CSCI-GA.3210-001 MATH-GA.2170-001	**Introduction to Cryptography**	April 25, 2012

Lecture 14

Lecturer: Yevgeniy Dodis *Spring 2012*

This lecture is on Commitment Schemes. Informally, a commitment scheme abstracts the notion of a "locked box": the contents of the box are hidden (without the key), but can be opened in only one way. First, a formal definition of a non-interactive Commitment Scheme is given along with some explanation. Then some examples of Commitment Schemes are given: based on "committing" encryption, PRG's (thus, OWF's), OWP's, CRHF's, discrete log (Pedersen's commitment), random oracle. Compositions of Commitment schemes are considered including bit-by-bit (ala encryption) and "hash-then-commit" (ala signatures) methods. We also briefly talk about a slightly relaxed notion of commitment, which allows us to use UOWHF's in place of CRHF's, and suffices for some of the applications of commitment. Finally, several applications of Commitment Schemes are given along with a brief explanation of zero knowledge proofs.

1 Commitment Schemes

1.1 Introduction

Commitment schemes arise out of the need for parties to commit to a choice or value and later communicate that value to the other parties involved in such a way that is fair to all the parties. The main problem here is that we do not want one party to find out about any other party's commitment before the latter opens this value itself. On the other hand, we do not want a party to be able to open its commitment in multiple ways (then, there is no point in "committing" in the first place). Therefore, we want our Commitment Scheme to somewhat resemble a locked box that contains some value. This locked box is given to the parties but it does not reveal anything about the commitment contained in it until the key for the locked box is released so the parties can open it. The "digital" implementation of this locked box however introduces the possibility for the locked box to contain multiple values (i.e., have several valid "keys" that open it in different ways). Therefore we have to ensure that each locked box can only hold one value.

We only consider so called "non-interactive" schemes, where all the communication goes from the sender to the receiver.[1] We will omit the word non-interactive from most of the discussion though. A bit more formally, a *Commitment Scheme* transforms a value m into a pair (c, d) here c is the locked box and d is the key such that (1) c reveals no information about m, but (2) together (c, d) reveal m, and it is infeasible to find d' such that (c, d') reveals $m' \neq m$. This is defined formally next.

[1] This can be contrasted with a more general *protocols* when the sender and the recipient can send messages to each other in multiple rounds.

1.2 Definition

DEFINITION 1 [Commitment Scheme] A (non-interactive) Commitment Scheme (for a message space M) is a triple $(\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ such that:

(a) $\mathsf{CK} \leftarrow \mathsf{Setup}(1^k)$ generates the public commitment key.

(b) for any $m \in M$, $(c, d) \leftarrow \mathsf{Commit}_{\mathsf{CK}}(m)$ is the commitment/opening pair for m. $c = c(m)$ serves as the commitment value, and $d = d(m)$ as the opening value. We often omit mentioning the public key CK when it is clear from the context.

(c) $\mathsf{Open}_{\mathsf{CK}}(c, d) \rightarrow \tilde{m} \in M \cup \{\perp\}$, where $\perp$ is returned if c is not a valid commitment to any message. We often omit mentioning the public key CK when it is clear from the context.

(d) Correctness: for any $m \in M$, $\mathsf{Open}_{\mathsf{CK}}(\mathsf{Commit}_{\mathsf{CK}}(m)) = m$

$\diamondsuit$

Here is how a commitment scheme is used. If Bob wants to commit a value m to Alice (using the commitment key CK which we don't explicitly mention below), he first generates the pair $(c, d) \leftarrow \mathsf{Commit}(m)$, and sends c to Alice. Naturally, this is called the *commit* stage. Later, when he wants to open m, he sends d to Alice, who runs $\tilde{m} \leftarrow \mathsf{Open}(c, d)$, and accepts the value $\tilde{m}$ provided that $\tilde{m} \neq \perp$. This is called the *reveal* (or opening) stage. By correctness, $\tilde{m} = m$ if everybody is honest.

Security. As we stated informally, we want two *security* properties: (1) c gives Alice no information about m, and (d) Bob cannot open c in two different ways. The properties stated above are called *hiding* and *binding*.

1. **Hiding**. It is computationally hard for any adversary A to generate two messages $m_0, m_1 \in M$ such that A can distinguish between their corresponding locked boxes c_0, c_1. That is, $c(m)$ reveals no information about m. Formally, for any PPT $A = (A_1, A_2)$ we require:

$$\Pr\left[\, b = \tilde{b} \;\middle|\; \begin{array}{c} \mathsf{CK} \leftarrow \mathsf{Setup}(1^k),\ (m_0, m_1, \alpha) \leftarrow A_1(\mathsf{CK}),\ b \leftarrow_r \{0,1\}, \\ (c, d) \leftarrow \mathsf{Commit}_{\mathsf{CK}}(m_b),\ \tilde{b} \leftarrow A_2(c;\ \alpha) \end{array} \right] \leq \frac{1}{2} + \mathsf{negl}(k)$$

   We write $c(m_0) \approx c(m_1)$, for any (m_0, m_1) chosen by A.

