CSE 5854: Class 02

Benjamin Fuller

January 17, 2018

1 Building Commitments

In the previous set of notes we defined two properties for a commitment scheme: hiding and binding. In this class we'll turn to constructing commitments and talk about strengthening the definition.

We start by a natural question, can a commitment scheme be built out of an encryption scheme? Certainly, the hiding property is exactly the definition of a secure public key encryption scheme, namely that the encryption of any two messages is computationally indistinguishable.

We'll now consider the binding property. For the moment, we will assume that the pair of public and secret key (pk, sk) is held by the receiver R. (This doesn't make sense as this would violate the hiding property, we'll start from here.) Then the open protocol would simply be to decrypt the message. In order for the encryption scheme to be binding we need that the encryption scheme *always* decrypts properly. That is,

$$\Pr[\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = 1] = 1.$$

Such a property prevents there from being two secret keys sk_1, sk_2 such that for some message m $\mathsf{Dec}_{sk_1}(\mathsf{Enc}_{pk}(m)) \neq \mathsf{Dec}_{sk_2}(\mathsf{Enc}_{pk}(m))$.

Discussion Question 1: Why does perfect correctness ensure that two private keys will decrypt the same?

We now consider the question of who stores the secret key sk? It seems important for the receiver to have the public key pk. For opening it seems important for the sender to have sk. However, if the sender generates sk then the receiver has to worry about the public key being honestly generated. That is, we can guarantee binding for public keys generating properly, but what we can say for arbitrary public keys? For example, if we are using a Diffie-Hellman type construction we need to consider whether they have actually sent a generator of the group. For the moment, we will ignore this problem and assume that the public key is output by a trusted one-time process. Let $(G, \mathsf{Enc}, \mathsf{Dec})$ be a public key encryption scheme. Then we can have the following scheme:

1. Let $\mathit{Setup}(1^k)$ output $ck = pk$ where $(pk, sk) \leftarrow G(1^k)$.
2. Let $(c, (m; r)) \leftarrow \mathit{Commit}(m; r)$ where r is chosen at random and $c = \mathsf{Enc}_{pk}(m; r)$.
3. Let $\tilde{m} \leftarrow \mathit{Open}(c, (m; r))$ where $\tilde{m} = m$ if $c = \mathsf{Enc}_{pk}(m; r)$ and $\tilde{m} = \perp$ otherwise.

Note here we are making the randomness used in algorithms explicit and it appears following a semicolon in the inputs to the algorithm. Note that the secret key and the decryption algorithm are never used. However, the fact that the encryption scheme is committing is crucial in proving security here.

Discussion Question 2: How would this scheme work with El Gamal encryption?

1.1 Commitment using a pseudorandom generator

We'll now define a commitment scheme that only relies on a pseudorandom generator. Consider some $G : \{0,1\}^k \to \{0,1\}^{3k}$ that is a PRG. Define a commitment scheme $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ as follows:

1. $r \leftarrow \mathsf{Setup}(1^k)$ where $r \leftarrow \{0,1\}^{3k}$.
2. $(c, (s, b)) \leftarrow \mathsf{Commit}(b)$, where $s \leftarrow \{0,1\}^k$ and $c = G(s) \oplus r$ if $b = 1$ and $c = G(s)$ otherwise.
3. $\tilde{m} \leftarrow \mathsf{Open}(c, (s, b))$, where $\tilde{m} = b$ if $c = G(s)$ output $b = 0$ and if $c = G(s) \oplus r$ set $b = 1$.

Theorem 1. *If G is a pseudorandom generator then the commitment scheme above is computationally hiding and statistically binding.*

Proof. We first consider hiding. It suffices to show for any fixed $r \in \{0,1\}^{3k}$ the distributions $G(S)$ and $G(S) \oplus r$ are computationally indistinguishable. We have the following:

$$G(S) \approx_c U_{3k} \overset{d}{=} U_{3k} \oplus r \approx_c G(S) \oplus r.$$

Where the first and third equalities proceed by security of the pseudorandom generator and the second inequality proceeds because a random string offset by any fixed value is distributed the same as a random string.

We now consider binding. The only way for a malicious sender to cheat in the opening phase is to send a string y such that there exists $s_1, s_2 \in \{0,1\}^k$ such that $G(s_1) = y = G(s_2) \oplus r$. We now ask how frequently this is possible.[1] We assume the adversary can cheat if they can find s_1, s_2 such that $z = G(s_1) \oplus G(s_2)$. Since there are at most 2^k values for each term in this sum there are at most 2^{2k} values for z. There are 2^{3k} values for r so a randomly chosen r there will exist such s_1, s_2 with probability at most 2^{-k}. Thus with overwhelming probability a malicious sender cannot find such s_1, s_2 (because they don't exist). Thus, with overwhelming probability the scheme is binding. $\square$

1.2 Pedersen Commitment

In this subsection we consider a second commitment scheme based on the hardness of discrete logarithm (our first example was adapting the El Gamal encryption scheme to be a commitment). This scheme appears similar to an encryption scheme but it has very different properties. This scheme is due to Pedersen [Ped91].

Define the scheme $C = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ as follows:

[1]This argument does not consider the computational power of the malicious sender.

1. $(p, g, y) \leftarrow \mathsf{Setup}(1^k)$ where p is a prime, y is a randomly chosen element of $\mathbb{Z}_p^*$ and g is a randomly chosen generator of $\mathbb{Z}_p^*$.
2. $(c, (r, b)) \leftarrow \mathsf{Commit}(b)$ where $r \overset{\$}{\leftarrow} \mathbb{Z}_p^*$ and $c = g^r y^b \mod p$.
3. $\tilde{m} \leftarrow \mathsf{Open}(c, (r, b))$, where $\tilde{m} = b$ if $c = g^r y^b$ and $\tilde{m} = \perp$ otherwise.

