COM S 6830 – Cryptography	Nov 24, 2009

Lecture 24: Secure Computation

Instructor: Rafael Pass *Scribe: Matt Weinberg*

1 Two Party Protocols

1.1 definitions

In this lecture we look at two party protocols for secretly computing a function on two inputs. In other words, Alice has a private value, a, Bob has a private value b, and they would both like to compute $f(a, b)$, without revealing any unnecessary information. In this lecture, we only consider honest-but-curious adversaries.

Definition 1 *Let A be the algorithm intended for use by Alice in an interactive protocol, and let B be the algorithm intended for use by Bob. Then we say that an interactive protocol is honest-but-curious-secure if $\exists\ S_A, S_B, \forall\ a, b$:*

$$\{View_A(A(a), B(b), 1^n)\} \sim \{S_A(a, f(a, b), 1^n)\}$$
$$\{View_B(A(a), B(b), 1^n)\} \sim \{S_B(b, f(a, b), 1^n)\}$$

Where the $View_A$ consists of $a, f(a, b)$, and the transcript of the protocol. In words, this means that anything A learns from the protocol, A could have figured out using only a and $f(a, b)$.

1.2 Garbled Circuits

Now we describe a protocol for securely computing $f(a, b)$. Because f is poly-time computable, we can write a circuit that computes f. For every wire, w, in the circuit, attach two keys, k_0^w and k_1^w. Intuitively, we use the first key when w carries the value 0, and the second when w carries the value 1. Have A compute this circuit, and then hard-code a to make the circuit only take b as input. Then A will send this circuit over to B and B will evaluate it using the appropriate keys for his input. This is just the intuition, there are several details that need to be addressed.

First, how do we deal with gates? We need to somehow take two input keys and get the correct output key. The idea is to have each gate store a table of size 4. The correct output wire will be encrypted under the two input keys sequentially. For example, say we had an AND gate with input wires x and y, and output wire z. Then the gate would store the following 4 elements in its table:

1) $Enc_{k_0^x}(Enc_{k_0^y}(k_0^z))$
2) $Enc_{k_0^x}(Enc_{k_1^y}(k_0^z))$
3) $Enc_{k_1^x}(Enc_{k_0^y}(k_0^z))$

4) $Enc_{k_1^x}(Enc_{k_1^y}(k_1^z))$
There is a slight problem with this scheme. If the table is always ordered in the same way, then when Bob figures out which entry he correctly decrypted, there will be no point to having keys, because he'll know exactly what the two input wires were. So this table must be permuted.
Now we run into a slightly different problem, how does Bob know whether he successfully decrypted an entry? Every entry in the table can be decrypted under any pair of keys, so how will Bob know the difference between decrypting under the right keys and the wrong ones? To solve this problem, we can add an extra column to the table which contains a MAC of the proper key-pair. At the beginning of the protocol, choose a single key that will be used to authenticate the key-pairs, and create a MAC for each key-pair to put in the table. Now, when Bob wants to know which entry in the table he's supposed to decrypt, he just tries all four MACs to verify his key-pair. If exactly one of them is correct, then he knows exactly which entry in the table he should decrypt. If we are in the unlucky case that more than one are correct with non-negligible probability, then we have an algorithm that can forge a MAC for this key-pair.
Assume for contradiction that with non-negligible probability, there existed, in some table, two key-pairs, k, k' such that $Tag(k')$ was a valid authentication for k. Then we now have an algorithm that can forge a MAC for k. Just use the same efficient algorithm that is used to generate all the key pairs, and ask the signing oracle to authenticate k'. Then with non-negligible probability, we are in the unlucky situation that $Tag(k')$ is a valid authentication for k, which contradicts the security of the MAC scheme. So it must be the case that we are only in this unlucky situation with negligible probability, and we may assume that there is a unique MAC that authenticates Bob's key-pair.
So now we have a protocol that, for every gate, allows Bob to correctly compute the key for the output wire using only the keys for the input wires, without Bob learning what the input wires were.
There is one final problem. Bob needs to know the key values for all of his input wires, but Alice cannot know which key values Bob is asking for. Similarly, Alice cannot send all the key values to Bob, or else (because he is curious) he will evalute the circuit on more values than intended, and learn something about Alice's input. To solve this problem, we just need to use 1/2 Oblivious Transfer. Alice stores k_0^w and k_1^w, and Bob asks for k_i^w where i is his input to wire w. Now we can describe the entire protocol:
1) Alice write a garbled circuit to compute f, and hardcodes her input (resulting in a new garbled circuit that depends only on input from Bob). Then, for each remaining wire, Alice assigns a key pair k_0^w and k_1^w. To each logic gate, she assigns a table defined as above. Each table entry consists of the correct key encrypted sequentially, followed by a MAC to authenticate that the two correct keys were used to decrypt. Each MAC uses the same key. Alice sends over the entire garbled circuit to Bob.
2) Bob requests the correct keys for his input using 1/2-Oblivious Transfer
3) Alice sends the correct keys for Bob's input using 1/2-Oblivious Transfer
4) Bob evaluates the garbled circuit using the keys he received from Alice and shares the