2. **Binding**. It is computationally hard for the adversary A to come up with a triple (c, d, d'), referred to as a *collision*, such that (c, d) and (c, d') are valid commitments for m and m' and $m \neq m'$. Formally, for any PPT A we require:

$$\Pr\left[\, \begin{array}{l} m \neq m' \wedge \\ m, m' \neq \perp \end{array} \;\middle|\; \begin{array}{c} \mathsf{CK} \leftarrow \mathsf{Setup}(1^k),\ (c, d, d') \leftarrow A(\mathsf{CK}) \\ m \leftarrow \mathsf{Open}_{\mathsf{CK}}(c, d),\ m' \leftarrow \mathsf{Open}_{\mathsf{CK}}(c, d') \end{array} \right] \leq \mathsf{negl}(k)$$

1.3 Comments

Commitment and Encryption. The hiding property of commitments is exactly the same as for (CPA-secure) public-key encryption: namely, $c(m_0) \approx c(m_1)$, for any m_0 and m_1. The binding property also seems similar. For commitments it says that every c can be opened in at most one way. Translated to encryption, it says that any encryption can be decrypted in at most one way. The types of encryptions we studied in this class indeed satisfy this property. Indeed, in our definitions Alice — the owner of the secret key — can *always* correctly decrypt any message sent to her using her public key PK): for any m, SK and PK, $D_{SK}(E_{PK}(m)) = m$. Thus, if there was a ciphertext c and two secret keys SK_0 and SK_1 — both corresponding to PK — such that $m_0 = D_{SK_1}(c) \neq D_{SK_2}(c) = m_1$, then the sender Bob of m_0 cannot be sure that $E_{PK}(m_0)$ will not decrypt to m_1. This is because it could happen that Alice's secret key corresponding to PK is SK_1, and Bob was unlucky to generate $E_{PK}(m_0) = c$. Hence, the type of encryption we studied so far is always binding. Not surprisingly, it is called a *committing* encryption.[2] Summarizing our comparison so far, we conclude

Lemma 1 *A committing encryption implies a secure commitment scheme.*

Proof: Assume $\mathcal{E} = (G, E, D)$ that is a CPA-secure PKE. We want to define a secure commitment scheme $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$. This seems straightforward, since we can let the encryption $E_{PK}(m)$ be our commitment, and the secret key SK be a "universal" opening. There is only one minor difficulty in the this: where do we store the secret key SK? Certainly, we cannot store it as part of the commitment key (why?). But then where do we get it in order to open the commitment then? The answer is simple. We don't need the secret key: we can open c by demonstrating the randomness r used for encryption! We get the following scheme:

(i) Let $\mathsf{Setup}(1^k)$ output $\mathsf{CK} = PK$, where $(PK, SK) \leftarrow G(1^k)$.

(ii) Let $(c, (m;\ r)) \leftarrow \mathsf{Commit}(m;\ r)$, where r is chosen at random and $c = E_{PK}(m;\ r)$.

(iii) Let $\tilde{m} \leftarrow \mathsf{Open}(c, (m;\ r))$, where $\tilde{m} = m$ if $c = E_{PK}(m;\ r)$ and $\tilde{m} = \perp$ otherwise.

The fact that this is a secure commitment follows easily from the discussion above (and the fact that $\mathcal{E}$ is committing). □

If we examine the proof above, we see that the secret key was never used! This exemplifies the crucial difference between commitment and encryption: encryption requires also the ability to decrypt based on c and "universal" secret key (*independent of the message*), while commitment allows to "decrypt" with *message-dependent* "secret key" d. In particular, almost always d contains the message m in the clear. In fact, we can assume without loss of generality that $d = (m, r)$, and $\mathsf{Open}(c;\ (m, r))$ simply checks if $c = \mathsf{Commit}(m;\ r)$ (notice, the proof above followed this format), and outputs m if the check succeeds.

[2] In turns out that often people allow a more relaxed notion of encryption, where one can have a negligible probability of decryption errors. We will not talk about this further in this introductory course, but remark that in certain applications, like secure multi-party computation and electronic voting, such "non-committing" encryption is extremely useful.

This makes the design of commitment schemes much easier that that of public-key (committing) encryption. In particular, we will shortly see that the construction in Lemma 1 is an "overkill": much simpler construction exist. However, we will also see that the close similarity between commitment and encryption will make the former inherit some properties of the latter.

Commitment and Signatures. There is also a much less obvious similarity between commitments and (public-key) signatures. The bonding property of a commitment scheme in some sense implies that the commitment c "validates" m, since c cannot be open to a different value. However, this similarity is a bit far fetched. First, the "signature" $c = c(m)$ is not publicly verifiable. Namely, it requires the "signer" to release d. In fact, the hiding property even implies that c does not even reveal the message "signed"! Also, the security of signatures says that it is hard to forge a "new" signature. Here everyone can forge signature, since the commitment key is public. Instead, it only says that it is hard to forge a "signature" of a message which is equal to a "signature" of a different message. However, the fact that both signatures and commitments cannot be shorter than the message makes collision-resistant hash function very useful for both, as we shall see.

Who runs the setup algorithm $\mathsf{Setup}(1^k)$? In our definition CK is public information. However it is not clear who generates it: the sender or the receiver. The problem is that if a single party generates it, it can potentially generate it way that benefits this party. For example, the recipient Alice might generate a CK that would always allow her to see inside the locked box c and determine (partial information about) m; thus, breaking the hiding property. On the other hand, the sender Bob might generate a CK that would allow him to generate different values d that would open the locked box c in different ways; thus, breaking the binding property.

