}]$.

multiplication: For $(gid, \texttt{mul}, gid_1, gid_2) \in$ Circ all parties proceed as follows:
1. Compute $[\alpha_{gid}] = [x_{gid_1}] + [a_{gid}]$ and $[\beta_{gid}] = [x_{gid_2}] + [b_{gid}]$.
2. Run $\alpha_{gid} \leftarrow \textsc{Open}([\alpha_{gid}])$ and $\beta_{gid} \leftarrow \textsc{Open}([\beta_{gid}])$.
3. Let $[x_{gid}] = \alpha_{gid}\beta_{gid} - \alpha_{gid}[b_{gid}] - \beta_{gid}[a_{gid}] + [c_{gid}]$.

output: For $(\texttt{out}, gid) \in$ Circ: Run $x_{gid} \leftarrow \textsc{Open}([x_{gid}])$.

Fig. 6. Eval(Circ)

The correctness of the protocol is straight-forward except for Step 3 in **multiplication**. The correctness of that step follows from [BB89] which introduced this preprocessed multiplication protocol. The privacy of the protocol follows from the fact that r_{gid} in the input protocol and a_{gid} and b_{gid} in the multiplication protocol are uniformly random elements from $\mathbb{F}$, in the view of the corrupted parties. Therefore $\delta_{gid} = x_{gid} + r_{gid}$ and $\alpha_{gid} = x_{gid_1} + a_{gid}$ and $\beta_{gid} = x_{gid_2} + b_{gid}$ leak no information on x_{gid} respectively x_{gid_1} and x_{gid_2}. Therefore the values of all gates are hidden, except for the output gates, whose values are allowed (required) to leak.

The communication complexity, including preprocessing, is seen to be $\mathcal{O}(n\mathcal{C}k + n^2k)$, where $\mathcal{C} = |\,\text{Circ}\,|$ is the number of gates in the circuit.

4 Robust, $t < n/4$

By now we have a circuit-evaluation protocol which is private and correct as long a no party deviates from the protocol. In this section we add mechanisms to ensure robustness. Throughout this section we assume that there are at most $t = \lfloor (n-1)/4 \rfloor$ corrupted parties. In the following section we then extend the solution to handle $t < n/3$.

4.1 Error Points

In the passive secure protocol there are several points where a party could deviate from the protocol to make it give a wrong output. We comment on two of these here and sketch how they are dealt with. More details follow later. A party which was asked to perform a d-sharing could distribute values which are not d-consistent. We are going to detect a cheater by asking the parties to open a random polynomial combination of all sharings they have dealt. Also, $P_{\texttt{king}}$

could fail to send the right value in OPEN (Fig. 3). We are going to use error correction to make sure this does not matter, by opening a Van der Monde code of the sharings to be opened and then correcting the t mistakes that the corrupted parties might have introduced.

4.2 Coin-Flip

In the protocol opening a polynomial combination of sharings we need a random value $x \in \mathbb{G}$ from the extension field. Therefore we need a protocol, FLIP(), for flipping a random value from $\mathbb{G}$. The standard protocol does this using a VSS protocol as subprotocol: Each $P_i \in \mathcal{P}$: Pick uniformly random $x_i \in_R \mathbb{G}$ and deal a VSS of x_i among the parties in $\mathcal{P}$. All parties: Reconstruct each x_i and let $x = \sum_{P_i \in \mathcal{P}} x_i$. Any of the known VSS's will do, e.g., [BGW88], since we only call FLIP a small number of times, and so its precise complexity is not important.

4.3 Dispute Control

In the following we use the technique of dispute control[BH06]. We keep a dispute set Disputes, initially empty, consisting of sets $\{P_i, P_j\}$ with $P_i, P_j \in \mathcal{P}$. If $\{P_i, P_j\} \in$ Disputes, then we write $P_i \nleftrightarrow P_j$. If during a protocol a dispute arises between P_i and P_j, then $\{P_i, P_j\}$ is added to Disputes. This is done in such a way that: (1) All parties in $\mathcal{P}$ agree on the value of Disputes. (2) If $P_i \nleftrightarrow P_j$, then P_i is corrupted or P_j is corrupted. For a given dispute set Disputes and $P_i \in \mathcal{P}$ we let Disputes_i be the set of $P_j \in \mathcal{P}$ for which $P_i \nleftrightarrow P_j$, and we let $\text{Agree}_i = \mathcal{P} \setminus \text{Disputes}_i$.

