CSE 5854: Class 10

Benjamin Fuller

February 19, 2018

1 Moving to secrecy

Zero knowledge proofs (including proofs of knowledge and non-interactive proofs) will be crucial tools in ensuring honest behavior of participants in interactive protocols. What we're going to turn to now is how to perform computation of any type in secret. We'll start with a basic situation with two parties P_1 and P_2 where P_1 holds a circuit gate with two inputs (OR, AND, XOR) and P_2 holds two input bits. Note that we can view the gate as P_1's input or we can consider a fixed gate and have P_1 provide one input to the protocol. For the moment we are only going to consider parties who follow the protocol but try and learn information they are not entitled to. The first thing that P_1 is going to do is write the truth table for the circuit as follows:

Input 1	Input 2	Output
0	0	b_{00}
0	1	b_{01}
1	0	b_{10}
1	1	b_{11}

The idea behind the scheme is for P_1 to encrypt each row of this table with keys that correspond to the inputs of P_2. P_2 will then ask for keys according to its input. So the encrypted table is developed as follows:

Input 1	Input 2	Output
r_{10}	r_{20}	b_{00}
r_{10}	r_{21}	b_{01}
r_{11}	r_{20}	b_{10}
r_{11}	r_{21}	b_{11}

This table is then turned into encryptions:

$\mathsf{Enc}_{r_{10}}(\mathsf{Enc}_{r_{20}}(b_{00}))$
$\mathsf{Enc}_{r_{10}}(\mathsf{Enc}_{r_{21}}(b_{01}))$
$\mathsf{Enc}_{r_{11}}(\mathsf{Enc}_{r_{20}}(b_{10}))$
$\mathsf{Enc}_{r_{11}}(\mathsf{Enc}_{r_{21}}(b_{11}))$

These four encryptions are delivered to P_2 (in a random order). Now for P_2 to evaluate this computation it suffices for them to get the correct r_{1*} and r_{2*} according to their input. Then P_2 will just try and decrypt each of the four values they are given (note we require it is possible to detect "bad" decryptions). We have two privacy properties we are concerned about:

1. P_2 should only learn information about a single label for each input.

2. P_1 should not learn which input P_2 is evaluating the gate on.

A protocol does this is called *oblivious transfer.* P_2 gets to pick one value without seeing the other and P_1 learns nothing about which value was picked by P_1.

2 Oblivious Transfer

In oblivious transfer we call the two parties the sender S and receiver R. We think of an interactive protocol between the two parties.

Definition 1. *A pair of interactive Turing machines (S, R) is an oblivious transfer with error δ if the following hold:*

1. ***Correctness:*** *For all messages $m_0, m_1 \in \{0,1\}^\ell$ and $b \in \{0,1\}$,*

$$\Pr[\langle S(m_0, m_1), R(b)\rangle_1 = m_b] = 1.$$

That is, R gets the message m_b with probability 1.

2. ***Receiver privacy:*** *The receivers bit is hidden. That is, for all PPT S^*,*

$$\langle S^*(\cdot), R(0)\rangle_1 \approx_{c,\delta} \langle S^*(\cdot), R(1)\rangle_1.$$

3. ***Sender privacy:*** *The not received message is hidden from the sender. That is, for all pairs m_0, m_0', m_1, m_1' for all R^* one the following holds:*

$$\langle S(m_0, m_1), R^*(\cdot)\rangle_2 \approx_{c,\delta} \langle S(m_0, m_1'), R^*(\cdot)\rangle_2,$$

or

$$\langle S(m_0, m_1), R^*(\cdot)\rangle_2 \approx_{c,\delta} \langle S(m_0', m_1), R^*(\cdot)\rangle_2.$$

Note: Our definitions are getting more and more complicated as we move on. This is the last definition that we'll explicitly state what can't be learned or done by the parties. From here on we'll state the intended behavior of the protocol and say that learning anything other than the intended behavior constitutes a breach of security. This will allow us to simplify exposition of security definitions considerably.

We will occasionally consider the easier task when we assume one or both of the parties can be trusted to not deviate from the protocol. This is called *semi-honest sender* or *semi-honest receiver* oblivious transfer. In this setting we simply remove the universal quantifier and consider an S^* or R^* that outputs their entire view (and actually has inputs that they use in the protocol). As you'll see on the homework it is possible to construct a semi-honest receiver oblivious transfer using a public-key encryption scheme. Roughly, the sender has their bit and generates two public keys one where they know the secret key and one where they don't. These are then sent (in the appropriate order) to the receiver who encrypts both messages. The idea is that the receiver can only decrypt one.