value with Alice

2 Non-Malleability

An intuitive definition of non-malleability is that given an encryption of m, no nuPPT can find an encryption of $f(m)$. Formally, we define the following experiment:
Let $\pi = (gen, enc, dec)$, $NM^{\pi}(m, A)$ is the following experiment.
1) $k \leftarrow gen(1^{|m|})$
2) $c \leftarrow Enc_k(m)$
3) $c'_1, \ldots, c'_l \leftarrow A(c, 1^{|m|})$
4) If $c'_i = c$, let $m'_i = \perp$. Otherwise let $m'_i = Dec_k(c'_i)$
5) Output $m'_1, \ldots, m'_l$

Definition 2 *π is NM if $\forall$ nuPPT $A, \forall\ m_0, m_1$, $\{NM^{\pi}(m_0, A)\} \sim \{NM^{\pi}(m_1, A)\}$.*

Definition 3 *π is $rel-NM$ if $\forall$ PPT $A, \exists$ PPT S, $\forall$nuPPT relations R, $\exists\ \epsilon, \forall n, \forall\ m \in \{0,1\}^n, \forall\ z$. ϵ is negligible and*

$$|Pr[NM^{\pi}(A(z), m) \in R(m)] - Pr[S(1^n) \in R(m)]| \leq \epsilon(n)$$

Proposition 1 *π is $rel - NM \Leftrightarrow \pi$ is NM.*

Proof. Say π is NM. The NM experiment can be simulated on any m by a PPT. So just let S be the simulator that runs the NM experiment on $A(z), 0$. Then because π is NM, we must have $|Pr[NM^{\pi}(A(z), m) \in R(m)] - Pr[S(1^n) \in R(m)]|$ negligible, or else we cuold distinguish $NM^{\pi}(A(z), m)$ from $NM^{\pi}(A(z), 0)$.
Say π is $rel-NM$, then there exists S for every relation R such that $|Pr[NM^{\pi}(A(z), m) \in R(m)] - Pr[S(1^n) \in R(m)]|$ is negligible. Now this also means that for any m_0,m_1, for every R, $|Pr[NM^{\pi}(A(z), m_0) \in R(m_0)] - |Pr[NM^{\pi}(A(z), m_1) \in R(m_0)]$ is negligible. Now we have that π is NM for experiments where $l = 1$. To see this, assume for contradiction that this were not the case. Then because $\{NM^{\pi}(A, m_0)\} \not\sim \{NM^{\pi}(A, m_1)\}$, it must be the case that there is some subset of $\{NM^{\pi}(A, m_0)\}$ such that the probability of $NM^{\pi}(A, m_1)$ landing in this subset is "distinguishable" from the probability that $NM^{\pi}(A, m_0)$ does. So let $R(m_0)$ be this subset, then we have a contradiction. To generalize this to any NM experiment, we can just use the hybrid lemma.
Finally, if π is CCA-2 secure, then π is NM. This is because running the NM experiment is a CCA-2 attack (IE: it can be executed with oracle access to decryption of non-challenge cipher-text). So if π were not NM, then we would have a CCA-2 attack to distinguish $E(m_0)$ and $E(m_1)$ for whichever pair breaks NM.

2.1 A non-malleable encryption scheme

$Gen(1^n)$: For $i \in [n]$, $b \in \{0,1\}$, use gen to generate pk_b^i, sk_b^i.
$Enc(m)$: Pick $vk, sk \leftarrow gen^sig(1^n)$ (sig is some secure signature scheme). Send over $Enc_{pk^i_{vk_i}}(m)$, for all $i \in [n]$, then sign the entire message with sk. Also send vk.
$Dec(m)$: verify using vk that the signature is valid, then decrypt each message individually.
Here's an intuition as to why this scheme is non-malleable. Say an attacker could, given $Enc(m)$, generate $Enc(f(m))$. There are two possibilities. Either the attacker keeps the same vk, which means he has found a way to sign a new message under the same key, breaking the security of the signature scheme. Or maybe the attacker changes the vk. But in this case, the attacker has to find $Enc(f(m))$ for at least one key for which he hasnt even seen $Enc(m)$ for. Because the keys are randomly generated, this attack could be used to break semantic security (just generate random keys on your own, and get an encryption of $f(m)$. Now because you generated the keys on your own, you can decrypt to find $f(m)$).
There is a slight change to the scheme that needs to happen for everything to work. Maybe each individual encryption scheme is malleable (like el-gamal for instance). Then as long as the vk the attacker chooses to use shares at least 1 bit with the original vk, then the attacker can generate a value that is either an encryption of $f(m)$, or an invalid encryption indistinguishable from the original encryption (because maybe now it's the case that not every individual encryption is of the same message $f(m)$). To fix this, add to the encryption scheme a Non-Interactive Zero-Knowledge proof that each individual encryption is of the same message. Now an attacker cannot get away with generating an invalid encryption.