CMSC 858K — Advanced Topics in Cryptography January 27, 2004

Lecture 1

Lecturer: Jonathan Katz *Scribe(s): Jonathan Katz*

1 Introduction to These Notes

These notes are intended to supplement, not replace, the lectures given in class. In particular, only the technical aspects of the lecture are reproduced here; much of the surrounding discussion that took place in class is not.

2 Trapdoor Permutations

We give two definitions of trapdoor permutations. The first is completely formal, and maps well onto the (conjectured) trapdoor permutations that are used in practice. The second is slightly less formal, but is simpler to use and somewhat easier to understand. Generally speaking, however, proofs of security using the second of the two definitions can be easily modified to work for the first definition as well. (Stronger definitions of trapdoor permutations are sometimes also considered; see [1, 2] for some examples.)

We begin with a syntactic definition, and then give a concrete example. Before giving the definition we introduce two pieces of notation. First is the abbreviation PPT which stands for "probabilistic, polynomial-time". This, of course, refers to algorithms which may make random choices during their execution but which always terminate in a polynomial number of steps. This begs the following question: polynomial in *what*? To deal with this, we introduce the notion of a *security parameter* k which will be provided as input to all algorithms. For technical reasons, the security parameter is given in unary and is thus represented as 1^k. In some sense, as we will see, a larger value of the security parameter results in a "more secure" scheme. (Hopefully, the concrete example that follows will give some more motivation for the purpose of the security parameters.)

Definition 1 A *trapdoor permutation* family is a tuple of PPT algorithms (Gen, Sample, Eval, Invert) such that:

1. $\mathsf{Gen}(1^k)$ is a probabilistic algorithm which outputs a pair (i, td). (One can think of i as indexing a particular permutation f_i defined over some domain D_i, while td represents some "trapdoor" information that allows inversion of f_i.)

2. $\mathsf{Sample}(1^k, i)$ is a probabilistic algorithm which outputs an element $x \in D_i$ (assuming i was output by Gen). Furthermore, x is uniformly distributed in D_i. (More formally, the distribution $\{\mathsf{Sample}(1^k, i)\}$ is equal to the uniform distribution over D_i.)

3. $\mathsf{Eval}(1^k, i, x)$ is a deterministic algorithm which outputs an element $y \in D_i$ (assuming i was output by Gen and $x \in D_i$). Furthermore, for all i output by Gen, the function $\mathsf{Eval}(1^k, i, \cdot) : D_i \to D_i$ is a permutation. (Thus, one can view $\mathsf{Eval}(1^k, i, \cdot)$ as corresponding to a permutation f_i mentioned above.)

4. $\mathsf{Invert}(1^k, \mathsf{td}, y)$ is a deterministic algorithm which outputs an element $x \in D_i$, where (i, td) is a possible output of Gen.

Furthermore, we require that for all k, all (i, td) output by Gen, and all $x \in D_i$ we have $\mathsf{Invert}(1^k, \mathsf{td}, \mathsf{Eval}(1^k, i, x)) = x$. (This is our **correctness** requirement.) $\diamondsuit$

The correctness requirement enables one to associate $\mathsf{Invert}(1^k, \mathsf{td}, \cdot)$ with f_i^{-1}. However, it is crucial to recognize that while, as a mathematical function, f_i^{-1} always exists, this function is not necessarily efficiently computable. The definition above, however, guarantees that it *is* efficiently computable, given the "trapdoor" information td (we will see below that, informally, if the trapdoor permutation family is secure then f_i^{-1} is *not* efficiently computable *without* td).

Before going further, we give as a concrete example one of the most popular trapdoor permutations used in practice: RSA [3] (some basic familiarity with RSA is assumed; we merely show how RSA fits into the above framework).

1. $\mathsf{Gen}(1^k)$ chooses two random, k-bit primes p and q, and forms their product $N = pq$. It then computes $\varphi(N) = (p-1)(q-1)$, chooses e relatively prime to $\varphi(N)$, and computes d such that $ed = 1 \bmod \varphi(N)$. Finally, it outputs $((N, e), (N, d))$ (note that i in the definition above corresponds to (N, e) while td corresponds to (N, d)). The domain $D_{N,e}$ is just $\mathbb{Z}_N^*$. (We can also see how the security parameter k comes into play: it determines the length of the primes making up the modulus N, and thus directly affects the "hardness" of factoring the resulting modulus.)

2. $\mathsf{Sample}(1^k, (N, e))$ simply chooses a uniformly-random element from $\mathbb{Z}_N^*$. We noted in class that it is possible to do this efficiently.

3. $\mathsf{Eval}(1^k, (N, e), x)$, where $x \in \mathbb{Z}_N^*$, outputs $y = x^e \bmod N$.

4. $\mathsf{Invert}(1^k, (N, d), y)$, where $y \in \mathbb{Z}_N^*$, outputs $x = y^d \bmod N$.

It is well-known that Invert indeed computes the inverse of Eval. Hence, RSA (as described above) is a trapdoor permutation family.

2.1 Trapdoor (One-Way) Permutations

The definition above was simply syntactic; it does not include any notion of "hardness" or "security". However, when cryptographers talk about trapdoor permutations they always mean *one-way* trapdoor permutations. Informally, this just means that a randomly-generated trapdoor permutation is hard to invert when the trapdoor information td is not known. In giving a formal definition, however, we must be careful: what do we mean by "hard to invert"? Roughly speaking, we will say this means that any "efficient" algorithm succeeds in inverting a randomly-generated f_i (at a random point) with only "very small" probability. (Note that it only makes sense to talk about the hardness of inverting a randomly-generated trapdoor permutation. If we *fix* a trapdoor permutation f_i then it may very well be the case that an adversary knows the associated trapdoor. A similar argument shows that the point to be inverted must be chosen at random as well.) It should be no surprise that we associate "efficient" algorithms with PPT ones. Our notion of "small" is made precise via the class of *negligible functions*, which we define now.

Definition 2 A function $\varepsilon : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ is *negligible* if it is asymptotically smaller than any inverse polynomial. More formally, this means that for all $c > 0$ there exists an integer N_c such that:

$$N > N_c \Rightarrow f(N) < 1/N^c.$$

$\diamondsuit$

We will now formally define the notion of being hard to invert, and thus formally define the notion of one-way trapdoor permutation families.

Definition 3 A trapdoor permutation family $(\mathsf{Gen}, \mathsf{Sample}, \mathsf{Eval}, \mathsf{Invert})$ is *one-way* if for any PPT A the following is negligible (in k):

$$\Pr[(i, \mathsf{td}) \leftarrow \mathsf{Gen}(1^k); y \leftarrow \mathsf{Sample}(1^k, i); x \leftarrow A(1^k, i, y) : \mathsf{Eval}(1^k, i, x) = y]. \tag{1}$$

$\diamondsuit$

A few words are in order to explain Eq. (1), especially since this notation will be used extensively throughout the rest of the semester. The equation represents the probability of a particular event following execution of a particular experiment; the experiment itself is written to the left of the colon, while the event of interest is written to the right of the colon. Furthermore, individual components of the experiment are separated by a semicolon. We use the notation "$\leftarrow$" to denote a randomized procedure: if S is a set, then "$x \leftarrow S$" denotes selecting x uniformly at random from S; if A is a randomized algorithm, then "$x \leftarrow A(\cdots)$" represents running A (with uniformly-chosen randomness) to obtain output x. Finally, in an *experiment* (i.e., to the left of the colon) "=" denotes assignment (thus, e.g., if A is a *deterministic* algorithm then we write $x = A(\cdots)$); on the other hand, in an event (i.e., to the right of the colon), "=" denotes a test of equality.

Thus, we may express Eq. (1) in words as follows:

> The probability that $\mathsf{Eval}(1^k, i, x)$ is equal to y upon completion of the following experiment: run $\mathsf{Gen}(1^k)$ to generate (i, td), run $\mathsf{Sample}(1^k, i)$ to generate y, and finally run $A(1^k, i, y)$ to obtain x.

Note also that Eq. (1) is indeed a function of k, and hence it makes sense to talk about whether this expression is negligible or not.

From now on, when we talk about "trapdoor permutations" we always mean "a one-way trapdoor permutation family".

2.2 A Simplified Definition of Trapdoor Permutations

The above definition is somewhat cumbersome to work with, and we therefore introduce the following simplified definition. As noted earlier, this definition does not map well (and sometimes does not map at all) to the trapdoor permutations used in practice; yet, proofs of security using this definition are (in general) easily modified to hold with regard to the more accurate definition given above. (Of course, when giving proofs of security based on trapdoor permutations, one should always be careful to make sure that this is the case.)

The following definition introduces two simplifying assumptions and one notational simplification: our first assumption is that all D_i are the same for a given security parameter

k. Furthermore, for a given security parameter k we will simply assume that $D_i = \{0,1\}^k$ (i.e., the set of strings of length k). We simplify the notation as follows: instead of referring to an index i and a trapdoor td, we simply refer to a permutation f and its inverse f^{-1}. (Technically, one should think of f as a *description* of f, which in particular allows for efficient computation of f; analogously, one should think of f^{-1} as a description of (an efficient method for computing) f^{-1}. In particular, it should always be kept in mind that the mathematical function f^{-1} will not, in general, be computable in polynomial time without being given some "trapdoor" information (which we are here representing by "f^{-1}").)

Definition 4 A trapdoor permutation family is a tuple of PPT algorithms $(\mathsf{Gen}, \mathsf{Eval}, \mathsf{Invert})$ such that:

1. $\mathsf{Gen}(1^k)$ outputs a pair (f, f^{-1}), where f is a permutation over $\{0,1\}^k$.
2. $\mathsf{Eval}(1^k, f, x)$ is a deterministic algorithm which outputs some $y \in \{0,1\}^k$ (assuming f was output by Gen and $x \in \{0,1\}^k$). We will often simply write $f(x)$ instead of $\mathsf{Eval}(1^k, f, x)$.
3. $\mathsf{Invert}(1^k, f^{-1}, y)$ is a deterministic algorithm which outputs some $x \in \{0,1\}^k$ (assuming f^{-1} was output by Gen and $y \in \{0,1\}^k$). We will often simply write $f^{-1}(y)$ instead of $\mathsf{Invert}(1^k, f^{-1}, y)$.
4. (**Correctness.**) For all k, all (f, f^{-1}) output by Gen, and all $x \in \{0,1\}^k$ we have $f^{-1}(f(x)) = x$.
5. (**One-wayness**.) For all PPT A, the following is negligible:

$$\Pr[(f, f^{-1}) \leftarrow \mathsf{Gen}(1^k); y \leftarrow \{0,1\}^k; x \leftarrow A(1^k, f, y) : f(x) = y].$$

♢

Given the above notation, we can just as well associate our trapdoor permutation family with Gen (and let the algorithms Eval and Invert be entirely implicit).

References

[1] O. Goldreich. *Foundations of Cryptography, vol. 1: Basic Tools*, Cambridge University Press, 2001.

[2] O. Goldreich. *Foundations of Cryptography, vol. 2: Basic Applications*, Cambridge University Press, to appear.

[3] R. Rivest, A. Shamir, and L.M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM* 21(2): 120–126 (1978).

CMSC 858K — Advanced Topics in Cryptography January 29, 2004

Lecture 2

Lecturer: Jonathan Katz

Scribe(s): Alvaro A. Cardenas, Avinash J. Dalal, Julie Staub

1 Summary

In the last set of notes the concept of a trapdoor permutation was discussed. In this set of lecture notes we begin by defining a *public-key encryption scheme*, and what it means for that scheme to be *semantically secure*. We show that if a public-key encryption scheme is secure under this definition then the encryption algorithm cannot be deterministic. We then define a *hardcore bit* and use it to build a provably-secure public-key encryption scheme.

2 Public-Key Cryptography

Definition 1 A **public-key encryption scheme** is a triple of PPT algorithms $(\mathsf{Gen}, \mathcal{E}, \mathcal{D})$, where

1. Gen is the key generation algorithm. $\mathsf{Gen}(1^k)$ outputs a pair (pk, sk). We assume for simplicity that $|pk| = k$.
2. $\mathcal{E}$ is the encryption algorithm. Given a plaintext m from a message space $\mathcal{M}$, algorithm $\mathcal{E}_{pk}(m)$ returns a ciphertext C of polynomial length $p(k)$.
3. $\mathcal{D}$ is the decryption algorithm. $\mathcal{D}_{sk}(C)$ returns a message m or the symbol $\perp$ representing incorrect decryption. Incorrect decryption can happen for example if C is not a valid ciphertext. (We will assume for simplicity — unless stated otherwise — that the decryption algorithm is deterministic.)
4. The public-key encryption scheme must satisfy **correctness:** i.e., for all $m \in \mathcal{M}$ and all possible (pk, sk) output by Gen, we have $\mathcal{D}_{sk}(\mathcal{E}_{pk}(m)) = m$.

$\diamondsuit$

In the following we assume that authentication of the public keys is possible and thus our main security concern is an attacker with access to the public key who attempts to obtain information about the plaintext from the ciphertext. Since the adversary has access to the public key pk, she can encrypt any message she wants and thus this scenario is sometimes known as a **chosen plaintext attack (CPA)**.

Our following definition of a secure public-key encryption scheme is strong in the sense that we do not only require that an adversary cannot obtain the plaintext m from the

knowledge of the public-key pk and the ciphertext C, but also that an adversary cannot obtain any partial information about m (except probably some information about the length). This security notion is known as *semantic security* or *indistinguishability* [3, 1].

The security of the scheme is stated as a game in which an adversary has the ability to select two messages. One of the messages is randomly selected and encrypted. The encryption is then called secure if the adversary cannot do better than a random guess in finding out which message was encrypted. Before we formalize the game we recall what a negligible function is.

Definition 2 A function $\varepsilon(\cdot) : \mathbb{N} \to [0,1]$ is **negligible** iff $\forall\ c > 0$, there exists an $N_c > 0$ such that $\forall\ N > N_c$ we have $\varepsilon(N) < 1/N^c$. $\diamondsuit$

An easier way of saying this is that $\varepsilon(\cdot)$ is negligible iff it grows smaller than any inverse polynomial. A very common example of a negligible function is the inverse exponential, $\varepsilon(k) = 2^{-k}$. Note that $2^{-k} = \mathcal{O}(1/k^c)$ for any c. We will use this definition of a negligible function to explicitly define what it means for an encryption scheme to be secure.

Definition 3 A public-key encryption scheme $(\mathsf{Gen}, \mathcal{E}, \mathcal{D})$ is **semantically secure** if for all PPT algorithms A, the following is negligible:

$$\left| \Pr \left[\begin{array}{l} (pk, sk) \leftarrow \mathsf{Gen}(1^k); (m_0, m_1) \leftarrow A(pk); \\ b \leftarrow \{0,1\}; C \leftarrow \mathcal{E}_{pk}(m_b); b' \leftarrow A(pk, C) \end{array} \quad : \quad b = b' \right] - \frac{1}{2} \right|.$$

$\diamondsuit$

Theorem 1 *If a public-key encryption scheme is semantically secure, then the encryption algorithm is not deterministic.*

Proof Consider an adversary who outputs (m_0, m_1) with $m_0 \neq m_1$. When presented with a ciphertext C, which is either an encryption of m_0 or m_1, compute $C_0 = \mathcal{E}_{pk}(m_0)$. If $C = C_0$ output 0 else output 1. This adversary succeeds in guessing b (cf. the above game) with probability 1; we use the fact that decryption succeeds with probability 1 and hence the space of encryptions of m_0 must be disjoint from the space of encryptions of m_1. ■

In the first lecture we defined what one-way trapdoor permutations are. Intuitively a one-way trapdoor permutation seems to be a good suggestion for a public-key encryption scheme as it easy to evaluate the function (encrypt) and hard to invert without the trapdoor (decrypt). More formally, given a one-way trapdoor permutation Gen_{td}, it is tempting to use the following encryption scheme: to generate keys, run $\mathsf{Gen}_{td}(1^k)$ to obtain (f, f^{-1}). Set $pk = f$ and $sk = f^{-1}$. Set the encryption algorithm $\mathcal{E}_f(\cdot) = f(\cdot)$, and set the decryption algorithm $\mathcal{D}_{f^{-1}}(\cdot) = f^{-1}(\cdot)$. However, from Theorem 1 we can conclude that a one-way trapdoor permutation cannot be used as a semantically secure public-key encryption scheme because the evaluation algorithm (i.e., computing $f(\cdot)$) is deterministic. Note in particular that "textbook RSA" (where encryption is $\mathcal{E}_{(N,e)}(m) = m^e \bmod N$) is susceptible to the adversary in the proof of Theorem 1. (It should also be clear, however, that the problems of the above approach — and in particular the case of "textbook RSA" — go beyond the fact that encryption is deterministic. For example, randomly padding the message before encrypting is not sufficient to guarantee semantic security either.)

However not all hope for using one-way trapdoor permutations as a basis for a secure encryption scheme is lost. First we will need to define hard-core bits.

3 Hard-Core Bits

Another problem with using one-way trapdoor permutations to encrypt (as suggested above) is that they can potentially reveal some information about the input when we have access to the output. For example, if $f(x)$ is a one-way trapdoor permutation, then it is easy to show that the function $f'(x_1|x_2) = x_1|f(x_2)$ (for $|x_1| = |x_2|$) is also a one-way trapdoor permutation. Here, however, we see that f' reveals half of the bits of its input directly. A *hardcore bit* of a one-way permutation is a bit of information that cannot be correctly identified better than with random guessing. Hardcore bits help us to formalize the notion of a single bit of information about the input x that is effectively obscured by the action of a one-way trapdoor permutation. More formally we have:

Definition 4 Let $H = \{h_k : \{0,1\}^k \to \{0,1\}\}_{k\geq 1}$ be a collection of efficiently-computable functions and let $\mathcal{F} = (\mathsf{Gen}_{td})$ be a trapdoor permutation. H is a **hard-core bit** for $\mathcal{F}$ if for all PPT algorithms A, the following is negligible (in k):

$$\left|\Pr[(f, f^{-1}) \leftarrow \mathsf{Gen}_{td}(1^k); x \leftarrow \{0,1\}^k; y = f(x) : A(f, y) = h_k(x)] - \frac{1}{2}\right|.$$

♢

Theorem 2 ([2]) ***Existence of hard-core bits.*** *Let $\mathcal{F} = (\mathsf{Gen}_{td})$ be a trapdoor permutation with $f : \{0,1\}^k \to \{0,1\}^k$ (for security parameter k). Consider the permutation family $\mathcal{F}' = (\mathsf{Gen}'_{td})$ with $f' : \{0,1\}^{2k} \to \{0,1\}^{2k}$ defined as $f'(x|r) \stackrel{\text{def}}{=} f(x)|r$, and the function family $\mathcal{H} = \{h_k : \{0,1\}^{2k} \to \{0,1\}\}$ defined by $h_k(x|r) \stackrel{\text{def}}{=} x \cdot r$ (where "$\cdot$" represents the binary dot product). Then $\mathcal{F}'$ is a trapdoor permutation with hard-core bit $\mathcal{H}$.*

Recall that if $x = x_1 x_2 \ldots x_k \in \{0,1\}^k$ and $r = r_1 r_2, \ldots r_k \in \{0,1\}^k$ then $x \cdot r \stackrel{\text{def}}{=} x_1 r_1 \oplus x_2 r_2 \oplus \cdots \oplus x_k r_k = \bigoplus_{i=1}^k x_i r_i$ (where $\oplus$ represents binary exclusive-or). For example, $1101011 \cdot 1001011 = 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 1 = 0$.

4 Public-Key Encryption From Trapdoor Permutations

In the following we assume for simplicity that $\mathcal{M} = \{0,1\}$, i.e. we only are interested in encrypting single-bit messages (we will later show how any single-bit encryption scheme can be used to derive an encryption scheme for poly-many bits). Given a trapdoor permutation $\mathcal{F} = (\mathsf{Gen}_{td})$, construct the following encryption scheme

1. $\mathsf{Gen}(1^k)$:
   $(f, f^{-1}) \leftarrow \mathsf{Gen}_{td}(1^k)$
   Select a random r: $r \leftarrow \{0,1\}^k$
   Output $pk = (f, r)$ and $sk = f^{-1}$

2. $\mathcal{E}_{pk}(m)$ (where $m \in \{0,1\}$):
   pick $x \leftarrow \{0,1\}^k$
   Compute $y = f(x)$
   Compute $h' = x \cdot r$
   Output $C = \langle y | h' \oplus m \rangle$

3. $D_{sk}(y|b)$ (where $|y| = k$ and $|b| = 1$):
   Output $(f^{-1}(y) \cdot r) \oplus b$

Correctness Note that if $y|b$ is a valid encryption of m then $f^{-1}(y) = x$ and $b = (x \cdot r) \oplus m$. So the decryption algorithm will output $(x \cdot r) \oplus (x \cdot r) \oplus m = m$.

Theorem 3 *Assuming $\mathcal{F}$ is a trapdoor permutation, the encryption scheme presented above is semantically secure.*

Proof Assume toward a contradiction that the encryption scheme is not semantically secure. Then there exists a PPT algorithm A such that

$$\left| \Pr[(pk, sk) \leftarrow \mathsf{Gen}(1^k); b \leftarrow \{0,1\}; C \leftarrow \mathcal{E}_{pk}(b); b' \leftarrow A(pk, C) : b = b'] - \frac{1}{2} \right| \qquad (1)$$

is not negligible. For simplicity, we simply assume $m_0 = 0$ and $m_1 = 1$ (recall we are working over a single-bit message space anyway, and the adversary cannot possibly succeed with better than half probability if $m_0 = m_1$).

Let the one-way trapdoor permutation that we are using for the encryption scheme be $\mathcal{F} = (\mathsf{Gen}_{td})$. With this trapdoor permutation we construct $\mathcal{F}' = (\mathsf{Gen}'_{td})$ with hard-core bit $\mathcal{H} = \{h_k\}$ as in Theorem 2; i.e., the hardcore bit for $f'(x|r) = f(x)|r$ is $h_k(x|r) = x \cdot r$. We know that for any PPT adversary A', the probability of guessing the hardcore bit $x \cdot r$ given $f_k(x)|r$ is negligible; that is, the following is negligible for any PPT A':

$$\left| \Pr \left[\begin{array}{cc} (f', f'^{-1}) \leftarrow \mathsf{Gen}'_{td}(1^k); x \leftarrow \{0,1\}^k; & \\ r \leftarrow \{0,1\}^k; y = f(x) & : \quad A'(f', y|r) = x \cdot r \end{array} \right] - \frac{1}{2} \right| . \qquad (2)$$

Given A as above, our goal is to construct a PPT algorithm A' contradicting the above equation. We proceed as follows:

$A'(f', y|r)$
   $\alpha \leftarrow \{0,1\}$
   Define $pk = (f, r)$ and $C = (y|\alpha)$
   run $A(pk, C)$
   if the output of A equals 0 then output α
   else output the complement $\bar{\alpha}$

We may also rephrase the execution of A' as follows: it runs $A(pk, C)$ as above, and then outputs $\alpha \oplus A(pk, C)$ (this gives exactly the same output as above). Note also that A' runs in probabilistic polynomial time, assuming A does.

Let us first examine the intuition behind this construction of A'. We have $A(pk, C) = A((f, r), (y|\alpha))$; thus, if A always correctly guessed which message was encrypted, then A would always output $(f^{-1}(y) \cdot r) \oplus \alpha$, and hence A' would always output $f^{-1}(y) \cdot r$. The key thing to notice here is that $f^{-1}(y) \cdot r$ is the hardcore bit $h_k(x|r)$ of $f'(x|r)$ (where we let $x \stackrel{\text{def}}{=} f^{-1}(y)$). So, if A always succeeds (in "breaking" the encryption scheme) then A' always succeeds in guessing the hardcore bit. Of course, there is no reason to assume that A *always* succeeds and a formal proof is needed.