It turns out this question is non-trivial. There are several answers. For the simplest answer, we can assume it is done by a trusted third party, and then such key can be subsequently used by any pair of (possibly untrusting) players. Such commitment scheme are said to have "public parameters" (initialized by the trusted party). Of course, we often would like to avoid this assumption. For another answer, in many of commitment schemes (and most of the ones we present) it turns out that it is actually safe to let one specific party (either sender or the recipient depending on the protocol) to run the key generation, and simply announce the commitment key. For example, with committing encryption of Lemma 1 we can let the sender generate this key (but cannot let the recipient do it; why?). More generally, when the scheme is *information-theoretically binding* (i.e., the message is hidden only computationally, but in theory is embedded into c) it is often the case that *any* commitment key gives binding. Thus, the sender cannot choose a bad "binding key", and it is in his interests to choose a good "hiding key". Similarly, when the scheme is *information-theoretically hiding* (i.e., the message is independent from c, but it is computationally hard to break the binding property) it is often the case that *any* commitment key gives hiding (or it is possible to give a certificate that the key is "good hiding"). Thus, the recipient cannot choose a bad "hiding key", and it is in his interests to choose a good "binding key". In such cases we can let the recipient choose the key. Notice, however, if the recipient chooses a new key for every message, the commitment scheme becomes *interactive*. And this is the basis

for the third general answer. Namely, in some cases, the sender and the recipient choose the key (or even perform the whole commitment stage) jointly, by running some *interactive protocol*.

To summarize, the solution depends on the scheme in question. For simplicity, we assume that the key is generated correctly by a trusted third party.

Asymmetry. Finally, a commitment scheme is slightly unfair to the recipient because even though the sender commits to a value when sending c, the recipient has no way of knowing what value the sender has committed to until the sender sends d. Therefore, the sender can simply refuse to send the recipient d. It turns out, such asymmetry is inevitable in two party protocols: one party always has an advantage in a sense of aborting the protocol before the other party learned its output.[3]

2 Examples of Commitment Schemes

We already gave an example in Lemma 1 based on any committing encryption. As we mentioned, this scheme is a bit of an "overkill". We now give simpler constructions.

2.1 Commitment using PRG

2.1.1 One-bit Scheme

Given G that is a $\mathsf{PRG} : \{0,1\}^k \to \{0,1\}^{3k}$, define $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ for $M = \{0,1\}$ such that:

(i) $R \leftarrow \mathsf{Setup}(1^k)$, where $R \leftarrow^r \{0,1\}^{3k}$.

(ii) $(c,\ (s,b)) \leftarrow \mathsf{Commit}(b)$, where $s \leftarrow^r \{0,1\}^k$ and $c = G(s) \oplus (b \cdot R)$. The notation $b \cdot R$ means $0 \cdot R = 0^{3k}$, and $1 \cdot R = R$. Thus, $c = G(s)$ if $b = 0$, and $c = G(s) \oplus R$ if $b = 1$.

(iii) $\tilde{m} \leftarrow \mathsf{Open}(c,\ (s,b))$, where $\tilde{m} = b$ if $c = G(s) \oplus (b \cdot R)$, and $\tilde{m} = \perp$ otherwise.

2.1.2 Security

Hiding is achieved because $c(0) = G(s) \oplus (0 \cdot R) = G(s)$ and $c(1) = G(s) \oplus (1 \cdot R) = G(s) \oplus R \equiv R$ and $G(s) \approx R$ by definition of G being a PRG. Now the question is whether or not the scheme achieves binding. Consider two valid commitments $(c, (s_0, G))$ and $(c, (s_1, G))$ such that $\mathsf{Open}(c, (s_0, G)) = 0$ and $\mathsf{Open}(c, (s_1, G)) = 1$ then $G(s_0) = c$ and $G(s_1) \oplus R = c$. Thus $G(s_0) = G(s_1) \oplus R$ or written in a different way $G(s_0) \oplus G(s_1) = R$. Now there are at most 2^k possible values for each $G(s_1)$ and $G(s_2)$ and at most 2^{2k} possible values for $G(s_0) \oplus G(s_1)$ while there are 2^{3k} possible values for R. Thus,

$$Pr_R(\exists\ s_0, s_1 \text{ s.t. } G(s_0) \oplus G(s_1) = R) \leq \frac{2^{2k}}{2^{3k}} = \frac{1}{2^k} = \mathsf{negl}(k)$$

[3] This "advantage" can be made less and less at the expense of increasing the number of rounds, but it will not concern us.

The probability that s_0 and s_1 that would produce a collision with a random R even exist is negligible, so the scheme achieves the binding property (information-theoretically).

Notice, this is an example of a scheme, where it is safe for the recipient to generate the commitment key R.

2.1.3 More Bits

The are two ways to extend this scheme to commit to more bits. One is bit-by-bit composition (see Lemma 2), which would make the sender commit to each bit individually. It turns out we can do better by directly extending the scheme above. For concreteness, assume the message space is $M = \{0,1\}^k$ (the method easily extends to any polynomial message size). Now, consider the finite field F of cardinality 2^{5k}. The elements of this field are naturally represented as $5k$-bit string. Moreover, both the addition and the subtraction in such representation coincide with the XOR operation $\oplus$ (because the field has characteristic 2). Let $\cdot$ now denote multiplication, and interpret every string $m \in M = \{0,1\}^k$ as $0^{4k} \circ m \in \{0,1\}^{5k}$, so we can view m as a member of F. Now, we directly extend our scheme:

(i) $R \leftarrow \mathsf{Setup}(1^k)$, where $R \leftarrow^r \{0,1\}^{5k}$.

(ii) $(c,\ (s,m)) \leftarrow \mathsf{Commit}(m)$, where $s \leftarrow^r \{0,1\}^k$, $c = G(s) \oplus (m \cdot R)$ and addition and multiplication are done in F.