All sub-protocols will use the same dispute set Disputes. We say that a sub-protocol has dispute control if (1) It can never halt due to a dispute between P_i and P_j if $\{P_i, P_j\}$ is already in Disputes. (2) If it does not generate a dispute, then it terminates with the correct result. (3) If it generates a dispute, then it is secure to rerun the sub-protocol (with the now larger dispute set).

We also keep a set Corrupt $\subset \mathcal{P}$. If during the run of some protocol a party P_i is detected to deviate from the protocol, then P_i is added to Corrupt. This is done in such a way that: (1) All parties in $\mathcal{P}$ agree on the value of Corrupt. (2) If $P_i \in$ Corrupt, then P_i is corrupted.

We enforce that when it happens for the first time that $|\,\text{Disputes}_i\,| > t$, where t is the bound on the number of corrupted parties, then P_i is added to Corrupt. It is easy to see that if $|\,\text{Disputes}_i\,| > t$, then indeed P_i is corrupted.

Secret Sharing with Dispute Control. We use a technique from [BH06] to perform secret sharing with dispute control. When a party P_i is to deal a d-sharing $[x]$ then P_i uses a random d-polynomial where $f(0) = x$ and $f(j) = 0$ for $P_j \in \text{Disputes}_i$. Since all sharings will have $d \geq t$ and $P_i \in$ Corrupt if $|\,\text{Disputes}_i\,| > t$, this type of dealing is possible for all $P_i \notin$ Corrupt. The advantage is that all parties will agree on what P_i sent to all $P_j \in \text{Disputes}_i$. This can then be exploited to ensure that P_i will never get a new dispute with some $P_j \in \text{Disputes}_i$.

4.4 Dealing Consistent Sharings

The robust protocol for sharing values will run the private protocol for dealing sharings followed by a check that the generated sharings are consistent. In Fig. 7 we consider the case where P_i shares ℓ values $(y_1, \ldots, y_\ell)$.

1. If $P_i \in$ Corrupt, then output $([y_1], \ldots, [y_\ell]) = ([0], \ldots, [0])$, where $[0] = (0, \ldots, 0)$ is the dummy sharing of 0. Otherwise, proceed as below.
2. P_i: Deal d-sharings $[y_1], \ldots, [y_\ell]$ over $\mathbb{F}$ among the parties in $\mathcal{P}$ along with a d-sharing $[r]$ over $\mathbb{G}$, where $r \in_R \mathbb{G}$ is a uniformly random element from the extension field. By definition all parties in Disputes$_i$ get 0-shares.
3. All parties in $\mathcal{P}$: Run $x \leftarrow$ Flip(), and compute $[y] = [r] + \sum_{l=1}^{\ell} x^l [y_l]$ in $\mathbb{G}$.
4. All parties in Agree$_i$: Broadcast the share of $[y]$. All parties in Disputes$_i$ are defined to broadcast 0.
5. P_i: In parallel with the above step, broadcast all shares of $[y]$.
6. All parties in $\mathcal{P}$: If the sharing $[y]$ broadcast by P_i is not a d-sharing with all parties in Disputes$_i$ having a 0-share, then output $([y_1], \ldots, [y_\ell]) = ([0], \ldots, [0])$. Otherwise, if the shares broadcast by the other parties are identical to those broadcast by P_i, then output $([y_1], \ldots, [y_\ell])$. Otherwise, let Disputes$' =$ Disputes$\cup\{(P_i, P_j)\}$ for each $P_j \in$ Agree$_i$ broadcasting a share different from that broadcast by P_i.