To complete our proof we need to massage Eq. (2) into Eq. (1). We are interested in the probability that A' correctly predicts the hard-core bit (this is just Equation (2), replacing f' by its definition in terms of f), i.e., we are interested in the following:

$$\left|\Pr[(f, f^{-1}) \leftarrow \mathsf{Gen}_{td}(1^k); x, r \leftarrow \{0,1\}^k; y = f(x) : A'(f, y|r) = x \cdot r] - \frac{1}{2}\right|.$$

Re-writing the above in terms of how A' was constructed, we obtain:

$$\begin{aligned}
&\left|\Pr[(f, f^{-1}) \leftarrow \mathsf{Gen}_{td}(1^k); x, r \leftarrow \{0,1\}^k; y = f(x) : A'(f, y|r) = x \cdot r] - \frac{1}{2}\right| \\
&= \left|\Pr\left[\begin{array}{c|c} (f, f^{-1}) \leftarrow \mathsf{Gen}_{td}(1^k); x, r \leftarrow \{0,1\}^k; & A((f, r), (y|\alpha)) \oplus \alpha = x \cdot r \\ y = f(x); \alpha \leftarrow \{0,1\} & \end{array}\right] - \frac{1}{2}\right|.
\end{aligned}$$

Next, we modify the experiment syntactically by choosing a bit b at random and setting $\alpha = (x \cdot r) \oplus b$ (I will omit the parentheses from now on). Note, however, that from the point of view of A this is *exactly* equivalent to the above (because α is still uniformly distributed over $\{0,1\}$). Thus, we obtain (after some algebraic simplification):

$$\begin{aligned}
&\left|\Pr\left[\begin{array}{c|c} (f, f^{-1}) \leftarrow \mathsf{Gen}_{td}(1^k); x, r \leftarrow \{0,1\}^k; & A((f, r), (y|\alpha)) \oplus \alpha = x \cdot r \\ y = f(x); \alpha \leftarrow \{0,1\} & \end{array}\right] - \frac{1}{2}\right| \\
&= \left|\Pr\left[\begin{array}{c|c} (f, f^{-1}) \leftarrow \mathsf{Gen}_{td}(1^k); x, r \leftarrow \{0,1\}^k; & A((f, r), (y|\alpha)) \oplus \alpha = x \cdot r \\ y = f(x); b \leftarrow \{0,1\}; \alpha = x \cdot r \oplus b & \end{array}\right] - \frac{1}{2}\right| \\
&= \left|\Pr\left[\begin{array}{c|c} (f, f^{-1}) \leftarrow \mathsf{Gen}_{td}(1^k); x, r \leftarrow \{0,1\}^k; & A((f, r), (y|x \cdot r \oplus b)) = b \\ y = f(x); b \leftarrow \{0,1\} & \end{array}\right] - \frac{1}{2}\right|. \quad (3)
\end{aligned}$$

Finally, let us look at the inputs given to A in the last expression above. The first input (f, r) is exactly a public-key for the encryption scheme under consideration. Furthermore, the second input $y|x \cdot r \oplus b$ given to A (with x, y and r chosen at random) is *exactly* a (random) encryption of the bit b with respect to the given public key. Thus, Equation (3) is exactly equal to Equation (1) (and hence Equation (2) is equal to Equation (1)). But we began by assuming that Equation (1) was non-negligible; this means that we have a particular PPT adversary A' for which Equation (2) is non-negligible. But this contradicts the assumed security of the trapdoor permutation (i.e., Theorem 2). ■

References

[1] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Adv. in Cryptology — CRYPTO 1998.*

[2] O. Goldreich and L. Levin, A hard-core predicate for all one-way functions, *Proc. 21st Ann. ACM Symp. on Theory of Computing*, 1989, pp. 25–32.

[3] S. Goldwasser and S. Micali, Probabilistic encryption, *Journal of Computer and System Sciences,* 28 (1984), pp. 270–299.

CMSC 858K — Advanced Topics in Cryptography	February 3, 2004
Lecture 3	
Lecturer: Jonathan Katz	*Scribe(s): Abheek Anand, Gelareh Taban, Radostina Koleva*

1 Introduction

In the last lecture, we introduced the notion of semantic security and gave a formal definition for semantic security with respect to public-key encryption schemes. It was shown that given the existence of hard-core bits, it is possible to construct a *semantically secure public-key encryption scheme* for messages of length one bit. In this lecture, we introduce the *hybrid technique* and use it to prove that semantic security of the constructed encryption scheme can be extended to polynomially-many messages (of arbitrary polynomial length).

We begin by reviewing the construction of a semantically-secure public-key encryption scheme from a trapdoor permutation. Let $F = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Invert})$ be a trapdoor permutation family and $H = \{h_k\}$ be a hard-core bit for F. Then we can construct the following public-key encryption scheme $PKE = (\mathsf{KeyGen}, \mathcal{E}, \mathcal{D})$ for the encryption of 1-bit messages:

$\mathsf{KeyGen}(1^k)$: $(f, f^{-1}) \leftarrow \mathsf{Gen}(1^k)$
$PK = (f, h_k)$
$SK = f^{-1}$

$\mathcal{E}_{PK}(m)$: $r \leftarrow \{0,1\}^k$
output $\langle f(r), h_k(r) \oplus m \rangle$

$\mathcal{D}_{SK}(\langle y, b \rangle)$: output $b \oplus h_k(f^{-1}(y))$

We showed in class last time that the encryption scheme above is semantically secure. In particular, this implies the following theorem:

Theorem 1 *Assuming trapdoor permutations exist, there exists a public-key encryption scheme achieving semantic security (or security in the sense of indistinguishability).*

2 Security for Multiple Messages

The encryption scheme above is semantically secure for messages of length one bit. Would the scheme remain secure if it is applied (in the natural bit-by-bit fashion[1]) to messages of length longer than one bit? Equivalently, is the scheme still secure if it is used to encrypt multiple messages, each of length one bit? If the adversary is able to eavesdrop

[1]Here, encryption of $m = m_1 \cdots m_\ell$ (with $m_i \in \{0,1\}$) is given by $\mathcal{E}_{PK}(m_1) \cdots \mathcal{E}_{PK}(m_\ell)$. The only subtlety here is that *independent* random coins must be used for every invocation of the encryption algorithm.

on these messages, will she obtain extra information (perhaps correlated information about the various messages) and be able to break the semantic security of the encryption scheme?

To model this stronger attack scenario where the adversary can intercept multiple messages via eavesdropping, we introduce the concept of an *encryption oracle* $\mathcal{E}_{PK,b}(\cdot,\cdot)$ which the adversary can query as many times as it wants. This oracle takes as input two messages m_0, m_1 of equal length, and we define $\mathcal{E}_{PK,b}(m_0, m_1) \stackrel{\text{def}}{=} \mathcal{E}_{PK}(m_b)$ (where new random coins are used to encrypt m_b each time the oracle is invoked). A scheme is secure if the adversary cannot guess the value of the bit b used by the encryption oracle (with much better than probability 1/2). A formal definition follows.

Definition 1 A public-key encryption scheme $PKE = (\mathsf{KeyGen}, \mathcal{E}, \mathcal{D})$ is secure in the sense of left-or-right indistinguishability if the following is negligible (in k) for any PPT adversary:

$$\left|\Pr\left[(PK, SK) \leftarrow \mathsf{KeyGen}(1^k); b \leftarrow \{0,1\} : A^{\mathcal{E}_{PK,b}(\cdot,\cdot)}(PK) = b\right] - 1/2\right|.$$

$\diamondsuit$

Theorem 2 *If a public-key encryption scheme $PKE = (\mathsf{KeyGen}, \mathcal{E}, \mathcal{D})$ is semantically secure, then it is also secure in the sense of left-or-right indistinguishability.*

Proof To prove this theorem, the *hybrid argument* is introduced. This technique plays a central role in demonstrating the indistinguishability of complex ensembles based on the indistinguishability of simpler ensembles. However, before we define the technique and prove the more general case, we will show that a semantically secure encryption scheme (for one message) is secure in the sense of left-or-right indistinguishability when *two* messages are encrypted. This is represented by allowing the adversary to have oracle access to the encryption oracle twice.

We will, as usual, perform a proof by contradiction: assume toward a contradiction that PKE is semantically secure but not left-or-right secure. This means that we have a PPT adversary A that can break the PKE in the left-or-right indistinguishability sense with non-negligible probability. Using this adversary we will construct a PPT algorithm that breaks the semantic security of the PKE with non-negligible probability. This is a contradiction as according to the theorem PKE is semantically secure.

In what follows we will let the key generation step be implicit in order to make the notation more readable. Construct adversary $\hat{A}_1$ that can access the encryption oracle just once, and tries to break semantic security as follows:

$\hat{A}_1^{\mathcal{E}_{PK,b}(\cdot,\cdot)}(PK)$
Run $A(PK)$
At some point A asks for $\mathcal{E}_{PK,b}(m_0, m_1)$
$\hat{A}_1$ queries its own encryption oracle and returns $c \leftarrow \mathcal{E}_{PK,b}(m_0, m_1)$ to A
Later, A requests a second encryption $\mathcal{E}_{PK,b}(m_0', m_1')$
$\hat{A}_1$ returns $c' \leftarrow \mathcal{E}_{PK}(m_0')$ to A (i.e., it encrypts m_0' *itself*)
$\hat{A}_1$ outputs the final output of A

Since A runs in polynomial time so does $\hat{A}_1$. The probability that $\hat{A}_1$ succeeds is:

$$\begin{aligned}
\mathsf{Succ}_{\hat{A}_1} &\stackrel{\text{def}}{=} \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k); b \leftarrow \{0,1\} : \hat{A}_1^{\mathcal{E}_{PK,b}(\cdot,\cdot)}(PK) = b\right] \qquad (1)\\
&= \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k); b \leftarrow \{0,1\} : A^{\mathcal{E}_{PK,b}(\cdot,\cdot),\mathcal{E}_{PK,0}(\cdot,\cdot)}(PK) = b\right]\\
&= \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k) : A^{\mathcal{E}_{PK,0}(\cdot,\cdot),\mathcal{E}_{PK,0}(\cdot,\cdot)}(PK) = 0\right] \times \frac{1}{2}\\
&\quad + \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k) : A^{\mathcal{E}_{PK,1}(\cdot,\cdot),\mathcal{E}_{PK,0}(\cdot,\cdot)}(PK) = 1\right] \times \frac{1}{2},
\end{aligned}$$

where we have abused notation and written $A^{\mathcal{E}_{PK,b_1}(\cdot,\cdot),\mathcal{E}_{PK,b_2}(\cdot,\cdot)}$ to indicate that the first time A accesses its oracle, the oracle uses bit b_1, whereas the second time A accesses its oracle, the oracle uses bit b_2.

Similarly, we construct an adversary $\hat{A}_2^{\mathcal{E}_{PK,b}(\cdot,\cdot)}$ that accesses the encryption oracle just once and runs as follows:

$\underline{\hat{A}_2^{\mathcal{E}_{PK,b}(\cdot,\cdot)}(PK)}$
Run $A(PK)$
At some point A asks for $\mathcal{E}_{PK,b}(m_0, m_1)$
$\hat{A}_2$ returns $c \leftarrow \mathcal{E}_{PK}(m_1)$ to A (i.e., it encrypts m_1 *itself*)
Later, A requests a second encryption $\mathcal{E}_{PK,b}(m_0', m_1')$
$\hat{A}_2$ queries its own encryption oracle and returns $c' \leftarrow \mathcal{E}_{PK,b}(m_0', m_1')$ to A
$\hat{A}_2$ outputs the final output of A

Again, $\hat{A}_2$ is clearly a PPT algorithm. The probability that $\hat{A}_2$ succeeds is:

$$\begin{aligned}
\mathsf{Succ}_{\hat{A}_2} &\stackrel{\text{def}}{=} \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k); b \leftarrow \{0,1\} : \hat{A}_2^{\mathcal{E}_{PK,b}(\cdot,\cdot)}(PK) = b\right] \qquad (2)\\
&= \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k); b \leftarrow \{0,1\} : A^{\mathcal{E}_{PK,1}(\cdot,\cdot),\mathcal{E}_{PK,b}(\cdot,\cdot)}(PK) = b\right]\\
&= \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k) : A^{\mathcal{E}_{PK,1}(\cdot,\cdot),\mathcal{E}_{PK,0}(\cdot,\cdot)}(PK) = 0\right] \times \frac{1}{2}\\
&\quad + \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k) : A^{\mathcal{E}_{PK,1}(\cdot,\cdot),\mathcal{E}_{PK,1}(\cdot,\cdot)}(PK) = 1\right] \times \frac{1}{2}.
\end{aligned}$$

We now express A's advantage[2] in breaking the left-or-right indistinguishability of the scheme in terms of Equations (1) and (2):

$$\begin{aligned}
\mathsf{Adv}_A &\stackrel{\text{def}}{=} \left|\Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k); b \leftarrow \{0,1\} : A^{\mathcal{E}_{PK,b}(\cdot,\cdot)}(PK) = b\right] - \frac{1}{2}\right|\\
&= \left|\frac{1}{2} \cdot \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k) : A^{\mathcal{E}_{PK,1}(\cdot,\cdot),\mathcal{E}_{PK,1}(\cdot,\cdot)}(PK) = 1\right]\right.\\
&\quad \left. + \frac{1}{2} \cdot \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k) : A^{\mathcal{E}_{PK,0}(\cdot,\cdot),\mathcal{E}_{PK,0}(\cdot,\cdot)}(PK) = 0\right] - \frac{1}{2}\right|
\end{aligned}$$

[2] An adversary's *advantage* in this setting is simply the absolute value of its success probability (i.e., the probability that it correctly guesses b) minus 1/2.

$$
\begin{aligned}
= \quad & \Bigg| \frac{1}{2} \cdot \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k) : A^{\mathcal{E}_{PK,1}(\cdot,\cdot),\mathcal{E}_{PK,1}(\cdot,\cdot)}(PK) = 1\right] \\
& + \frac{1}{2} \cdot \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k) : A^{\mathcal{E}_{PK,1}(\cdot,\cdot),\mathcal{E}_{PK,0}(\cdot,\cdot)}(PK) = 0\right] \\
& + \frac{1}{2} \cdot \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k) : A^{\mathcal{E}_{PK,1}(\cdot,\cdot),\mathcal{E}_{PK,0}(\cdot,\cdot)}(PK) = 1\right] \\
& + \frac{1}{2} \cdot \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k) : A^{\mathcal{E}_{PK,0}(\cdot,\cdot),\mathcal{E}_{PK,0}(\cdot,\cdot)}(PK) = 0\right] - 1 \Bigg|,
\end{aligned} \tag{3}
$$

where we use the fact (from basic probability theory) that

$$
\Pr\left[A^{\mathcal{E}_{PK,1}(\cdot,\cdot)\mathcal{E}_{PK,0}(\cdot,\cdot)}(PK) = 0\right] + \Pr\left[A^{\mathcal{E}_{PK,1}(\cdot,\cdot)\mathcal{E}_{PK,0}(\cdot,\cdot)}(PK) = 1\right] = 1.
$$

Continuing, we obtain:

$$
\begin{aligned}
\mathsf{Adv}_A \quad \leq \quad & \Bigg| \frac{1}{2} \cdot \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k) : A^{\mathcal{E}_{PK,1}(\cdot,\cdot),\mathcal{E}_{PK,1}(\cdot,\cdot)}(PK) = 1\right] \\
& + \frac{1}{2} \cdot \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k) : A^{\mathcal{E}_{PK,1}(\cdot,\cdot),\mathcal{E}_{PK,0}(\cdot,\cdot)}(PK) = 0\right] - \frac{1}{2} \Bigg| \\
& + \Bigg| \frac{1}{2} \cdot \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k) : A^{\mathcal{E}_{PK,1}(\cdot,\cdot),\mathcal{E}_{PK,0}(\cdot,\cdot)}(PK) = 1\right] \\
& + \frac{1}{2} \cdot \Pr\left[(PK,SK) \leftarrow \mathsf{KeyGen}(1^k) : A^{\mathcal{E}_{PK,0}(\cdot,\cdot),\mathcal{E}_{PK,0}(\cdot,\cdot)}(PK) = 0\right] - \frac{1}{2} \Bigg| . \\
= \quad & \mathsf{Adv}_{\hat{A}_2} + \mathsf{Adv}_{\hat{A}_1}.
\end{aligned}
$$

Since (by our initial assumption) Adv_A was non negligible, the above implies that at least one of $\mathsf{Adv}_{\hat{A}_1}$ or $\mathsf{Adv}_{\hat{A}_2}$ must be non negligible. However, this would imply that at least one of $\hat{A}_1$ or $\hat{A}_2$ violate the semantic security of the encryption scheme, contradicting the assumption of the theorem. ■

2.1 The Hybrid Argument

We can generalize the proof technique used above so that it applies to any indistinguishable distributions (the technique is referred to as the "hybrid argument"). We formalize this idea now by first defining *computational indistinguishability*.

Definition 2 Let $\mathcal{X} = \{X_k\}$ and $\mathcal{Y} = \{Y_k\}$ be ensembles of distributions, where for all k, X_k and Y_k are distributions over the same space. $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable (written $\mathcal{X} \stackrel{c}{\equiv} \mathcal{Y}$) if the following is negligible (in k) for all PPT A:

$$
\left| \Pr\left[x \leftarrow X_k; A(x) = 1\right] - \Pr\left[y \leftarrow Y_k; A(y) = 1\right] \right|. \tag{4}
$$

♢

We will sometimes be informal and refer to "distributions" instead of "ensembles of distributions".

As an example of how this notation may be used, we give an equivalent definition of semantic security (for a single bit) in terms of computational indistinguishability. Namely, let

$$X_k \stackrel{\text{def}}{=} \{(PK, SK) \leftarrow \mathsf{KeyGen}(1^k); C \leftarrow \mathcal{E}_{PK}(0) : (PK, C)\}$$

and

$$Y_k \stackrel{\text{def}}{=} \{(PK, SK) \leftarrow \mathsf{KeyGen}(1^k); C \leftarrow \mathcal{E}_{PK}(1) : (PK, C)\}.$$

Then encryption scheme $(\mathsf{KeyGen}, \mathcal{E}, \mathcal{D})$ is semantically-secure (for encryption of a single bit) if and only if $\{X_k\} \stackrel{\text{c}}{\equiv} \{Y_k\}$.

Before continuing with our discussion of the "hybrid argument", we note the following useful properties of computational indistinguishability:

Claim 3 *If $\mathcal{X} \stackrel{\text{c}}{\equiv} \mathcal{Y}$ and $\mathcal{Y} \stackrel{\text{c}}{\equiv} \mathcal{Z}$ then $\mathcal{X} \stackrel{\text{c}}{\equiv} \mathcal{Z}$.*

Sketch of Proof (Informal) The proof relies on the *triangle inequality* (namely, the fact that for any real numbers a, b, c we have $|a - b| \leq |a - b| + |b - c|$) and the fact that the sum of two negligible functions is negligible. □

In fact, we can extend this claim as follows:

Claim 4 (Transitivity) *Given polynomially many distributions $\mathcal{X}_1, \ldots, \mathcal{X}_{\ell(k)}$ for which $\mathcal{X}_i \stackrel{\text{c}}{\equiv} \mathcal{X}_{i+1}$ for $i = 1, \ldots, \ell(k) - 1$, then $\mathcal{X}_1 \stackrel{\text{c}}{\equiv} \mathcal{X}_{\ell(k)}$*

Sketch of Proof (Informal) The proof again uses the triangle inequality along with the fact that the sum of a polynomial number of negligible functions remains negligible. □

Note that the claim does *not* hold for a super-polynomial number of distributions.

We now formalize the hybrid argument.

Claim 5 (Hybrid argument) *Let $\mathcal{X}^1, \mathcal{X}^2, \mathcal{Y}^1, \mathcal{Y}^2$ be efficiently sampleable[3] distributions for which $\mathcal{X}^1 \stackrel{\text{c}}{\equiv} \mathcal{Y}^1$ and $\mathcal{X}^2 \stackrel{\text{c}}{\equiv} \mathcal{Y}^2$. Then $(\mathcal{X}^1, \mathcal{X}^2) \stackrel{\text{c}}{\equiv} (\mathcal{Y}^1, \mathcal{Y}^2)$. (Note: if $\mathcal{X} = \{X_k\}$ and $\mathcal{Y} = \{Y_k\}$ are two distribution ensembles, the notation $(\mathcal{X}, \mathcal{Y})$ refers to the distribution ensemble $\{(X_k, Y_k)\}$ where the distribution (X_k, Y_k) is defined by $\{x \leftarrow X_k; y \leftarrow Y_k : (x, y)\}$.)*

Proof Instead of proving this by contradiction, we prove it directly. Let A be an arbitrary PPT algorithm trying to distinguish $(\mathcal{X}^1, \mathcal{X}^2)$ and $(\mathcal{Y}^1, \mathcal{Y}^2)$. We may construct a PPT adversary A_1 trying to distinguish $\mathcal{X}^1$ and $\mathcal{Y}^1$ as follows:

$$\begin{array}{l} \underline{A_1(1^k, z)} \\ \text{Choose random } x \leftarrow X_k^2 \\ \text{output } A(z, x) \end{array}$$

Clearly, A_1 runs in polynomial time (here is where we use the fact that all our distributions are efficiently sampleable). Since $\mathcal{X}^1 \stackrel{\text{c}}{\equiv} \mathcal{Y}^1$ we therefore know that the following must be

[3] A distribution ensemble $\mathcal{X} = \{X_k\}$ is *efficiently-sampleable* if we can generate an element according to distribution X_k in time polynomial in k.

negligible:

$$\begin{aligned}
&\left|\Pr\left[z \leftarrow X_k^1 : A_1(z) = 1\right] - \Pr\left[z \leftarrow Y_k^1 : A_1(z) = 1\right]\right| \qquad (5)\\
&= \left|\Pr\left[z \leftarrow X_k^1; x \leftarrow X_k^2 : A(z,x) = 1\right] - \Pr\left[z \leftarrow Y_k^1; x \leftarrow X_k^2 : A(z,x) = 1\right]\right|\\
&= \left|\Pr\left[x_1 \leftarrow X_k^1; x_2 \leftarrow X_k^2 : A(x_1,x_2) = 1\right] - \Pr\left[y_1 \leftarrow Y_k^1; x_2 \leftarrow X_k^2 : A(y_1,x_2) = 1\right]\right|,
\end{aligned}$$

where the last line is simply a renaming of the variables.