(iii) $\tilde{m} \leftarrow \mathsf{Open}(c,\ (s,m))$, where $\tilde{m} = m$ if $c = G(s) \oplus (m \cdot R)$, and $\tilde{m} = \perp$ otherwise.

The hiding is proven as before, since $G(s)$ plus any fixed string looks pseudorandom. As for binding, in order for $G(s_0) \oplus (m_0 \cdot R) = G(s_1) \oplus (m_1 \cdot R)$, where $m_0 \neq m_1$, we must have

$$R = (G(s_0) \oplus G(s_1)) \cdot (m_0 \oplus m_1)^{-1}$$

where the inverse is taken in our field F. There are at most 2^{4k} values for the quantity $(G(s_0) \oplus G(s_1)) \cdot (m_0 \oplus m_1)^{-1}$, so a random $R \in \{0,1\}^{5k}$ can be of the above form with probability at most $2^{4k}/2^{5k} = 2^{-k} = \mathsf{negl}(k)$, as before.

2.2 One-bit Commitment using OWP

2.2.1 The Scheme

Given f that is a OWP and h that is a hardcore bit for f, define $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ for $M = \{0,1\}$ as follows:

(i) $\mathsf{Setup}(1^k)$ outputs the descryption of f and h (in case those are chosen from a family of OWP's).

(ii) $(c,x) \leftarrow \mathsf{Commit}(b)$ where $x \leftarrow^r \{0,1\}^k$ subject to $h(x) = b$, and $c = f(x)$.

(iii) $\tilde{m} \leftarrow \mathsf{Open}(c,x)$, where $\tilde{m} = h(x)$ if $c = f(x)$ and $\tilde{m} = \perp$ otherwise.

In other words, we use the value $f(x)$ to commit to its hardcore bit $h(x)$.

2.2.2 Security

Hiding is achieved because determining b from $c(b)$ for a random b is equivalent to determing $h(x)$ from $f(x)$ for a random x. Since h is a hardcore bit for f, no adversary can determine b from $c(b)$ and equivalently distinguish between $c(0)$ and $c(1)$ with greater than $1/2+\mathsf{negl}(k)$ probability. Binding is achieved information-theoretically, because f is a permutation, so $c = f(x)$ uniquely determines x, and thus $b = h(x)$.

Notice, in this scheme either player can typically run the setup algorithm. The disadvantage of the scheme is that it only allows one to commit to one bit. If several bits of f are simultaneously hardcore, we can use this scheme to commit to more bits, but one typically does not use this scheme for committing to more than one (or very few) bits. Instead, schemes given below are used.

2.3 Commitment using CRHF

Assume $\mathcal{H}$ is a collision-resistant family. Intuitively, it is very easy to achieve (computational) binding using a random $h \in \mathcal{H}$, since $h(x)$ commits one to the value of x. Unfortunately, $h(x)$ need not (and actually does not) hide all partial information about x, so we need to do something more complicated to achieve hiding.

2.3.1 The Scheme

So assume $\mathcal{H}$ that is a CRHF from L to ℓ bits, let $M = \{0,1\}^n$ and assume $L \gg \ell + n$ (the reason for this will be given later later). Finally, let $\mathcal{U}$ be a family of perfect universal hash functions from L to n bits.[4] Then we define the commitment scheme $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ for $M = \{0,1\}^n$ as follows:

(i) $h \leftarrow \mathsf{Setup}(1^k)$, where $h \leftarrow^r \mathcal{H}$.

(ii) $(c,(u,x)) \leftarrow \mathsf{Commit}(m)$, where $x \leftarrow^r \{0,1\}^k$, $c = (u, h(x))$ and u is a universal hash function chosen fro $\mathcal{U}$ at random subject to $u(x) = m$.

(iii) $\tilde{m} \leftarrow \mathsf{Open}(c,(u,x))$, where $\tilde{m} = u(x)$ if $c = (u, h(x))$ and $\tilde{m} = \perp$ otherwise.

2.3.2 Security

Biding is achieved because a collision — necessarily of the form $c = (u,y)$, $d = (u,x)$, $d' = (u,x')$ — implies that $h(x) = h(x') = y$, and since h is chosen at random from CRHF family, the adversary is "forced" to use $x = x'$, but then we get $m = u(x) = u(x') = m'$, so the messages are not distinct. Hiding (which is information-theoretic, up to a negligible statistical advantage) is much more difficult to prove. We will not do it, but notice that this is where the condition $L \gg \ell + n$. Intuitively, $h(x)$ reveals ℓ out of L bits of information about randomly chosen x. Then, universally of $\mathcal{U}$ and the fact that $L - \ell \gg n$ imply that the choice of u — even subject to $u(x) = m$ — still leaves the distribution of u look "almost uniform" to the adversary, *independent* of m (this is the hard part). Thus, irrespective of

[4] This means that for any $x_0, x_1 \in \{0,1\}^L$, $m_0, m_1 \in \{0,1\}^n$ we have $\Pr_u(u(x_0) = m_0 \wedge u(x_1) = m_1) = \Pr_u(u(x_0) = m_0) \cdot \Pr_u(u(x_1) = m_1)$.

what the message $m \in \{0,1\}^n$ is being committed, the adversary sees a randomly looking u and the value $h(x)$, where both h and x where random and independent of m.

We notice that the size of commitment for n-bit message is $O(L) \gg n$ (since u takes $O(L)$ bits to represent). We will see later how CRHF's can also be used to decrease the commitment size to $O(\ell)$, which could be much less than n.