Fig. 7. Share$(P_i, \text{Disputes})$

We first argue that if any of the sharings dealt by P_i are not d-sharings, then a dispute will be generated, except with probability $\text{poly}(\kappa)2^{-\kappa}$. Namely, let $f_0(X) \in \mathbb{G}[X]$ be the lowest degree polynomial consistent with the honest shares of $[r]$, and for $i = 1, \ldots, \ell$ let $f_l(X) \in \mathbb{G}[X]$ be the lowest degree polynomial consistent with the honest shares of $[y_l]$. It can be seen that $f_l(X)$ is also the lowest degree polynomial $f_l^{(\mathbb{F})}(X) \in \mathbb{F}[X]$ consistent with the honest shares of $[y_l]$.[3] It follows that if the sharings dealt by P_i are not all d-consistent, then one of the polynomials $f_l(X)$ has degree larger than d. Let m be such that $f_m(X)$ has maximal degree among $f_0(X), \ldots, f_\ell(X)$, let d_m be the degree of $f_m(X)$ and write each $f_l(X)$ as $f_l(X) = \alpha_l x^{d_m} + f_l'(X)$, where $f_l'(X)$ has degree lower than d_m. By definition $\alpha_m \neq 0$. Therefore $g(Y) = \sum_{i=0}^{\ell} \alpha_l Y^l$ is a non-zero polynomial over $\mathbb{G}$ with degree at most ℓ, and since x is uniformly random in $\mathbb{G}$, it follows that $g(x) = 0$ with probability at most $\ell/|\mathbb{G}| = \text{poly}(\kappa)2^{-\kappa}$. So, we can assume that $g(x) \neq 0$. This implies that $f(X) = \sum_{l=0}^{\ell} x^l f_l(X)$ has degree $d_m > d$. Note that $f(X)$ is consistent with the honest shares of $[y] = [r] + \sum_{l=1}^{\ell} x^l [y_l]$, and let $g(X) \in \mathbb{G}[X]$ be the lowest degree polynomial which is consistent with the honest shares of $[y]$. Let $h(X) = f(X) - g(X)$. Clearly $h(i) = 0$ for all honest

[3] The polynomial $f_l(X)$ can be computed from the indexes $i \in \mathbb{F}$ of the honest parties P_i and the shares $x_l^{(i)} \in \mathbb{F}$ of the honest parties P_i using Lagrange interpolation, which is linear. Therefore the coefficients of $f_l(X)$ ends up in $\mathbb{F}$, even when the interpolation is done over the extension $\mathbb{G}$.

parties P_i. Since $h(i)$ has degree at most $d_m < |H|$, where H is the set of honest parties, and $h(i) = 0$ for $i \in H$ it follows that $h(X)$ is the zero-polynomial. So, $g(X) = f(X)$ and $[y]$ thus has degree d_m. Therefore the honest shares of $[y]$ are not on a d-polynomial, and thus some dispute will be generated. It follows that when no dispute is generated, then all the sharings dealt by P_i are d-consistent, except with probability $\text{poly}(\kappa)2^{-\kappa}$. This in particular applies to the sharings $[y_1], \ldots, [y_\ell]$.

As for the privacy, note that when P_i is honest, $\sum_{l=1}^{\ell} x^l[y_l]$ is a d-sharing over $\mathbb{G}$ and $[r]$ is an independent uniformly random d-sharings over $\mathbb{G}$ of a uniformly random $r \in_R \mathbb{G}$. Therefore $[y]$ is a uniformly random d-sharing over $\mathbb{G}$ and leaks no information to the corrupted parties when reconstructed.

If the protocol $\textsc{Share}(P_i, \text{Disputes})$ fails, it is rerun using the new larger $\text{Disputes}'$. Since Disputes_i grows by at least 1 in each failed attempt, at most t failed attempts will occur. So, if $\lceil \ell/t \rceil$ values are shared in each attempt, the total number of attempts needed to share ℓ values will be $2t$. Since each attempt has communication complexity $\mathcal{O}(\lceil \ell/t \rceil nk) + \text{poly}(n\kappa)$ the total complexity is $\mathcal{O}(\ell nk) + \text{poly}(n\kappa)$, where $\text{poly}(n\kappa)$ covers the cost of the n broadcasts and the run of $\textsc{Flip}()$ in each attempt. The round complexity is $\mathcal{O}(t)$.

Dealing Inter-Consistent Sharings. The above procedure allows a party P_i to deal a number of consistent d-sharings. This can easily be extend to allow a party P_i to deal consistent t-sharings $[y_1], \ldots, [y_\ell]$ and $2t$-sharings $\langle Y_1 \rangle, \ldots, \langle Y_\ell \rangle$ with $Y_l = y_l$. The check uses a random t-sharing $[r]$ and a random $2t$-sharing $\langle R \rangle$ with $R = r$, and then $[y] = [r] + \sum_{l=1}^{\ell} x^l[y_l]$ and $\langle Y \rangle = \langle R \rangle + \sum_{l=1}^{\ell} x^l[Y_l]$ are opened as above. In addition to the sharings being t-consistent (respectively $2t$-consistent), it is checked that $Y = y$. Note that if $R = r$ and $X_l = x_l$, then indeed $Y = y$. On the other hand, if $R \neq r$ or some $X_l \neq x_l$, then $Y - y = (R - r) + \sum_{l=1}^{\ell} x^l(Y_l - y_l)$ is different from 0 except with probability $\ell/|\mathbb{G}|$, giving a soundness error of $\text{poly}(\kappa)2^{-\kappa}$.

