Secure Multi-Party Computation
(Excerpts of Chapter 2)

Sandro Coretti

June 20, 2011

Contents

1 Introduction

This is a selection of the key parts of Chapter 2 of the lecture notes of the course on *Cyptographic Protocols*, intended as a reference for non-German speakers. The parts that are covered are the ones we feel cannot be easily understood by just taking notes during the lecture. Therefore, this summary does not contain all material relevant for the exam.

2 Sharing Schemes

A t-out-of-n secret-sharing scheme allows an *honest* dealer D to distribute a secret s among n players, such that any subset of t players has no information about s, but every set of $t+1$ players can collaboratively reconstruct the secret. The most famous secret-sharing scheme is Shamir's Sharing Scheme [Sha79] (see Section 2.2). It uses polynomials to obtain the desired properties. Before presenting the scheme, we first take a look at Lagrange interpolation.

2.1 Lagrange Interpolation

Consider a field $\mathbb{F}$ and n points $(\alpha_1, s_1), \ldots, (\alpha_n, s_n)$. Lagrange interpolation can be used to find the unique polynomial over $\mathbb{F}$ of degree at most $n-1$ that contains these points: For $i = 1, \ldots, n$, consider the polynomial

$$\ell_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - \alpha_j}{\alpha_i - \alpha_j}.$$

Clearly, $\ell_i(\alpha_i) = 1$ and $\ell_i(\alpha_j) = 0$ for $j \neq i$. Therefore, a polynomial that goes through the points (α_i, s_i) for $i = 1, \ldots, n$ is given by

$$g(x) = \sum_{i=1}^{n} \ell_i(x) \cdot s_i.$$

The polynomial g can be shown to be unique (see Exercise 8). Moreover, note that the value of g at some fixed position is a linear function of the s_i.

2.2 Shamir's Scheme

Consider a finite field $\mathrm{GF}(q)$ of size greater than n.[1] Assume there is a unique, fixed, and known value $\alpha_i \in \mathrm{GF}(q) \setminus \{0\}$ associated with every player P_i. The dealer D chooses a random polynomial $p(\cdot)$ over $\mathrm{GF}(q)$ and of degree at most t such that $p(0) = s$. This can be done by choosing $r_1, \ldots, r_t$ at random from $\mathrm{GF}(q)$ and defining

$$p(x) := s + r_1 x + r_2 x^2 + \ldots + r_t x^t.$$

Then, for $i = 1, \ldots, n$, he sends the value $s_i := p(\alpha_i)$, called the i^{th} *share*, to player P_i. All shares together are called a *sharing* of s. Sometimes, we denote a sharing of s by $[s]$.

Given any $t+1$ shares one can compute the secret s using Lagrange interpolation. Furthermore, one can show that up to t shares give no information about s (see Exercise 8).

Shamir sharings are linear. That is, the sharing $\mathbf{s} = (s_1, \ldots, s_n)^\intercal$ can be obtained by multiplying the vector $\mathbf{r} = (s, r_1, \ldots, r_t)^\intercal$ consisting of the secret and the random coefficients with a matrix $\mathbf{M}$ (which the reader can find as an exercise). That is, $\mathbf{s} = \mathbf{M} \cdot \mathbf{r}$. From two sharings $\mathbf{s} = \mathbf{M}\mathbf{r}$ of s and $\mathbf{s}' = \mathbf{M}\mathbf{r}'$ of s', a sharing of the sum $s + s'$ can be computed by having each player adding his shares. This is due to the fact that $\mathbf{s} + \mathbf{s}' = \mathbf{M}\mathbf{r} + \mathbf{M}\mathbf{r}' = \mathbf{M}(\mathbf{r} + \mathbf{r}')$ and thus the secret in $\mathbf{s} + \mathbf{s}'$ is $s + s'$.

3 Secure Multi-Party Computation

3.1 Overview

The task of multi-party computation (MPC) is to enable n mutually distrusting parties $P_1, \ldots, P_n$ to compute some function on their inputs, which however they do not wish to reveal. This task can be reformulated using a trusted third party (TTP) that takes the inputs of the parties, computes the function, and returns the output(s). The goal of MPC is to simulate such a TTP.

We start by introducing the model and the security requirements for MPC protocols and then describe solutions for the various settings.

[1] Recall that n denotes the number of players.

3.1.1 Model

In the following we will make some (more or less natural) assumptions on our communication network.