2.4 Using Random Oracle

We leave this an a simple exercise to show that *optimal*[5] commitment schemes are completely trivial to build in the random oracle model (why?).

2.5 Pedersen commitment (using discrete log)

Finally, we give an example of a commitment scheme based on a specific number theoretic assumption – the discrete log assumption. The scheme is called Pedersen commitment.

2.5.1 One-bit Scheme

Define $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ such that:

(i) $(p, g, y) \leftarrow \mathsf{Setup}(1^k)$ where p is a prime, y is a randomly chosen element of $\mathbb{Z}_p^*$, and g is a randomly chosen generator of $\mathbb{Z}_p^*$.

(ii) $(c, (r, b)) \leftarrow \mathsf{Commit}(b)$, where $r \leftarrow^r \mathbb{Z}_p^*$ and $c = g^r y^b \bmod p$.

(iii) $\tilde{m} \leftarrow \mathsf{Open}(c, (r, b))$, where $\tilde{m} = b$ if $c = g^r y^b$ and $\tilde{m} = \perp$ otherwise.

2.5.2 Security

Hiding is achieved (information-theoretically) because r is randomly chosen from $\mathbb{Z}_p^*$, and therefore both $c(0) = g^r$ and $c(1) = g^r y$ are also random elements of $\mathbb{Z}_p^*$. On the other hand, finding r_0, r_1 such that $\mathsf{Open}(c, (r_0, 0)) = 0$ and $\mathsf{Open}(c, (r_1, 1)) = 1$ would require that $g^{r_0} = g^{r_1} y$. Then $y = g^{r_0 - r_1}$, and the adversary would have computed the discrete log of randomly chosen y: $DL(y) = (r_0 - r_1) \bmod (p-1)$. Hence, under the assumption that discrete log is computationally hard the binding property is achieved.

2.5.3 Commitment for many bits

We extend the commitment scheme above to many bits by using the discrete log assumption over $\mathbb{Z}_p$ where $p = 2q + 1$ is a strong $(k+1)$-bit prime (recall, this means that $p = 2q + 1$ where q is prime). We define $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ over $M = \mathbb{Z}_q$ as follows:

(i) $(p, g, y) \leftarrow \mathsf{Setup}(1^k)$, where $p = 2q + 1$ is a strong $(k+1)$-bit prime , and g is a random generator of $G = QR(\mathbb{Z}_p^*)$, and y is a random element of G.

(ii) $(c, (r, m)) \leftarrow \mathsf{Commit}(m)$, where $r \leftarrow^r \mathbb{Z}_q^*$ and $c = g^r y^m \bmod p$.

(iii) $\tilde{m} \leftarrow \mathsf{Open}(c, (r, m))$, where $\tilde{m} = m$ if $c = g^r y^m$, and $\tilde{m} = \perp$ otherwise.

[5] It is easy to see that in an optimal commitment scheme we have $|c| \approx k$ and $|d| \approx |m| + k$, where k is the security parameter.

2.5.4 Security

Hiding is achieved as before information-theoretically, because r is chosen at random from $\mathbb{Z}_q^*$, so $g^r y^m$ is random in G, irrespective of m. On the other hand, finding (r_0, m_0) and (r_1, m_1) such that $m_0 \neq m_1$, $\mathsf{Open}(c, (r_0, m_0)) = m_0$, and $\mathsf{Open}(c, (r_1, m_1)) = m_1$ would require that

$$g^{r_0} y^{m_0} = g^{r_1} y^{m_1} \bmod p$$

Then $g^{r_0 - r_1} = y^{m_1 - m_0} \bmod p$, which implies that

$$y = g^{(r_0 - r_1)\cdot (m_1 - m_0)^{-1} \bmod q} \bmod p$$

Notice, $(m_1 - m_0)^{-1} \bmod q$ exists since $m_0 \neq m_1$ and q is prime. Thus, the adversary would have computed the discrete log of y base g: $DL_g(y) = (r_0 - r_1) \cdot (m_1 - m_0)^{-1} \bmod q$. Since y is randomly chosen, this contradicts the discrete log assumption over strong primes, so the binding property is achieved as well.

3 Composition Properties of Commitment Schemes

3.1 Bit-by-bit Composition (many times usage)

First, we consider the question of whether the same commitment scheme could be securely used multiple times. Equivalently, assuming we have a secure commitment scheme for small message space — for concreteness, $M = \{0, 1\}$ — can we build a secure commitment scheme for larger message space by a simple bit-by-bit composition of the base commitment scheme. As we saw, the answer to this question was positive in case of CPA-secure encryption, but negative for the case of signatures. Luckily, the answer is also positive for commitment schemes. Namely (for simplicity we state the result for $M = \{0, 1\}$, but it clearly holds for any base commitment scheme),

Lemma 2 *If $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ is a secure commitment scheme for $\{0, 1\}$, then $\mathcal{C}'$, obtained from $\mathcal{C}$ by bit-bit-bit composition for $p(k)$ times, is a secure commitment scheme for $\{0, 1\}^{p(k)}$, for any polynomial $p(k)$. In particular, a given commitment scheme can be securely used for committing to multiple messages.*

Proof Sketch: The proof that $\mathcal{C}'$ satisfies the hiding property is the same as for the case of encryption: use the hybrid argument and the fact that $\mathsf{Commit}_{\mathsf{CK}}(\cdot)$ is a public operation. The binding property is also similarly proven using the hybrid argument: finding a collision for distinct $m_0, m_1 \in \{0, 1\}^{p(k)}$ implies finding a 0/1-collision at some position $i \in \{1 \ldots p(k)\}$. □

3.2 Hash-then-commit with CRHF's

Secondly, recall that hashing allowed for very compact signature schemes via "hash-then-sign" paradigm. It turns out that the same can be done for commitment schemes, except it is now called "hash-then-commit" paradigm. In essence, similar to signatures and unlike encryption, we use the fact that commitment does not have to enable one to recover the message; it should only be hard to collide two messages.