4.5 Random Double Sharings

Recall that the purpose of this protocol is to generate a set of random values that are unknown to all parties and are both t- and $2t$-shared. The robust protocol for this is derived directly from the passive secure protocol, and we also denote it by $\textsc{Double-Random}(\ell)$. The only difference between the two is that the above robust procedure for dealing inter-consistent sharings is used as subprotocol. To generate ℓ double sharings, first each party deals $\lceil \ell/(n-t) \rceil$ random pairs using the procedure for dealing inter-consistent sharings. Then the first pair from each party is used to compute $n-t$ pairs as in the passive secure $\textsc{Double-Random}(\ell)$, using a matrix $\text{Van}^{(n,n-t)^\top}$. At the same time the second pairs from each party is used to compute $n-t$ more pairs, and so on. This yields a total of $(n-t)\lceil \ell/(n-t) \rceil \geq \ell$ pairs. The communication complexity is $\mathcal{O}(\ell nk) + \text{poly}(n\kappa)$.

4.6 Opening Sharings

We describe how sharings are opened. We assume that the sharings to be opened are consistent d-sharings of the same degree $d \leq 2t$. Reconstruction of a single sharing happens by sending all shares to some king, which then reconstructs. Since the sharing is d-consistent, the king receives at least $n - t > 3t = d + t$ correct sharings and at most t incorrect sharings. Therefore the king can always compute the d-polynomial $f(X)$ of the sharing using Berlekamp-Welch. The details are given in Fig. 8.

1. The parties agree on a consistent d-sharings $[x]$ with $d \leq 2t$.
2. Each $P_i \in \mathcal{P}$: Send the share x_i of $[x]$ to $P_{\texttt{king}}$.
3. $P_{\texttt{king}}$: Run Berlekamp-Welch on the received shares to get x, and send x to all parties.

Fig. 8. Open$(P_{\texttt{king}}, d, [x])$

The protocol in Fig. 8 has the obvious flaw that $P_{\texttt{king}}$ could send an incorrect value. This is handled by expanding $n - (2t + 1)$ sharings to n sharings using a linear error correcting code tolerating t errors. Then each P_i opens one sharing, and the possible t mistakes are removed by error correction. The details are given in Fig. 9.

1. The parties agree on consistent d-sharings $[x_1], \ldots, [x_\ell]$ with $\ell = n - (2t + 1)$ and $d \leq 2t$.
2. All parties: Compute $([y^{(1)}], \ldots, [y^{(n)}]) = M([x_1], \ldots, [x_\ell])$, where $M = \mathrm{Van}^{(n,\ell)}$.
3. All parties: For each $P_i \in \mathcal{P}$ in parallel, run $y^{(i)} \leftarrow \textsc{Open}(P_i, d, [y^{(i)}])$.
4. All parties: Run Berlekamp-Welch on the values $(y^{(1)}, \ldots, y^{(n)})$ to get $(x_1, \ldots, x_\ell)$.

Fig. 9. OpenRobust$(d, [x_1], \ldots, [x_\ell])$

The communication complexity of opening $\ell = n - (2t + 1)$ values is $\mathcal{O}(n^2 k)$, giving an amortized communication complexity of $\mathcal{O}(nk)$ per reconstruction. An arbitrary ℓ sharings can thus be reconstructed with communication complexity $\mathcal{O}(\ell nk + n^2 k)$.

4.7 Circuit Evaluation

We now have robust protocols Double-Random (and thus Random) and OpenRobust. This allows to implement a robust version of Triples exactly as in Fig. 4. Then a robust preprocessing can be run exactly as in Fig. 5. This in

turn almost allows to run a robust circuit-evaluation as in Fig. 6. Note in particular that since all sharings computed during the circuit evaluation are linear combinations of sharings constructed in the preprocessing, all sharings will be consistent t-sharings. Therefore Berlekamp-Welch can continuously be used to compute the openings of such sharings. Indeed, the only additional complication in running Eval(Circ) is in Step 2 in **input**, where it must be ensured that P_j sends the same δ_{gid} to all parties. This is handled by distributing all δ_{gid} using n parallel broadcasts (each P_j broadcast all its δ_{gid} in one message).