- Each pair of parties is connected by **secure channels**. When constructing computationally secure protocols, such channels can be implemented, e.g., using some (already present) public-key infrastructure. If one is interested in information-theoretic security, one assumes that the channels are somehow physically secured or realized using one-time pad encryption.
- The network used by the parties is perfectly **synchronous**. This means that messages arrive instantaneously at their destination. Protocols for this model proceed in so-called *rounds*.
- There is a so-called **broadcast channel**, which roughly corresponds to a megaphone some party can use to send the *same* message to all other parties. Under certain assumptions, such channels can be realized.

3.1.2 Security Requirements

Players participating in an MPC can be dishonest. This is modeled by a *central* a dversary that may corrupt up to t of the players, for some bound $t < n$. There are two main types of corruption:

- **Passive Corruption:** Passively corrupted parties follow the protocol exactly, but the adversary has access to their internal state and, thus, learns whatever they learn during the computation.
- **Active Corruption:** Actively corrupted parties can be instructed by the adversary to deviate from the protocol in arbitrary ways. Note that an actively corrupted party is also passively corrupted.

Parties not corrupted by the adversary are called *honest* or *uncorrupted* parties.
The typical security properties required of an MPC protocol are the following:

- **Privacy:** The adversary must not be able to learn any information about the inputs and the outputs of the uncorrupted parties except for what he would learn from the inputs and outputs of the corrupted parties anyway.
- **Correctness:** The adversary cannot falsify the output of the computation.
- **Fairness:** The adversary cannot abort the protocol with an *advantage*. In other words, as soon as the adversary learns anything about the output, all honest players will learn the complete output.
- **Robustness:** The adversary cannot make the protocol abort at all. In particular, a robust protocol is fair.

Somewhat more precisely, the security of an MPC protocol is defined relative to a so-called *specification*. A specification is a protocol between the players and a TTP that computes the function the players want to evaluate. An MPC protocol is said to securely realize some specification if everything the adversary could achieve in an execution of the protocol he could also achieve in the specification by corrupting the same players.

3.2 Passively Secure Multi-Party Computation

In this section we present a simplified version of the MPC protocol by [BGW88], which provides security against up to $t < n/2$ *passively* corrupted players. This is optimal in the sense that there cannot exist a protocol with these properties that allows to evaluate an arbitrary function and tolerates more passively corrupted players.

The function that the players want to evaluate must be given as an arithmetic circuit over a finite field $\mathrm{GF}(q)$ with $q > n$, consisting of addition and multiplication gates. We will denote the player set by $\mathcal{P} = \{P_1, \ldots, P_n\}$. Without loss of generality we assume that player P_i holds input $x_i \in \mathrm{GF}(q)$.

The basic idea of the protocol is the following: At the beginning, every player shares his input using Shamir's sharing scheme. Then, in a gate-by-gate fashion, the players will evaluate every

Passively Secure MPC

Input

To share his input x_i, each player P_i chooses random polynomial $f(x)$ of degree at most t such that $f(0) = x_i$. Then, for $j = 1, \ldots, n$, he sends $x_{ij} := f(\alpha_j)$ to P_j.

Addition Gates (and Linear Functions)

Each player P_i starts out with the shares a_i and b_i of the inputs to the gate a and b, respectively. He computes a share c_i of $c = a + b$ by adding his shares, i.e., $c_i = a_i + b_i$.

Multiplication Gates

Each player P_i starts out with the shares a_i and b_i of the inputs to the gate a and b, respectively. He computes a share c_i of $c = a \cdot b$ by as follows:

1. Compute $d_i := a_i \cdot b_i$.
2. Compute a Shamir sharing $(d_{i1}, \ldots, d_{in})$ of d_i.
3. For $j = 1, \ldots, n$: Send share d_{ij} to player P_j.
   For $j = 1, \ldots, n$: Let d_{ji} be share of $d_j = a_j \cdot b_j$ received from player P_j.
4. Compute share of $c = a \cdot b$ as

$$c_i := \sum_{j=1}^{n} w_j d_{ji} \qquad \text{where} \qquad w_j = \prod_{\substack{k=1 \\ k \neq j}}^{n} \frac{\alpha_k}{\alpha_k - \alpha_j}.$$

Output

To reconstruct a value a, each player P_i sends his share a_i to all players. Then, he uses Lagrange interpolation to compute a from shares a_j received by players P_j.

Figure 1: *Passively secure MPC protocol.*

gate. Here, evaluate means that if the inputs to some gate are shared among the players, then, after the evaluation of the gate, the output of the gate is shared among the players. Once the last gate is evaluated, the players reconstruct the output using Lagrange interpolation. The full protocol is depicted in Figure 1.