Lemma 3 *If $\mathcal{H}$ is a* CRHF *from L to ℓ bits and $\mathcal{C}' = (\mathsf{Setup}', \mathsf{Commit}', \mathsf{Open}')$ is a secure commitment scheme for ℓ-bit messages, then the commitment scheme $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ defined below is secure for L-bit messages:*

(a) $\mathsf{CK} = (\mathsf{CK}', h)$, *where* $\mathsf{CK}' \leftarrow \mathsf{Setup}'(1^k)$ *and* $h \leftarrow \mathcal{H}$.

(b) $(c', (d', m)) \leftarrow \mathsf{Commit}_{\mathsf{CK}}(m)$, *where* $(c', d') \leftarrow \mathsf{Commit}'_{\mathsf{CK}'}(h(m))$.

(c) $\mathsf{Open}_{\mathsf{CK}}(c', (d', m)) = \tilde{m}$, *where* $\tilde{m} = m$ *if* $\mathsf{Open}'_{\mathsf{CK}'}(c', d') = h(m)$, *and else* $\tilde{m} = \perp$.

Proof Sketch: The hiding property follows easily from that of $\mathcal{C}'$: $\mathsf{Commit}'(h(m_0)) \approx \mathsf{Commit}'(h(m_1))$. For the binding property, a collision triple $(c', (d'_0, m_0), (d'_1, m_1)$ either implies that $h(m_0) = h(m_1)$ — a collision to h — or that (c', d'_0, d'_1) form a collision for the pair $h(m_0) \neq h(m_1)$. □

In particular, we remark that by combining the hash-then-commit technique with the commitment scheme of Section 2.3, we get extremely compact and efficient commitment schemes based on CRHF's, where the size of the commitment to arbitrarily long messages can be as small as the security parameter.

Using UOWHF's? Is is interesting to see if the technique above can work if we replace CRHF's with UOWHF's (picking a fresh $h \in \mathcal{H}$ for every commitment), like we had with digital signatures. A moment reflection shows that the answer is *negative* (why? take a look at the binding property). However, it turns out that UOWHF's can be used, as prescribed above, with a slight relaxation of regular commitment schemes, called *relaxed commitments*. As we will see, this relaxed notion suffices for some important applications of commitment, so we treat it next.

3.3 Relaxed Commitments and UOWHF's

We now consider *relaxed* commitment schemes, where the (strict) binding property of regular commitment schemes is replaced by the **Relaxed Binding** property. Informally, having the knowledge of CK, it is computationally hard for the adversary A to come up with a message m, such that when $(c, d) \leftarrow \mathsf{Commit}(m)$ is generated, $A(c, d, \mathsf{CK})$ produces, with non-negligible probability, a value d' such that (c, d') is a valid commitment to some $m' \neq m$. Formally, for any PPT $A = (A_1, A_2)$,

$$\Pr\left[\begin{array}{c} m \neq m' \wedge \\ m, m' \neq \perp \end{array} \;\middle|\; \begin{array}{c} \mathsf{CK} \leftarrow \mathsf{Setup}(1^k),\ (m, \alpha) \leftarrow A_1(\mathsf{CK}),\ (c, d) \leftarrow \mathsf{Commit}_{\mathsf{CK}}(m), \\ d' \leftarrow A_2(c, d;\ \alpha),\ m' \leftarrow \mathsf{Open}_{\mathsf{CK}}(c, d') \end{array}\right] \leq \mathsf{negl}(k)$$

Thus, A cannot find a collision using a *randomly generated* $c(m)$, even for m of its choice.

As we shall see, (1) relaxed commitment suffice for some important applications of commitment schemes (see authenticated encryption later), (2) UOWHF's can be used in the "hash-then-commit" paradigm, (3) relaxed commitments could be a bit "easier"[6] to construct than regular ones. We start with point (2).

[6] Obviously, both notions are equivalent to OWF's.

Lemma 4 *If $\mathcal{H}$ is a* UOWHF *from L to ℓ bits with key size p and $\mathcal{C}' = (\mathsf{Setup}', \mathsf{Commit}', \mathsf{Open}')$ is a secure* relaxed *commitment scheme for ℓ-bit messages, then the* relaxed *commitment scheme $\mathcal{C} = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Open})$ defined below is secure for L-bit messages:*

(a) $\mathsf{CK} = \mathsf{CK}'$, *where* $\mathsf{CK}' \leftarrow \mathsf{Setup}'(1^k)$.

(b) $((c', h), (d', m)) \leftarrow \mathsf{Commit}_{\mathsf{CK}}(m)$, *where* $(c', d') \leftarrow \mathsf{Commit}'_{\mathsf{CK}'}(h(m))$ *and* $h \leftarrow \mathcal{H}$.

(c) $\mathsf{Open}_{\mathsf{CK}}((c', h), (d', m)) = \tilde{m}$, *where* $\tilde{m} = m$ *if* $\mathsf{Open}'_{\mathsf{CK}'}(c', d') = h(m)$, *and else* $\tilde{m} = \perp$.