Since an ℓ-bit message can be broadcast with communication complexity $\mathcal{O}(\ell) + \text{poly}(n\kappa)$ (see [FH06]), the communication complexity of handling the input gates will be $\mathcal{O}(\ell_i nk) + \text{poly}(n\kappa)$, where ℓ_i is the number of input gates. The communication complexity of handling the remaining gates is seen to be $\mathcal{O}(n\mathcal{C}k + (D+1)n^2k) + \text{poly}(n\kappa)$, where $\mathcal{C} = |\,\text{Circ}\,|$ and D is the multiplicative depth of the circuit. The term $(D+1)n^2k$ comes from the fact that OpenRobust is run for each layer of multiplication gates and run once to handle the output gates in parallel. If we sum the above with the communication complexity of the preprocessing phase we get a communication complexity of $\mathcal{O}(n\mathcal{C}k + (D+1)n^2k) + \text{poly}(n\kappa)$. The round complexity is $\mathcal{O}(t + D + 1)$, where t comes from running the robust sharing protocol.

5 Robust, $t < n/3$

We sketch how the case $t < n/3$ is handled. Assume first that a preprocessing phase has been run where the "usual" consistent t-sharings have been associated to each gate. Since $t < n/3$ it follows that there are at least $2t+1$ honest shares in each sharing. Therefore Berlekamp-Welch can be used to reconstruct these t-sharings. Since the circuit-evaluation phase consists only of opening t-sharings, it can thus be run exactly as in the case $t < n/4$.

The main problem is therefore to establish the preprocessed t-sharings. The protocol Share can be run as for $t < n/4$ as can then Double-Random. Going over Triples it thus follows that the only problematic step is $D \leftarrow$ Open$(2t, \langle D \rangle)$, where a sharing of degree $2t$ is opened. Since there are only $2t+1$ honest parties, Berlekamp-Welch cannot be used to compute D in Step 3 in Fig. 8. An honest party $P_{\texttt{king}}$ can therefore find itself not being able to contribute correctly to the reconstruction. This is handled by ensuring that when this happens, then $P_{\texttt{king}}$ can complain and the parties together identify a new dispute.

In more detail, when $P_{\texttt{king}}$ is to reconstruct some $2t$-sharing $\langle y \rangle$ as part of the run of Open in Triples (Fig. 4), then $P_{\texttt{king}}$ collects shares y_j from each $P_j \in \text{Agree}_{\texttt{king}}$.[4] If these shares are on some $2t$-polynomial $D(X) \in \mathbb{F}[X]$, then $P_{\texttt{king}}$ sends $D(0)$ to all parties. Otherwise some $P_j \in \text{Agree}_{\texttt{king}}$ sent an incorrect y_j. This is used to find a new dispute, as detailed in Fig. 10.

[4] $P_{\texttt{king}}$ can safely ignore D_j from $P_j \in \text{Disputes}_{\texttt{king}}$ as $P_{\texttt{king}}$ knows that these P_j are corrupted.

0. Assume that $P_{\texttt{king}}$ was reconstructing some $2t$-sharing $\langle y \rangle$ as part of the opening in TRIPLES (Fig. 4), and assume that the shares y_j for $P_j \in \text{Agree}_{\texttt{king}}$ are $2t$-inconsistent.
1. Each $P_j \in$ Honest: Broadcast all the shares sent and received during RANDOM and DOUBLE-RANDOM.[a] At the same time $P_{\texttt{king}}$ broadcasts the $2t$-inconsistent shares $\langle y \rangle$.
2. All parties: If some $P_j \in$ Honest claims to have sent sharings which do not have the correct degree, then add P_j to Corrupt and terminate. Otherwise, if some P_i and P_j disagree on a share $R_j^{(i)}$ sent from P_i to P_j, then add the dispute $P_i \nleftrightarrow P_j$.[b] Otherwise, proceed as below.
3. All parties: Compute from the broadcast sharings the $2t$-sharing $\langle y' \rangle$ that $P_{\texttt{king}}$ was reconstructing. Since $\langle y' \rangle$ is $2t$-consistent and $\langle y \rangle$ is $2t$-inconsistent on $\text{Agree}_{\texttt{king}}$, there exists $P_i \in \text{Agree}_{\texttt{king}}$ where $y'_i \neq y_i$. For each such P_i, add the dispute $P_{\texttt{king}} \nleftrightarrow P_i$.[c]

[a] This is secure as the secret inputs did not enter any computation yet in the preprocessing phase.