We may similarly construct a PPT algorithm A_2 trying to distinguish $\mathcal{X}^2$ and $\mathcal{Y}^2$ that runs as follows:

$\underline{A_2(1^k, z)}$
Choose random $y \leftarrow \mathcal{Y}_k^1$
Output $A(y, z)$

Here, since $\mathcal{X}^2 \stackrel{c}{\equiv} \mathcal{Y}^2$ we know that the following is negligible:

$$\begin{aligned}
&\left|\Pr\left[z \leftarrow X_k^2 : A_2(z) = 1\right] - \Pr\left[z \leftarrow Y_k^2 : A_2(z) = 1\right]\right|\\
&= \left|\Pr\left[y \leftarrow \mathcal{Y}_k^1; z \leftarrow X_k^2 : A(y,z) = 1\right] - \Pr\left[y \leftarrow Y_k^1; z \leftarrow Y_k^2 : A(y,z) = 1\right]\right|\\
&= \left|\Pr\left[y_1 \leftarrow Y_k^1; x_2 \leftarrow X_k^2 : A(y_1,x_2) = 1\right] - \Pr\left[y_1 \leftarrow Y_k^1; y_2 \leftarrow Y_k^2 : A(y_1,y_2) = 1\right]\right|.
\end{aligned}$$

Of course, what we are really interested in is how well A does at distinguishing $(\mathcal{X}^1, \mathcal{X}^2)$ and $(\mathcal{Y}^1, \mathcal{Y}^2)$. We can bound this quantity as follows:

$$\begin{aligned}
&\left|\Pr[x_1 \leftarrow X_k^1; x_2 \leftarrow X_k^2 : A(x_1,x_2) = 1] - \Pr[y_1 \leftarrow Y_k^1; y_2 \leftarrow Y_k^2 : A(y_1,y_2) = 1]\right|\\
&= \left|\Pr[x_1 \leftarrow X_k^1; x_2 \leftarrow X_k^2 : A(x_1,x_2) = 1] - \Pr\left[y_1 \leftarrow Y_k^1; x_2 \leftarrow X_k^2 : A(y_1,x_2) = 1\right]\right.\\
&\quad \left. + \Pr\left[y_1 \leftarrow Y_k^1; x_2 \leftarrow X_k^2 : A(y_1,x_2) = 1\right] - \Pr[y_1 \leftarrow Y_k^1; y_2 \leftarrow Y_k^2 : A(y_1,y_2) = 1]\right|\\
&\leq \left|\Pr[x_1 \leftarrow X_k^1; x_2 \leftarrow X_k^2 : A(x_1,x_2) = 1] - \Pr\left[y_1 \leftarrow Y_k^1; x_2 \leftarrow X_k^2 : A(y_1,x_2) = 1\right]\right|\\
&\quad + \left|\Pr\left[y_1 \leftarrow Y_k^1; x_2 \leftarrow X_k^2 : A(y_1,x_2) = 1\right] - \Pr[y_1 \leftarrow Y_k^1; y_2 \leftarrow Y_k^2 : A(y_1,y_2) = 1]\right|
\end{aligned}$$

(where we have again applied the triangle inequality). The last two terms are exactly Equations (5) and (6), and we know they are negligible. Since the sum of two negligible quantities is negligible, the distinguishing advantage of A is negligible, as desired. ■

This is called a "hybrid argument" for the following reason: Looking at the structure of the proof, we introduced the "hybrid" distribution $(\mathcal{Y}^1, \mathcal{X}^2)$ (which is not equal to either of the distributions we are ultimately interested in) and noted that $(\mathcal{X}^1, \mathcal{X}^2) \stackrel{c}{\equiv} (\mathcal{Y}^1, \mathcal{X}^2)$ and $(\mathcal{Y}^1, \mathcal{X}^2) \stackrel{c}{\equiv} (\mathcal{Y}^1, \mathcal{Y}^2)$ (this was the purpose of A_1 and A_2, respectively). Applying Claim 3 (which we essentially re-derived above) gives the desired result.

A similar argument can be used for combinations of poly-many ensembles instead of two ensembles, but we omit the details. Furthermore, a corollary of the above is that if $\mathcal{X} \stackrel{c}{\equiv} \mathcal{Y}$ then polynomially-many copies of $\mathcal{X}$ are indistinguishable from polynomially-many copies of $\mathcal{Y}$. Formally, let $\ell(k)$ be a polynomial and define $\mathcal{X}^\ell = \{X_k^\ell\}$ as follows:

$$X_k^\ell \stackrel{\text{def}}{=} (\overbrace{X_k, \ldots, X_k}^{\ell(k) \text{ times}})$$

(and similarly for $\mathcal{Y}^\ell$). Then $\mathcal{X}^\ell \stackrel{c}{\equiv} \mathcal{Y}^\ell$.

Strictly speaking, Claim 5 is not quite enough to yield Theorem 2 directly. The problem is the following: recall that if $(\mathsf{KeyGen}, \mathcal{E}, \mathcal{D})$ is a semantically-secure encryption scheme for a single bit then the following ensembles are computationally indistinguishable:

$$X_k \stackrel{\text{def}}{=} \{(PK, SK) \leftarrow \mathsf{KeyGen}(1^k); C \leftarrow \mathcal{E}_{PK}(0) : (PK, C)\}$$
$$Y_k \stackrel{\text{def}}{=} \{(PK, SK) \leftarrow \mathsf{KeyGen}(1^k); C \leftarrow \mathcal{E}_{PK}(1) : (PK, C)\}.$$

But then applying the hybrid argument directly only tells us that the following are indistinguishable:

$$(X_k, X_k) = \left\{ \begin{array}{c} (PK, SK), (PK', SK') \leftarrow \mathsf{KeyGen}(1^k) \\ C \leftarrow \mathcal{E}_{PK}(0); C' \leftarrow \mathcal{E}_{PK'}(0) \end{array} \quad : \quad (PK, C, PK', C') \right\}$$
$$(Y_k, Y_k) = \left\{ \begin{array}{c} (PK, SK), (PK', SK') \leftarrow \mathsf{KeyGen}(1^k) \\ C \leftarrow \mathcal{E}_{PK}(1); C' \leftarrow \mathcal{E}_{PK'}(1) \end{array} \quad : \quad (PK, C, PK', C') \right\};$$

here, encryption is done with respect to two *different* public keys, not a single key as desired. Even so, the hybrid technique is essentially what is used to prove Theorem 2 and we therefore refer to it as such. As a final remark, note that it is crucial in the proof of Theorem 2 that an adversary can generate random encryptions[4] as can be done in any public-key encryption scheme. In particular, an analogue of Theorem 2 does *not* hold for the case of private-key encryption, where an adversary may be unable to generate legal ciphertexts corresponding to an unknown key.

[4] In fact, this directly parallels the requirement in Claim 5 that the distribution ensembles be efficiently sampleable.

CMSC 858K — Advanced Topics in Cryptography	February 5, 2004

Lecture 4

Lecturer: Jonathan Katz

Scribe(s): Chiu Yuen Koo, Nikolai Yakovenko, Jeffrey Blank

1 Summary

The focus of this lecture is efficient public-key encryption. In the previous lecture, we discussed a public-key encryption scheme for 1-bit messages. However, to encrypt an ℓ-bit message, we can simply encrypt ℓ one-bit messages and send these (and we proved last time that this remains secure in the case of public-key encryption). Here, we first describe (briefly) how to combine public and private key encryption to obtain a public-key encryption scheme with the efficiency of a private-key scheme (for long messages). Next, we describe an efficient public key encryption scheme called *El Gamal encryption* [2] which is based on a particular number-theoretic assumption rather than the general assumption of trapdoor permutations. In the course of introducing this scheme, we discuss how it relies on the Discrete Logarithm Problem and the Decisional Diffie-Hellman Assumption.

2 Hybrid Encryption

A hybrid encryption scheme uses public-key encryption to encrypt a random symmetric key, and then proceeds to encrypt the message with that symmetric key. The receiver decrypts the symmetric key using the public-key encryption scheme and then uses the recovered symmetric key to decrypt the message.

More formally, let (KeyGen, $\mathcal{E}$, $\mathcal{D}$) be a secure public- key encryption scheme and ($\mathcal{E}'$, $\mathcal{D}'$) be a secure private-key encryption scheme. We can construct a secure hybrid encryption scheme (KeyGen", $\mathcal{E}''$,$\mathcal{D}''$) as follows:

- KeyGen" is the same as *KeyGen*, generating a public key pk and a secret key sk.
- $\mathcal{E}''_{pk}(m)$:
  1. $sk' \leftarrow \{0,1\}^k$
  2. $C_1 \leftarrow \mathcal{E}_{pk}(sk')$
  3. $C_2 \leftarrow \mathcal{E}'_{sk'}(m)$
- $\mathcal{D}''_{sk}(C_1, C_2)$:
  1. $sk' = \mathcal{D}_{sk}(C_1)$
  2. $m = \mathcal{D}'_{sk'}(C_2)$

The above scheme can be proven semantically-secure, assuming the semantic security of the underlying public- and private-key schemes. However, we will not give a proof so here.

3 The El Gamal Encryption Scheme

3.1 Groups

Before describing the El Gamal encryption scheme, we give a very brief overview of the group theory necessary to understand the scheme. We will need to concept of a *finite, cyclic group*, so we first introduce the concept of a finite group (actually, we will introduce what is known as an *Abelian group*, since we will not use non-Abelian groups in this class).

Definition 1 An Abelian group $\mathcal{G}$ is a finite set of elements along with an operation $*$ (written multiplicatively, although not necessarily corresponding to integer multiplication!) such that:

Closure For all $a, b \in \mathcal{G}$ we have $a * b \in \mathcal{G}$. Since we are using multiplicative notation, we also write $a * b$ as ab when convenient.

Associativity For all $a, b, c \in \mathcal{G}$, we have $(ab)c = a(bc)$.

Commutativity For all $a, b \in \mathcal{G}$ we have $ab = ba$.

Existence of identity There exists an element $1 \in \mathcal{G}$ such that $1 * a = a$ for all $a \in \mathcal{G}$. This element is called the identity of $\mathcal{G}$.

Inverse For all $a \in \mathcal{G}$ there exists an element $a^{-1} \in \mathcal{G}$ such that $aa^{-1} = 1$.

$\diamondsuit$

If n is a positive integer and $a \in \mathcal{G}$, the notation a^n simply refers to the product of a with itself n times. Just as we are used to, we have $a^0 = 1$ and $a^1 = a$ for all $a \in \mathcal{G}$. Let $q = |\mathcal{G}|$ (i.e., the number of elements in $\mathcal{G}$); this is known as the *order* of $\mathcal{G}$. A useful result is the following theorem:

Theorem 1 *Let $\mathcal{G}$ be a finite Abelian group of order q. Then $a^q = 1$ for all $a \in \mathcal{G}$.*

Proof We may actually give a simple proof of this theorem. Let $a_1, \ldots, a_q$ be the elements of $\mathcal{G}$, and let $a \in \mathcal{G}$ be arbitrary. We may note that the sequence of elements $aa_1, aa_2, \ldots, aa_q$ also contains exactly the elements of $\mathcal{G}$ (clearly, this sequence contains at most q distinct elements; furthermore, if $aa_i = aa_j$ then we can multiply by a^{-1} on both sides to obtain $a_i = a_j$ which is not the case). So:

$$\begin{aligned} & a_1 \cdot a_2 \cdots a_q \\ = \; & (aa_1) \cdot (aa_2) \cdots (aa_q) \\ = \; & a^q (a_1 \cdot a_2 \cdots a_q). \end{aligned}$$

Multiplying each side by $(a_1 \cdots a_q)^{-1}$, we see that $a^q = 1$. ■

We mention the following easy corollary:

Corollary 2 *Let $\mathcal{G}$ be a finite Abelian group of order q, and let n be a positive integer. Then $g^n = g^{n \bmod q}$.*

Proof Let $n = n_q \bmod q$ so that n can be written as $n = aq + n_q$ for some integer a. We then have $g^n = g^{aq+n_q} = (g^a)^q g^{n_q} = g^{n_q}$. ■

A finite group $\mathcal{G}$ of order q is *cyclic* if there exists an element $g \in \mathcal{G}$ such that the set $\{g^0, g^1, \ldots, g^{q-1}\}$ is all of $\mathcal{G}$. (Note that $g^q = g^0 = 1$ by the above theorem, so that the sequence would simply "cycle" if continued.) If such a g exists, it is called a *generator* of $\mathcal{G}$. As an example, consider the group $\mathbb{Z}_5^* = \{1, 2, 3, 4\}$ under multiplication modulo 5. Since $4^2 = 1$, the element 4 is *not* a generator. However, since $2^1 = 2, 2^2 = 4$, and $2^3 = 3$, element 2 is a generator and $\mathbb{Z}_5^*$ is cyclic.

All the groups we are going to deal with here will be cyclic, and will additionally have prime order (i.e., $|\mathcal{G}| = q$ and q is prime.) The following is a simple fact in such groups:

Lemma 3 *If $\mathcal{G}$ is an Abelian group with prime order q, then (1) $\mathcal{G}$ is cyclic; furthermore, (2) every element of $\mathcal{G}$ (except the identity) is a generator.*

Proof We do not prove that $\mathcal{G}$ is cyclic, but instead refer the reader to any book on group theory. However, assuming $\mathcal{G}$ is cyclic, we may prove the second part. Let g be a generator of $\mathcal{G}$, and consider an element $h \in \mathcal{G} \setminus \{1\}$. We know that $h = g^i$ for some i between 1 and $q - 1$. Consider the set $\mathcal{G}' = \{1, h, h^2, \ldots, h^{q-1}\}$. By the closure property, $\mathcal{G}' \subseteq \mathcal{G}$. On the other hand, if $h^x = h^y$ for some $1 \leq x < y \leq q - 1$ then $g^{xi} = g^{yi}$. By the Corollary given above, this implies that $xi = yi \bmod q$, or, equivalently, that q divides $(y - x)i$. However, since q is prime and both $(y - x)$ and i are strictly less than q, this cannot occur. This implies that all elements in $\mathcal{G}'$ are unique, and hence h is a generator. ■

An important fact about a cyclic group $\mathcal{G}$ of order q is that given a generator g, every element of $h \in \mathcal{G}$ satisfies $h = g^s$ for exactly one s between 0 and $q - 1$ (this follows immediately from the definition of a generator). In this case, we say $\log_g h = s$ (the previous fact indicates that this is well-defined as long as g is a generator) and call s the discrete logarithm of h to the base g. Discrete logarithms satisfy many of the rules you are familiar with for logarithms over the reals; for example (always assuming g is a generator), we have $\log_g(h_1 h_2) = \log_g h_1 + \log_g h_2$ for all $h_1, h_2 \in \mathcal{G}$.

For our applications to cryptographic protocols, we will consider groups of very large order (on the order of $q \approx 2^{100}$ or more!). When dealing with such large numbers, it is important to verify that the desired arithmetic operations can be done *efficiently*. As usual, we associate "efficient" with "polynomial time". The only subtlety here is that the running time should be polynomial in the *lengths* of all inputs (and not their absolute value); so, for example, computing g^x for some generator g in some group $\mathcal{G}$ should require time polynomial in $|x|$ (equivalently, the number of bits needed to describe x) and $|q|$, rather than polynomial in x and q. This clearly makes a big difference — an algorithm running in 2^{100} steps is infeasible, while one running in 100 steps certainly is!

We will take it as a given that all the groups with which we will deal support efficient "base" operations such as multiplication, equality testing, and membership testing. (Yet, for some groups that are used in cryptography, verifying these properties is non-trivial!) However, we do *not* assume that exponentiation is efficient, but will instead show that it can be done efficiently (assuming that multiplication in the group can be done efficiently).

To see that this is not entirely trivial, consider the following naive algorithm to compute g^x for some element g in some group $\mathcal{G}$ (the exponent x is, of course, a positive integer):

```
exponentiate_naive( g , x ) {
    ans = 1;
    if x ?= 0 return 1;
    while(x ≥ 1){
        ans = ans * g;
        x = x − 1;
    }
    return ans; }
```

(In the above algorithm, the expression $\mathsf{ans} * g$ means multiplication in the *group* and not over the integers.) However, this algorithm requires time $O(x)$ to run (there are x iterations of the loop), which is unacceptable and not polynomial time! However, we can improve this by using *repeated squaring*. First, note that:

$$g^x = \begin{cases} (g^{\frac{x}{2}})^2 & \text{if x is even} \\ g(g^{\frac{x-1}{2}})^2 & \text{if x is odd} \end{cases}.$$

This leads us to the following algorithm for exponentiation:

```
exponentiate_efficient (g, x) {
        if (x ?= 0) return 1;
        tmp = 1, ans = g;
        // we maintain the invariant that tmp * ans^x is our answer
        while (x > 1) {
                if (x is odd) {
                        tmp = tmp * ans;
                        x = x − 1; }
                if (x > 1) {
                        ans = ans * ans;
                        x = x/2; }
                }
        return tmp * ans; }
```

(Again, the " $*$ " in the above code refers to multiplication in the group, not over the integers. However, expressions involving x *are* performed over the integers.) Note that in each execution of the loop the value of x is decreased by at least half, and thus the number of executions of this loop is $O(\log x) = O(|x|)$. Thus, the algorithm as a whole runs in polynomial time.

From the above, we see that given a generator g and an integer x, we can compute $h = g^x$ in polynomial time. What about the inverse problem of finding x given h and g? This is known as the *discrete logarithm problem* which we define next.

3.2 The Discrete Logarithm Problem

The discrete logarithm problem is as follows: given generator g and random element $h \in \mathcal{G}$, compute $\log_g h$. For many groups, this problem is conjectured to be "hard"; this is referred

to as the *discrete logarithm assumption* which we make precise now. In the following, we let $\mathsf{GroupGen}$ be a polynomial time algorithm which on input 1^k outputs a description of a cyclic group $\mathcal{G}$ of order q (with $|q| = k$ and q not necessarily prime), and also outputs q and a generator $g \in \mathcal{G}$. (Note that $\mathsf{GroupGen}$ may possibly be deterministic.) The *discrete logarithm assumption* is simply that the assumption that the discrete logarithm problem is hard for $\mathsf{GroupGen}$, where this is defined as follows:

Definition 2 The *discrete logarithm problem is hard for* $\mathsf{GroupGen}$ if the following is negligible for all PPT algorithms A:

$$\Pr[(\mathcal{G}, q, g) \leftarrow \mathsf{GroupGen}(1^k); h \leftarrow \mathcal{G}; x \leftarrow A(\mathcal{G}, q, g, h) : g^x = h].$$

$\diamondsuit$

Sometimes, if the discrete logarithm problem is hard for $\mathsf{GroupGen}$ and $\mathcal{G}$ is a group output by $\mathsf{GroupGen}$, we will informally say that the discrete logarithm problem is hard in $\mathcal{G}$.

We provide an example of a $\mathsf{GroupGen}$ for which the discrete logarithm assumption is believed to hold: Let $\mathsf{GroupGen}$ be an algorithm which, on input 1^k, generates a random prime q of length k (note that this can be done efficiently via a randomized algorithm), and let $\mathcal{G} = \mathbb{Z}_q^*$. It is known that this forms a cyclic group of order $q - 1$ (not a prime). It is also known how to efficiently find a generator g of $\mathcal{G}$ via a randomized algorithm which we do not describe here. Let $(\mathcal{G}, q, g)$ be the output of $\mathsf{GroupGen}$.

We now describe the El Gamal encryption scheme whose security is related to (but does *not* follow from) the discrete logarithm assumption:

Key generation $\mathsf{Gen}(1^k)$:
$(\mathcal{G}, q, g) \leftarrow \mathsf{GroupGen}(1^k)$
Choose $x \leftarrow \mathbb{Z}_q$; set $y = g^x$
Output $PK = (\mathcal{G}, q, g, y)$ and $SK = x$

Encryption $\mathcal{E}_{pk}(m)$ (where $m \in \mathcal{G}$):
Pick $r \leftarrow \mathbb{Z}_q$
Output $\langle g^r, y^r m \rangle$

Decryption $\mathcal{D}_{sk}(A, B)$:
Compute $m = \frac{B}{A^x}$

Correctness of decryption follows from $\frac{y^r m}{(g^r)^x} = \frac{y^r m}{(g^x)^r} = \frac{y^r m}{y^r} = m$.

The discrete logarithm problem implies that no adversary can determine the secret key given the public key (can you prove this?). However, this alone is *not* enough to guarantee semantic security! In fact, we can show a particular group for which the discrete logarithm assumption (DLA) is believed to hold, yet the El Gamal encryption scheme is *not* semantically secure. Namely, consider groups $\mathbb{Z}_p^*$ for p prime, as discussed earlier. We noted that the DLA is believed to hold in groups of this form. However, it is also known how to determine in polynomial time whether a given element of $\mathbb{Z}_p^*$ is a *quadratic residue* or not (an element $y \in \mathbb{Z}_p^*$ is a quadratic residue if there exists an $x \in \mathbb{Z}_p^*$ such that $x^2 = y$). Furthermore, a generator g of $\mathbb{Z}_p^*$ cannot be a quadratic residue. These observations leads to a direct attack on the El Gamal scheme (we sketch the attack here, but let the reader

fill in the details or refer to [1, Section 9.5.2]): output (m_0, m_1) such that m_0 is a quadratic residue but m_1 is not. Given a ciphertext $\langle A, B\rangle$, where $A = g^r$ and $B = y^r m_b$ for some r, we can determine in polynomial time whether y^r is a quadratic residue or not (for example, if A or y are quadratic residues then at least one of r, x is even and thus $y^r = y^{xr}$ is also a quadratic residue). But then by looking at B we can determine whether m_b is a quadratic residue or not (e.g., if y^r is a non-residue and B is a residue, then it must be the case that m_b was not a quadratic residue), and hence determine which message was encrypted.

Evidently, then, we need need a stronger assumption about $\mathsf{GroupGen}$ in order to prove that El Gamal encryption is semantically secure.

3.3 The Decisional Diffie-Hellman (DDH) Assumption

Informally, the DDH assumption is that it is hard to distinguish between tuples of the form (g, g^x, g^y, g^{xy}) and (g, g^x, g^y, g^z), where g is a generator and x, y, z are random. More formally, $\mathsf{GroupGen}$ satisfies the DDH assumption if the DDH problem is hard for $\mathsf{GroupGen}$, where this is defined as follows:

Definition 3 The *DDH problem is hard for* $\mathsf{GroupGen}$ if the following distributions are computationally indistinguishable (cf. the definition from Lecture 3):

$$\{(\mathcal{G}, q, g) \leftarrow \mathsf{GroupGen}(1^k); x, y, z \leftarrow \mathbb{Z}_q : (\mathcal{G}, q, g, g^x, g^y, g^z)\}$$

and

$$\{(\mathcal{G}, q, g) \leftarrow \mathsf{GroupGen}(1^k); x, y \leftarrow \mathbb{Z}_q : (\mathcal{G}, q, g, g^x, g^y, g^{xy})\}.$$

♢

We call tuples chosen from the first distribution "random tuples" and tuples chosen from the second distribution "DH tuples". (Note that this is an abuse of terminology, since there exist tuples in the support of both distributions. But when we say that $\vec{g}$ is a random tuple we simply mean that $\vec{g}$ was drawn from the first distribution above.) Also, if the DDH assumption holds for $\mathsf{GroupGen}$ and $\mathcal{G}$ is a particular group output by $\mathsf{GroupGen}$ then we informally say that the DDH assumption holds in $\mathcal{G}$.

Security of the El Gamal encryption scheme turns out to be equivalent to the DDH assumption. We prove the more interesting direction in the following theorem.

Theorem 4 *Under the DDH assumption, the El Gamal encryption scheme is secure in the sense of indistinguishability.*

Proof (Note: what we really mean here is that if the DDH assumption holds for $\mathsf{GroupGen}$, and this algorithm is used in the key generation phase of El Gamal encryption as described above, then that particular instantiation of El Gamal encryption is secure.)

Assume a PPT adversary A attacking the El Gamal encryption scheme in the sense of indistinguishability. Recall this means that A outputs messages (m_0, m_1), is given a random encryption of m_b for random b, and outputs guess b'. We will say that A succeeds if $b' = b$ (and denote this event by SUCC), and we are ultimately interested in $\Pr_A[\mathsf{Succ}]$.