The hope would be that we can easily transform this protocol into a maliciously secure protocol using a zero-knowledge proof of knowledge.

Discussion Question: Why can't we apply a ZK PoK to this protocol to get malicious security?

2.1 A DDH based protocol

We start by introducing a sender semi-honest protocol using DDH and El Gamal encryption. We then show how to convert this protocol into malicious security using the properties of DDH. As usual let G be a group of prime order p with generator g.

S

1. Generate $c \xleftarrow{\$} G$ and send to R.

R

2. Receive c, pick $k \xleftarrow{\$} \mathbb{Z}_p$, set $pk_b = g^k$ and $pk_{1-b} = c \cdot g^{-k}$. Send pk_0, pk_1.

3. Check that $c = pk_0 \cdot pk_1$. If not abort.

4. Encrypt m_0, m_1 using El Gamal encryption with public keys pk_0, pk_1 respectively.

5. Receive c_0, c_1, decrypt c_b.

The idea behind the sender's first message is to keep R from choosing two public keys for which they know the secret key. Because the two public keys must be offset by $\log(c)$ this notionally would require them to solve the discrete logarithm problem. However, this does not keep them from choosing the two public keys in such a way that they learn some information about both message x_0, x_1 without being able to completely decrypt either message.

The goal of the revised protocol is to ensure that R learns partial information if and only if they knew the exponents of the items they sent (this protocol is due to Naor and Pinkas [NP99]). (We can think of this as a proof of non-knowledge.) To do this we will randomize exponents in such a way that if R knows the exponents it does not hurt and otherwise it destroys all structure and keeps R from learning anything. In doing so we are also able to let R send the first message instead of S.

R

1. Generate c, d compute $g^c = x, g^d = y$ set $pk_b = g^{cd}$ and $pk_{1-b} = g^z$ where z is random. Send pk_0, pk_1.

S

2. Receive x, y, pk_0, pk_1. Verify that $pk_0 \neq pk_1$. Check that $c = pk_0 \cdot pk_1$. Randomize $(y_0', pk_0') \leftarrow Rand(g, x, y, pk_0)$ and $(y_1', pk_1' \leftarrow Rand(g, x, y, pk_1)$.

3. Set $c_0 = (y_0, x_0 \cdot pk_0')$ and $c_1 = (y_1', x_1 \cdot pk_1')$.

4. Send c_0, c_1.

5. Receive c_0, c_1, decrypt c_b.

As stated above the idea behind this protocol is to random pk_b in a way that preserves decryption but randomize pk_{1-b} in a way that destroys decryption (information-theoretically). The trick is to do this "obliviously" without knowing which is which. The key to the protocol is that only one of the public keys can be a DDH triple that is g^x, g^y, g^z where $xy = z$. Rand works as follows:

1. Input g, g^x, g^y, g^z.
2. Pick random $a, b \leftarrow \mathbb{Z}_p^*$
3. Output $g^{y'} = g^{(y+b)a} = (g^y)^a \cdot g^{ab}$,
   $g^{z'} = g^{(z+bx)a} = (g^z)^a \cdot (g^x)^{ba}$.

Note if $z = xy$ then the new triple is of the form $x, (y + b)a, (xy + bx)a = x(y + b)a$. Furthermore, given the new y value and the ciphertext it is possible to recover $g^{z'}$ if x is known. That is, when Rand is applied on pk_b it preserves the ability to decrypt based on the secret key.

The second part we want to show is that when run on pk_{1-b} this destroys any partial information. To show this we show that if $z \neq xy$ (which must be true for at least one of public keys) then there is a unique a, b that produce the y', z' based on the inputs (that is, this is a random triple of exponents). This is because $(y+b)a = y'$ and $(z+bx)a = z'$ are a linear system with two unknowns. In particular, $a = (z' - y'x)/(z - xy)$ which is solvable if and only if $z \neq xy$.

Discussion Question: Why is this protocol still secure against a malicious sender? What can the sender do to disrupt the protocol?

References

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 245–254. ACM, 1999.