Lecture 20 — Homomorphic Encryption 2: Two party secure computation.

Boaz Barak

April 14, 2010

Multi-party secure computation We've considered a great many cryptographic applications in this course, encryption, signatures, coin tossing, commitments, zero knowledge, private information retrieval,... and there are more such as electronic elections, electronic auctions, etc.. that we didn't consider.

Today we consider one framework that captures all of them.

Cryptography in the presence of a completely trusted party We observe that all the cryptographic problems we considered become trivial if there is a completely trusted party $\mathcal{F}$ that has a secure private channel to every one of the participants:

Coin tossing The participants ask $\mathcal{F}$ to toss a coin, $\mathcal{F}$ tosses it and broadcasts the result.

Authenticated Encryption Alice sends a message x to $\mathcal{F}$ and asks to relay it to Bob and no one else, $\mathcal{F}$ relays x to Bob and tells him this message was received from Alice.

Zero knowledge To prove that she knows x such that $C(x) = 1$, Alice sends C, x to $\mathcal{F}$. $\mathcal{F}$ verifies that $C(x) = 1$, then sends only C to Bob that it can guarantee that there is x such that $C(x) = 1$.

And here are some we didn't consider:

Electronic voting Everyone sends their votes to $\mathcal{F}$, that announces the winner.

Electronic auctions Everyone sends their bids to $\mathcal{F}$, that announces who was the highest bidder, and what was the value of the second highest bid.

Poker $\mathcal{F}$ chooses a random permutation of the 52 cards, sends each party their cards, parties send their choices to $\mathcal{F}$ that announces the public information and sends to individual parties their secret cards, etc..

Yao's Miliionaire's problem Alice and Bob want to compare who has a higher salary. Alice sends her salary x to $\mathcal{F}$, and Bob sends his salary y to $\mathcal{F}$, $\mathcal{F}$ announces whether $x > y$ or not.

(In fact, we allow here for the possibility of Alice and/or Bob to cheat in the number they provide $\mathcal{F}$, but at least they will not learn more than this one bit about the other person's salary, one can also think of a functionality that gets as input a public key of say the IRS, and checks that x and y are signed with this key.)

Distributed signature and decryption Three parties each has strings s_1, s_2, s_3 and a public message x, they all send their strings to $\mathcal{F}$ that broadcasts a signature on x using the key $s_1 \oplus s_2 \oplus s_3$. One can similarly have $\mathcal{F}$ decrypt a ciphertext using the a secret key obtained by the XOR of the inputs.

Virtual trusted party The notion of *secure multi-party computation* is to allow k parties to create a virtual trusted party out of thin air. The basic setting is assuming that these parties have identities and can talk privately and securely to one another (e.g., there is a public key infrastructure) though this has been generalized to weaker notions as well. There are many variants of the definition and the one we'll use is the following. Below for simplicity we assume that $\mathcal{F}$ is a deterministic stateless function: that is $\mathcal{F}$ takes input x_i from the i^{th} party, and after receiving all inputs, sends the output y_j to the j^{th} party, where $(y_1, .., y_k) = \mathcal{F}(x_1, ..., x_k)$. We'll remark later how to generalize this for randomized and stateful functionalities.

Let $\mathcal{F} : (\{0,1\}^n)^k \rightarrow (\{0,1\}^n)^k$ be some function. A k-party protocol is a *secure function evaluation* protocol for $\mathcal{F}$ if for every subset $S \subsetneq [k]$ and coordinated cheating strategy A^* for the parties in S, there exists a simulator SIM and a set of inputs $\{x_i\}_{i \in S}$ such that:

Correctness For every set of inputs $\{x_i\}_{i \in \overline{S}}$, if the parties in $\overline{S}$ follow the protocol, then we have a guarantee that for every $i \in \overline{S}$, the output the i^{th} party obtains is either $f(x_1 ... x_k)_i$ or $\perp$. Here $\overline{S}$ denotes $[k] \setminus S$.