[b] Note that the dispute will be new, as $P_i \nleftrightarrow P_j$ implies that $R_j^{(i)}$ is defined to be 0, and thus no dispute can arise.

[c] The dispute is new as $P_i \in \text{Agree}_{\texttt{king}}$. Note that we cannot add P_i to Corrupt, as $P_{\texttt{king}}$ could be lying about y_i.

Fig. 10. Detect-Dispute

The protocol in Fig. 10 can be greatly optimized, using a more involved protocol avoiding many of the broadcasts. However, already the simple solution has communication complexity $\text{poly}(n\kappa)$. Since the protocol always leads to a new dispute and at most $\mathcal{O}(t^2)$ disputes are generated in total, it follows that the protocol contributes with a term $\text{poly}(n\kappa)$ to the overall communication complexity. This gives us a robust protocol with communication complexity $\mathcal{O}(n\mathcal{C}k + (D+1)n^2k) + \text{poly}(n\kappa)$ for the case $t < n/3$.

6 Robust, $t < n/2$

It is yet an open problem to get an unconditionally secure protocol with linear communication complexity for the case $t < n/2$. One can however construct a protocol withstanding a computationally bounded adversary *during* the protocol and a computationally unbounded adversary *after* the protocol execution. Due to space limitations we can only sketch the involved ideas.

Transferable Secret Sharing. To allow some $P_{\texttt{king}}$ to reconstruct a sharing and, verifiably, transfer the result to the other parties, each t-sharing $[x]$ is augmented by a Pedersen commitment $C = \text{commit}(x; r)$, known by all parties, and the randomness r is shared as $[r]$. We write $[\![x]\!] = (C, [x], [r])$. In the preprocessing, each P_j generates many random t-sharings $[\![x]\!]$ and corresponding, normal

$2t$-sharings $\langle x \rangle$, where $P_i \in \text{Disputes}_j$ get $x_i = r_i = 0$. By committing using a group of prime order q and secret sharing in $\text{GF}(q)$, the sharings $[\![x]\!] = (C, [x], [r])$ are linear modulo q, and the usual protocols can be used for checking consistency of sharings and for combining random sharings using a Van der Monde matrix (Sec. 4.4 and 4.5). When a consistency check fails, a new dispute is identified using a technique reminiscent of that used when $t < n/3$, but slightly more involved. Then a new attempt is made at generating random shared values. The total overhead of identifying disputes is kept down by generating the random shared values in phases, each containing a limited number of sharings.

Multiplication Triples. After enough pairs $([\![x_l]\!], \langle x_l \rangle)$ have been generated, a multiplication protocol to be described below is run to generate multiplication triples. Generation of triples are done in n sequential phases, where for efficiency checks for correct behavior are done simultaneously for all triples in a phase.

The input to the generation of a multiplication triple is sharings $[\![a]\!]$, $[\![b]\!]$, $[\![r]\!]$, $\langle r \rangle$, $[\![\tilde{b}]\!]$, $[\![\tilde{r}]\!]$, $\langle \tilde{r} \rangle$. The parties compute their shares in $[a][b] + \langle r \rangle$, $[a][\tilde{b}] + \langle \tilde{r} \rangle$, send these to a selected party $P_{\texttt{king}}$, who reconstructs values D, respectively $\tilde{D}$, and sends these values to all players. Players now compute $[\![c]\!] = D - [\![r]\!]$ and $[\![\tilde{c}]\!] = \tilde{D} - [\![\tilde{r}]\!]$. For simplicity we assume that $2t+1 = n$, in which case the shares received by $P_{\texttt{king}}$ will always be $2t$-consistent. Hence even an honest $P_{\texttt{king}}$ might be tricked into reconstructing wrong D and $\tilde{D}$, this problems is handled below. Also, a dishonest $P_{\texttt{king}}$ may distribute inconsistent values for $D, \tilde{D}$. We therefore run the share consistency check from Sec. 4.4 over all $[\![c]\!], [\![\tilde{c}]\!]$ in the current phase and disqualify $P_{\texttt{king}}$ if it fails. Now the (supposed) multiplication triples $([\![a]\!], [\![b]\!], [\![c]\!])$ and $([\![a]\!], [\![\tilde{b}]\!], [\![\tilde{c}]\!])$ are checked. First a uniformly random value $X \in_R \text{GF}(q)$ is flipped. Then it is checked that $([\![a]\!], [\![\tilde{b}]\!] + X[\![b]\!], [\![\tilde{c}]\!] + X[\![c]\!])$ is a multiplication triple: Compute $b = \textsc{Open}([\![\tilde{b}]\!] + X[\![b]\!])$, compute $d = \textsc{Open}([\![a]\!]b - ([\![\tilde{c}]\!] + X[\![c]\!]))$, and check that $d = 0$. If $d = 0$, then $([\![a]\!], [\![b]\!], [\![c]\!])$ is taken to be the generated triple. Here Open refers to the reconstruction procedure described below, which either disqualifies $P_{\texttt{king}}$ or lets at least one honest player compute d plus a proof that it is correct. He can therefore alert all parties if $d \neq 0$.