Since it is clear from the description given in Section 2 how the sharing and reconstruction phases work, we now focus on how to evaluate addition and multiplication gates.

Evaluating addition gates is easy as already pointed out in Section 2.2: Each player just adds his shares of the summands to obtain a share of the sum. This can obviously be done without communication. Moreover, it easy to see that this argument extends to any linear function, not just addition.

Evaluating multiplication gates is somewhat more challenging. Suppose that from two sharings $[a] = (a_1, \ldots, a_n)$ and $[b] = (b_1, \ldots, b_n)$ the players wish to compute a sharing $[c] = (c_1, \ldots, c_n)$ of the product $c = a \cdot b$. Following the same approach as with addition, one might be tempted to simply have each player P_i compute the product $d_i = a_i \cdot b_i$ of his two shares. However, now the values d_i lie on a polynomial of degree $2t$, which implies that the players could only compute one multiplication before they would become unable to reconstruct the sharing. Moreover, the product polynomial is no longer random, which violates privacy (see Exercise 8).

Therefore, we are forced to take a different approach. Let $f(x)$ and $g(x)$ denote the polynomials (of degree at most t) used to share a and b, respectively. Consider the product polynomial $h(x) := f(x) \cdot g(x)$, which has degree at most $2t$. Clearly, we have $h(0) = c$ and $h(\alpha_i) = d_i$ for $i = 1, \ldots, n$. That is, since $n > 2t$, the points (α_i, d_i) define the polynomial $h(x)$. Hence, Lagrange interpolation (see Section 2.1) can be used to compute

$$c = h(0) = \sum_{i=1}^{n} w_i \cdot d_i \qquad \text{where} \qquad w_i = \ell_i(0) = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{\alpha_j}{\alpha_j - \alpha_i}.$$

Thus, the product c can be expressed as a linear combination $\mathcal{L}$ of the values d_i. All one needs to do now is to come up with a way to evaluate $\mathcal{L}$ on the inputs d_i. But we already know how to securely evaluate linear functions: Each player P_i shares his input d_i and applies $\mathcal{L}$ to his shares. At this point the players have a sharing (of degree t) of c.

Analysis

The correctness of the protocol can easily be verified since we consider passive adversaries only.

In order prove that the protocol is private, one needs to show that the adversary, until before the reconstruction phase, does not obtain any information on the inputs of uncorrupted parties or intermediate results. During reconstruction the adversary only learns information he is allowed to know according to the specification.

In the protocol there are three places in which the adversary might potentially obtain unauthorized information. It is easily seen, however, that privacy is not violated:

- When an honest party *shares* a value (either an input or a product share), it uses fresh randomness and thus the shares of the corrupted parties are statistically independent of the shared value.
- *Local computation* obviously does violate privacy.
- When *reconstructing*, only players who are supposed to learn the output receive shares by honest players. Thus, privacy is maintained.

3.3 Actively Secure Multi-Party Computation

In this section we modify the protocol secure against passive adversaries from Section 3.2 to provide security against actively corrupted players. We will do so in a generic fashion and then provide two instantiations which lead to cryptographic and information-theoretic security, respectively.

3.3.1 Basic Idea

There are three forms of active misbehavior: divulging secret information, not sending values, and sending incorrect values. In the following we will incrementally improve the passive protocol to be secure against each of these forms.

Divulging Secret Information. In our protocol, the adversary could, e.g., use non-random coefficients to share values, or he could send values he is not supposed to send. Clearly, the worst-case scenario is if up to t players publish all of their information. But since the adversary already knows this information and honest players must ignore it, this kind of active corruption does not pose a problem.

Not Sending Values. There are three points in the protocol at which players can refuse to send values:

1. While sharing a value, a corrupted dealer might not send a share to some of the players.
2. In the multiplication protocol, a corrupted player might not send his product share to some of the players.
3. During reconstruction a corrupted player might not send his share to some of the players.

Case 3 is easy to solve: As $n > 2t$, an honest player receives at least $n - t \geq t + 1$ shares, which suffices to reconstruct a polynomial of degree at most t.

In Case 1, a player who did not receive a share accuses the dealer publicly, using broadcast. Then, the accused must broadcast the corresponding share. If he refuses to comply, he is disqualified and a default value is assumed as his input. This can be done using a degree-0 Shamir sharing, i.e., a polynomial that evaluates to this default value at every point. Note that the publication of the share using broadcast does not constitute a problem as the value broadcast is already known to the adversary (either the dealer is corrupted, or the accusing player is corrupted and has already received the value from the dealer).