We construct an adversary A' as follows:

$\underline{A'(\mathcal{G}, q, g_1, g_2, g_3, g_4)}$
$PK = (g_1, g_2)$
run $A(PK)$ and get messages (m_0, m_1)
$b \leftarrow \{0,1\}$
$C = \langle g_3, g_4 m_b \rangle$
run $A(PK, C)$ to obtain b'
output 1 iff $b' = b$

Let Rand be the event that (g_1, g_2, g_3, g_4) are chosen from the distribution of random tuples, and let DH be the event that they were chosen from the distribution on DH tuples. Since the DDH assumption holds in $\mathcal{G}$ and A' is a PPT algorithm we know that the following is negligible:

$$\left|\Pr[A' = 1|\mathsf{DH}] - \Pr[A' = 1|\mathsf{Rand}]\right|.$$

Next, we claim that $\Pr[A' = 1|\mathsf{DH}] = \Pr_A[\mathsf{Succ}]$. To see this, note that when DH occurs we have $g_2 = g_1^x$, $g_3 = g_1^r$, and $g_4 = g_1^{xr} = g_2^r$ for some x and r chosen at random. But then the public key and the ciphertext are distributed exactly as they would be in a real execution of the El Gamal encryption scheme, and since A' outputs 1 iff A succeeds, the claim follows.

To complete the proof, we show that $\Pr[A' = 1|\mathsf{Rand}] = 1/2$ (do you see why this completes the proof?). Here, we know that g_4 is uniformly distributed in $\mathcal{G}$ independent of g_1, g_2, or g_3. In particular, then, the second component of the ciphertext given to A is uniformly distributed in $\mathcal{G}$ independent of the message being encrypted (and, in particular, independent of b). Thus, A' has *no information* about b — even an all-powerful A cannot predict b in this case with probability different from $1/2$ (we assume that A must always output some guess $b' \in \{0,1\}$). Since A' outputs 1 iff A succeeds, we may conclude that $\Pr[A' = 1|\mathsf{Rand}] = 1/2$. ■

References

[1] M. Bellare and P. Rogaway. Introduction to Modern Cryptography. Notes available from http://www-cse.ucsd.edu/users/mihir/cse207/classnotes.html.

[2] T. El Gamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory* 31(4): 469–472 (1985).

CMSC 858K — Advanced Topics in Cryptography February 10, 2004

Lecture 5

Lecturer: Jonathan Katz *Scribe(s):* *Rengarajan Aravamudhan*
Morgan Kleene
Nan Wang
Aaron Zollman

1 Semantic Security May Not be Enough

The following trick can be played on the El-Gamal encryption scheme which, under the Decisional Diffie-Hellman Hypothesis (DDH) is semantically secure. We show how an attacker can subvert a sealed-bid auction conducted using El-Gamal encryption, where the auction is won by the bidder who submits the higher bid. Assume the auctioneer has public key $PK = (g, y = g^x)$, and let the bid of the first bidder be m.

Bidder 1	$C \leftarrow (g^r, y^r \cdot m)$ (where r is random)	$\overset{C=(C_1,C_2)}{\longrightarrow}$	Auctioneer decrypts m
Bidder 2	$C' = (C_1, C_2 \cdot \alpha)$ (where $\alpha = 2$)	$\overset{C'}{\longrightarrow}$	Auctioneer decrypts $m' = m \cdot \alpha$

Although Bidder 2 has no idea what was bid (he doesn't even know his own bid!), he is still able to outbid bidder 1 by a factor of α.

The following system for verifying credit cards is also malleable. A user has a credit card number $C_1, C_2, C_3, ..., C_{48}$ (where each C_i represents one bit) which is encrypted, bit-wise, with the merchant's public key pk and sent to the merchant as follows:

$$E_{pk}(C_1), E_{pk}(C_2), E_{pk}(C_3), ..., E_{pk}(C_{48})$$

The merchant then immediately responds ACCEPT or REJECT, indicating whether the credit card is valid. Now, an adversary need not decrypt the message to recover the credit card: consider what happens if the first element of the above ciphertext is replaced by $E_{pk}(0)$ (which an attacker can compute since the public key is available!) — if the message is accepted by the merchant, the first bit of the credit card must be zero; if rejected, it is one. Continuing in this way, the adversary learns the entire credit card number after 48 such attempts.

These two examples motivate the concept of *malleability*. Informally, an encryption scheme is *malleable* if, given an encryption C of some message M, it is possible to construct a different ciphertext C' decrypting to some "related" message M'. Non-malleability precludes the attacks shown above (in particular). Attacks that are thematically similar to the ones given above have been implemented [2], although they are much more complicated than the above examples.

This motivates the development of stronger notions of security preventing the above attacks. It turns out that non-malleability is (for the cases of interest here) equivalent [1]

to a security property which is simpler to define called *security against chosen-ciphertext attacks*. We may further consider security against *non-adaptive chosen-ciphertext attacks* (**CCA1**) or security against *adaptive chosen-ciphertext attack* (**CCA2**); we define both of these now.

Definition 1 [IND-CCA2] An encryption scheme is secure against adaptive chosen-ciphertext attacks (CCA2) if the following is negligible for all PPT adversaries $\mathcal{A}$:

$$\left|\Pr\left[\begin{array}{c}(pk,sk)\leftarrow \mathsf{KeyGen}(1^k);(m_0,m_1)\leftarrow \mathcal{A}^{\mathcal{D}_{sk}(\cdot)}(pk);\\ b\leftarrow\{0,1\};c\leftarrow\mathcal{E}_{pk}(m_b);b'\leftarrow\mathcal{A}^{\mathcal{D}_{sk}(\cdot)}(pk,c)\end{array} \;:\; b=b'\right]-\frac{1}{2}\right|$$

where $\mathcal{A}$ cannot query $\mathcal{D}_{sk}(c)$. ♢

We note that for non-adaptive chosen-ciphertext attacks (CCA1) the adversary is only allowed to query $\mathcal{D}_{sk}(\cdot)$ in the first stage (i.e., before being given the ciphertext c).

2 Zero-Knowledge Proofs

Toward our eventual goal of designing encryption schemes secure against chosen-ciphertext attacks, we define a class of exchanges for which it holds that one party is able to convince another that he holds some information without revealing the information itself. We first review what kinds of computation we consider feasible and then discuss the actual exchanges that have been devised.

2.1 NP-Completeness

The kinds of computations that can be carried out efficiently are typically considered to be those that can be done in polynomial time. We consider computational problems as the recognition of a set of strings, referred to as a *language*. We say that a Turing machine M accepts a language L if: $x \in L \Leftrightarrow M(x)$ outputs "accept". We will simply let a "1" signify acceptance.

There are two sets of computational problems which are of special importance. The first is the set of languages that can be decided in polynomial time, denoted P. Formally, a language L is in P if there exists a Turing machine M which takes at most $p(|x|)$ steps for some polynomial p (where $|x|$ denotes the length of its input string x), and accepts if and only if $x \in L$. The class NP is the set of languages for which there exist proofs of membership that can be checked in polynomial time. Formally, a language L is in the class NP if there exists a polynomial-time Turing machine M[1] such that:

$$x \in L \Leftrightarrow \text{there exists a string } w_x \text{ s.t. } M(x, w_x) = 1.$$

A w_x of this sort is called a *witness for* x. One can think of this as an efficiently-verifiable "proof" that $x \in L$.

Intuitively, if we can use a solution to problem A to solve problem B it seems as if problem A is in some sense "(at least) as hard as" problem B. This is formalized by the

[1] By convention, the running time of a Turing machine taking multiple inputs is measured as a function of the length of its *first* input.

notion of a *polynomial-time reduction* between two languages. We say that language L_1 is poly-time reducible to language L_2 if there exists a function $f : \{0,1\}^* \to \{0,1\}^*$ such that: (1) f is computable in polynomial time, and (2) $x \in L_1$ if and only if $f(x) \in L_2$. We will sometimes abbreviate this by writing $L_1 \leq_p L_2$. Note that if $L_1 \leq_p L_2$ and $L_2 \in P$ (i.e., L_2 can be decided in polynomial time) then L_1 can be decided in polynomial time using the following algorithm: Given a string x, compute $x' = f(x)$ and then decide whether $x' \in L_2$. Similarly, if $L_1 \leq_p L_2$ and $L_2 \in NP$ then $L_1 \in NP$ as well.

There are languages which, in a certain sense, are "the hardest languages" in NP in the sense that all problems in NP are poly-time reducible to them. These problems are called NP *complete*. Note that if an NP-complete problem could be shown to be in P, then *all* of NP would be in P, by the discussion above. A classic example of an NP-complete language is satisfiability (i.e., given a boolean formula does there exist an assignment of truth values to variables such that the formula evaluates to true). There are a variety of other well-known NP-complete problems; the ones we will encounter in this class are:

- Hamiltonian Cycle: This is the lnaguage $\{ G : G$ is a graph which contains a Hamilton cycle $\}$. (A *Hamiltonian cycle* is a cycle in which each vertex appears exactly once.)
- 3-colorability: This is the language $\{ G = (V, E) : G$ is a graph and there exists a function $\varphi : V \to \{Red,\ Green,\ Blue\}$ such that if $\{u, v\} \in E$, $\varphi(u) \neq \varphi(v)$ $\}$

Looking (way) ahead, we will eventually show a proof system for all of NP by showing a proof system for a particular NP-complete language.

2.2 Non-Interactive Proof Systems

We first informally discuss the notion of an *interactive proof system*. Here we have a prover P and a polynomial-time verifier V who both have some common input x, and the prover wants to convince V that $x \in L$ for some agreed-upon language L. The prover will attempt to do this by interacting with the verifier in multiple communication rounds. Informally, we would like that if $x \in L$ then the prover can always make the verifier accept, while if $x \notin L$ then no matter what the prover does the verifier should reject with high probability.

We first give two trivial examples: we claim that all languages in P have an interactive proof system with 0 rounds. Here, P does not communicate with V at all, but V merely decides on its own whether $x \in L$ (it can do this since $L \in P$). Next, we claim that any $L \in NP$ has a 1-round interactive proof system in which P simply sends to V the witness for x (assuming $x \in L$), and V verifies this. Note that the definition of NP implies that V will always accept if $x \in L$ (assuming P wants to make V accept) and that P can *never* fool V into accepting if $x \notin L$.

Things get more interesting when we allow more rounds of interaction (and languages outside of NP can be shown to have interactive proof systems), but this is for a class in complexity theory. We will be more interested in strengthening the model of interactive proofs so as to require also that V does not learn anything from interacting with the prover other than the fact that $x \in L$. So far we have described a scheme that does not specify what V may learn from the interaction. The below definition is motivated by the desire to let V ascertain with high probability whether a particular string is in its language of interest without actually learning anything about *why* it is in the language. To put it another way, V

should not learn anything from P that he could not have figured out himself. We formalize this now for the case of non-interactive proofs (where there is additionally a common random string available to both parties), and later in the course we will formalize it for interactive proofs. See [3, 4] for more details.

Definition 2 A pair of PPT algorithms[2] (P, V) is a *non-interactive zero-knowledge (NIZK) proof system* for a language $L \in NP$ if:

Completeness For any $x \in L$ (with $|x| = k$) and witness w for x, we have:

$$\Pr\left[r \leftarrow \{0,1\}^{\mathsf{poly}(k)}; \pi \leftarrow P(r, x, w) : V(r, x, \pi) = 1\right] = 1.$$

In words: a random string r is given to both parties. P is given r, x, and the witness that $x \in L$, and produces a proof π which he sends to V. The verifier, given r, x, and π, decides whether to accept or reject. The above just says that if $x \in L$ and everyone is honest, then V always accepts.

Soundness If $x \notin L$ then $\forall P^*$ (even all-powerful P^*), the following is negligible (in $|x| = k$):

$$\Pr\left[r \leftarrow \{0,1\}^{\mathsf{poly}(k)}; \pi \leftarrow P^*(r, x) : V(r, x, \pi) = 1\right].$$

Zero-knowledge There exists a PPT simulator S such that for all $x \in L$ (with $|x| = k$, the security parameter) and any witness w for x, the following distributions are computationally indistinguishable:

1. $\{r \leftarrow \{0,1\}^{\mathsf{poly}(k)}; \pi \leftarrow P(r, x, w) : (r, x, \pi)\}$
2. $\{(r, \pi) \leftarrow S(x) : (r, x, \pi)\}$.

$\diamondsuit$

The last condition restricts the information V may obtain from P. Intuitively, it says that if V has "learned" anything from interacting with P he could also have learned it by himself, using the polynomial-time simulator S.

References

[1] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. Crypto '98.

[2] D. Bleichenbacher. Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS. Crypto '98.

[3] M. Blum, P. Feldman, and S. Micali. Non-interactive Zero-Knowledge and its Applications. STOC '88.

[4] O. Goldreich. *Foundations of Cryptography, vol. 1: Basic Tools*. Cambridge University Press, 2001.

[2] Here, we require that P run in probabilistic polynomial time as well, since we are going to eventually want to use P to construct efficient cryptographic protocols!

CMSC 858K — Advanced Topics in Cryptography February 12, 2004

Lecture 6

Lecturer: Jonathan Katz *Scribe(s):* *Omer Horvitz* *Zhongchao Yu* *John Trafton* *Akhil Gupta*

1 Introduction

In this lecture, we show how to construct a public-key encryption scheme secure against non-adaptive chosen-ciphertext attacks, given a semantically-secure public-key encryption scheme and an adaptively-secure non-interactive zero-knowledge proof system (in the common random string model).

2 Adaptively-Secure Non-Interactive Zero-Knowledge

We begin with a definition of a basic case of non-interactive zero-knowledge.

Definition 1 A pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$ is a *non-interactive zero-knowledge* (*NIZK*) proof system for a language $L \in \text{NP}$ if there exists some polynomial poly such that:

1. Completeness (for $x \in L$, $\mathcal{P}$ generates proofs that $\mathcal{V}$ accepts): For all $x \in L \cap \{0,1\}^k$ and all witnesses w for x,

$$\Pr[r \leftarrow \{0,1\}^{\text{poly}(k)}; \Pi \leftarrow \mathcal{P}(r, x, w) : \mathcal{V}(r, x, \Pi) = 1] = 1.$$

2. Soundness (for $x \notin L$, no prover can generate proofs that $\mathcal{V}$ accepts with better than negligible probability): For all $x \in \{0,1\}^k \setminus L$ and all (possibly unbounded) algorithms $\mathcal{P}^*$, the following is negligible in k:

$$\Pr[r \leftarrow \{0,1\}^{\text{poly}(k)}; \Pi \leftarrow \mathcal{P}^*(r, x) : \mathcal{V}(r, x, \Pi) = 1].$$

3. Zero-knowledge (for $x \in L$, the view of any verifier can be efficiently simulated without knowledge of a witness): There exists a PPT algorithm Sim such that for any $x \in L \cap \{0,1\}^k$ and any witness w for x, the following ensembles are computationally indistinguishable:

$$(1) \quad \left\{ r \leftarrow \{0,1\}^{\text{poly}(k)}; \Pi \leftarrow \mathcal{P}(r, x, w) : (r, x, \Pi) \right\}_k$$
$$(2) \quad \{(r, \Pi) \leftarrow \mathsf{Sim}(x) : (r, x, \Pi)\}_k .$$

$\diamondsuit$

For our purposes, we need to strengthen the definition in two ways. In the soundness requirement, we would like to also protect against a prover who chooses $x \notin L$ *after* seeing the common random string r. In the zero-knowledge requirement, we would like to make the simulator's job a little harder by making it output a simulated common random string

r first, and only then supplying it with an $x \in L$ for which it needs to generate a simulated proof. In particular, such an x may be chosen adaptively (by an adversary, say) based on r. In the following, we use $\Pr_{\mathsf{E}}[\mathsf{A}]$ to denote the probability that event A occurs in the probabilistic experiment E.

Definition 2 A pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$ is an *adaptive, non-interactive zero-knowledge* (*aNIZK*[1]) proof system for a language $L \in \text{NP}$ if there exists a polynomial poly such that:

1. Completeness: Same as above.

2. Soundness (now, the cheating prover may choose $x \notin L$ *after* seeing r): For all (possibly unbounded) algorithms $\mathcal{P}^*$, the following is negligible in k:

$$\Pr\left[r \leftarrow \{0,1\}^{\text{poly}(k)}; (x, \Pi) \leftarrow \mathcal{P}^*(r) : \mathcal{V}(r, x, \Pi) = 1 \wedge x \in \{0,1\}^k \setminus L\right].$$

3. Zero-knowledge (simulator must output r first, is then given x, and asked to produce a simulated proof): Let $(\mathsf{Sim}_1, \mathsf{Sim}_2)$, (A_1, A_2) be a pair of two-staged algorithms (we may assume that the first stage outputs some state information which is passed as input to the second stage; this will be implicit). Consider the following experiments:

Game $\mathsf{ZK}_{\text{real}}$		Game ZK_{sim}	
$r \leftarrow \{0,1\}^{\text{poly}(k)}$		$r \leftarrow \mathsf{Sim}_1(1^k)$	
$(x, w) \leftarrow A_1(r)$	$(x \in L \cap \{0,1\}^k)$	$(x, w) \leftarrow A_1(r)$	$(x \in L \cap \{0,1\}^k)$
$\Pi \leftarrow \mathcal{P}(r, x, w)$		$\Pi \leftarrow \mathsf{Sim}_2(x)$	
$b \leftarrow A_2(r, x, \Pi)$		$b \leftarrow A_2(r, x, \Pi)$	

   We require that there exist a PPT *simulator* $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that for any PPT algorithm (A_1, A_2) the following is negligible in k:

$$\left|\Pr_{\mathsf{ZK}_{\text{real}}}[A_2 \text{ outputs } 0] - \Pr_{\mathsf{ZK}_{\text{sim}}}[A_2 \text{ outputs } 0]\right|.$$

   Equivalently, the "real" game $\mathsf{ZK}_{\text{real}}$ is computationally indistinguishable from the "simulated" game ZK_{sim}. (In other words, the adversary cannot tell whether it is participating in the first or the second experiment (except with negligible probability). A_2's output can be thought of as its guess towards which experiment it is in. This means that the simulator is able to simulate the adversary's view in the real execution.)

$\diamondsuit$

To simplify the notation a little, we will usually drop the stage identifier for A; when we refer to A's output in the experiment, we will mean A_2's output.

3 A Public-Key Encryption Scheme Secure Against Non-Adaptive Chosen-Ciphertext Attacks

Let $(\mathsf{Gen}, \mathcal{E}, \mathcal{D})$ be a public-key encryption scheme and $(\mathcal{P}, \mathcal{V})$ be an adaptively-secure NIZK proof system for languages in NP. The following construction is due to Naor and Yung [1].

[1] This notation is non-standard.

$\underline{\mathsf{Gen}^*(1^k)}$
$(pk_1, sk_1) \leftarrow \mathsf{Gen}(1^k)$;
$(pk_2, sk_2) \leftarrow \mathsf{Gen}(1^k)$;
$r \leftarrow \{0,1\}^{\mathrm{poly}(k)}$;
$pk^* = (pk_1, pk_2, r)$;
$sk^* = sk_1$

$\underline{\mathcal{E}^*_{(pk_1,pk_2,r)}(m)}$
$w_1, w_2 \leftarrow \{0,1\}^*$;
$c_1 = \mathcal{E}_{pk_1}(m; w_1)$;
$c_2 = \mathcal{E}_{pk_2}(m; w_2)$;
$\Pi \leftarrow \mathcal{P}(r, (c_1, c_2), (w_1, w_2, m))$;
Output (c_1, c_2, Π)

$\underline{\mathcal{D}^*_{sk_1}(c_1, c_2, \Pi)}$
If $\mathcal{V}(r, (c_1, c_2), \Pi) = 0$
  Output $\perp$;
else
  Output $\mathcal{D}_{sk_1}(c_1)$

A few words of explanation are due here. For key generation, we use the underlying key-generation algorithm to produce two pairs of (public, private) keys, publish the public keys and a random string r (to serve as the common random string for the proof system), and keep the first underlying private key as our private key (we discard the second private key). For encryption, we use the underlying algorithm to encrypt the given message m *twice*, under both pk_1 and pk_2, with the random tapes of the encryption algorithm fixed to w_1, w_2, respectively.[2] (For a probabilistic algorithm $A(\cdot)$, the notation $A(\cdot; w)$ is used to denote that A's random tape is fixed to a particular $w \in \{0,1\}^*$.) We then use our prover to generate a proof that (c_1, c_2) are encryptions of the same message under pk_1, pk_2 in the underlying scheme; i.e., $(c_1, c_2) \in L$ where

$$L = \{(c_1, c_2) \,|\exists m, w_1, w_2 \text{ such that } c_1 = \mathcal{E}_{pk_1}(m; w_1), c_2 = \mathcal{E}_{pk_2}(m; w_2)\}\,,$$

using w_1, w_2, and m as witnesses. Note that $L \in$ NP. We send the ciphertexts and the proof to the receiver. For decryption, we use the underlying decryption algorithm on the first ciphertext, if the provided proof verifies correctly.

Theorem 1 *Assuming that* $(\mathsf{Gen}, \mathcal{E}, \mathcal{D})$ *is semantically secure and that* $(\mathcal{P}, \mathcal{V})$ *is an adaptively-secure NIZK proof system,* $(\mathsf{Gen}^*, \mathcal{E}^*, \mathcal{D}^*)$ *is secure against non-adaptive ("lunchtime"*[3]*) chosen-ciphertext attacks (i.e., is CCA1 secure).*

The remainder of these notes is the beginning of a proof of this theorem (we continue the proof next lecture). Let A be a two-staged PPT algorithm, where the first stage has access to an oracle. Consider the following two experiments (their differences are in boldface):

Game $\mathsf{CCA1}_0$
$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^k)$
$r \leftarrow \{0,1\}^{\mathrm{poly}(k)}$
$pk^* = (pk_1, pk_2, r)$, $sk^* = sk_1$
$(m_0, m_1) \leftarrow A^{\mathcal{D}^*_{sk^*}(\cdot)}(pk^*)$
$w_1, w_2 \leftarrow \{0,1\}^*$
$c_1 = \mathcal{E}_{pk_1}(\boldsymbol{m_0}; w_1)$, $c_2 = \mathcal{E}_{pk_2}(\boldsymbol{m_0}; w_2)$
$\Pi \leftarrow \mathcal{P}(r, (c_1, c_2), (w_1, w_2, \boldsymbol{m_0}))$
$b \leftarrow A(pk^*, c_1, c_2, \Pi)$

Game $\mathsf{CCA1}_1$
$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^k)$
$r \leftarrow \{0,1\}^{\mathrm{poly}(k)}$
$pk^* = (pk_1, pk_2, r)$, $sk^* = sk_1$
$(m_0, m_1) \leftarrow A^{\mathcal{D}^*_{sk^*}(\cdot)}(pk^*)$
$w_1, w_2 \leftarrow \{0,1\}^*$
$c_1 = \mathcal{E}_{pk_1}(\boldsymbol{m_1}; w_1)$, $c_2 = \mathcal{E}_{pk_2}(\boldsymbol{m_1}; w_2)$
$\Pi \leftarrow \mathcal{P}(r, (c_1, c_2), (w_1, w_2, \boldsymbol{m_1}))$
$b \leftarrow A(pk^*, c_1, c_2, \Pi)$

To prove the scheme CCA1 secure, we need to show that A cannot distinguish the above two games; i.e.,to show that the following is negligible: $|\Pr_{\mathsf{CCA1}_0}[b = 0] - \Pr_{\mathsf{CCA1}_1}[b = 0]|$.

[2]The notation $w_1 \leftarrow \{0,1\}^*$ just means that a "long enough" random string is chosen.

[3]That is, the adversary is assumed to be able to "play" with the decryption oracle while people are out for lunch, but not afterward when he gets the challenge ciphertext.

To that effect, we introduce a sequence of intermediate games, and show that A cannot distinguish each game from its subsequent one; the theorem will then follow.

Let $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ be the simulator for our proof system. In the first game, we replace the random string and legitimate proof of game $\mathsf{CCA1}_0$ with a simulated random string and simulated proof. Once again, the differences between the new game and game $\mathsf{CCA1}_0$ are highlighted.