Simulation For every set of inputs $\{x_i\}_{i \in \overline{S}}$, if SIM obtains as output $\{f(x_1 ... x_k)_i\}_{i \in S}$ and a parameter $\epsilon > 0$ then SIM runs in poly$(1/\epsilon, n, k)$-time and the output of SIM is computationally indistinguishable from the view of all the parties in S in an interaction where for every i, if $i \in \overline{S}$ then the i^{th} party follows the protocol and uses x_i as input, and if $i \in S$ then the i^{th} party follows the coordinated strategy A^*.

(It is possible to combine the two conditions together in one requirement, though we will not follow this route.)

The validity requirement ensures that the cheating parties have no control over the output the honest parties receive beyond their obvious power to choose their own inputs. The ability to cause the output to be $\perp$ comes from the fact that it's always possible for an attacker to halt the protocol and stop communicating. In fact, it is possible in certain settings (in particular when $|S| < k/2$) to get rid of that possibility, but we will not go into these variants.

The simulation requirement ensures that the cheating parties do not learn anything about the honest parties' inputs beyond what $\mathcal{F}$ tells them in the "ideal" setting.

Honest but curious simulation Just like zero knowledge, there is a simpler, but still interesting variant where we don't assume that A^* deviates from the protocol, but rather just want to make sure the parties in S don't learn anything about the inputs of the other parties aside from what $\mathcal{F}$ provides. We'll focus on this variant today.

Obtaining randomized or stateful functionalities To obtain the coin tossing functionality, consider $\mathcal{F}(x_1, x_2) = x_1 \oplus x_2$. If one party chooses its bit at random then it's guaranteed that the outcome is random (or $\perp$), no matter what the other party does. This can be generalized to any randomized functionality, by having additional inputs $r_1, \ldots, r_k$ to all the parties, where the honest parties choose them at random, and having $\mathcal{F}$ use $r_1 \oplus \cdots \oplus r_k$ for its random tape.

For stateful functionalities, $\mathcal{F}$ can broadcast after each step its state in an encrypted and authenticated form, and require this state to be provided in the next step (aborting if all the parties didn't provide $\mathcal{F}$ the same state). Similar to the above, the secret key used can be generated as $r_1 \oplus \cdots \oplus r_k$ where r_i is chosen at random by the i^{th} party.

We see that multi-party computation can basically solve every cryptographic task possible. Thus the following theorem is pretty amazing:

Fundamental Theorem of Cryptography (Yao 82, Goldreich-Micali-Wigderson 87) Assuming trapdoor permutations exist, for every polynomial-size $\mathcal{F} : (\{0,1\}^n)^k \to (\{0,1\}^n)^k$, there exists a secure multiparty protocol for $\mathcal{F}$.

What's left Did this theorem finish all there is to be done in cryptography? Below are a few reasons why cryptography is still an active research area...

Efficiency: the biggest caveat with this theorem was that the protocol is quite inefficient, requiring running a fairly complex mini-protocol for each gate of the circuit for $\mathcal{F}$. Indeed, there has been a lot of research on obtaining efficient variants of this theorem for specific functionalities that arise in practice (we'll talk about voting protocols, see link on website on Danish sugar beets auction). Indeed, this is a common theme in cryptography— first people come up with a polynomial time but fairly impractical solution for a problem, and only much later more efficient solutions are discovered. But here the very generality of the problem seem to hinder truly efficient solutions (though there have been significant improvements in this regard).

We remark that the homomorphic encryption based protocols we'll see will have an important efficiency advantage over the original protocol, in the sense that we will not have communication proportional to the number of gates in C. As I mentioned, these were only discovered only last year.

Interaction, infrastructure: in real life we sometimes want to restrict the interaction to be of a certain form, such as a single message from sender to receiver, as in encryption. Indeed, there are variants of this theorem with reduced interaction requirements. Also, the assumption of private secure channels between the parties is not always justified, since not every party has public keys, though again there are by now variants of this theorem with reduced assumptions on authentication.