To deal with Case 2, one first proceeds as in Case 1, i.e., by accusation and broadcast. However, in this case, if the accused player refuses to broadcast the necessary value, the remaining players cannot just assume a default value since they need his product share d_i in the multiplication protocol.

There is a number of different ways to handle this:

- The entire protocol is repeated without the fallible player (decreasing both the number of total players and the number of cheating players by one. Since the fault occurred during a multiplication, no players have received information about the output yet.
- The factor shares a_i and b_i of the cheating player (and only those) are reconstructed. Then, each player can use a degree-0 sharing of the product share $d_i = a_i b_i$ in remainder of the computation. The cheater is, however, not eliminated. Such an approach, in particular how to reconstruct some uncooperating player's share, is discussed in Exercise 9.
- The cheating player is eliminated and all of his relevant shares are interpolated. In the remainder of the protocol, each player simulates the disqualified one locally. When a value is to be sent to that player, it has to be broadcast to all parties instead.
- The fallible player is eliminated but his shares are *not* interpolated. Instead, the players re-share all intermediate values with degree $t - 1$. The resharing can be done in a fashion similar to reconstructing a share.

The former two approaches are the most common ones in the literature. However, none of them is particularly elegant. The first one will only work for secure function evaluation but not for general MPC, i.e., for simulating interactive specifications with intermediate results. As to the second one: It is unclear why one would allow an obvious cheater to further disrupt the protocol. The latter two solutions are more elegant, while the last one turns out to be more efficient than any of the others.

Sending Wrong Values. After having introduced measures that make our protocol secure against players who refuse to send certain values, it now remains to deal with players who send wrong values. The idea will be to make sure that the sending of incorrect values is detected by honest parties who then treat this as if no value had been sent (using the solutions described above).

Roughly speaking, to ensure that players cannot send incorrect values without being detected, every player will be *committed* to every value he knows at any given time. Whenever a player is supposed to send some value to some other player, the commitment must be *transferred* to the recipient. When a value is to be broadcast, the sender must open the commitment to all players. Finally, whenever some player performs some internal computation, he commits to the result and proves (e.g., using a general zero-knowledge proof of knowledge) that the result was computed correctly.

Depending on the level of security one wants to achieve, different commitment schemes are used: If one desires computational security only, one uses normal, cryptographically secure commitment schemes like, e.g., Pedersen or ElGamal Commitments (see Section 3.3.3). To obtain information-theoretic security, one uses special *distributed* commitment schemes (see Section 3.3.4).

3.3.2 Generic Actively Secure MPC

Motivated by the discussion in the previous section, we postulate a commitment scheme with the following sub-protocols:

- COMMIT: A player can *commit* to some value.
- OPEN: A player committed to some value can *open* the commitment to some other player or to all players.
- CTP (Commitment Transfer Protocol): A commitment of some player to some value can be *transferred* to some other player.
- CMP (Commitment Multiplication Protocol): Two commitments of some player to two values can be transformed into a commitment of the *product* of these two values.

Finally, we require that the scheme be *homomorphic*. That is, from two commitments of some player to two values, one can compute a commitment to the *sum* of these two values (without communication).

Recall that the basic idea is to detect the sending of wrong values and treat it as if none had been sent. Therefore, it is imperative that for each of the sub-protocols, the honest parties are in agreement whether it was executed successfully or not.

The operations COMMIT and OPEN are part of any commitment scheme. If one uses commitment schemes made for a single receiver, one needs to ensure that the sender commits to the same values with respect to all players.

The protocol to transfer commitments will be used whenever some party sends a value to some other party: If the protocol is successful, the recipient will be committed to the same value (and know how to open the commitment to that value).

Finally, the protocol for commitment multiplication allows some player to provably commit to the product of two shares.

We are now ready to present the generic protocol, which we later instantiate with cryptographically and information-theoretically secure commitment schemes. First, we construct (from the postulated commitment scheme) a commitment sharing protocol (CSP, see Figure 2), which allows a dealer committed to some value to "share" this value in such a way that every player ends up being committed to his share.

Commitment Sharing Protocol

Starting point: Dealer is committed to some value s.
Goal: Every player has a share of s and is committed to it.

1. The dealer chooses the random coefficients used in the secret sharing scheme and commits to them.
2. Each player (locally) computes the commitments to all shares (using the homomorphic property).
3. For every player, the dealer transfers the commitment to the corresponding share to that player using the CTP.