Game 1
$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^k)$
$r \leftarrow \mathbf{Sim_1(1^k)}$
$pk^* = (pk_1, pk_2, r)$, $sk^* = sk_1$
$(m_0, m_1) \leftarrow A^{\mathcal{D}^*_{sk^*}(\cdot)}(pk^*)$
$w_1, w_2 \leftarrow \{0,1\}^*$
$c_1 = \mathcal{E}_{pk_1}(m_0; w_1)$, $c_2 = \mathcal{E}_{pk_2}(m_0; w_2)$
$\Pi \leftarrow \mathbf{Sim_2((c_1, c_2))}$
$b \leftarrow A(pk^*, c_1, c_2, \Pi)$

Claim 2 $|\mathrm{Pr}_{\mathsf{CCA1}_0}[A \text{ outputs } 0] - \mathrm{Pr}_1[A \text{ outputs } 0]|$ *is negligible.*

Proof By reduction to the zero-knowledge property of the proof system: we use A to construct an algorithm B that attempts to distinguish real from simulated proofs.

$B(1^k)$
Receive r as first-stage input;
$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^k)$;
$pk^* = (pk_1, pk_2, r)$;
$sk^* = sk_1$;
$(m_0, m_1) \leftarrow A^{\mathcal{D}^*_{sk^*}(\cdot)}(pk^*)$; // note that B has no trouble
// simulating the decryption oracle for A
$w_1, w_2 \leftarrow \{0,1\}^*$;
$c_1 = \mathcal{E}_{pk_1}(m_0; w_1)$, $c_2 = \mathcal{E}_{pk_2}(m_0; w_2)$;
Output $((c_1, c_2), (w_1, w_2, m_0))$ as first-stage output;
Receive Π as second-stage input;
$b \leftarrow A(pk^*, c_1, c_2, \Pi)$;
Output b as second-stage output.

Now, when the inputs to B are a random string r and a real proof Π, then A's view in the above experiment is precisely its view in game $\mathsf{CCA1}_0$, and so $\mathrm{Pr}_{\mathsf{ZK}_{\mathrm{real}}}[B \text{ outputs } 0] = \mathrm{Pr}_{\mathsf{CCA1}_0}[A \text{ outputs } 0]$. On the other hand, when the inputs to B are a simulated string and a simulated proof, A's view in B is precisely its view in game 1, and so $\mathrm{Pr}_{\mathsf{ZK}_{\mathrm{sim}}}[B \text{ outputs } 0] = \mathrm{Pr}_1[A \text{ outputs } 0]$. Since $|\mathrm{Pr}_{\mathsf{ZK}_{\mathrm{real}}}[B \text{ outputs } 0] - \mathrm{Pr}_{\mathsf{ZK}_{\mathrm{sim}}}[B \text{ outputs } 0]|$ is negligible (since the proof system is adaptively-secure NIZK), we have that

$$|\mathrm{Pr}_{\mathsf{CCA1}_0}[A \text{ outputs } 0] - \mathrm{Pr}_1[A \text{ outputs } 0]|$$

is negligible as well. ■

The second game differs from game 1 in that it does not double-encrypt m_0, but instead encrypts m_0 once and m_1 once.

Game 2
$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^k)$
$r \leftarrow \mathsf{Sim}_1(1^k)$
$pk^* = (pk_1, pk_2, r)$, $sk^* = sk_1$
$(m_0, m_1) \leftarrow A^{\mathcal{D}^*_{sk^*}(\cdot)}(pk^*)$
$w_1, w_2 \leftarrow \{0,1\}^*$
$c_1 = \mathcal{E}_{pk_1}(m_0; w_1)$, $\boldsymbol{c_2 = \mathcal{E}_{pk_2}(m_1; w_2)}$
$\Pi \leftarrow \mathsf{Sim}_2((c_1, c_2))$
$b \leftarrow A(pk^*, c_1, c_2, \Pi)$

Note that in the above, the simulator is given as input encryptions of two *different* messages. Such an input is *not* in L, and in general there is not much we can say about the simulator's output in this case. However, we will see that in this particular case the game is indistinguishable to A because the semantic security of the underlying encryption scheme implies that encryptions of m_0 are indistinguishable from encryptions of m_1. Of course, this will require a formal proof.

Claim 3 $|\Pr_1[A \text{ outputs } 0] - \Pr_2[A \text{ outputs } 0]|$ *is negligible.*

Proof We use A to construct B that attempts to break the semantic security of the underlying scheme. Recall that B is given a public key, outputs two messages (m_0, m_1), is given the encryption of one of these, and has to guess which one. But B does *not* have access to a decryption oracle.

$B(pk)$
Set $pk_2 = pk$;
$(pk_1, sk_1) \leftarrow \mathsf{Gen}(1^k)$;
$r \leftarrow \mathsf{Sim}_1(1^k)$;
$pk^* = (pk_1, pk_2, r)$, $sk^* = sk_1$;
$(m_0, m_1) \leftarrow A^{\mathcal{D}^*_{sk^*}(\cdot)}(pk^*)$; // note that B has no trouble
// simulating the decryption oracle for A
Output (m_0, m_1);
Receive c_2 (an encryption of either m_0 or m_1 using (unknown) random tape w_2);
$w_1 \leftarrow \{0,1\}^*$;
$c_1 = \mathcal{E}_{pk_1}(m_0; w_1)$;
$\Pi \leftarrow \mathsf{Sim}_2((c_1, c_2))$;
$b \leftarrow A(pk^*, c_1, c_2, \Pi)$;
Output b.

Now, when c_2 is an encryption of m_0, then A's view above is precisely its view in game 1. On the other hand, when c_2 is an encryption of m_1, then A's view above is precisely its view in game 2. Therefore, the probability that A distinguishes game 1 from game 2 is precisely the probability that B distinguishes an encryption of m_0 from an encryption of m_1, which is negligible by the semantic security of the underlying encryption scheme. ■

In the same way as above, we would now like to "switch" c_1 from being an encryption of m_0 to being an encryption of m_1. Here, however, a potential problem arises! To prove

a claim analogous to Claim 3, we would need to construct some adversary B that gets $pk = pk_1$ and then has to distinguish whether the ciphertext c_1 it receives is an encryption of m_0 or m_1. But, in order to do this it has to somehow simulate a decryption oracle for A — and this seems to require sk_1, which B does not have! (If B has sk_1 then it would be easy for B to break semantic security of the scheme....) So, we will have to do a little more work before continuing.

Let Fake be the event that A submits a query (c_1, c_2, Π) to its decryption oracle (in stage 1) such that $\mathcal{D}_{sk_1}(c_1) \neq \mathcal{D}_{sk_2}(c_2)$ but $\mathcal{V}(r, (c_1, c_2), \Pi) = 1$. Note that Π is then a valid-looking proof for a false statement (since $(c_1, c_2) \notin L$).

Claim 4 $\Pr_2[\mathsf{Fake}]$ *is negligible.*

Proof First, note that $\Pr_2[\mathsf{Fake}] = \Pr_1[\mathsf{Fake}]$. This is because A submits oracle queries only in its first stage, and up to that stage the games are *identical*.

Next, we show that $|\Pr_1[\mathsf{Fake}] - \Pr_{\mathsf{CCA1}_0}[\mathsf{Fake}]|$ is negligible. Up to A's first stage, the games differ only in r being a random string or a simulated string. Construct an algorithm B that attempts to distinguish random from simulated strings, as follows:

$B(r)$
  $(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^k)$;
  $pk^* = (pk_1, pk_2, r)$;
  Run $A^{\mathcal{D}^*_{sk^*}(\cdot)}(pk^*)$, simulating the oracle for A normally except that
    if for any decryption query (c_1, c_2, Π) it is the case that
    $\mathcal{V}(r, (c_1, c_2), \Pi) = 1$ but $\mathcal{D}_{sk_1}(c_1) \neq \mathcal{D}_{sk_2}(c_2)$,
    then output 1 and stop (note that now B does *not* throw away sk_2);
  Otherwise, once A is done with its first stage simply output 0

Now, $\Pr_{\mathsf{ZK}_{\mathrm{sim}}}[B \text{ outputs } 0] = \Pr_1[\mathsf{Fake}]$. Similarly, $\Pr_{\mathsf{ZK}_{\mathrm{real}}}[B \text{ outputs } 0] = \Pr_{\mathsf{CCA1}_0}[\mathsf{Fake}]$. Since $|\Pr_{\mathsf{ZK}_{\mathrm{real}}}[B \text{ outputs } 0] - \Pr_{\mathsf{ZK}_{\mathrm{sim}}}[B \text{ outputs } 0]|$ is negligible (since the proof system is adaptively-secure NIZK), we have that $|\Pr_1[\mathsf{Fake}] - \Pr_{\mathsf{CCA1}_0}[\mathsf{Fake}]|$ is negligible as well.

Finally, note that $\Pr_{\mathsf{CCA1}_0}[\mathsf{Fake}]$ is negligible. This is because Fake occurs when A produces a valid proof for a $(c_1, c_2) \notin L$, which can only happen with a negligible probability because of the soundness of the NIZK proof system (note that r now is a truly random string). The claim follows. ■

We pick up the proof from here in the next lecture. Informally, the next game we introduce differs from game 2 only in that sk_2 is used for decryption instead of sk_1. The adversary's view in the games only differs if Fake occurs, which happens with negligible probability. All that's left to be done is to switch c_1 to an encryption of m_1 (similar to the introduction of game 2), switch the decryption key back to sk_1, and then go back to a random string and a real proof (similar to the introduction of game 1). This gets us back to game $\mathsf{CCA1}_1$ as desired, and will complete the proof.

References

[1] M. Naor and M. Yung. Public-Key Cryptosystems Provably Secure Against Chosen Ciphertext Attacks. In *Proceedings of the ACM Symposium on the Theory of Computing*, pages 427-437, 1990.

CMSC 858K — Advanced Topics in Cryptography February 17, 2004

Lecture 7

Lecturer: Jonathan Katz

Scribe(s): *Nagaraj Anthapadmanabhan*
Minkyoung Cho
Ji Sun Shin
Nick L. Petroni, Jr.

1 Introduction

In the last set of lectures, we introduced definitions of adaptively-secure non-interactive zero knowledge and semantically-secure encryption. Based on these, we presented a construction of a public-key encryption scheme secure against *non-adaptive* chosen-ciphertext attacks (CCA1). This encryption scheme was proposed by Naor and Yung in 1990 [3].

In this lecture, we complete the proof of non-adaptive chosen-ciphertext security for the Naor-Yung construction from the previous lecture. Next, we show that the scheme is *not* secure against adaptive chosen-ciphertext attacks by showing a counterexample; we also examine where the proof breaks down. Then, we introduce the definition of a digital signature scheme and the notion of security for a one-time strong signature scheme. Finally, we present a public key encryption scheme secure against *adaptive* chosen-ciphertext attacks (CCA2). This encryption scheme was constructed by Dolev, Dwork, and Naor in 1991 [1].

2 Naor-Yung Construction

The Naor-Yung construction relies on an underlying semantically-secure public-key encryption scheme $(\mathsf{Gen}, \mathcal{E}, \mathcal{D})$ and an adaptively-secure non-interactive zero-knowledge proof system $(\mathcal{P}, \mathcal{V})$. Given these, the scheme is defined as follows:

$\mathsf{Gen}^*(1^k)$: $(pk_1, sk_1) \leftarrow \mathsf{Gen}(1^k)$
$(pk_2, sk_2) \leftarrow \mathsf{Gen}(1^k)$
$r \leftarrow \{0,1\}^{\mathrm{poly(k)}}$
$pk^* = (pk_1, pk_2, r)$
$sk^* = sk_1$

$\mathcal{E}^*_{(pk_1,pk_2,r)}(m)$: pick $w_1, w_2 \leftarrow \{0,1\}^*$
$c_1 \leftarrow \mathcal{E}_{pk_1}(m; w_1)$
$c_2 \leftarrow \mathcal{E}_{pk_2}(m; w_2)$
$\Pi \leftarrow \mathcal{P}\left(r, (c_1, c_2), (m, w_1, w_2)\right)$
output (c_1, c_2, Π)

$\mathcal{D}^*_{sk_1}(c_1, c_2, \Pi)$: if $\mathcal{V}\left(r, (c_1, c_2), \Pi\right) = 0$ then (Verify proof)
    output $\perp$
else
    output $\mathcal{D}_{sk_1}(c_1)$

2.1 CCA1-Security

Theorem 1 *Assuming* $(\mathsf{Gen}, \mathcal{E}, \mathcal{D})$ *is a semantically-secure encryption scheme and* $(\mathcal{P}, \mathcal{V})$ *is an adaptively-secure NIZK proof system, then* $(\mathsf{Gen}^*, \mathcal{E}^*, \mathcal{D}^*)$ *is secure against non-adaptive chosen-ciphertext attacks.*

Recall that in a CCA1 attack an adversary is given access to a decryption oracle before choosing two messages. He does not, however, have access to this oracle after being presented the ciphertext (which he then has to use to guess which message was encrypted). The formal definition is as follows:

Definition 1 An encryption scheme $(\mathsf{Gen}, \mathcal{E}, \mathcal{D})$ is secure against chosen-ciphertext attacks ("CCA1-Secure") if the following is negligible for all PPT algorithms A:

$$\Big|\Pr[(pk, sk) \leftarrow \mathsf{Gen}(1^k);\ (m_0, m_1) \leftarrow A^{\mathcal{D}_{sk}(\cdot)}(pk);\ b \leftarrow \{0,1\};$$

$$C \leftarrow \mathcal{E}_{pk}(m_b);\ b' \leftarrow A(pk, C) :\ b = b'] \Big| - \frac{1}{2}.$$

$\diamondsuit$

We provide here the remainder of the proof of Theorem 1, noting where we left off last lecture. Recall that the idea behind the proof is as follows. We construct a set of games that differ slightly from each other. We show at each step, with the construction of each new game, that the ability of the adversary to distinguish between the two games is negligible (or, in other words, the games are computationally indistinguishable). Transitivity then implies that the first and last games are indistinguishable. But the first game will correspond to the view of an adversary when m_0 is encrypted, while the final game will correspond to the view of an adversary when m_1 is encrypted; thus, this completes the proof.

For the six different games, informal descriptions are as follows:

- Game 0: This is the real game, with the adversary getting an encryption of m_0.
- Game 1: Like Game 0 except that instead of $(\mathcal{P}, \mathcal{V})$, its simulator is used to provide the proof Π.
- Game 2: Like Game 1 except that c_2 is computed as an encryption of m_1.
- Game 2′: Like Game 2 except use sk_2 to decrypt instead of sk_1.
- Game 3: Like Game 2′ except that c_1 is computed as an encryption of m_1.
- Game 3′ Like Game 3 except use sk_1 to decrypt instead of sk_2.
- Game 4: Like Game 3′ except use $(\mathcal{P}, \mathcal{V})$ to provide the proof Π.

Note that Game 4 corresponds exactly to the real game when the adversary gets an encryption of m_1.

Proof of Theorem 1: We first considered the following two games. In each game the adversary is given a pk based on two valid runs of Gen along with a value r. The adversary, with the help of a decryption oracle, then outputs two messages and has to guess which

of those messages was encrypted. The only difference between the two games is that in Game 1 the values for r and Π come from a simulator (and are not a true random string and a real proof, respectively). However, we claim that the probability of the adversary's guess being $b' = 0$ is the essentially same in each case (i.e., only negligibly different).

Game 0 :	Game 1 :
$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^k)$	$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^k)$
$\boxed{r \leftarrow \{0,1\}^{\mathrm{poly}(k)}}$	$\boxed{r \leftarrow \mathsf{Sim}_1(1^k)}$
$pk^* = (pk_1, pk_2, r); \quad sk^* = sk_1$	$pk^* = (pk_1, pk_2, r); \quad sk^* = sk_1$
$(m_0, m_1) \leftarrow A^{\mathcal{D}_{sk^*}(\cdot)}(pk)$	$(m_0, m_1) \leftarrow A^{\mathcal{D}_{sk^*}(\cdot)}(pk)$
$w_1, w_2 \leftarrow \{0,1\}^{poly(k)}$	$w_1, w_2 \leftarrow \{0,1\}^{poly(k)}$
$c_1 \leftarrow \mathcal{E}_{pk_1}(m_0; w_1); \quad c_2 \leftarrow \mathcal{E}_{pk_2}(m_0; w_2)$	$c_1 \leftarrow \mathcal{E}_{pk_1}(m_0; w_1); \quad c_2 \leftarrow \mathcal{E}_{pk_2}(m_0; w_2)$
$\boxed{\Pi \leftarrow \mathcal{P}(r, (c_1, c_2), (m, w_1, w_2))}$	$\boxed{\Pi \leftarrow \mathsf{Sim}_2(c_1, c_2)}$
$b \leftarrow A(c_1, c_2, \Pi)$	$b \leftarrow A(c_1, c_2, \Pi)$

Claim 2 *Let* $\Pr_0[\cdot]$ *and* $\Pr_1[\cdot]$ *represent the probability of event* $\cdot$ *in games 0 and 1 respectively. Then* $|\Pr_0[b = 0] - \Pr_1[b = 0]|$ *is negligible.*

Proof *Summary from last time.* To show that the adversary's choice is not affected by the use of a simulator, the only change from Game 0 to Game 1, we showed that if such a difference *could* be detected by the adversary then the adversary could be used by another adversary, A', to distinguish real proofs from simulated proofs. Using the adversary who can distinguish between games 0 and 1, A' can easily simulate the decryption oracle, obtain messages to encrypt and pass on for a proof, and give the proof to A along with valid ciphertext. The advantage of A' in distinguishing real/simulated proofs is then $|\Pr_0[b = 0] - \Pr_1[b = 0]|$, which is negligible by the security of $(\mathcal{P}, \mathcal{V})$ as an adaptively-secure NIZK proof system. ■

We then made a change to Game 1, where we encrypt message m_1 to get c_2, and called this Game 2.

Game 2 :
$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^k)$
$r \leftarrow \mathsf{Sim}_1(1^k)$
$pk^* = (pk_1, pk_2, r); \quad sk^* = sk_1$
$(m_0, m_1) \leftarrow A^{\mathcal{D}_{sk^*}(\cdot)}(pk)$
$w_1, w_2 \leftarrow \{0,1\}^{poly(k)}$
$c_1 \leftarrow \mathcal{E}_{pk_1}(m_0; w_1); \quad \boxed{c_2 \leftarrow \mathcal{E}_{pk_2}(m_1; w_2)}$
$\Pi \leftarrow \mathsf{Sim}_2(c_1, c_2)$
$b \leftarrow A(c_1, c_2, \Pi)$

Claim 3 *Let* $\Pr_1[\cdot]$ *and* $\Pr_2[\cdot]$ *represent the probability of event* $\cdot$ *in games 1 and 2 respectively. Then* $|\Pr_1[b = 0] - \Pr_2[b = 0]|$ *is negligible.*

Proof *Summary from last time.* As with the previous claim, we wish to show that the existence of an adversary who can distinguish between the two games is an impossibility. This time, instead of the zero-knowledge property, we attacked the semantic security of the

underlying encryption scheme. Specifically, we constructed an adversary A' who wishes to guess which of two messages he generated was encrypted with a given key. To achieve his goal, A' uses the adversary who can distinguish between games 1 and 2 to pick two messages. A' passes these on directly and then gets back an encrypted message that is either m_0 or m_1. Giving this to A as c_2 (note that A' has no problem running the simulator), we see that the advantage of A' is exactly $|\Pr_1[b=0] - \Pr_2[b=0]|$. And this must be negligible by semantic security of the underlying encryption scheme. ■

We next defined an event Fake as the event that A submits (c_1, c_2, Π) to the decryption oracle with $\mathcal{D}_{sk_1}(c_1) \neq \mathcal{D}_{sk_2}(c_2)$, but $\mathcal{V}(r, (c_1, c_2), \Pi) = 1$. This represents the event where the adversary is able to trick the verifier into returning true on a bad pair of ciphertexts (i.e., a pair not in the language). We continued with the claim that the probability of Fake occurring in Game 2 is negligible.

Claim 4 $\Pr_2[\mathsf{Fake}]$ *is negligible.*

Proof Since Fake has to do with the use of the decryption oracle, we note that the only important event from A's perspective at the point he uses the oracle is the generation of pk^*. Furthermore, we note that pk^* is created *identically* in games 1 and 2. Therefore $\Pr_1[\mathsf{Fake}] = \Pr_2[\mathsf{Fake}]$. Last time, we went on to show that $|\Pr_1[\mathsf{Fake}] - \Pr_0[\mathsf{Fake}]|$ is negligible by the zero-knowledge property of the proof system and that $Pr_0[\mathsf{Fake}]$ is negligible because of the soundness of the proof system. Therefore $\Pr_2[\mathsf{Fake}]$ is negligible. ■

At this point, we continue our proof from last time (and begin the new material from this lecture). The proof continues by constructing another game, Game $2'$, which is the same as Game 2 *except* we use sk_2 to decrypt instead of sk_1 (in the obvious way).

Game 2':
$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^k)$
$r \leftarrow \mathsf{Sim}_1(1^k)$
$pk^* = (pk_1, pk_2, r)$; $\boxed{sk^* = sk_2}$
$(m_0, m_1) \leftarrow \boxed{A^{\mathcal{D}_{sk^*}(\cdot)}(pk)}$
$w_1, w_2 \leftarrow \{0,1\}^{poly(k)}$
$c_1 \leftarrow \mathcal{E}_{pk_1}(m_0; w_1)$; $c_2 \leftarrow \mathcal{E}_{pk_2}(m_1; w_2)$
$\Pi \leftarrow \mathsf{Sim}_2(c_1, c_2)$
$b \leftarrow A(c_1, c_2, \Pi)$

Corollary 5 $|\Pr_{2'}[b=0] - \Pr_2[b=0]|$ *is negligible.*

Proof From an adversary's point of view, a difference between Game 2 and Game $2'$ occurs only if the event Fake occurs. This is because, as long as both c_1 and c_2 are encryptions of the same message, decryption using sk_1 or sk_2 will make no difference. It is not hard to see that $\Pr_2[\mathsf{Fake}] = \Pr_{2'}[\mathsf{Fake}]$. Since in either game the event Fake occurs with only negligible probability, the corollary follows. ■

We now modify Game $2'$, encrypting m_1 for both c_1 and c_2, and call this Game 3.

