ETH Zurich, Department of Computer Science
SS 2018

Prof. Ueli Maurer
Dr. Martin Hirt
Chen-Da Liu Zhang

Cryptographic Protocols
Notes 6

Scribe: Sandro Coretti (modified by Chen-Da Liu Zhang)

About the notes: These notes serve as written reference for the topics not covered by the papers that are handed out during the lecture. The material contained therein is thus a *strict* subset of what is relevant for the final exam.

This week, the notes contain an introduction to secure multi-party computation (MPC) and treat oblivious transfer, which is an important primitive in the area of MPC.

6.1 Secure Multi-Party Computation

The task of *secure multi-party computation (MPC)* is to enable n mutually distrusting parties $P_1, \ldots, P_n$ to compute some function on their inputs without revealing any information about them that could not already be inferred from the output of the function. 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.

6.1.1 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 parties can use to send the *same* message to all other parties. Under certain assumptions, such channels can be realized.

6.1.2 Security Requirements

Players participating in an MPC can be dishonest. This is modeled by a *central* adversary 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 consequently 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.

6.2 Oblivious Transfer

One of the most fundamental MPC primitives is *oblivious transfer (OT)*. An OT protocol is executed between two mutually distrusting parties Alice and Bob. It comes in several variants:

- *Rabin OT:* Alice transmits a bit b to Bob, who receives b with probability $1/2$ while Alice does not know which is the case. That is, the output of Bob is either b or $\perp$ (indicating that the bit was not received).
- 1-*out-of*-2 *OT:* Alice holds two bits b_0 and b_1. For a bit $c \in \{0, 1\}$ of Bob's choice, he can learn b_c but not b_{1-c}, and Alice does not learn c.
- 1-*out-of-k OT for* $k > 2$*:* Alice holds k bits $b_1, \ldots, b_k$. For $c \in \{1, \ldots, k\}$ of Bob's choice, he can learn b_c but none of the others, and Alice does not learn c.

All variants also exist as *string OT*, i.e., the values transmitted are not bits but bitstrings.

1-out-of-k OT can be used by Alice and Bob to securely evaluate an arbitrary function $g : \mathcal{X} \times \mathcal{Y} \to \Omega$, where $\mathcal{X}$ is Alice's input domain, $\mathcal{Y}$ is Bob's input domain with $|\mathcal{Y}| = k$, and Ω is the output domain.

Let $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ be the inputs of Alice and Bob, respectively. The protocol for securely computing $g(x, y)$ works as follows: Alice sends the function table of $g(x, \cdot)$ to Bob using 1-out-of-k string OT, and Bob chooses to receive the entry in the position corresponding to y, which equals $g(x, y)$. Bob then sends this value back to Alice.

The above protocol is secure against a passive adversary.