Figure 2: *The commitment sharing protocol.*

The actively secure MPC protocol is outlined in Figure 3.

3.3.3 Cryptographic Security

Given the considerations above, to obtain cryptographic security, it suffices to use some homomorphic commitment scheme that satisfies the requirements given in Section 3.3.2. This results in (a modification of) the protocol by [GMW87]. If the scheme is of type H (e.g., Pedersen Commitments), then the *secrecy* of the resulting protocol holds unconditionally. Conversely, if the scheme is of type B (e.g., ElGamal Commitments), then the *correctness* of the resulting protocol is unconditional. For further discussion, in particular for the construction of a CMP from a normal homomorphic commitment scheme, see Exercise 9.

3.3.4 Information-Theoretic Security

In this section we provide a homomorphic, information-theoretically secure commitment scheme along with commitment multiplication (CMP) and transfer (CTP) protocols. The way around the fact that there can be no commitment scheme that is both information-theoretically binding and hiding is to construct a *distributed* scheme that is based on Shamir's sharing scheme. Instatiating our generic protocol with this commitment scheme results (in a variant) of the protocol by [BGW88].

In the following we assume that the number of corrupted parties is $t < n/3$. We start by constructing protocols COMMIT and OPEN, which allow some dealer D to commit to a value and

Actively Secure MPC

Whenever some other player refuses to execute any of the steps below, proceed as described in Section 3.3.1 (sending wrong values).

Input

Each player gives input by committing to it and then executing the CSP to commit all other players to their share of the input.

Addition Gates (and Linear Functions)

Two values are added by having each player commit to the sum of his two shares using the homomorphic property of the commitment scheme (i.e., without communication).

Multiplication Gates

To multiply two values, each player first uses the CMP to commit to the product of his shares. Then he uses to the CSP to share this value. Finally, the players compute the product using Lagrange interpolation (as in the passive protocol), which is a linear function and can be evaluated using the homomorphic property.

Output

To reconstruct a value towards some player, all other players open their commitment to the share of that value to that player. Since there are at least $t + 1$ honest players, the player receives enough shares.

Figure 3: *Actively secure MPC protocol tolerating at most $t < n/2$ corrupted players (or less if the commitment scheme requires a lower threshold).*

to open it, respectively. Recall that an execution of them can be rejected by the players. We require that they satisfy the following properties:

- Consistency: If Commit or Open is rejected by some honest player, all honest players do so. If D is honest, no honest player rejects.
- Uniqueness: If Commit is successful, D is committed to some value, i.e., there exists only one value that is accepted in Open.
- Privacy: If D is honest an commits to some value, then, in Commit, the adversary obtains no information about this value.

It is easily seen that these properties imply that the scheme is perfectly binding and hiding.

Commit. The main idea is that the dealer D, to commit to some value s, chooses a random polynomial $g(x)$ of degree at most t with $g(0) = s$ and sends the *commit share* $s_i := g(\alpha_i)$ to P_i. Furthermore, D somehow "proves" that the commit shares indeed lie on a polynomial of degree at most t. The protocol is given in Figure 4.

Open. Assume the dealer D is committed to some value s by a polynomial $g(x)$ and each player holds the share $s_i = g(\alpha_i)$. To open the commitment, D first broadcasts $g(x)$ (by broadcasting the coefficients). Each player P_i then checks if his share lies on the broadcasted polynomial and accuses the dealer via broadcast if this is not the case. If at most t players accuse D, the opening is accepted. The protocol is depicted in Figure 5.

Let us turn to the analysis of the commitment scheme. It is straight-forward to verify that the presented commitment scheme satisfies the consistency property. Also, it is easily seen that the presented commitment scheme maintains privacy: At the beginning, the adversary gets up to t pairs of polynomials $(h_i(x), k_i(y))$, which give no information about the secret (more details on this in the lecture). The reader can easily verify that, after this, the only values are revealed that the adversary already knows.

Distributed Commit Protocol COMMIT

1. Distribution: To share secret s, dealer D chooses a random bivariate polynomial of (single) degree at most t

$$f(x,y) = \sum_{i,j=0}^{t} f_{ij}\, x^i y^j, \quad \text{where } f_{00} = s \text{ and } f_{ij} \in_R \mathrm{GF}(q) \text{ for } i,j \neq 0$$

   and sends the polynomials $h_i(x) := f(x, \alpha_i)$ and $k_i(y) := f(\alpha_i, y)$ to P_i for $i = 1, \ldots, n$.