$\underline{\text{Game } 3}$:
$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^k)$
$r \leftarrow \mathsf{Sim}_1(1^k)$
$pk^* = (pk_1, pk_2, r); \quad sk^* = sk_2$
$(m_0, m_1) \leftarrow A^{\mathcal{D}_{sk^*}(\cdot)}(pk)$
$w_1, w_2 \leftarrow \{0,1\}^{poly(k)}$
$\boxed{c_1 \leftarrow \mathcal{E}_{pk_1}(m_1; w_1)}; \quad c_2 \leftarrow \mathcal{E}_{pk_2}(m_1; w_2)$
$\Pi \leftarrow \mathsf{Sim}_2(c_1, c_2)$
$b \leftarrow A(c_1, c_2, \Pi)$

Claim 6 $|\Pr_3[b = 0] - \Pr_{2'}[b = 0]|$ *is negligible.*

Proof We note that the argument used in this proof is similar to the one used in between games 1 and 2. Assume that there exists a PPT adversary A such that $|\Pr_3[b = 0] - \Pr_{2'}[b = 0]|$ is NOT negligible. We can then construct an adversary $A'(pk_1)$ which breaks the semantic security of the underlying encryption scheme, thus generating a contradiction, as follows:

$\underline{A'(pk_1)}$:
$(pk_2, sk_2) \leftarrow \mathsf{Gen}(1^k)$
$r \leftarrow \mathsf{Sim}_1(1^k)$
$pk^* = (pk_1, pk_2, r); \quad sk^* = sk_2$
Run $A^{\mathcal{D}_{sk^*}(\cdot)}(pk)$ until it outputs (m_0, m_1)
Query $\mathcal{E}_{pk_1,b}(m_0, m_1)$ to get c_1
$c_2 \leftarrow \mathcal{E}_{pk_2}(m_1)$
$\Pi \leftarrow \mathsf{Sim}_2(c_1, c_2)$
Output $A(c_1, c_2, \Pi)$

Note that $A'(pk_1)$ is PPT because A is PPT. Also, it is possible for A' to simulate the decryption oracle for A: A' can decrypt using sk_2 since (pk_2, sk_2) is generated locally by A'. Next, we can see that if $c_1 = \mathcal{E}_{pk_1}(m_0)$, then this becomes equivalent to Game 2'; if $c_1 = \mathcal{E}_{pk_1}(m_1)$, then this becomes equivalent to Game 3. So, A' distinguishes encryptions of m_0 from encryptions of m_1 with probability $|\Pr_3[b = 0] - \Pr_{2'}[b = 0]|$, which must be negligible by semantic security of the underlying encryption scheme. ■

We next imagine a game Game 3′ in which we revert back to using sk_1 to decrypt rather than sk_2. As in the proof of indistinguishability between Game 2 and Game 2′, it is not hard to see that Game 3′ is indistinguishable from Game 3.

We next construct another game, Game 4, in which we switch back to *real* proofs from *simulated* proofs.

Game 4 :
$(pk_1, sk_1), (pk_2, sk_2) \leftarrow \mathsf{Gen}(1^k)$
$\boxed{r \leftarrow \{0,1\}^{\mathrm{poly}(k)}}$
$pk^* = (pk_1, pk_2, r); \quad sk^* = sk_1$
$(m_0, m_1) \leftarrow A^{\mathcal{D}_{sk^*}(\cdot)}(pk)$
$w_1, w_2 \leftarrow \{0,1\}^{poly(k)}$
$c_1 \leftarrow \mathcal{E}_{pk_1}(m_1; w_1); \quad c_2 \leftarrow \mathcal{E}_{pk_2}(m_1; w_2)$
$\boxed{\Pi \leftarrow \mathcal{P}(r, (c_1, c_2), (w_1, w_2))}$
$b \leftarrow A(c_1, c_2, \Pi)$

So, Game 4 is basically the actual situation where the adversary gets a *real* encryption of m_1 along with a *real* proof.

Claim 7 $|\mathrm{Pr}_4[b = 0] - \mathrm{Pr}_3[b = 0]|$ *is negligible*

Proof The proof is exactly analogous to that used in studying the transition from Game 0 to Game 1. As there, if an adversary could distinguish between the two games, it could be used by another adversary to distinguish real from simulated proofs. However, this violates the (adaptive) zero-knowledge property of the underlying proof system. ■

From the sequence of preceding claims, we can conclude that $|\mathrm{Pr}_4[b = 0] - \mathrm{Pr}_0[b = 0]|$ is negligible. But since the final game is just the real game when the adversary gets an encryption of m_1, and the original game is just the real game when the adversary gets an encryption of m_0, we see that we have proved that $(\mathsf{Gen}^*, \mathcal{E}^*, \mathcal{D}^*)$ is secure against *non-adaptive* chosen-ciphertext attacks. ■

2.2 CCA2-Security

In this section, we will examine why the Naor-Yung construction is not secure against *adaptive* chosen-ciphertext attacks by giving a counter-example. Recall the formal definition of such attacks:

Definition 2 An encryption scheme $(\mathsf{Gen}, \mathcal{E}, \mathcal{D})$ is secure against adaptive chosen-ciphertext attacks ("CCA2-Secure") if the following is negligible for all PPT algorithms A:

$$\Big|\mathrm{Pr}[(pk, sk) \leftarrow \mathsf{Gen}(1^k);\ (m_0, m_1) \leftarrow A^{\mathcal{D}_{sk}(\cdot)}(pk);\ b \leftarrow \{0,1\};$$

$$C \leftarrow \mathcal{E}_{pk}(m_b);\ b' \leftarrow A^{\mathcal{D}_{sk}(\cdot)}(pk, C) :\ b = b'] \Big| - \frac{1}{2},$$

where A cannot query $\mathcal{D}_{sk}(C)$. ♢

Theorem 8 *The Naor-Yung scheme* $(\mathsf{Gen}^*, \mathcal{E}^*, \mathcal{D}^*)$ *is not secure against adaptive chosen-ciphertext attacks (in general). More precisely, for any semantically-secure encryption scheme* $(\mathsf{Gen}, \mathcal{E}, \mathcal{D})$ *there exists an adaptively-secure NIZK proof system* $(\mathcal{P}, \mathcal{V})$ *such that the resulting Naor-Yung construction is demonstrably insecure against adaptive chosen-ciphertext attacks.*

Proof Let $(\mathcal{P}', \mathcal{V}')$ be any adaptively-secure NIZK proof system. Define the proof system $(\mathcal{P}, \mathcal{V})$ as follows:

$$\underline{\mathcal{P}(r, (c_1, c_2), (w_1, w_2))}:$$
$$\text{Output } \mathcal{P}'(r, (c_1, c_2), (w_1, w_2))|0$$

$$\underline{\mathcal{V}(r, (c_1, c_2), \Pi|b)}:$$
$$\text{Output } \mathcal{V}'(r, (c_1, c_2), \Pi)$$

I.e., we introduce a spurious bit in $\mathcal{P}$ and have $\mathcal{V}$ ignore it. (here, "|" denotes concatenation). It is not hard to show that $(\mathcal{P}, \mathcal{V})$ is also an adaptively-secure NIZK proof system. However, if this new proof system is used in the Naor-Yung construction we can construct an adversary A (making a CCA2 attack) which breaks the encryption as follows:

$\underline{A(pk)}$:
Output (m_0, m_1)
Get back $(c_1, c_2, \Pi|0)$
Submit $(c_1, c_2, \Pi|1)$ to the decryption oracle
Get back m_b

The adversary just modifies the last bit of the challenged ciphertext and submits it to the decryption oracle (note that this is allowed under the definition of CCA2 security). By this method, the adversary will get back the actual message as the last bit was just a spurious bit. So, this construction is not secure against *adaptive* chosen-ciphertext attacks. ■

If we examine the proof of security for the Naor-Yung construction and try to analyze where it breaks down in this case, we see that the $\Pr_2[\mathsf{Fake}]$ is no longer negligible. This is because, if the adversary gets a fake proof (say, $(c_1, c_2, \Pi|0)$), he can construct *another* fake proof by changing just the last bit (i.e., $(c_1, c_2, \Pi|1)$).

We mention in passing that one fix this problem by constructing a proof system satisfying a stronger notion of security: namely, that even when an adversary is given a fake proof, it should be unable to construct a *different* fake proof. See [4, 2] for work along this line. Here, however, we discuss a different method which was historically first.

3 Signature Schemes

For the construction that follows, we will need to notion of a digital signature scheme. Of course, such schemes are also very useful in their own right, and maybe we will return to them later in the course.

Definition 3 A *signature scheme* (over some message space $\mathcal{M}$) is a triple of PPT algorithms $(\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Verify})$ such that:

1. SigGen is a randomized algorithm which outputs a verification key vk and a secret key sk (denoted by $(vk, sk) \leftarrow \mathsf{SigGen}(1^k)$).

2. Sign is a (possibly) randomized algorithm which takes a secret key sk and a message $m \in \mathcal{M}$ and outputs a signature σ (denoted by $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$).

3. Verify is a deterministic algorithm which takes a verification key vk, a message $m \in \mathcal{M}$, and a signature σ and outputs 1 or 0 (denoted by $\mathsf{Verify}_{vk}(m, \sigma)$). A 1 indicates that the signature is *valid* and a 0 indicates that the signature is *invalid*.

We require that for all k, for all (vk, sk) output by $\mathsf{SigGen}(1^k)$, and for all $m \in \mathcal{M}$ we have $\mathsf{Verify}_{vk}(m, \mathsf{Sign}_{sk}(m)) = 1$. $\diamondsuit$

The above merely defines the semantics of signing, but does not give any notion of security. Many such definitions are possible, but we will only require a fairly weak definition of security for the present application (note that this definition of security is too weak for signature schemes used to sign, say, documents).

Definition 4 A signature scheme $(\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Verify})$ is a *one-time, strong signature scheme* if the following is negligible for all PPT adversaries A:

$$\Pr\left[\begin{array}{l}(vk, sk) \leftarrow \mathsf{SigGen}(1^k); m \leftarrow A\,(vk)\,; \sigma \leftarrow \mathsf{Sign}_{sk}(m); \\ (m', \sigma') \leftarrow A\,(vk, \sigma) : \mathsf{Verify}_{vk}\,(m', \sigma') = 1 \wedge (m', \sigma') \neq (m, \sigma)\end{array}\right].$$

$\diamondsuit$

What this means is that, given a signature of a message he chooses, an adversary cannot forge a signature for a different message without knowledge of the secret key. (Also, the adversary cannot even forge a *different* signature on the same message.) While we do not prove it here, it is known that one-time, strong signature schemes exist assuming the existence of one-way functions.

Theorem 9 *If one-way functions exist then one-time, strong signature schemes exist.*

4 Dolev-Dwork-Naor Construction

Danny Dolev, Cynthia Dwork, and Moni Naor [1] constructed an encryption scheme secure against adaptive chosen-ciphertext attacks beginning from any underlying semantically-secure scheme, a one-time, strong signature scheme, and an adaptively-secure NIZK proof system. Their construction is discussed in this section.

Let $(\mathsf{Gen}, \mathcal{E}, \mathcal{D})$ be a *semantically secure* encryption scheme. We construct a new encryption scheme $(\mathsf{Gen}', \mathcal{E}', \mathcal{D}')$ as follows:

$\mathsf{Gen}'(1^k)$: $r \leftarrow \{0,1\}^{\mathrm{poly}(k)}$
for $i = 1$ to k
  for $b = 0$ to 1
    $(pk_{i,b}, sk_{i,b}) \leftarrow \mathsf{Gen}(1^k)$ (Generate $2k$ pairs of keys)

$$pk = \begin{pmatrix} pk_{1,0} & pk_{2,0} & \dots & pk_{k,0} \\ pk_{1,1} & pk_{2,1} & \dots & pk_{k,1} \end{pmatrix}, r$$

$$sk = \begin{pmatrix} sk_{1,0} & sk_{2,0} & \dots & sk_{k,0} \\ sk_{1,1} & sk_{2,1} & \dots & sk_{k,1} \end{pmatrix}$$

${\mathcal{E}'}_{pk}(m)$: $(vk, sk) \leftarrow \mathsf{SigGen}(1^k)$
Let $vk = v_1|v_2|\cdots|v_k$ be the binary representation of vk
  (we assume for simplicity that $|vk| = k$)
for $i = 1$ to k
  $w_i \leftarrow \{0,1\}^*$; $c_i \leftarrow \mathcal{E}_{pk_{i,v_i}}(m; w_i)$
$\Pi \leftarrow \mathcal{P}(r, \vec{C}, (m, \vec{w}))$
  (this is a proof that all ciphertexts correspond to same message)
$\sigma \leftarrow \mathsf{Sign}_{sk}(\vec{C}|\Pi)$
output $vk, \vec{C}, \Pi, \sigma$

${\mathcal{D}'}_{sk}(vk, \vec{C}, \Pi, \sigma)$: if $\mathsf{Verify}_{vk}(\vec{C}|\Pi, \sigma) = 0$ (Verify signature)
  output $\perp$
else
  if $\mathcal{V}(r, \vec{C}, \Pi) = 0$ (Verify proof)
    output $\perp$
  else
    output $\mathcal{D}_{sk_{1,v_1}}(c_1)$

Note that the attack we showed on the Naor-Yung scheme fails here since the attack would require an adversary to forge a signature with respect to vk (which is infeasible). Of course, we need a formal proof to show that the scheme resists *all* adaptive chosen-ciphertext attacks.

Theorem 10 *Assuming* $(\mathsf{Gen}, \mathcal{E}, \mathcal{D})$ *is a* semantically secure *encryption scheme,* $(\mathcal{P}, \mathcal{V})$ *is an adaptively-secure NIZK proof system, and a one-time, strong signature scheme is used, then* $(\mathsf{Gen}', \mathcal{E}', \mathcal{D}')$ *is secure against* adaptive *chosen-ciphertext attacks.*

Proof The proof uses the same structure as that of the Naor-Yung construction. We have a PPT adversary A making an adaptive chosen-ciphertext attack on the encryption scheme. We show that the probability that the adversary will succeed is negligible by constructing a series of games and showing that they are all computationally indistinguishable. We begin by defining our original game, which corresponds to the real encryption scheme when the adversary gets an encryption of m_0:

Game 0 :
$\{(pk_{i,b}, sk_{i,b})\}_{1\leq i\leq k; b\in\{0,1\}} \leftarrow \mathsf{Gen}(1^k)$
$r \leftarrow \{0,1\}^{\mathrm{poly}(k)}$
$pk = \{pk_{i,b}\}_{1\leq i\leq k; b\in\{0,1\}}, r; \quad sk = \{sk_{i,b}\}_{1\leq i\leq k; b\in\{0,1\}}$
$(m_0, m_1) \leftarrow A^{\mathcal{D}'_{sk}(\cdot)}(pk)$
$(vk, sk) \leftarrow \mathsf{SigGen}(1^k)$
for $i = 1$ to k
  $c_i \leftarrow \mathcal{E}_{pk_{i,v_i}}(m_0)$
$\boxed{\Pi \leftarrow \mathcal{P}(r, \vec{C}, (m_0, \vec{w}))}$
$\sigma \leftarrow \mathsf{Sign}_{sk}(\vec{C}|\Pi)$
$b \leftarrow A^{\mathcal{D}'_{sk}(\cdot)}(vk, \vec{C}, \Pi, \sigma)$

We begin by stating a technical lemma. Let Forge be the event that A submits a ciphertext $(vk', \vec{C}', \Pi', \sigma')$ to the decryption oracle with:

- $vk' = vk$
- $(\vec{C}', \Pi', \sigma') \neq (\vec{C}, \Pi, \sigma)$
- $\mathsf{Verify}_{vk}(\vec{C}'|\Pi', \sigma') = 1$

Claim 11 $\Pr_0[\mathsf{Forge}]$ *is negligible.*

This follows from the 1-time strong security of the signature scheme. Details omitted.

The proof of security for the Dolev-Dwork-Naor scheme will be completed in the following lecture. ■

References

[1] D. Dolev, C. Dwork, and M. Naor. *Non-Malleable Cryptography*, proceedings of the twenty-third annual ACM Symposium on Theory of Computing (1991).

[2] Y. Lindell. *A Simpler Construction of CCA2-Secure Public-Key Encryption under General Assumptions*, Eurocrypt 2003: 241-254 (2003).

[3] M. Naor and M. Yung. *Public-key Cryptosystems Provably Secure against Chosen-Ciphertext Attacks*, proceedings of the twenty-second annual ACM Symposium on Theory of Computing (1990).

[4] A. Sahai. *Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security*, FOCS 1999: 543-553 (1999).

CMSC 858K — Advanced Topics in Cryptography February 19, 2004

Lecture 8

Lecturer: Jonathan Katz

Scribe(s): *Alvaro A. Cardenas*
Nicholas Sze
Yinian Mao
Kavitha Swaminathan

1 Introduction

Last time we introduced the Naor-Yung construction of a CCA1-secure cryptosystem, and gave a proof of security. We also gave the construction of a CCA2-secure cryptosystem by Dolev-Dwork-Naor. Here, we complete the proof that this cryptosystem is indeed CCA2 secure.

2 The Dolev-Dwork-Naor (DDN) Scheme [1]

Given a public-key encryption scheme $(\mathsf{Gen}', \mathcal{E}', \mathcal{D}')$, an adaptively-secure NIZK proof system $(\mathcal{P}, \mathcal{V})$, and a (strong, one-time) signature scheme $(\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Vrfy})$, the DDN encryption scheme is constructed as follows (in the following, $poly(k)$ represents some unspecified polynomial which is not necessarily always the same):

- $\mathsf{Gen}(1^k)$:
  for $i = 1$ to k do $(pk_{i,0}, sk_{i,0}) \leftarrow \mathsf{Gen}'(1^k)$, $(pk_{i,1}, sk_{i,1}) \leftarrow \mathsf{Gen}'(1^k)$
  Select a random r: $r \leftarrow \{0,1\}^{poly(k)}$
  Output $pk^* = \begin{bmatrix} pk_{1,0} & pk_{2,0} & \cdots & pk_{k,0} \\ pk_{1,1} & pk_{2,1} & \cdots & pk_{k,1} \end{bmatrix}, r$
  and $sk^* = \begin{bmatrix} sk_{1,0} & sk_{2,0} & \cdots & sk_{k,0} \\ sk_{1,1} & sk_{2,1} & \cdots & sk_{k,1} \end{bmatrix}$
  (in fact, we may simplify things and let $sk^* = (sk_{1,0}, sk_{1,1})$; see below).

- $\mathcal{E}_{pk^*}(m)$:
  $(vk, sk) \leftarrow \mathsf{SigGen}(1^k)$
  view vk as a sequence of k bits[1]; i.e., $vk = vk_1|vk_2|\cdots|vk_k$
  for $i = 1$ to k: $w_i \leftarrow \{0,1\}^{poly(k)}$; $c_i \leftarrow \mathcal{E}'_{pk_{i,vk_i}}(m; w_i)$
  $\pi \leftarrow \mathcal{P}(r, \vec{c}, \vec{w})$
  $\sigma \leftarrow \mathsf{Sign}_{sk}(\vec{c}|\pi)$
  Output $(vk, \vec{c}, \pi, \sigma)$

- $\mathcal{D}_{sk^*}(vk, \vec{c}, \pi, \sigma)$:
  If $\mathsf{Vrfy}_{vk}(\vec{c}|\pi, \sigma) = 0$ then output $\perp$
  If $\mathcal{V}(r, \vec{c}, \pi) = 0$ then output $\perp$
  Else output $\mathcal{D}'_{sk_{1,vk_1}}(c_1)$

[1] The scheme can be modified in the obvious way for vk of arbitrary (polynomial) length.

Theorem 1 *The encryption scheme presented above is CCA2 secure if* $(\mathsf{Gen}', \mathcal{E}', \mathcal{D}')$ *is semantically secure,* $(\mathcal{P}, \mathcal{V})$ *is an adaptively-secure NIZK proof system, and* $(\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is a strong, one-time signature scheme.*

Proof Consider an arbitrary PPT adversary A with adaptive access to a decryption oracle. We will use a sequence of games in which the first game will correspond to a real encryption of m_0, the final game will correspond to a real encryption of m_1 (here, m_0, m_1 are the messages output by A), and in each stage along the way we show that the difference in the adversary's probability of outputting "1" is negligible. This then implies that the difference between the probability that it outputs 1 when it gets an encryption of m_0 and the probability it outputs 1 when it gets an encryption of m_1 is also negligible, and that is exactly the definition of CCA2 security.

Game 0 is the encryption of m_0 using the real cryptosystem:

Game 0: Stage 1

$$\begin{aligned} \{(pk_{i,b}, sk_{i,b})\} &\leftarrow \mathsf{Gen}'(1^k), \text{ for } i = 1, 2, \ldots, k \text{ and } b = 0, 1 \\ r &\leftarrow \{0,1\}^{\mathsf{poly}(k)} \\ (pk^*, sk^*) &= \left(\left(\{pk_{i,b}\}, r\right), \{sk_{i,b}\}\right) \\ (m_0, m_1) &\leftarrow A^{\mathcal{D}_{sk^*}(\cdot)}(pk^*) \end{aligned}$$

Stage 2

$$\begin{aligned} (vk, sk) &\leftarrow \mathsf{SigGen}(1^k) \\ w_i &\leftarrow \{0,1\}^{poly(k)}, \text{ for } i = 1, 2, \cdots, k \text{ (from now on we let this step be implicit)} \\ c_i &\leftarrow \mathcal{E}'_{pk_{i,vk_i}}(m_0; w_i), \text{ for } i = 1, 2, \cdots, k \\ \pi &\leftarrow \mathcal{P}(r, \vec{c}, \vec{w}) \\ \sigma &\leftarrow \mathsf{Sign}_{sk}(\vec{c}|\pi) \\ b^* &\leftarrow A^{\mathcal{D}_{sk^*}(\cdot)}(pk^*, vk, \vec{c}, \pi, \sigma) \end{aligned}$$

Then, we modify Game 0 by simulating r and π to obtain Game 1. Simulator Sim_1 generates r and simulator Sim_2 outputs π without any witness.

Game 1: Stage 1

$$\begin{aligned} \{(pk_{i,b}, sk_{i,b})\} &\leftarrow \mathsf{Gen}'(1^k), \text{ for } i = 1, 2, \cdots, k \text{ and } b = 0, 1 \\ r &\leftarrow \boxed{\mathsf{Sim}_1(1^k)} \\ (pk^*, sk^*) &= \left(\left(\{pk_{i,b}\}, r\right), \{sk_{i,b}\}\right) \\ (m_0, m_1) &\leftarrow A^{\mathcal{D}_{sk^*}(\cdot)}(pk^*) \end{aligned}$$

Stage 2

$$\begin{aligned} (vk, sk) &\leftarrow \mathsf{SigGen}(1^k) \\ c_i &\leftarrow \mathcal{E}'_{pk_{i,vk_i}}(m_0; w_i), \text{ for } i = 1, 2, \cdots, k \\ \pi &\leftarrow \boxed{\mathsf{Sim}_2(\vec{c})} \\ \sigma &\leftarrow \mathsf{Sign}_{sk}(\vec{c}|\pi) \\ b^* &\leftarrow A^{\mathcal{D}_{sk^*}(\cdot)}(pk^*, vk, \vec{c}, \pi, \sigma) \end{aligned}$$

Claim 2 *Let* $\Pr_i[\cdot]$ *denote the probability of a given event in game* i*. Then for any* PPT A *the following is negligible:* $|\Pr_0[b^* = 1] - \Pr_1[b^* = 1].|$

Sketch of Proof (Informal) The validity of this claim is intuitively clear as if the probabilities are substantially different then A can be used as a distinguisher between a real NIZK proof and a simulated proof. distinguish a simulated proof from a real proof. We provide more details now.

Given a PPT adversary A, construct the following PPT adversary A' (adversary A' will attempt to distinguish between real/simulated proofs):

$A'(r)$**:** // r is either a truly random string or a string output by Sim_1
$\{(pk_{i,b}, sk_{i,b})\} \leftarrow \mathsf{Gen}'(1^k)$, for $i = 1, 2, \cdots, k$ and $b = 0, 1$
$pk^* = (\{pk_{i,b}\}, r)$
$(m_0, m_1) \leftarrow A(pk^*)$
$(vk, sk) \leftarrow \mathsf{SigGen}(1^k)$
$\forall i\ c_i \leftarrow \mathcal{E}'_{pk_{i,,vk_i}}(m_0; w_i)$
Output $(\vec{c}, \vec{w})$
get π //π is either a real proof or a simulated proof
$\sigma \leftarrow \mathsf{Sign}_{sk}(\vec{c}|\pi)$
$b^* \leftarrow A^{\mathcal{D}_{sk^*}(\cdot)}(pk^*, vk, \vec{c}, \pi, \sigma)$
Output b^*

Note that A' has no problems simulating the decryption oracle for A, since it has all necessary secret keys. If (r, π) are a real string/proof, then A is interacting in Game 0 and so the probability that A' outputs 1 is the probability that A outputs 1 in Game 0. On the other hand, if (r, π) are a simulated string/proof, then A is interacting in Game 1 and so the probability that A' outputs 1 is the probability that A outputs 1 in Game 1. Since the NIZK proof system is adaptively-secure, we must have $|\Pr_0[b^* = 1] - \Pr_1[b^* = 1].|$. □

We construct Game $1'$ as Game 1 except that if A ever makes valid decryption oracle query using vk (where vk is the verification key used to construct the challenge ciphertext), then we simply return $\perp$ in response to this query. We claim that $|\Pr_{1'}[b^* = 1] - \Pr_1[b^* = 1]|$ is negligible. Note that the only difference between the games occurs if A is able to forge a new, valid signature with respect to vk (since ciphertexts submitted to the decryption oracle must be *different* from the challenge ciphertext, and since ciphertexts are only valid if the signature verifies correctly); furthermore, the security of the signature scheme ensures that this event occurs with only negligible probability. Details omitted.