For efficiency the same X is used for all checks done in the same phase. If $d \neq 0$, then all messages sent and received during the multiplication and the generating of the sharings involved in the multiplication are broadcast and some new dispute is found. Since $\mathcal{O}(1)$ sharings are involved in multiplication, each being a linear combination of at most $\mathcal{O}(n)$ sharings, at most $\mathcal{O}(n)$ sharings are broadcast to find a dispute. Since at most t^2 disputes are generated, the total overhead is thus independent of the circuit size.

Reconstructing. As usual, the evaluation phase proceeds by computing affine combinations of the t-sharings $[\![x]\!]$ generated in the preprocessing and by opening such sharings. All that we need is thus a protocol for opening such t-sharings. From a beginning some fixed reconstructor $P_{\texttt{king}} \in \text{Honest}$ is chosen. In a reconstruction of $[\![x]\!] = (C, [x], [r])$, each P_i sends (x_i, r_i) to $P_{\texttt{king}}$, who computes (x, r)

and sends (x, r) to all parties. The receivers can check that $C = \text{commit}(x; r)$. If the shares received by $P_{\texttt{king}}$ are not t-consistent, then $P_{\texttt{king}}$ complains and a special protocol FIND-CORRUPT-SHARE is run to remove some corrupted party from Honest. The details of how FIND-CORRUPT-SHARE identifies a new corrupted party from the incorrect shares is rather involved, and the details will be given in the full version due to space limitations. Most important is that the communication complexity of FIND-CORRUPT-SHARE is bounded by $\mathcal{O}(|\,\text{Circ}\,|k) + \text{poly}(n\kappa)$. Since each run of FIND-CORRUPT-SHARE removes one new party from Honest, it is run at most $t = \mathcal{O}(n)$ times, giving a total overhead of at most $\mathcal{O}(|\,\text{Circ}\,|nk) + \text{poly}(n\kappa)$. The procedure FIND-CORRUPT-SHARE will be run until $P_{\texttt{king}}$ sees that the shares (x_i, r_i) from parties in Honest are t-consistent. At this point $P_{\texttt{king}}$ can then interpolate (x, r) and send it to all parties. The above procedure always allows an honest $P_{\texttt{king}}$ to compute (x, r) and send it to all parties, maybe after some runs of FIND-CORRUPT-SHARE. A party P_i not receiving (x, r) such that $C = \text{commit}(x; r)$ therefore knows that $P_{\texttt{king}}$ is corrupt. If P_i does not receive an opening it therefore signs the message "P_i `disputes` $P_{\texttt{king}}$" and sends it to all parties. All parties receiving "P_i `disputes` $P_{\texttt{king}}$" signed by P_i adds the dispute $P_i \nleftrightarrow P_{\texttt{king}}$ and sends the signed "P_i `disputes` $P_{\texttt{king}}$" to all parties. Any party which at some point sees that $P_{\texttt{king}}$ has more than t disputes stops the execution. Then $P_{\texttt{king}}$ is removed from Honest, and some fresh $P_{\texttt{king}} \in$ Honest is chosen to be responsible for reconstructing the sharings. Then the computation is restarted. When a new reconstructor $P_{\texttt{king}}$ is elected, all gates that the previous reconstructor have (should have) handled might have to be reopened by the new $P_{\texttt{king}}$. To keep the cost of this down, each reconstructor will handle only $\mathcal{O}(|\,\text{Circ}\,|/n)$ gates before a new reconstructor is picked.