Composition, rationality, leakage: While the security requirements of multi-party protocol seem quite impressive, they do suffer from an important problem which is that they do not guarantee security when multiple instances of this or other protocols are run simultaneously. (As I mentioned in the context of zero knowledge, sometimes in crypto $0 + 0 \neq 0$.) Obtaining versions of this theorem with such security guarantees has been a very active area of research since the late 1990's. Other security concerns that are actively researched include the following: (1) we assume that honest parties follow the protocol, but in real life we should assume that even non-hacked parties will only follow the protocol if it's in their interest to do so, thus several works try to combine multi-party secure computation with the rationality framework from game theory (2) we assume that the adversary has either complete control of a machine or no control at all, but what if he's able to get some partial information (via side channels such as radiation, timing, power consumption, sound etc..) even on the secret inputs of the honest parties? new works try to handle such issues as well.

Computational assumptions: It is of course a basic question of cryptography to understand what assumptions are necessary for such a result.

Proof of the theorem We'll prove this theorem using homomorphic encryption instead. We start with the case $k = 2$ and honest-but-curious adversaries (this is the case originally shown by Yao, whereas GMW extended this to more parties, and more importantly, malicious adversaries). We have two parties, Alice and Bob with inputs x, y respectively and they wish to compute $\mathcal{F}(x, y)$ where $\mathcal{F}$ is some Boolean circuit. We can make the following simplifying assumptions: (1) only Alice learns an output in the protocol (after that they can run another iteration in which Bob will learn his output) and (2) Bob "hardwires" his inputs into the circuit. So we are in the setting where Alice knows an input x, Bob knows a circuit C, and their goal is for Alice to learn $C(x)$ and Bob to learn nothing at all.

Protocol Basic-SFE:

Alice's input $x \in \{0,1\}^n$

Bob's input Circuit $C : \{0,1\}^n \to \{0,1\}$

Step 1: Alice generates $(e, d) \leftarrow_{\mathrm{R}} G(1^n)$ keys for the homomorphic encryption, and sends e to Bob.

Step 2: Alice sends $\hat{x} = E_e(x_1) \cdots E_e(x_n)$ to Bob.

Step 3: Bob computes $\hat{c} = EVAL_e(C, \hat{x})$ and sends $\hat{c}$ to Alice.

Step 4: Alice decrypts $\hat{c}$ to obtain her output.

We have the following theorem: **Theorem:** Basic-SFE is an honest-but-curious secure two party computation protocol.

(Note that there are many chances for Alice and Bob to cheat if they deviate from the protocol, but this is not part of the notion of honest-but-curious security.)

The GMW compiler The protocol we presented above strongly depends on Alice and Bob not deviating from the protocol.[1] So it might seem useless as a starting point for a full fledges multi-party computation protocol. It turns out this is not the case:

Theorem (Goldreich-Micali-Wigderson): Assuming one-way functions exist, there is a general transformation converting any multi-party computation protocol for $\mathcal{F}$ that is honest-but-curious secure into a full-fledged secure protocol.

We illustrate this theorem by showing how this compiler works for our Basic-SFE protocol.

From 2 parties to k parties Let's first discuss how we'll go from 2 parties to 3 parties, and then continue. The compiler works in the same way for every k, and so we can focus on the honest but curious case. We basically split Alice into two pieces, Alice 1 and Alice 2 that have inputs x^1, x^2 and the goal is that Alice 1 learns $C(x^1, x^2)$ where C is the circuit Bob has. Alice 1 and Alice 2 run a two party secure protocol to jointly generate a key pair (e, d) for the homomorphic encryption such that Alice 1 gets d_1 and Alice 2 gets d_2 s.t. d_1, d_2 are random subject to $d_1 \oplus d_2 = d$. Then each of them encrypts and sends her input to Bob, who applies EVAL on both inputs, and sends it to them. They run again a two party secure protocol to decrypt the output and have Alice 1 of them learn the decryption.

[1] As a very advanced comment we note that there are protocol that depend on this property even in a stronger way, requiring the parties to even generate their own randomness completely uniformly.