We construct a new game, Game $1''$, which is the same as Game $1'$ except that instead of using sk_{1,vk'_1} to decrypt a ciphertext $(vk', \vec{c'}, \pi', \sigma')$ (i.e., to answer decryption oracle queries for this ciphertext), we look for the first bit position i where vk and vk' differ[2] (i.e., $vk_i \neq vk'_i$) and use key sk_{i,vk'_i} to decrypt. I.e., the decryption oracle now works as follows:

$$\mathcal{D}''_{sk^*}(vk', \vec{c'}, \pi', \sigma') = \begin{cases} \perp & \text{if } vk' = vk; \\ \perp & \text{if } \mathsf{Vrfy}_{vk'}(\vec{c'}|\pi', \sigma') = 0 \text{ or } \mathcal{V}(r, \vec{c'}, \pi') = 0; \\ \mathcal{D}'_{sk_{i,vk'_i}}(c'_i) & \text{otherwise (where } i \text{ is as discussed above)} \end{cases}.$$

Claim 3 *For any* PPT *A the following is negligible:* $|\Pr_{1''}[b^* = 1] - \Pr_{1'}[b^* = 1]|$.

[2]Here, vk is again the verification key used for the challenge ciphertext; note that there must be a bit position where they differ since if $vk = vk'$ we abort anyway.

Sketch of Proof (Informal) In a given query to the decryption oracle, if all ciphertexts decrypt to the same thing then it doesn't really matter what secret key we use. The only difference between Game $1''$ and Game $1'$ occurs if the adversary queries a vector of ciphertexts $\vec{c'}$ where different ciphertexts decrypt to different messages. So the only possible way to distinguish between Game 1 and Game $1'$ is if a decryption query is ever made for which there exists two different indices i and j where the decryption of c'_i is not equal to the decryption of c'_j and yet the proof is valid (i.e., $V(r, \vec{c'}, \pi') = 1$). We argue that this event occurs with negligible probability.

Let Fake be the event that A requests a decryption query $(vk', \vec{c'}, \pi', \sigma')$ s.t. π' is a valid proof and $\exists i, j$ s.t. $\mathcal{D'}_{sk_{i,vk'_i}}(c_i) \neq \mathcal{D'}_{sk_{j,vk'_j}}(c_j)$. Note that $\Pr_{1''}[\mathsf{Fake}]] = \Pr_{1'}[\mathsf{Fake}]$ (since there is no difference between the games until Fake occurs). Furthermore, we claim that $|\Pr_{1'}[\mathsf{Fake}] - \Pr_1[\mathsf{Fake}]|$ is negligible. This is so because (as before) the only difference between these games occurs if the adversary forges a signature using vk, which happens with only negligible probability. We also claim that $|\Pr_1[\mathsf{Fake}] - \Pr_0[\mathsf{Fake}]|$ is negligible, since otherwise we can construct an adversary A' distinguishing real from simulated proofs, similar to the proof of Claim 1 (it is essential here that A' knows all secret keys, so can check when event Fake occurs). Finally, note that $\Pr_0[\mathsf{Fake}]$ is negligible by the (adaptive) soundness of the NIZK proof system. We conclude that $\Pr_{1''}[\mathsf{Fake}]$ is negligible, and this is sufficient to complete the proof of the claim. □

We construct Game 2 which is the same as Game $1''$ except that we form the challenge ciphertext by encrypting (k copies of) m_1 instead of m_0. I.e., for all i: we compute $c_i \leftarrow \mathcal{E'}_{pk_{i,vk_i}}(m_1)$

Claim 4 *For any* PPT A *the following is negligible:* $|\Pr_2[b^* = 1] - \Pr_{1''}[b^* = 1]|$.

Sketch of Proof (Informal) If A can distinguish between these two games we construct an adversary A' attacking the semantic security of the underlying encryption scheme. Actually, instead of attacking a *single* instance of the encryption scheme it will attack k instances of the encryption scheme; i.e., it gets k independently-generated public keys, outputs m_0, m_1, gets back either an encryption of m_0 (with respect to all k keys) or an encryption of m_1, and then has to guess which is the case. Note, however, that by a standard hybrid argument the semantic security of a single instance implies the semantic security of poly-many instances.

We construct our A' as follows:

$\underline{A'(pk_1, \cdots, pk_k) :}$
$(vk, sk) \leftarrow \mathsf{SigGen}(1^k)$
$\{(pk'_i, sk'_i)\} \leftarrow \mathsf{Gen}'(1^k)$, for $i = 1, 2, \cdots, k$
$r \leftarrow \mathsf{Sim}_1(1^k)$
$pk^* = (\{pk_{i,b}\}, r)$, where $pk_{i,b} = \begin{cases} pk_i & \text{if } b = vk_i \\ pk'_i & \text{otherwise} \end{cases}$
$(m_0, m_1) \leftarrow A^{\mathcal{D}^*(\cdot)}(pk^*)$
Output (m_0, m_1), get back $\vec{c}$
$\pi \leftarrow \mathsf{Sim}_2(\vec{c})$
$\sigma \leftarrow \mathsf{Sign}_{sk}(\vec{c}|\pi)$
Output whatever$A^{\mathcal{D}^*(\cdot)}(vk, \vec{c}, \pi, \sigma)$ outputs

It is crucial to note here that A' *can simulate the decryption oracle* $\mathcal{D}^*$ — in particular, for any ciphertext $(vk', \vec{c}', \pi', \sigma')$ submitted by A, if $vk' = vk$ then A' just returns $\perp$ (as in the previous game), whereas if $vk' \neq vk$ then there is a bit i where they differ (i.e., $vk'_i \neq vk_i$) and A' can use the secret key $sk_{i,vk'_i} = sk'_i$ (which is knows!) to decrypt. This is by construction: A' knows exactly half the secret keys (i.e., those in positions not overlapping with vk) and can use those to decrypt.

Notice that if $\vec{c}$ is an encryption of m_1 then A is essentially interacting in Game 2, whereas if it is an encryption of m_0 then A is in Game $1''$. So, if A can distinguish between Game 1" and Game 2 then A' can distinguish the encryptions and break the semantic security of the underlying encryption scheme. □

Let Game 3 correspond to an encryption of m_1 in the real encryption scheme. We jump ahead and claim the following:

Claim 5 *For any* PPT *A, the following is negligible:* $|\Pr_3[b^* = 1] - \Pr_2[b^* = 1]|$.

Sketch of Proof (Informal) Technically, the proof would proceed by a sequence of games exactly analogous to games 1, $1'$, and $1''$ that we introduced previously. In particular, we would first revert back to decrypting using either $sk_{1,0}$ or $sk_{1,1}$; would then revert back to decrypting even if $vk' = vk$; and, finally, would go back to using a real random string/proof rather than simulated ones. Because these games (and the proofs that they all do not affect the probability that $b^* = 1$ by more than a negligible amount) are essentially the same as before, we do not repeat the arguments here. □

The above sequence of claims implies (by multiple applications of the triangle inequality) that $|\Pr_0[b^* = 1] - \Pr_3[b^* = 1]|$ is negligible; this is exactly equivalent to saying that the scheme is secure against adaptive chosen-ciphertext attacks. ■

3 Summary

We give a definition of a one-way function.

Definition 1 A function $f : \{0,1\}^* \to \{0,1\}^*$ is a *one-way function* if the following hold:

1. $f(x)$ is computable in time polynomial in $|x|$.
2. For all PPT algorithms A, the following is negligible (in k):

$$\mathsf{Pr}[x \leftarrow \{0,1\}^k ; y = f(x); x' \leftarrow A(y) : f(x') = y].$$

$\diamondsuit$ It is not hard to show that if a one-way function exists, then $P \neq NP$. The converse (i.e., whether $P \neq NP$ implies the existence one-way functions), is not known to hold.

Since the existence of semantically-secure public-key encryption schemes implies the existence of one-way functions[3], which furthermore implies the existence of one-time strong signature schemes, we may restate the result of the previous section as follows:

[3]Prove it as an exercise!

Theorem 6 *If there exists a semantically-secure public-key encryption scheme and an adaptively-secure NIZK proof system, then there exists a CCA2-secure encryption scheme.*

Later in the course, we will show:

Theorem 7 *If there exist trapdoor permutations, then there exists an adaptively-secure NIZK proof system.*

We have shown in a previous lecture that the existence of trapdoor permutations implies the existence of semantically-secure public-key encryption. The gives the following corollary:

Corollary 8 *If there exist trapdoor permutations, then there exists a CCA2-secure encryption scheme.*

The following important question is still open:

> *Does semantically-secure public-key encryption imply CCA2-secure public-key encryption?*

In particular, can we construct adaptively-secure NIZK proof systems based on semantically-secure public-key encryption? Note that these questions are especially interesting since we do have examples of public-key encryption schemes which are not based (or, at least, so not seem to be based) on trapdoor permutations; El Gamal encryption is probably the best-known example.

References

[1] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. In *23rd ACM Symposium on the Theory of Computing*, pages 543-552, 1991.

[2] M. Naor and M. Yung. Public-Key Cryptosystems Provably Secure Against Chosen Ciphertext Attacks. In *22nd ACM Symposium on the Theory of Computing*, pages 427-437, 1990

CMSC 858K — Advanced Topics in Cryptography February 24, 2004

Lecture 9

Lecturer: Jonathan Katz

Scribe(s): *Julie Staub*
Avi Dalal
Abheek Anand
Gelareh Taban

1 Introduction

In previous lectures, we constructed public-key encryption schemes which were provably secure against non-adaptive chosen-ciphertext (CCA-1) attacks, and also adaptive chosen-ciphertext (CCA-2) attacks. However, both constructions used generic non-interactive zero-knowledge proof systems which — although poly-time — are not very efficient (as we will see later in the course). Therefore, the constructions are not very practical.

In 1998, Cramer and Shoup proposed an encryption scheme [1] which was provably secure against adaptive chosen-ciphertext attacks and was also practical. The proof of security relies on the hardness of the Decisional Diffie-Hellman (DDH) problem in some underlying group.

In this lecture, we will first review the Decisional Diffie-Hellman assumption and the El-Gamal cryptosystem. Then we will modify the El-Gamal encryption scheme to construct a scheme secure against non-adaptive chosen-ciphertext attacks. This will be a step toward the full Cramer-Shoup cryptosystem, which we will cover in later lectures.

2 Background

The Cramer-Shoup cryptosystem relies on the DDH assumption in some finite group. In Lecture 4, we defined the Discrete Logarithm (DL) problem and DDH problem; we review them here. Let $\mathcal{G}$ be a finite cyclic group of prime order q, and let $g \in \mathcal{G}$ be a generator. Given $h \in \mathcal{G}$, the discrete logarithm problem requires us to compute $x \in \mathbb{Z}_q$ such that $g^x = h$. We denote this (unique) x by $\log_g h$. In particular groups $\mathcal{G}$ and for q large, it is assumed *hard* to compute x (this was formalized in Lecture 4).

A stronger assumption is the Decisional Diffie-Hellman (DDH) assumption. Here, given $\mathcal{G}$, a generator g of $\mathcal{G}$, and three elements $a, b, c \in \mathcal{G}$, we are asked (informally) to decide whether there exist x, y such that $a = g^x, b = g^y$ and $c = g^{xy}$. More formally, the DDH assumption states that the following two distributions are computationally indistinguishable:

- $\{\mathcal{G}, g, g^x, g^y, g^{xy}\}$
- $\{\mathcal{G}, g, g^x, g^y, g^z\}$

where g is a generator of $\mathcal{G}$ and x, y, z are chosen at random from $\mathbb{Z}_q$. (Again, see Lecture 4 for more formal definitions.)

Clearly, the DDH assumption implies the DL assumption. In fact, it appears to be considerably stronger. In particular, there are groups where DDH is false, but DL is still believed to hold. For example[1], let $\mathcal{G} = \mathbb{Z}_p^*$ for a prime p. On the one hand, the DL problem is believed to be hard in this group. Yet given g^a, g^b (for generator g) one can easily deduce the Legendre symbol of g^{ab} (which we denote by $\mathcal{L}(g^{ab})$). This observation gives an immediate method for distinguishing $\{\mathcal{G}, g, g^x, g^y, g^{xy}\}$ and $\{\mathcal{G}, g, g^x, g^y, g^z\}$ with non-negligible probability; namely, guess "DDH tuple" iff $\mathcal{L}(g^z) = \mathcal{L}(g^{xy})$.

A group in which DDH is assumed to hold is the following: Let $p = 2q+1$ where p, q are both prime. Let $\mathcal{G}$ be the subgroup of quadratic residues in $\mathbb{Z}_p^*$. Then $\mathcal{G}$ is a cyclic group of prime order q in which the DDH assumption is believed to hold.

The El-Gamal cryptosystem. In Lecture 4, we introduced the El-Gamal encryption scheme and proved that it was semantically secure under the DDH assumption; it may be useful to review that proof before continuing. We recall the scheme now (here, g is a generator of a group $\mathcal{G}$):

$\mathsf{KeyGen}(1^k)$: $x \leftarrow \mathbb{Z}_q$
$y = g^x$
$PK = \langle g, y \rangle$
$SK = \langle x \rangle$

$\mathcal{E}_{PK}(m)$: $r \leftarrow \{0,1\}^k$
output $\langle g^r, y^r \cdot m \rangle$

$\mathcal{D}_{SK}(u, v)$: output $\frac{v}{u^x}$

Correctness: Assuming an honest execution of the protocol, we have

$$\frac{v}{u^x} = \frac{y^r \cdot m}{(g^r)^x} = \frac{(g^x)^r \cdot m}{(g^r)^x} = m.$$

3 Modifying El-Gamal

To build up to the Cramer-Shoup scheme, we first modify the El-Gamal encryption scheme and prove that the modified scheme is semantically secure under the DDH assumption. Although we achieve the same result, we introduce the modified scheme because the proof technique used to prove the modified scheme secure is different than that used to prove security of the El Gamal scheme in Lecture 4. The same sort of techniques will later be used to analyze the Cramer-Shoup scheme.

Consider the following scheme, where g_1, g_2 are two randomly-chosen generators in $\mathcal{G}$:

$\mathsf{KeyGen}(1^k)$: $x, y \leftarrow \mathbb{Z}_q$
$h = g_1^x g_2^y$
$PK = \langle g_1, g_2, h \rangle$
$SK = \langle x, y \rangle$

[1] Note that in this example, the order of $\mathcal{G}$ is not prime. However, all groups we use in our constructions will have prime order.

$\mathcal{E}_{PK}(m)$: $r \leftarrow \mathbb{Z}_q$
output $\langle g_1^r,\ g_2^r,\ h^r \cdot m \rangle$

$\mathcal{D}_{SK}(u, v, e)$: output $\frac{e}{u^x v^y}$

CORRECTNESS: Assuming an honest execution of the protocol, we have

$$\frac{e}{u^x v^y} = \frac{h^r m}{(g_1^r)^x (g_2^r)^y} = \frac{h^r m}{(g_1^x g_2^y)^r} = m.$$

Theorem 1 *The above encryption scheme is semantically secure, assuming the DDH assumption in $\mathcal{G}$.*

Proof We prove security of this scheme by a reduction to the DDH problem. Suppose a PPT algorithm A can break the semantic security of the modified scheme. We then construct a PPT adversary $\hat{A}$ that can break the DDH problem by distinguishing a DDH tuple from a random tuple. Thus by contradiction, the security of the new scheme is proved.

The input to algorithm $\hat{A}$ is (g_1, g_2, g_3, g_4), which is either a DDH tuple or a random tuple. The algorithm $\hat{A}$ runs the following experiment.

$\hat{A}(g_1, g_2, g_3, g_4)$
$x, y \leftarrow \mathbb{Z}_q$
$h = g_1^x g_2^y$
$PK = \langle g_1, g_2, h \rangle$
$(m_0, m_1) \leftarrow A(PK)$
$b \leftarrow \{0, 1\}$
$C = \langle g_3,\ g_4,\ g_3^x g_4^y \cdot m_b \rangle$
$b' \leftarrow A(PK, C)$
if $b = b'$, then guess "DDH tuple"
else guess "random tuple"

Claim 2 *If adversary $\hat{A}$ gets a DDH tuple, then A's view of the game is the same as in an execution of the real encryption scheme.*

Assume $\hat{A}$ gets a DH tuple. Then there exist α, $r \in \mathbb{Z}_q$, such that:

$$\langle g_1, g_2 = g_1^\alpha, g_3 = g_1^r, g_4 = g_1^{\alpha r} = g_2^r \rangle.$$

Therefore, the constructed public key and ciphertext have the following forms:

$$PK = \langle g_1,\ g_2,\ h = g_1^x g_2^y \rangle \text{ and } C = \langle g_1^r,\ g_2^r,\ (g_1^r)^x\,(g_2^r)^y \cdot m_b \rangle = \langle g_1^r,\ g_2^r,\ h^r \cdot m_b \rangle\,.$$

Thus, the distribution of the public key and the ciphertext correspond exactly to A's view in the real world. (It should be noted that this occurs even though $\hat{A}$ does not know nor use the value of r.) □

Note that the claim implies:

$$\Pr[\hat{A} \text{ outputs "DDH tuple"} \mid \text{DDH tuple}] = \Pr[b' = b \mid A \text{ attacks real encryption scheme}].$$

Claim 3 *If adversary $\hat{A}$ gets a random tuple, then (with all but negligible probability) even an all-powerful A has no information about the bit b chosen by $\hat{A}$. In other words, b is information-theoretically hidden from A with all but negligible probability.*

An immediate corollary is that the probability that A correctly guesses b must be negligibly close to $1/2$ in this case (note that this holds even if A is all powerful). We continue with the proof of the claim.

Assume $\hat{A}$ gets a random tuple. Then there exist α, r, β chosen uniformly from $\mathbb{Z}_q$ such that (g_1, g_2, g_3, g_4) have the following form:

$$\langle g_1, g_2 = g_1^{\alpha}, g_3 = g_1^{r}, g_4 = g_1^{\beta} \rangle.$$

Note that with all but negligible probability, $\beta \neq \alpha r \pmod{q}$ and $\alpha \neq 0$. This is because, for example, $\beta = \alpha r$ with probability $1/q$, and q is exponentially large. From now on, we simply assume that these hold. Re-writing, this means that there exist $r, r' \in \mathbb{Z}_q$ with $r \neq r'$ such that $g_3 = g_1^{r}$ and $g_4 = g_2^{r'}$. We now look at A's information about x and y. Given the public key $PK = \langle g_1, g_2, h \rangle$, note that there are exactly q possible pairs (x, y) that could have been chosen by A. This is because h satisfies $h = g_1^x g_2^y$, and hence x and y satisfy

$$\log_{g_1} h = x + (\log_{g_1} g_2) \cdot y = x + \alpha y \,. \tag{1}$$

Now, for every $x \in \mathbb{Z}_q$ there is a unique $y \in \mathbb{Z}_q$ satisfying the above equation (and similarly for y). (We use the fact here that $\alpha \neq 0$.) In particular, then, there are exactly q solutions to the above equation and furthermore each of these possibilities are equally likely from the point of view of A.

Now, consider the term $g_3^x g_4^y$. We will be interested in the probability that $g_3^x g_4^y = \mu$, where μ is an arbitrary group element. In order for this to occur, we must have $\log_{g_1} \mu = \log_{g_1}(g_3^x g_4^y)$; i.e.:

$$\begin{aligned} &\log_{g_1} \mu \\ &\quad = \quad x \cdot \log_{g_1} g_3 + y \cdot \log_{g_1} g_4 \\ &\quad = \quad r \cdot x + r'\alpha \cdot y \,. \end{aligned} \tag{2}$$

Let $z_1 = \log_{g_1} h$ and $z_2 = \log_{g_1} \mu$. Then Equations (1) and (2) form a system of linear equations in x and y (over $\mathbb{Z}_q$) given by $\mathbf{B}\vec{x} = \vec{z}$, where

$$\mathbf{B} = \begin{pmatrix} 1 & \alpha \\ r & r'\alpha \end{pmatrix}, \quad \vec{x} = [x \;\; y]^T, \quad \vec{z} = [z_1 \;\; z_2]^T.$$

Assuming $r' \neq r$ and $\alpha \neq 0$ (see above), the matrix B above has rank 2 and therefore the above system of equations always has a (unique) solution in x, y. But since μ was an arbitrary group element, this means that *each possible value μ is possible* and moreover, *each value of μ is equally likely.* In other words, what we are saying is the following: given g_1, g_2, g_3, g_4, and $h = g_1^x g_2^y$ for x and y chosen uniformly at random from $\mathbb{Z}_q$ (and assuming $\log_{g_1} g_3 \neq \log_{g_2} g_4$), *even an all-powerful algorithm* cannot predict the value of $g_3^x g_4^y$ with probability better than $1/q$. (again, this is because all values of $g_3^x g_4^y$ are equally likely).

Since $g_3^x g_4^y$ is distributed uniformly in the group (from the point of view of A), it essentially acts like a "one-time pad" and thus A has no information (in an information-theoretic sense) about which message was encrypted, and hence no information about the value of b. This implies the claim. □

The above claim implies:

$$\Pr[\hat{A} \text{ outputs "DDH tuple"} \mid \text{random tuple}] = 1/2 \pm negl(k).$$

Thus, the advantage of $\hat{A}$ is negligibly close to:

$$\left|\Pr[b = b' \mid A \text{ attacks real scheme}] - 1/2\right|.$$

Since we know that the advantage of $\hat{A}$ must be negligible, this implies that the probability that A correctly guess the value of b must be negligibly close to $1/2$. But this exactly means that the encryption scheme is semantically secure, as desired. ■

4 The Cramer-Shoup-"Lite" Cryptosystem

We next define the Cramer-Shoup "lite" encryption scheme. This is a step toward the full Cramer-Shoup scheme, but is only secure against *non-adaptive* chosen-ciphertext attacks. The scheme is defined as follows (g_1, g_2 are randomly-chosen generators of group $\mathcal{G}$):

$\mathsf{KeyGen}(1^k)$: $x, y, a, b \leftarrow \mathbb{Z}_q$
$h = g_1^x g_2^y$
$c = g_1^a g_2^b$
$PK = \langle g_1, g_2, h, c \rangle$
$SK = \langle x, y, a, b \rangle$

$\mathcal{E}_{PK}(m)$: $r \leftarrow \mathbb{Z}_q$
output $\langle g_1^r,\ g_2^r,\ h^r \cdot m,\ c^r \rangle$

$\mathcal{D}_{SK}(u, v, e, w)$: // Verify w has the correct form
if $(w = u^a v^b)$, then output $\frac{e}{u^x v^y}$
else output $\perp$

Correctness: If the ciphertext is computed honestly, the validity check succeeds since

$$w = c^r = \left(g_1^a g_2^b\right)^r = (g_1^r)^a (g_2^r)^b = u^a v^b$$

and the message is then recovered as

$$\frac{e}{u^x v^y} = \frac{h^r m}{(g_1^r)^x (g_2^r)^y} = \frac{h^r m}{(g_1^x g_2^y)^r} = m.$$

We now prove the security of the scheme.