2. Consistency Checks: For $1 \leq i, j \leq n$: Players (P_i, P_j) verify that $h_i(\alpha_j) = k_j(\alpha_i)$. To that end, P_i sends the value $h_i(\alpha_j)$ to P_j, who compares the received value to $k_j(\alpha_i)$. Each player P_i broadcasts a complaint for all coordinates (i, j) where the values do not match. Dealer D must answers such complaints by broadcasting $f(\alpha_i, \alpha_j)$.

3. Accusations: If, in Step 2, dealer D broadcasts values that are not consistent with polynomials $h_i(x)$ and $k_i(y)$ of some player P_i, this player accueses the dealer via broadcast. The dealer must answer such an accusation by broadcasting both $h_i(x)$ and $k_i(y)$. If the broadcasted polynomials are inconsistent with the polynomials of some other player P_j (i.e., $h_i(\alpha_j) \neq k_j(\alpha_i)$ or $k_i(\alpha_j) \neq h_j(\alpha_i)$), this player also accuses the dealer. This continues until no new players accuse D.

4. Determine Commit Share: If dealer D, in Step 3, was accused by more than t players, refused to answer any of the accusations, or the answers contradicted each other, he is disqualified.

   Otherwise, each player computes his commit share $s_i = k_i(0)$ as the output of the protocol, where, if P_i accused D, he uses the polynomial broadcast by D. The dealer outputs the polynomial $g(x) = f(x, 0)$.

Figure 4: *Distributed commit protocol secure against up to $t < n/3$ corrupted players.*

To show that the uniqueness property is satisfied, suppose some dealer D is not disqualified. After Step 3, the polynomials of all honest players are pairwise consistent. Consider the polynomial $f'(x, y)$ defined by the polynomials $h_i(x), k_i(y)$ of the $t + 1$ honest players with the lowest indices. Moreover, consider the polynomial $h_j(x)$ of some other honest player P_j. It is consistent with the polynomials $k_i(y)$ of the $t + 1$ first players. Thus $h_j(x) = f'(x, \alpha_j)$. Similarly, $k_j(y) = f'(\alpha_j, y)$. It follows that the polynomials of all honest players lie on a well-defined polynomial $f'(x, y)$ of degree at most t. In particular, the shares $k_i(0)$ lie on a polynomial $g(x) = f'(x, 0)$ of degree at most t and the secret is uniquely defined.

Suppose now that a corrupted dealer broadcasts a polynomial $g'(x) \neq g(x)$ in the open protocol. Since they must disagree in at least $n - t$ positions α_i, there will be at least $n - 2t \geq t + 1$ honest players accusing the dealer, who will thus be disqualified.

Commitment Transfer Protocol. Such a protocol will be discussed in Exercise 10.

Commitment Multiplication Protocol. As a final step, we will construct a commitment multiplication protocol that allows a player committed to two values a, b to commit to their product $c = ab$ (see Figure 6). The protocol is generic, that is, it works with any commitment scheme (also cryptographic ones). It requires, however, that $t < n/3$. We defer its analysis to Exercise 10.

Distributed Open Protocol OPEN

Starting point: Dealer D is committed to some value s by a polynomial $g(x)$ of degree at most t and each player P_i holds the commit share $s_i = g(\alpha_i)$.
Goal: Every player learns s.

1. Dealer D broadcasts $g(x)$.
2. Every player P_i checks that his share lies on the broadcasted polynomial, i.e., whether $s_i = g'(\alpha_i)$. If not, P_i accuses D via broadcast.
3. If more than t players accused D the protocol is rejected. Otherwise the opened value is $s = g'(0)$.

Figure 5: *Distributed open protocol secure against up to $t < n/3$ corrupted players.*

Commitment Multiplication Protocol CMP

Starting point: Dealer D is committed to two values a and b.
Goal: D is committed to the product $c = ab$ of the two values.

1. D computes $c = ab$ and commits to it.
2. D uses the CSP for both for a and b, which results in every player P_i being committed to two shares a_i and b_i.
3. D uses the CSP for c. However, he uses a polynomial of degree $2t$ that is the product of the two polynomials used to share a and b. P_i is now committed to c_i.
4. Each player P_i checks whether $c_i = a_i b_i$. If not, he opens the commitments to the three values, thus proving that D has cheated. If no player provides such a proof, the commitment to c from Step 1 is accepted.

Figure 6: *Commitment multiplication protocol secure against up to $t < n/3$ corrupted players.*

4 Broadcast

This section presents Broadcast and Consensus. The former is a primitive that allows a certain player, called sender, to distribute a value to all players with the guarantee that all honest players receive the *same* value in the end. Moreover, if the sender is honest, then the players agree on the value sent by the sender. In Consensus, every player holds an input and the goal is that, in the end, the honest players agree on a value while preserving so-called *pre-agreement*.