Distributing Results. At the end of the evaluation, each sharing $[\![y]\!]$ associated to an output gate has been opened by some $P_{\texttt{king}}$ which was in Honest at the end of his reign. This means that $P_{\texttt{king}}$ was, at the end of his reign, disputed by at most t parties, which in turn implies that at least one of the $t+1$ *honest* parties holds the opening of the $[\![y]\!]$ handled by $P_{\texttt{king}}$. But it is not guaranteed that *all* honest parties hold the opening. Therefore, each $P_i \in$ Honest is made responsible for $O/|\,\text{Honest}\,|$ of the O output sharings $[\![y]\!]$. All parties holding an opening of $[\![y]\!]$ for which P_i is responsible sends the opening to P_i; At least the one honest party will do so, letting P_i learn all openings. Then each P_i sends the opening of each $[\![y]\!]$ for which it is responsible to all parties. The total communication for this is $\mathcal{O}(Onk)$. Since there is honest majority in Honest, all parties now hold the opening of more than half the outputs, and they know which ones are correct. To ensure that this is enough, a Van der Monde error correction code is applied to the outputs before distribution. This is incorporated into the circuit, as a final layer of affine combinations. The only cost of this is doubling the number of output gates in the circuit. Note that the code only has to correct for erasures and hence can work for $t < n/2$.

References

[BB89] Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: PODC'89, pp. 201–209 (1989)

[Bea89] Beaver, D.: Multiparty protocols tolerating half faulty processors. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 560–572. Springer, Heidelberg (1990)

[Bea91] Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 420–432. Springer, Heidelberg (1992)

[BFKR90] Beaver, D., Feigenbaum, J., Kilian, J., Rogaway, P.: Security with low communication overhead (extended abstract). In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 62–76. Springer, Heidelberg (1991)

[BGW88] Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th STOC, pp. 1–10 (1988)

[BH06] Beerliova-Trubiniova, Z., Hirt, M.: Efficient multi-party computation with dispute control. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 305–328. Springer, Heidelberg (2006)

[BMR90] Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: 22nd STOC, pp. 503–513 (1990)

[Can01] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145 (2001)

[CCD88] Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th STOC, pp. 11–19 (1988)

[CDD+99] Cramer, R., Damgård, I., Dziembowski, S., Hirt, M., Rabin, T.: Efficient multiparty computations secure against an adaptive adversary. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 311–326. Springer, Heidelberg (1999)

[CDD00] Cramer, R., Damgård, I., Dziembowski, S.: On the complexity of verifiable secret sharing and multiparty computation. In: 32nd STOC, pp. 325–334 (2000)

[CDG87] Chaum, D., Damgård, I., van de Graaf, J.: Multiparty computations ensuring privacy of each party's input and correctness of the result. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 87–119. Springer, Heidelberg (1988)

[CDM00] Cramer, R., Damgård, I., Maurer, U.: 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 (2000)

[DI06] Damgård, I., Ishai, Y.: Scalable secure multiparty computation. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 501–520. Springer, Heidelberg (2006)

[FH06] Fitzi, M., Hirt, M.: Optimally efficient multi-valued byzantine agreement. In: PODC 2006 (2006)

[GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: 19th STOC (1987)

[GRR98] Gennaro, R., Rabin, M., Rabin, T.: Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In: PODC'98 (1998)

[GV87] Goldreich, O., Vainish, R.: How to solve any protocol problem - an efficiency improvement. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 73–86. Springer, Heidelberg (1988)

[HM01] Hirt, M., Maurer, U.: Robustness for free in unconditional multi-party computation. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 101–118. Springer, Heidelberg (2001)

[HMP00] Hirt, M., Maurer, U.M., Przydatek, B.: Efficient secure multi-party computation. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 143–161. Springer, Heidelberg (2000)

[HN05] Hirt, M., Nielsen, J.B.: Upper bounds on the communication complexity of optimally resilient cryptographic multiparty computation. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 79–99. Springer, Heidelberg (2005)

[HN06] Hirt, M., Nielsen, J.B.: Robust multiparty computation with linear communication complexity. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 463–482. Springer, Heidelberg (2006)

[RB89] Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority. In: 21th STOC, pp. 73–85.

[Yao82] Yao, A.C.-C.: Protocols for secure computations (extended abstract). In: 23rd FOCS.