Proof Sketch: The hiding property follows easily from that of $\mathcal{C}'$: $\mathsf{Commit}'(h(m_0)) \approx \mathsf{Commit}'(h(m_1))$. For the relaxed binding property, since a fresh $h \in \mathcal{H}$ is chosen for every message and is part of the commitment $c = (c', h)$, a collision triple $((c', h), (d'_0, m_0), (d'_1, m_1)$ either implies that $h(m_0) = h(m_1)$ — a collision to h that was chosen *at random and after* m_0 — or that (c', d'_0, d'_1) form a collision for the pair $h(m_0) \neq h(m_1)$, where again c' was chosen honestly corresponding to $h(m_0)$. □

The result above is not surprising, since in the relaxed binding property, just like in the security of UOWHF's, the commitment/hash function is chosen honestly after the first message is selected. This finishes point (2) above.

As for point (3), consider the construction of commitments from CRHF's, explained in Section 2.3. We notice that replacing a CRHF by a UOWHF (where a fresh function is chosen per every message) will result in a secure *relaxed* commitment. The proof is left as an exercise. To summarize, the relation between CRHF's and UOWHF is very similar to that between regular and relaxed commitments.

4 Applications of Commitment Schemes

4.1 Bidding and Auctions

Consider the following example. A potential buyer Bob is happy to buy some item for any price less than the buying price b. A potential seller Alice is happy to sell the item for any asking price greater than the asking price a. Let us assume that $a \leq b$, but the quantities a and b are initially kept secret by Alice and Bob. Assume Alice and Bob agree on a fair protocol where the item is traded for the average price $p = \frac{a+b}{2}$. One naive protocol would be for Alice tell a to Bob, then for Bob tell b to Alice, and then compute the average c. Unfortunately, in this case Bob will certainly report $b' = a$, making $p' = a$ as well. Similarly, if Bob goes first, Alice will report $a' = b$ resulting in $p' = b$. Clearly, what we need is exactly a commitment. First, Alice commits to the value a and tells her commitment to Bob: $c = c(a)$. The hiding property ensures that Bob learns nothing about a from c. Then Bob tells the value b to Alice. Then Alice opens c to a by sending the opening information $d = d(a)$. The binding property ensures that she can open it in only one way. Then both players compute $p = \frac{a+b}{2}$.

The example above can be generalized to more complicated situations, like auctions. But the point is clear. First, one party commits to some value, then it learns some other information, after which it opens the committed value. We notice the (necessary) weakness of this: the player can always refuse to open the commitment. In some applications, like the

simple buyer/seller game, this does not create serious problems, but in more complicated examples it could result in some unfairness.

We only remark that one has to be careful in such applications. Aside from the problem of commitment key generation discussed earlier, there is also the problem of *non-malleability*, similar to the case of encryption. We will not deal with it here.

4.2 Coin-Flipping

Assume Alice and Bob want to jointly flip a fair coin. For example, they agree that if the coin comes heads, they go to the opera, else they go to the soccer game. Naturally, Bob wants the opera, so he is bound to cheat if Alice asks him to select the (digital) outcome of the coin. Similarly, letting Alice do so will certainly result in a field trip to a soccer stadium. This shows that both Alice and Bob should somehow participate. But what can they do together?

The answer, next best to actually flipping a physical coin, is that they should design an *interactive protocol* where neither player can *influence* the outcome by a non-negligible amount. A first naive attempt is the following: Alice sends Bob a random bit $a \in \{0,1\}$, then Bob tells Alice a random bit $b \in zo$, and they output a joint coin flip $f = a \oplus b$. Clearly, however, in this case we might as well let Bob — the second player to go — select the coin. However, using commitments we can actually fix this protocol.

First, Alice commits to her (supposedly random) bit a, and sends $c = c(a)$ to Bob. Bob then sends a (supposedly random) bit b to Alice. Then Alice opens c to a by sending the opening information $d = d(a)$. Finally, both player output the value $f = a \oplus b$.

Showing that the above simple protocol is "good" is actually non-trivial in the following sense: we first need to give a *formal* definition of what a secure coin-flipping protocol is! The latter is indeed tricky. For example, a naive attempt might be to say that both $\Pr(f = 0), \Pr(f = 1) \in [\frac{1}{2} - \mathsf{negl}(k), \frac{1}{2} + \mathsf{negl}(k)]$. However, the protocol above does not (and, in fact, no protocol *can*!) satisfy this strong property. The reason is that Alice, who learns the coin value first, might refuse to open a if the coin flip $f = 0$. This way, Bob will never see $f = 0$. It turns out that more or less the best we can do is that both $\Pr(f = 0), \Pr(f = 1) \leq \frac{1}{2} + \mathsf{negl}(k)$, so no value can be forced with "unreasonable" probability, even though one or both parties can prematurely terminate the protocol, possibly based on the final "ideal" outcome f. In our protocol, Alice has this ability, while Bob cannot abort the protocol in a way that is dependent on f (by the hiding property of commitments).

We omit more formal treatment, but mention that the protocol above can be shown to satisfy the formal definition of coin-flipping, as outlined above.

4.3 Authenticated Encryption and Relaxed Commitments

We have already earned something about *authenticated encryption* from the homework. While encryption provides privacy against eavesdropping, and signatures/MAC's validate the sender and the integrity of the message, in many situations one wants to achieve both privacy and authenticity *simultaneously*. Authenticated encryption makes sense for both private and public settings. For concreteness, we concentrate below on the public setting. In this case, authenticated encryption is also sometimes called *signcryption*.