More precisely, a *broadcast* protocol satisfies the following conditions:

- CONSISTENCY: All honest players output the same value, i.e., there is *agreement* at the end of the protocol.
- VALIDITY: If the sender is honest, the value the honest players decide on is the value sent by him.
- TERMINATION: All honest players decide on a value at some point.

Finally, a protocol achieves *consensus* if the follwing conditions are met:

- CONSISTENCY: All honest players output the same value, i.e., there is *agreement* at the end of the protocol.
- PERSISTENCY: If all honest players have the same input, they agree on this value.
- TERMINATION: All honest players decide on a value at some point.

Note that, if $t < n/2$, broadcast and consensus are equivalent in the sense that a broadcast protocol can easily be transformed into a consensus protocol and vice-versa:

Broadcast $\Rightarrow$ Consensus: Each player broadcasts his value and decides on the value received most often.

Consensus ⇒ Broadcast: The sender sends the value to be broadcast to all players. Then, the players run consensus, where each player inputs the value received from the sender.

In the following we present the broadcast protocol by [BGP89] which provides information-theoretic security provided that less than a third of the players are corrupted, i.e., $t < n/3$. We start out with a protocol that achieves a weak form of consensus and then build from it a full consensus protocol. Using the above construction, this can be transformed into a broadcast protocol.

Note that we construct a binary broadcast protocol, i.e., a protocol with input space $\{0,1\}$. A protocol with a larger input space can, e.g., be achieved by parallel exection of the one-bit protocol.

4.1 Weak Consensus

In Weak Consensus, each player has an input x_i and eventually decides on a value $y_i \in \{0,1,\perp\}$, where $\perp$ is to be considered "invalid". Weak Consensus achieves a weaker consistency condition:

Weak Consistency: If some player P_i decides on an output $y_i \in \{0,1\}$, then $y_j \in \{y_i, \perp\}$ for all honest players P_j.

In other words, no two honest players decide on two different "valid" outputs. A protocol achieves Weak Consensus if it satisfies weak consistency, persistency, and termination. The following is such a protocol:

Protocol `WeakConsensus` $(x_1, \ldots, x_n) \to (y_1, \ldots, y_n)$**:**

1. $\forall P_i$: send x_i to each P_j
2. $\forall P_j$: $y_j = \begin{cases} 0 & \text{if number of received zeros} \geq n-t \\ 1 & \text{if number of received ones} \geq n-t \\ \perp & \text{otherwise} \end{cases}$
3. $\forall P_j$: return y_j

Lemma 4.1. *If $t < n/3$, protocol* `WeakConsensus` *achieves persistency, weak consistency, and termination.*

Proof. Persistency: If all honest players input the same value x, each honest player receives x at least $n-t$ times (and at most $t < n-t$ times the value $1-x$) and thus decides on x.

Weak Consistency: Suppose an honest player P_i decides on y_i. Then, he received the value y_i at least $n-t$ times in Step 1. Since at least $n-2t$ of these messages are from honest players, it follows that every honest player has received y_i at least $n-2t$ times and thus $1-y_i$ at most $2t < n-t$ times. Hence, no honest player P_j decides $y_j = 1-y_i$.

Termination: Obvious. □

4.2 Graded Consensus

Each player starts out with an input $x_i \in \{0,1\}$ and eventually decides on a value $y_i \in \{0,1\}$ and on a grade $g_i \in \{0,1\}$. The grade $g_i = 0$ is to be consiedered as "consistency achieved", whereas $g_i = 0$ means "not sure that consistency is achieved".

We introduce the following requirements:

Graded Consistency: If some player P_i decides on an output $y_i \in \{0,1\}$ with grade $g_i = 1$, then $y_j = y_i$ for all honest players P_j.[2]

Graded Persistency: If all honest players P_i have the same input x, they all decide on the output $(y_i, g_i) = (x, 1)$.

A protocol achieves Graded Consensus if it satisfies graded consistency, graded persistency, and termination. Such a protocol can be constructed as follows:

[2]But not necessarily $g_j = 1$.

