COM S 6830 – Cryptography Nov 17, 2009

Lecture 22: Secure Computation

Instructor: Rafael Pass *Scribe: Stefano Ermon*

1 Recap on security primitives

So far we have seen a number of cryptographic primitives:

- PRG
- PRF
- Private key encryption
- Signatures
- Zero Knowledge proofs for NP
- OWF

This group of primitives are equivalent, in the sense that the existence of one of them implies the existence of the others (assuming NP is hard on average for ZK proofs).
We believe that collision resistant hash functions (CRH) are a stronger primitive, in the sense that it is unlikely that they can be constructed from OWF. On the other hand it is possible to construct a OWF from a CRH f.
The intuition of the construction is that a CRH f (that without loss of generality can be taken as strongly compressing) is itself a OWF. In fact if an adversary $\mathcal{A}$ is able to invert $f(x)$, with high probability $\mathcal{A}(f(x)) \neq x$ because by the compressing property $f^{-1}(f(x))$ is large, thus resulting in a collision.

Another set of cryptographic primitives include:

- Public Encryption
- OT
- Secure Computation

These primitives are believed to be stronger than the other ones described so far, but no theoretical result is known (in either direction). Their existence is based on assumptions such as Factoring, DDH or are Lattice-Based. While OWF and CRH seem to be reasonable existing objects, these assumptions are more difficult to accept.

2 Definition of Secure Computation

We consider m parties, each one with a private input x_i, that wish to compute a function $F(x_1, \ldots, x_m) = (f_1(\vec{x}), \ldots, f_m(\vec{x}))$ that depends on the vector of all inputs $\vec{x} = (x_1, \ldots, x_m)$. F is vector valued because we assume that each party i computes only the i-th component $f_i(\vec{x})$, so that it could be the case that each party gets a different result.
We would like to have protocols to perform this task with the following properties:

- Correctness
- Privacy

Intuitively privacy means that players do not learn other people's inputs. Correctness means that players get the right results and that the inputs are chosen independently, in the sense that people cannot decide their inputs as a function of inputs from other players they might have seen.

2.1 Definitions

A first approach to the definition of privacy inspired by ZK could be imposing that whatever a party sees in the execution of the protocols, he could just have generated it by himself with a simulator. This approach cannot work because a protocol for a multiparty computation cannot be Zero Knowledge, because at the end one gets the output of the function and therefore learns something (even when the protocol relies on a trusted party).
To formally define the privacy requirement, we notice that the most ideal setting we can expect is one where a honest trusted party T_F collects inputs from all the parties, computes $F(\vec{x})$ and distributes the result. Therefore on a high level we say that a protocol π securely computes F if the execution of π emulates the ideal execution (with a trusted party).
Consider $m(n)$ players, each one with an input $x_i \in \{0,1\}^n$, that wish to compute a function $F(x_1, \ldots, x_{m(n)})$ using a protocol π.

Definition 1 (Security) *A protocol π securely computes F if for all subsets of players $Z \subset [m(n)]$, for all PPT adversaries $\mathcal{A}$ controlling players Z, there exists a PPT $\tilde{\mathcal{A}}$ controlling the same subset of players such that for all nuPPT $\mathcal{D}$ there exists a negligible function ϵ such that for all n, for all inputs $\vec{x} \in (\{0,1\}^n)^{m(n)}$ and for all z, $\mathcal{D}$ distinguishes with probability at most $\epsilon(n)$*

$$REAL(\mathcal{A}(z), \vec{x})$$
$$IDEAL(\tilde{\mathcal{A}}(z), \vec{x})$$

where $REAL(\mathcal{A}(z), \vec{x})$ is defined as the output of all players when π is run on input $\vec{x}$ with $\mathcal{A}(z)$ controlling the subset of players. Similarly $IDEAL(\tilde{\mathcal{A}}, \vec{x})$ is defined as the

output of all players in the ideal situation when they compute F interacting with a trusted party, but where $\tilde{\mathcal{A}}$ controls the players Z.

This definition clearly captures the correctness property, because in the ideal world the adversary cannot affect in any way the honest parties. More precisely the controlled players can only affect the honest players by deciding their own inputs, but this can clearly not be prevented in any way. This definition also captures the privacy concept, because it implies that the view of all players in the two worlds are indistinguishable. This means that the views of the cheating players are the same in both worlds, and therefore they do not learn anything (as in the ideal case).
Notice that the case $Z = \emptyset$ is particularly interesting because it reflects the fact that F is correctly computed when all the parties are honest.

2.2 Secure Computations

Some common notions of secure computations are $(m-1)$-security, where the adversary $\mathcal{A}$ is allowed to control up to $m-1$ players, and $(m/2)$-security where $\mathcal{A}$ can control less than $m/2$ players (*Honest Majority*).
Unfortunately $(m-1)$-security is not achievable for many functions of interest. In many cases in fact there is a problem of fairness, where one party has to be the first one to send something useful (e.g. when two persons want to exchange goods electronically). For example the XOR function cannot be computed both fairly and securely.
Sometimes protocols are defined so that they are secure but not fair: in this case the adversary can also decide who gets the output from the trusted party in the ideal world. The drawback of honest majority is that it does not apply to two-party computations. In general it is possible to prove the following result:

Theorem 1 *Let $F : (\{0,1\}^n)^{m(n)} \to (\{0,1\}^n)^{m(n)}$ be an efficiently computable function. Given $t < m(n)/2$, if there exists a trapdoor permutation over $\{0,1\}^n$, then there exists an efficient protocol π to compute F that is t-secure.*

Thanks to this theorem, it is also possible to emulate the action of a trusted party (stateful and multi-round). The idea is that a suitable function F is used to update a state and generate a new shared state that is distributed to the parties.

2.3 Examples

- The *Set Intersection* protocol is used to compute the intersection $X \cap Y = F(X,Y)$ of two sets X,Y.
- In 1/4-OT, a player A has 4 bits $\vec{a} = (a_1, a_2, a_3, a_4)$, while B's input is $b \in \{1,2,3,4\}$. The function they wish to compute is $F(\vec{a}, b) = (\perp, a_b)$.

Theorem 2 *Assume the existence of a trapdoor permutation over* $\{0,1\}^n$*. Then there exists a* 2*-party protocol that securely computes* $1/4$*-*OT *with respect t a honest-but-curious adversary.*

The honest-but-curious adversary setting is similar to the honest zero knowledge concept, in the sense that the adversary follows the protocol but at the end tries to learn as much as possible on the other input. This notion is important because we can force players to act honestly by requiring ZK-proofs of their correct behavior at each round. Moreover there are many practical situations where a trusted hardware platform enforces that the protocols are respected.