We briefly sketch how a commitment scheme $\mathcal{C}$ can be used to achieve efficient secure signcryption — from a secure encryption scheme E and a secure signature scheme S (below, $E(m)$ denotes the encryption of m, and $S(m)$ denotes a message/signature pair (m, σ), i.e. the signature includes the message signed). The two naive way to build signcryption are those of *sequential composition*. Namely, $E(S(m))$ and $S(E(m))$. Under reasonable definitions, both of these methods indeed yield a secure signcryption. However, they have a potential disadvantage that two expensive public-key operations — signing and encrypting — are done sequentially one after another. Below we show a new method where (usually, quite cheap) commitments allow to perform these operations *in parallel*.

First, two auxiliary lemmas, each being interesting, but "useless" on its own.

Lemma 5 *$(c, E(d))$, where $(c, d) \leftarrow \mathsf{Commit}(m)$, is a secure encryption scheme if and only if $\mathcal{C}$ satisfies the hiding property of commitment schemes. Decryption is done by first decrypting d, and then returning $\mathsf{Open}(c, d)$.*

Proof Sketch: We use the hiding property of commitment scheme. Intuitively, c does not revel any information about m, so so does $E(d)$, since E is a secure encryption. The converse is clear as well. □

The reason the lemma by itself is "useless" is we might encrypt m directly using a secure encryption E: just return $E(m)$.

Lemma 6 *$(S(c), d)$, where $(c, d) \leftarrow \mathsf{Commit}(m)$, is a secure signature scheme if and only if $\mathcal{C}$ satisfies the relaxed binding property of commitment schemes. Verification is done be verifying the signature of c and checking $\mathsf{Open}(c, d) \neq \perp$.*

Proof Sketch: We use the relaxed binding property since the pair (c, d) binds one to the message, so it is hard to reuse a valid signature $(S(c), d)$ corresponding to some m (i.e., $c = c(m)$) to produce a forgery $(S(c), d')$ for m': this will create a collision (c, d, d'). Thus, the forger is forced to forge a "new" signature $S(c')$, but this is impossible since S is a secure signature scheme. The converse is simple as well. □

We make two comments here. First, as with the previous lemma, this lemma by itself is useless: one might either sign m directly, or use much cheaper hash-sign-method, if the size of the signature is an issue. Secondly, the lemma above shows that *relaxed* commitments are sufficient for the above applications (so UOWHF's can be used).

Now, we combine the above two lemmas and give the following theorem.

Theorem 1 *$(S(c), E(d))$, where $(c, d) \leftarrow \mathsf{Commit}(m)$, is a secure signcryption if and only if $\mathcal{C}$ is a secure relaxed commitment scheme (decryption and verification is as in the above lemmas). In particular, it is secure with any regular commitment scheme.*

This result is actually useful, since the expensive public key operations are indeed done in parallel. More optimizations of the above idea are possible (i.e., on-line/off-line signcryption), but we omit the details.

4.4 Zero-Knowledge

We will not give any details here, but zero-knowledge proofs allow one to proof the validity of some statement or possession of some secret, without revealing any information beyond a validity of the statement proved, or the possession of the secret in question. For a simple example, one can prove that a number $a \in \mathbb{Z}_n^*$ is a quadratic residue without revealing the square root of a! For another example in the second category (proofs of knowledge), one can proof the knowledge of the value x such that $g^x = y \bmod p$ without revealing x!

No need to say, the study of zero-knowledge proofs is of fundamental importance in cryptography. It turns out that commitment schemes allow one to prove an amazingly powerful statement (do not worry if this makes no sense now): any language in the (huge) complexity class NP has a (computational) zero-knowledge proof.

4.5 Password Authentication and Identification Schemes

Recall the usage of OWF's for password authentication. The server stores the value $f(x)$ in a public file, and the user authenticates by presenting a value x. As an alternative, we can use commitment schemes. Namely, we let $(c, d) \leftarrow \mathsf{Commit}(x)$, store c on the server, and present d during authentication (which succeeds if $\mathsf{Open}(c, d) \neq \perp$). This more complicated schemes has several minor advantages over the scheme with OWF's: (1) the distribution of x does not have to be uniform; (2) even knowing x does not let one authenticate successfully; (3) the value c reveals no partial information about x. In practice, however, the above advantages are not essential, since one still needs to remember the value d (which is longer than x, for example).

We remark that password authentication schemes above have a serious weakness in that they trust the server (or assume that nobody snoops during authentication). Specifically, snooping the password x or the opening value d allow the adversary unlimited future access. It turns out that by *combining* either one of the above scheme with an appropriate zero-knowledge proof — where the user proves to the server the *knowledge* of the corresponding authentication information (i.e., x such that $f(x) = y$, or d such that $\mathsf{Open}(c, d) \neq \perp$) *without revealing this information* — allow one to make up a secure *identification scheme*. Specifically, an identification scheme remains secure for the future, even if the adversary manages to listen in during user authentication, and even if it plays the role of the server with the honest user! Intuitively, the only thing such an adversary learns is that the user indeed possesses correct secret information, but this does not help the adversary to impersonate the user, since the adversary new that the user is "legal" to begin with!

We will study this more formally a bit later, after we talk about zero-knowledge in more detail.

4.6 Trapdoor Commitments

This is not an application by itself. Rather, it is a *stronger type* of commitment, with one additional property. We will not give more detail now, but remark that trapdoor commitments have found numerous applications including on-line/off-line signatures, chameleon signatures, certified e-mail, zero-knowledge and general multi-party computation.