Protocol $\texttt{GradedConsensus}\,(x_1, \ldots, x_n) \to ((y_1, g_1), \ldots, (y_n, g_n))$**:**

1. $(z_1, \ldots, z_n) = \texttt{WeakConsensus}(x_1, \ldots, x_n)$
2. $\forall P_i$: send z_i to each P_j.
3. $\forall P_j$

$$y_j = \begin{cases} 0 & \text{if \#zeros} \geq \text{\#ones} \\ 1 & \text{if \#zeros} < \text{\#ones} \end{cases}$$

$$g_j = \begin{cases} 1 & \text{if } \#y_j\text{'s} \geq n - t \\ 0 & \text{otherwise} \end{cases}$$

4. $\forall P_j$: return (y_j, g_j)

Lemma 4.2. *If $t < n/3$, protocol* `GradedConsensus` *achieves graded persistency, graded consistency, and termination.*

Proof. Graded Persistency: Assume all honest players input the same value x. The persistency of `WeakConsensus` guarantees that all honest players send x in Step 2. Thus, each honest player P_i receives x at least $n - t$ times (and $1 - x$ at most t times) and thus decides on $y_i = x$ und $g_i = 1$.

Weak Consistency: Suppose honest player P_i outputs $g_i = 1$. Then, he has received y_i from at least $n - t$ players in Step 2. Since at least $n - 2t$ of these players are honest, it follows that every honest player P_j has recieved y_j at least $n - 2t > t$ times in Step 2. Moreover, since at least $t + 1 \geq 1$ honest players held y_i after `WeakConsensus`, no honest player held $1 - y_i$. Hence every honest player gets the value $1 - y_i$ at most t times and thus decides on y_i in Step 3.

Termination: Obvious. □

4.3 King Consensus

In King Consensus some player P_k takes over the role of the king. If the king is honest, we require that the protocol achieve consensus. If the king is corrupted, we want that at least pre-agreement be preserved.

Thus, we introduce the following new requirement:

King Consistency: If the king P_k is honest, all players decide on the same value in the end.

A protocol achieves King Consensus if it satisfies king consistency, persistency, and termination. Such a protocol can be constructed as follows:

Protocol $\texttt{KingConsensus}_{P_k}\,(x_1, \ldots, x_n) \to (y_1, \ldots, y_n)$**:**

1. $((z_1, g_1), \ldots, (z_n, g_n)) = \texttt{GradedConsensus}(x_1, \ldots, x_n)$
2. P_k: send z_k to each player P_j.
3. $\forall P_j$: $\quad y_j = \begin{cases} z_j & \text{if } g_j = 1 \\ z_k & \text{otherwise} \end{cases}$
4. $\forall P_j$: return y_j

Lemma 4.3. *If $t < n/3$, protocol* `KingConsensus` *achieves persistency, king consistency, and termination.*

Proof. Persistency: If all honest players P_i start the protocol with the same input x, then graded persistency implies that after Step 1, they hold $(z_i, g_i) = (x, 1)$ and decide $y_i = z_i = x$ in Step 3.

King Consistency: Suppose the king is honest. If all honest players P_i have $g_i = 0$ in Step 1, then they all take king's value. Otherwise, let P_j be a player with $g_j = 1$. Graded consistency implies that in such a case all honest players P_i have $z_i = z_j$. In particular, this holds for the (honest) king P_k. Thus, all players decide on the same output.

Termination: Obvious. □

4.4 Consensus

Consensus can be achieved from King Consensus by running `KingConsensus` with $t+1$ different kings P_k, which guarantees that at least one of them is honest.

Protocol $\texttt{Consensus}\,(x_1, \ldots, x_n) \to (y_1, \ldots, y_n)$**:**

1. for $k := 1$ to $t+1$ do
   $(x_1, \ldots, x_n) := \texttt{KingConsensus}_{P_k}\,(x_1, \ldots, x_n)$
   od
2. $\forall P_j$: return $y_j = x_j$

Lemma 4.4. *Protocol* `Consensus` *achieves Consensus if at most* $t < n/3$ *players are corrupted.*

Proof. Persistency: Assume there is pre-agreement at the beginning of the protocol. Then, the consistency of King Consensus (Lemma 4.3) immediately implies that it is preserved until the end.

Consistency: At least one of the kings $P_k \in \{P_1, \ldots, P_{t+1}\}$ is honest. Thus, after the execution of $\texttt{KingConsensus}_{P_k}$, there is agreement among the honest parties, which is maintained until the end of the protocol due to the persistency condition.

Termination: Obvious. □

The efficiency of the protocol is discussed in Exercise 11.

References

[BGP89] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Towards optimal distributed consensus (extended abstract). In *FOCS*, pages 410–415. IEEE, 1989.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10. ACM, 1988.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.

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