Bonn-Aachen International Center for Information Technology
Cryptography and Game Theory — Raekow, Ziegler
IPEC Winter 2012

Scribe(s): Yona Raekow, Nevena Tacheva — Lecture 5, Date: 15.03.2012

Protocol for MPC for n players

In this lecture we discuss the following:

1. Some definitions
2. Protocol
3. Proof sketch

1 Preliminaries

1.1 Setting

We consider $n \geq 3$ players and we make the following assumptions:

1. There are secure channels between all players.
2. There is a subset C of players that are corrupted. The players behave semi-honest, meaning they follow the execution of the protocol as specified, but get together afterwards and pool their inputs and the outputs they received from the protocol.

1.2 Goal

We want to specify a protocol for n players, s.t. they can compute a function that is specified by an arithmetic circuit together with an execution order, that specifies the order in which the gates are executed. We want to make sure that nothing is revealed about the private inputs of the players and that the players learn only the outcome of the function evaluation, players should only learn their private result.

1.3 Notation

We write $[a; f]_d$, where $a \in \mathbb{F}$ is a polynomial over $\mathbb{F}[x]$ with $f(0) = a$ and $deg(f) = d$. So $[a; f]_d = (f(1), .., f(n))$ are the shares of a secret a computed using the polynomial f of degree d. With this notation we can define the following operations. Let f, g, be polynomials over $\mathbb{F}[x]$ and $a, b, \alpha \in \mathbb{F}$.

1. **Addition**: $[a; f]_d + [b; g]_d = (f(1) + g(1), .., f(n) + g(n)) = [a + b; f + g]_d$, i.e. adding shares of the secrets a and b in shares of $a + b$. Where the secret a was shared using f and b was shared using g, and the shares of the secret $a + b$ were computed from the polynomial $f + g$.

2. **Multiply by constant:** $\alpha \cdot [a; f]_d = (\alpha \cdot f(1), .., \alpha \cdot f(n)) = [\alpha \cdot a; \alpha \cdot f]_d$, i.e. multiplying a share of the secret a with a constant α results in a share of the secret $\alpha \cdot a$, where a was shared using f and $\alpha \cdot a$ is shared using $\alpha \cdot f$.

3. **Multiplication:** $[a; f]_d \cdot [b; g]_d = (fg(1), .., fg(n)) = [a \cdot b; f \cdot g]_{2d}$, i.e. given a share of the secret a and a share of the secret b, that were shared using the polynomials f and g respectively, multiplying a share of a and a share of b results in a share of the secret ab computed from the polynomial fg. Note that fg has degree $2d$.

We need the following statements in the protocol:

1. P_i **distributes** $[a; f_a]_d$: P_i choses f_a at random s.t. the degree of $(f_a) = d$, $f_a(0) = a$ and send shares $f_a(j)$ to P_j for $j = 1, .., n$.

2. Players **hold** $[a; f_a]_d$: Players have obtained the shares of a secret a that were computed using the polynomial f_a.

3. Players **compute** $[a; f_a]_d + [b; f_b]_d$: Suppose that the players have shares of the secrets a and b. Each player P_i computes $f_a(i) + f_b(i)$. So after that players hold $[a + b, f_a + f_b]_d$.

4. Players **compute** $\alpha \cdot [a; f_a]_d$: Suppose that the players have shares of the secrete a Each player P_i computes $\alpha \cdot f_a(i)$. So after that players hold $[\alpha \cdot a, \alpha \cdot f_a]_d$.

5. Players **compute** $[a; f_a]_d \cdot [b; f_b]_d$: Suppose that the players hold $[a; f_a]_d$ and $[b; f_b]_d$. Each player P_i computes $f_a(i) \cdot f_b(i)$. So after that players hold $[a \cdot b, f_a \cdot f_b]_{2d}$.

2 Circuit evaluation protocol with passive security (CEPS):

In this section we present a protocol for n players that can securely compute a function, given by an arithmetic circuit, even if t players are corrupted. The protocol together with detailed proofs is described in [Cramer et al., 2011].