This type of commitment is different from what we've seen before. It is information-theoretically hiding that is, even an all powerful malicious receiver can't learn anything about the bit b. However, it is (only) computationally hiding under the discrete log assumption. On the first homework you'll be asked to prove security of this scheme.

2 Composing Commitment

Previously, we saw that it is possible to build a multi bit encryption scheme by individually encrypting each message. This does not work for signature schemes as the attacker can reorder the message and change the meaning. Because we do not consider an attacker receiver (yet) composition works for commitment schemes.

Lemma 1. *If $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ is a secure commitment scheme for $\{0, 1\}$, then $\mathcal{C}'$ obtained from $\mathcal{C}$ from bit by bit composition for any polynomial $p(k)$ times is a secure commitment scheme for $\{0, 1\}^{p(k)}$. In particular, any commitment scheme can be used to commit and open multiple messages.*

Proof Sketch. The proof that $\mathcal{C}'$ satisfies hiding is the same as for encryption, we use the hybrid argument and the fact that a malicious receiver can run Commit on any values they wish. The binding property is also a natural hybrid argument, if the sender can find d_0, d_1 such that $m_0 = \mathsf{Open}(c, d_0), m_1 = \mathsf{Open}(c, d_1)$ then the messages m_0, m_1 must differ in some bit and we can use the part of the opening string as a break for the single bit decommitment scheme. □

3 Interactive Proofs

We now are going to change directions in what seems like a significant way, we will consider interactive proofs. Consider some language $\mathcal{L}$. In the computer science realm, a proof is a way of demonstrating if $x \in \mathcal{L}$. Take the setting of NP then we know that $x \in \mathcal{L}$ if and only if there exists a computable relation $R(\cdot, \cdot)$ such that $\exists w$ where $R(x, w) = 1$ if and only if $x \in \mathcal{L}$.

The most basic interactive proof between a prover and a verifier is as follows:

1. Assume both parties agree on the language $\mathcal{L}$, the relation R, and the statement x.
2. Further assume that the prover knows some w such that $R(x, w) = 1$.
3. The prover can "prove" to the sender by simply transmitting the value w.

This type of proof is elementary, we are simply writing down a proof that the verifier is able to then use to check membership. There is nothing special about the fact that P and V are interacting. In this class we'll see two types of proofs: proofs were the verifier is not able to check membership of the statement on their own and proofs where the prover convinces the verifier that something is true without revealing the entire truth of the statement (i.e. the entire witness).

Like commitments, we want two security properties from interactive proof, they are soundness and completeness. Roughly, soundness says that the prover should not be able to convince the verifier if $x \notin \mathcal{L}$ and completeness says that the prover should be able to convince the verifier if $x \in \mathcal{L}$. For these definitions we need notions of interactive Turing machines as we may have many messages going back and forth. Full specification of an interactive Turing machine is out of the scope of this class [Weg97]. We will add details as we progress.

Definition 1 (Interactive Proof System)**.** *A pair of interactive machines* (P, V) *is called* an interactive proof system *for a language* L *if machine* V *is polynomial time and the following conditions hold:*

- *Completeness: For every* $x \in L$*,*
$$\Pr[< P, V > (x) = 1] \geq 2/3.$$
- *Soundness: For every* $x \notin L$ *and every interactive machine* B*,*
$$\Pr[< B, V > (x) = 1] \leq 1/3.$$

.

Here the symbols $< P, V > (x)$ correspond to the complete interaction between machines P and V when both initialized with x. We say the interaction outputs 1 if the verifier V outputs 1 at the end of the process. We will often use the term $View$ to refer to the complete set of messages that pass between P and V.

3.1 Graph Non-Isomorphism

In this section we will show an example of an interactive proof for a language that is not known to know in BPP or NP. Thus, it is not possible for the verifier to compute membership or their own or for the prover to simply send a witness. The example we will use is that of graph non-isomorphism. Define two graphs $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ where each edge is between two vertices in the graph. We say that two graphs are isomorphic if there exists an isomorphism $\pi : V_0 \to V_1$ such that for every edge $u, v \in E_0$ $\pi(u), \pi(v) \in E_1$. We say that two graphs are non-isomorphic if no such isomorphism exists. No polynomial time solution for solving graph isomorphism is known. However, with an unbounded prover it is possible to convince a verifier that two graphs are not isomorphic. For convenience we assume that the sets $V_0 = V_1$ (why is it okay to make this assumption?). The protocol works as follows:

1. The verifier picks a random $b \in \{0, 1\}$ and a random isomorphism $\pi : V_b \to V_{1-b}$. The verifier computes a graph $G_2 = (\pi(V_b), \pi(E_b))$ and sends G_2 to P.

2. The prover P receives a graph G_2. P computes 2 isomorphisms π and ν between G_0 and G_1 to G_2 respectively. If both π and ν exist, the prover picks a random b' and sends this to the verifier. If only π exists P sends $b' = 0$ if only ν exists P sends $b' = 1$ to V.

Discussion Question: How frequently is the prover going to be right when the graphs are isomorphic? When they aren't isomorphic? If a prover B decides to cheat what can they do to effect success probability?

References

[Ped91] Torben P Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Crypto*, volume 91, pages 129–140. Springer, 1991.

[Weg97] Peter Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91, 1997.