1. Input sharing: Each player P_i holding input $x_i \in \mathbb{F}$ distributes $[x_i, f_{x_i}]_t$.

2. Computation phase: Repeat the following until all gates in the execution order have been processed. Pick the first gate that has not been processed yet. Depending on the gate do one the following:

   - Addition gate: The players *hold* $[a; f_a]_t$, $[b; f_b]_t$ for the two inputs to this gate. The players *compute* $[a; f_a]_t + [b; f_b]_t = [a + b; f_a + f_b]_t$.
   - Multiply by constant gate: The players *hold* $[a, f_a]_t$, where a is the input to this gate. The players *compute* $\alpha[a, f_a]_t = [\alpha a, \alpha f_a]_t$.
   - Multiplication gate:
     (a) The players *hold* $[a; f_a]_t$, $[b; f_b]_t$ for the two inputs to this gate. The players *compute* $[a; f_a]_t \cdot [b; f_b]_t = [ab; f_a f_b]_t$.
     (b) Let $h = f_a f_b$, then $h(0) = ab$. The players *hold* $[ab, f_a f_b]_{2t}$, i.e. they know $h(i)$. Each P_i *distributes* $[h(i), f_i]_t$.
     (c) We note that the degree of $h = 2t \leq n - 1$. Let $\mathbf{r} = (r_1, \ldots, r_n)$ be the recombination vector [1], that we saw in the last lecture. We know that $h(0) = \sum_{i=1}^{n} r_i h(i)$ for any

[1] $r_i = \prod_{i \neq j} \frac{-j}{i-j}$, so the r_i do not depend on h, they work for any polynomial of degree $\leq n - 1$.

polynomial h of degree $\leq n-1$. So the players can compute a share of the value ab that was distributed using a polynomial of degree only t in the following way:

$$\sum_i r_i[h(i); f_i]_t = [\sum_i r_i h(i); \sum_i r_i f_i]_t = [h(0); \sum_i r_i f_i]_t = [ab; \sum_i r_i f_i]_t.$$

3. Output reconstruction: After all gates – including the output gates have been processed, all players do the following: For each output gate y_i the players *hold* $[y_i; f_{y_i}]_t$, where y_i is the value assigned to the ith output gate. Each P_j sends f_{y_i} to P_i, who can use Lagrange interpolation to recover his output y_i. Note that $y_i = f_{y_i}(0)$.

3 Analysis of CEPS

We make the following remarks:

1. After the input phase all input gates have been processed. I.e. the input gates have specific values. Think of it in the following way: Each player holds locally a copy of the circuit. After the input phase he receives shares from all the players. The share he receives from player P_i is the input to the ith input gate of his local copy of the circuit.

2. We maintain the following invariant: Computing the circuit on $x_1 \dots x_n$ assigns a unique value to every wire in the circuit. Let $a \in \mathbb{F}$ a value assigned to some wire in the circuit during the computation. This wire can either be an input wire to a gate, or an output wire (or both). After the gate that is connected to this wire is processed, all players *hold* $[a, f_a]_t$, for some polynomial f_a.

We will show in the following that the protocol provides

1. Perfect correctness with probability 1. All players receive outputs that are correct based on the inputs that were supplied.

2. Perfect privacy: Any subset C of corrupt (semi-honest) players with $1 \leq |C| \leq t$, where $t < \frac{n}{2}$ learns no information beyond $\{x_i, y_i\}_{P_i \in C}$ from executing the protocol, regardless of the computational power.

3.1 Correctness

If the invariant is maintained then correctness holds.

3.2 Privacy

Observe that corrupted players receive two types of messages from the honest players.

1. In the input sharing an multiplication they receive $[x_i, f_{x_i}]_t$ or $[h(i), f_i]_t$, respectively.

2. In the output reconstruction they receive all shares $[y_i, f_{y_i}]_t$ for each output y_j which corresponds to a $P_j \in C$.

Privacy then follows from two observations:

Observation 1 All values sent by honest players to corrupted players in the input phase or when a multiplication gate is executed are shared using polynomials that were selected at random and that have degree t. Since there are at most t corrupted parties, they can pool at most t shares of any of these polynomials. Therefore, their shares look like an values that were selected at random.

Observation 2 Note that the values $[y_i, f_{y_i}]_t$ could have been computed by the corrupted parties themselves. Before the output reconstruction the parties know $f_{y_j}(j)$ for each $j \in C$. But they also know that $f_{y_j}(0) = y_j$. So they know the points of all players on f_{y_j}, i.e. they can compute $f_{y_j}(i)$ for all $P_i \notin C$.

References

[Cramer et al., 2011] Cramer, R., Damgrd, I., and Nielsen, J. B. (2011). Secure multiparty computation. Book Draft.