Theorem 4 *Under the DDH Assumption, the above encryption scheme is secure against non-adaptive chosen-ciphertext attack.*

Proof The proof is very similar to the proof of the previous theorem. As there, assume we are given a PPT algorithm A attacking the above encryption scheme. We construct algorithm $\hat{A}$ trying to distinguish DDH tuples from random tuples. As in the previous proof, we will argue that if the tuple given to $\hat{A}$ is a DDH tuple, then the view of A is identical to its view when attaching the above encryption scheme. On the other hand, if the tuple given to $\hat{A}$ is a random tuple, then A will have no information about the message that is encrypted in an information-theoretic sense. The difference here is that we will be considering the more difficult case of CCA-1 security, and we must show that the queries made to the decryption oracle by A will not reveal anything. With this in mind, let us begin a formal proof.

Given some adversary A attacking the above encryption scheme via a non-adaptive chosen-ciphertext attack, we construct an adversary $\hat{A}$ as follows:

$\underline{\hat{A}(g_1, g_2, g_3, g_4)}$
$x, y, a, b \leftarrow \mathbb{Z}_q$
$h = g_1^x g_2^y;\ c = g_1^a g_2^b$
$PK = \langle g_1, g_2, h, c \rangle$
$SK = \langle x, y, a, b \rangle$
$(m_0, m_1) \leftarrow A^{\mathcal{D}_{SK}(\cdot)}(PK)$
$b \leftarrow \{0, 1\}$
$C = \langle g_3,\ g_4,\ g_3^x g_4^y \cdot m_b,\ g_3^a g_4^b \rangle$
$b' \leftarrow A(PK, C)$
if $b = b'$, then guess "DDH tuple"
else guess "random tuple"

Claim 5 *If $\hat{A}$ gets a DDH tuple, then A's view of the game is the same as in an execution of the real Cramer-Shoup-lite encryption scheme.*

A corollary of this claim is that

$$\Pr[\hat{A} \text{ outputs "DDH tuple"} \mid \text{DDH tuple}] = \Pr[b' = b \mid A \text{ attacks real scheme}].$$

We now prove the claim.

Certainly the public key created by $\hat{A}$ is exactly identical to the public key seen by A in a real execution of the encryption scheme. In fact, the secret key held by $\hat{A}$ is also identical to that used in a real execution of the encryption scheme, and thus the decryption queries of A are answered exactly as they would be in a real execution of the encryption scheme. The only thing left to examine is the ciphertext. But if the input to $\mathcal{A}$ is a DDH tuple, then we can write $g_3 = g_1^r$ and $g_4 = g_2^r$ where r is uniformly distributed in $\mathbb{Z}_q$. But then simple algebra shows that the ciphertext is distributed identically to the challenge ciphertext in a real execution of the encryption scheme (details left to the reader). □

Claim 6 *If $\hat{A}$ gets a random tuple, then (with all but negligible probability) A has no information about the bit b chosen by $\hat{A}$. We remark that this holds in an information-theoretic sense, for all-powerful A, as long as A can only make polynomially-many queries to the decryption oracle.*

Before proving the claim, we show how this claim completes the proof of the theorem. The claim implies that the probability that A correctly guesses b is negligibly close to $1/2$ and therefore $\Pr[\hat{A}$ outputs "DDH tuple" | random tuple] is negligibly close to $1/2$ as well. Thus, the advantage of $\hat{A}$ is negligibly close to:

$$\left|\Pr[b = b' \mid A \text{ attacks real scheme}] - 1/2\right|.$$

Since the DDH assumption implies that the advantage of $\hat{A}$ is negligible, this implies that the probability that A correctly guesses the value of b when attacking the real scheme is negligibly close to $1/2$. But this is exactly the definition of CCA-1 security.

We return to the proof of the claim. The proof is similar to the analogous claim proven previously, in that we argue that A's information about x and y will not be enough to determine which message was encrypted. But we have to be a little more careful here because A can now potentially learn additional information about x and y from the decryption oracle queries it makes.

Let (g_1, g_2, g_3, g_4) be a random tuple. As before, we may write these as:

$$\langle g_1, g_2 = g_2^{\alpha}, g_3 = g_1^{r}, g_4 = g_2^{r'} \rangle,$$

where with all but negligible probability $\alpha \neq 0$ and $r \neq r'$ (and we assume this from now on). From PK, adversary A learns that $h = g_1^x g_2^y$ and this constrains x, y according to:

$$\log_{g_1} h = x + (\log_{g_1} g_2) \cdot y = x + \alpha y \tag{3}$$

exactly as before.

We now consider what additional information A learns about x, y from its queries to the decryption oracle. When A makes a decryption oracle query (μ, ν, e, w) there are two cases: either there exists an r'' such that $\mu = g_1^{r''}$ and $\nu = g_2^{r''}$ (and hence this ciphertext is "legal"), or not. We call queries of the latter form "illegal". We first show that A only learns additional information about x, y if it makes an illegal query which is not rejected. But we next show that (with all but negligible probability) all A's illegal queries are rejected. Putting this together will imply that A does not learn additional information about x, y with all but negligible probability.

Claim 7 *A gets additional information about x, y only if it submits a decryption query (μ, ν, e, w) such that:*

1. $\log_{g_1} \mu \neq \log_{g_2} \nu$ *(i.e., an illegal query), and*
2. $\mathcal{D}_{SK}(\cdot)$ *does not return* $\perp$.

To see this, first suppose that $\mathcal{D}_{SK}(\cdot)$ returns $\perp$. The only time this happens is when the decryption routine rejects because w is not of the correct form. But this check only involves a and b, and hence cannot reveal any information about x, y.

Next suppose that A submits a query for which $\log_{g_1} \mu = \log_{g_2} \nu = r''$ for some arbitrary r''. In this case, based on the output m of the decryption oracle, A learns that $m = \frac{e}{\mu^x \nu^y}$.

Taking logarithms of both sides means that A learns the following linear constraint on x and y:

$$\begin{aligned}&\log_{g_1} m\\ &= \log_{g_1} e - (\log_{g_1} \mu)x - (\alpha \log_{g_2} \nu)y\\ &= \log_{g_1} e - r''x - \alpha r''y\end{aligned}$$

(note that e and m are known, so x and y are the only variables here). But this equation is linearly *dependent* on Equation (3). Thus, this does *not* introduce any additional constraint on x, y and hence the adversary has not learned any additional information about x, y. □

Claim 8 *The probability that A submits a decryption query (μ, ν, e, w) for which $\log_{g_1} \mu \neq \log_{g_2} \nu$ but $\mathcal{D}_{SK}(\mu, \nu, e, w) \neq \perp$ is negligible.*

Let $\log_{g_1} \mu = r_1$ and $\log_{g_2} \nu = r_2$. In order for the decryption oracle to not reject, the adversary must "predict" the value of $\mu^a \nu^b$ (so that it can set w equal to this value). We show that it cannot do so with better than negligible probability.

Consider the information the adversary knows about a, b. From the public key, A learns that $c = g_1^a g_2^b$ and this constrains a, b according to:

$$\log_{g_1} c = a + (\log_{g_1} g_2) \cdot b = a + \alpha b\,. \tag{4}$$

The first time A makes an illegal decryption oracle query with $\log_{g_1} \mu \neq \log_{g_2} \nu$, the above equation represents all the information the adversary knows about a, b. Now, let w' be an arbitrary group element. The value of $\mu^a \nu^b$ is equal to w' exactly if:

$$\begin{aligned}&\log_{g_1} w'\\ &= a \log_{g_1} \mu + b \log_{g_1} \nu\\ &= r_1 \cdot a + \alpha r_2 \cdot b\,.\end{aligned} \tag{5}$$

But Equations (4) and (5) (viewed as equations in unknowns a, b over $\mathbb{Z}_q$) are linearly independent and hence have a solution in terms of a, b. Since this is true for *arbitrary* w', this means that any value of w' is possible (in fact, they are all equally likely) and hence A can only predict the correct value of w with probability $1/q$. (Note that this argument is substantially similar to the proof of Claim 3, above.)

Now, the above was true for the *first* illegal decryption query of A. However, each illegal decryption query of A *does* reveal some additional information about a, b. In particular, when an illegal query (μ, ν, e, w) is rejected the adversary learns that $w \neq \mu^a \nu^b$. At best, however, this eliminates one possibility for a, b. From Equation (4 alone, there are q possibilities for (a, b), and each rejected decryption query of A eliminates at most one of these solutions. Thus, at the time of the $(p+1)^{\text{th}}$ decryption query of A, assuming the first p of A's illegal decryption queries were rejected, there are (at least) $q - p$ possible solutions for (a, b). The argument of the previous paragraph now has to be modified to take this into account. But what we see is that eliminating one possibility for (a, b) has the effect of eliminating one possible value of w. So now the probability that A can correctly guess w is $1/(q-p)$.

Assume A makes a total of p decryption queries. Straightforward probability calculations show that the probability that *any* of A's illegal queries are *not* rejected is at most $p/(q-p)$ (in each of p illegal decryption queries, A has at best probability $1/(q-p)$ of the query not being rejected). But since p is polynomial and q is exponential, this is a negligible quantity. □

Putting the above two claims together shows that, with all but negligible probability, A never learns any additional information about x, y beyond that implied by Equation (3). Assuming this is the case, an argument exactly like that given in the proof of Claim 3 shows that $g_1^x g_2^y$ is uniformly distributed in the group (from the point of view of A) and hence A has no information about the value of b. This completes the proof of Claim 6, and thus the proof of the theorem. ■

References

[1] R. Cramer and V. Shoup. A Practical Public-Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. In *Adv. in Cryptology — CRYPTO 1998*.

CMSC 858K — Advanced Topics in Cryptography	February 26, 2004
Lecture 10	
Lecturer: Jonathan Katz	*Scribe(s): Jeffrey Blank, Chiu Yuen Koo, Nikolai Yakovenko*

1 Summary

We had previously begun to analyze the Cramer-Shoup encryption scheme by looking at a simplified version of the scheme that is secure only against *non-adaptive* chosen-ciphertext attacks. These notes revisit this discussion, and then go on to show the full Cramer-Shoup scheme that is secure against *adaptive* chosen-ciphertext attacks [1].

2 Simplified Cramer-Shoup

Recall the simplified Cramer-Shoup scheme:

- $PK = (g_1, g_2, h = g_1^x g_2^y, c = g_1^a g_2^b)$
- $SK = (x, y, a, b)$
- $\mathcal{E}_{PK}(m)$: choose random $r \in \mathbb{Z}_q$; set $C = (g_1^r,\ g_2^r,\ h^r \cdot m,\ c^r)$
- $\mathcal{D}_{SK}(u, v, w, e)$
  - If $e \neq u^a v^b$ then output $\perp$ (i.e., invalid)
  - else output $\frac{w}{u^x v^y}$

A proof of the following was given in the last lecture, but we briefly review it here.

Theorem 1 *Under the DDH assumption, the simplified Cramer-Shoup Scheme is secure against non-adaptive chosen-ciphertext attacks.*

Proof Based on a PPT adversary $\mathcal{A}$ attacking the simplified Cramer-Shoup scheme, we construct an adversary $\mathcal{A}'$ as follows:

$\underline{\mathcal{A}'(g_1, g_2, g_3, g_4)}$
$x, y, a, b \leftarrow \mathbb{Z}_q$
$PK = (g_1, g_2, h = g_1^x g_2^y, c = g_1^a g_2^b)$
$(m_0, m_1) \leftarrow \mathcal{A}(PK)$
$b \leftarrow \{0, 1\}$
$b' \leftarrow \mathcal{A}(PK,\ g_3,\ g_4,\ g_3^x g_4^y \cdot m_b,\ g_3^a g_4^b)$
output 1 iff $b' = b$

Let Rand be the event that (g_1, g_2, g_3, g_4) are chosen from the distribution of random tuples, and let DH be the event that they were chosen from the distribution of Diffie-Hellman tuples. We have the following claims:

Claim 2 $\left|\Pr[\mathcal{A}'=1|\mathsf{DH}] - \Pr[\mathcal{A}'=1|\mathsf{Rand}]\right| = \mathsf{negl}(k)$.

Proof This follows from the DDH assumption and the fact that $\mathcal{A}'$ is a PPT algorithm. ■

Claim 3 $\Pr[\mathcal{A}'=1|\mathsf{DH}] = \Pr_{\mathcal{A}}[\mathsf{Succ}]$.

Proof By inspection, if (g_1, g_2, g_3, g_4) is a DH tuple, then $g_3 = g_1^r$ and $g_4 = g_2^r$ for some (randomly distributed) $r \in \mathbb{Z}_q$. The claim follows. ■

Claim 4 $|\Pr[\mathcal{A}'=1|\mathsf{Rand}] - \frac{1}{2}| = \mathsf{negl}(k)$.

Proof Since $\Pr[\mathcal{A}'=1] = \Pr[\mathcal{A}=b]$, we will show that

$$|\Pr[\mathcal{A}=b \mid \mathsf{Rand}] - \frac{1}{2}| = \mathsf{negl}(k). \tag{1}$$

In fact, we will show that this is true even if $\mathcal{A}$ has unlimited computational power (but can only access the decryption oracle polynomially-many times). If $\mathcal{A}$ is all-powerful, we may assume it knows $\log_{g_1} g_2$; call this γ. From the public key, $\mathcal{A}$ learns that $h = g_1^x g_2^y$, or, equivalently (taking discrete logarithms of both sides):

$$\log_{g_1} h = x + y \cdot \gamma. \tag{2}$$

This collapses the space of (x, y) into q possible pairs, one for each value of $x \in \mathbb{Z}_q$ (and similarly for y). The following sub-claims show that $\mathcal{A}$ cannot learn any additional information about x and y using its decryption queries:

Sub-Claim Except with negligible probability, for all decryption queries (u, v, w, e) made by $\mathcal{A}$ such that $\log_{g_1} u \neq \log_{g_2} v$, decryption returns $\perp$.

Proof of Sub-Claim Before the first decryption query, the only thing $\mathcal{A}$ knows about a and b from the public key is that $c = g_1^a g_2^b$. Taking discrete logarithms of both sides gives:

$$\log_{g_1} c = a + b \cdot \gamma. \tag{3}$$

Consider some ciphertext (u, v, w, e) submitted to the decryption oracle, where $u = g_1^r$ and $v = g_2^{r'}$ with $r \neq r'$. For every $z \in \mathcal{G}$, there is exactly one pair $(a, b) \in \mathbb{Z}_q \times \mathbb{Z}_q$ satisfying Equation (3) and

$$z = u^a v^b \Leftrightarrow \log_{g_1} z = ar + br' \cdot \gamma \tag{4}$$

(this is because Equation 3 and Equation 4 are linearly independent in the unknowns a and b). Therefore, from the point of view of $\mathcal{A}$ the value of $u^a v^b$ is uniformly distributed in $\mathcal{G}$. Furthermore, the ciphertext (u, v, w, e) is rejected unless $e = u^a v^b$. Thus, the ciphertext is rejected except with (negligible) probability $1/q$ (i.e., unless $\mathcal{A}$ happens to choose e correctly).

Assuming the first decryption query of this form (i.e., with $\log_{g_1} u \neq \log_{g_2} v$) is rejected, all $\mathcal{A}$ learns is that $e \neq u^a v^b$. This eliminates one of the q possibilities for (a, b), but there are still $q - 1$ possibilities remaining. Thus, the same argument as above shows that for

the second decryption query of this form, the query will be rejected except with probability (at most) $1/(q-1)$. Continuing in this way, the n^{th} decryption query of this form will be rejected except with probability (at most) $1/(q-n+1)$. Thus, the probability that *one* of these queries is *not* rejected is at most $n/(q-n+1)$. Since q is exponential in k while n (the number of decryption queries) is polynomial in k, this proves the claim.

Sub-Claim Assuming all "bad" decryption queries of the form described above are rejected, $\mathcal{A}$ learns no additional information about x and y.

Proof of Sub-Claim Note that when a "bad" decryption query is rejected, $\mathcal{A}$ learns nothing about x and y (since the ciphertext is rejected based on a and b alone). So, we need only look at the "good" decryption queries (u, v, w, e); i.e., those for which $\log_{g_1} u = \log_{g_2} v = r$ (for some r). From the response m to such a query, $\mathcal{A}$ learns that $w/m = (g_1^r)^x (g_2^r)^y$, or:

$$\log_{g_1}(w/m) = xr + yr \cdot \gamma. \tag{5}$$

However, the above equation and Equation (2) are linearly *dependent*. Thus, no extra information about (x, y) is revealed.

The above claims show that, with all but negligible probability, when $\mathcal{A}$ gets the challenge ciphertext $(g_3,\ g_4,\ g_3^x g_4^y \cdot m_b,\ g_3^a g_4^b)$, there are q equally-likely possibilities for (x, y). Thus, with all but negligible probability $g_3^x g_4^y \cdot m_b$ is uniformly distributed over $\mathcal{G}$ and independent of b and hence $\Pr[\mathcal{A} = b] = \frac{1}{2}$. But this is equivalent to Equation (1), completing the proof for Claim 4 and Theorem 1. ■ ■

It is worth noting that the proof above fails to extend for the case of adaptive chosen-ciphertext attacks. The reason is as follows: in trying to extend the proof of Claim 4, we see now that the challenge ciphertext $(g_3,\ g_4,\ g_3^x g_4^y \cdot m_b,\ g_3^a g_4^b)$ gives an additional, independent linear constraint on the pair (a, b); namely:

$$\log_{g_1} e = a \log_{g_1} g_3 + \log_{g_1} g_4. \tag{6}$$

From Equations 3 and 6, $\mathcal{A}$ could (at least in theory) compute the values of a and b. Thus, the first sub-claim — though still true for the decryption queries made by $\mathcal{A}$ *before* seeing the challenge ciphertext — no longer holds for decryption queries made by $\mathcal{A}$ *after* seeing the challenge ciphertext (indeed, here we see exactly an example of the extra power given by an *adaptive* chosen-ciphertext attack). In particular, $\mathcal{A}$ can potentially make a query of the form $(g_1^r, g_2^{r'}, w, (g_1^r)^a (g_2^{r'})^b)$ (with $r \neq r'$), receive in return the answer m, and thus learn that:

$$\log_{g_1}(w/m) = xr + yr' \cdot \gamma. \tag{7}$$

Since Equations 2 and 7 are linearly independent, $\mathcal{A}$ can compute the values of x and y and decrypt the challenge ciphertext.

The above just shows where the proof breaks down, but does not show an explicit (poly-time) attack. However, such an attack on the simplified Cramer-Shoup scheme is easily derived, and is left as an exercise.

3 The Cramer-Shoup Cryptosystem

The discussion at the end of the last section illustrates that for the proof to extend, we need to ensure that the adversary still cannot submit "bad" ciphertexts which decrypt "properly", even after seeing the challenge ciphertext. We will achieve this by adding two more variables so that the number of unknowns will remain greater than the number of linear equations in these unknowns. The details follows.

Before fully describing the scheme, we note that the scheme will use a *collision-resistant hash function* H hashing arbitrary-length strings to $\mathbb{Z}_q$. We do not give a formal definition here, but content ourselves with the following, *informal* definition: a function is collision-resistant if an adversary cannot find two distinct inputs hashing to the same output in any "reasonable" amount of time. (In fact, a weaker assumption suffices to prove security for the Cramer-Shoup cryptosystem, but we do not explore this further here.)

The Cramer-Shoup scheme is as follows:

- $PK = (g_1, g_2, h = g_1^x g_2^y, c = g_1^a g_2^b, d = g_1^{a'} g_2^{b'}, H)$ where H is a collision-resistant hash function
- $SK = (x, y, a, b, a', b')$
- $\mathcal{E}_{PK}(m)$: Choose random $r \in \mathbb{Z}_q$, and set $C = (g_1^r, g_2^r, h^r \cdot m, (cd^\alpha)^r)$, where $\alpha = H(g_1^r, g_2^r, h^r \cdot m)$.
- $\mathcal{D}_{SK}(u, v, w, e)$
  - if $u^{a+\alpha a'} v^{b+\alpha b'} \neq e$ (where $\alpha = H(u, v, w)$) then output $\perp$ (i.e., invalid)
  - alse output $\frac{w}{u^x v^y}$

For an honestly-constructed ciphertext, we have:

$$\begin{aligned} u^{a+\alpha a'} v^{b+\alpha b'} &= u^a v^b (u^{a'} v^{b'})^\alpha \\ &= (g_1^a g_2^b)^r (g_1^{a'} g_2^{b'})^{r\alpha} \\ &= c^r d^{r\alpha} = (cd^\alpha)^r \end{aligned}$$

and so the validity check always succeeds. It can then be verified that decryption recovers the encrypted message.

Theorem 5 *Under the DDH assumption, the Cramer-Shoup scheme is secure against adaptive chosen-ciphertext attacks.*

Proof We proceed as we did in the previous proof, using the same notation as there. Given a PPT adversary $\mathcal{A}$ attacking the Cramer-Shoup Scheme that succeeds with $\Pr_{\mathcal{A}}[\mathsf{Succ}]$, we construct an adversary $\mathcal{A}'$ as follows:

$\underline{\mathcal{A}'(g_1, g_2, g_3, g_4)}$
$x, y, a, b, a', b' \leftarrow \mathbb{Z}_q$
$PK = (g_1, g_2, h = g_1^x g_2^y, c = g_1^a g_2^b, d = g_1^{a'} g_2^{b'}, H)$
$(m_0, m_1) \leftarrow \mathcal{A}(PK)$
$b \leftarrow \{0, 1\}$
$b' \leftarrow \mathcal{A}(PK,\ g_3,\ g_4,\ g_3^x g_4^y \cdot m_b,\ g_3^{a+\alpha a'} g_4^{b+\alpha b'})$
output 1 iff $b' = b$

The following two claims are exactly as in the previous proof:

Claim 6 $\left|\Pr[\mathcal{A}'=1|\mathsf{DH}] - \Pr[\mathcal{A}'=1|\mathsf{Rand}]\right| = \mathsf{negl}(k)$.

Claim 7 $\Pr[\mathcal{A}'=1|\mathsf{DH}] = \Pr_{\mathcal{A}}[\mathsf{Succ}]$.

To complete the proof, we prove the following claim. Note, however, that the proof is now more complicated than it was previously.

Claim 8 $\left|\Pr[\mathcal{A}'=1|\mathsf{Rand}] - \frac{1}{2}\right| = \mathsf{negl}(k)$.

Proof As before, we will show that the claim is true even if $\mathcal{A}$ were able to compute discrete logarithms (which, in general, it cannot since it runs in polynomial time). Furthermore, the overall structure of the proof will be the same — namely, we show that $\mathcal{A}$ cannot make any "bad" decryption queries that do not get rejected, and that conditioned on this fact $\mathcal{A}$ has no information about the encrypted message (since it does not have enough information about x, y). Since the latter part of the proof is the same, we only consider the first part of the proof here. The discussion below assumes the reader is familiar with the proof for the simplified Cramer-Shoup scheme given earlier.

Let us look at the information $\mathcal{A}$ possibly learns about a, b, a', b' during the course of the experiment. From the public key, $\mathcal{A}$ learns:

$$\log_{g_1} c = a + b \cdot \gamma \tag{8}$$
$$\log_{g_1} d = a' + b' \cdot \gamma\,, \tag{9}$$

where we again let $\gamma = \log_{g_1} g_2$. Let $g_3 = g_1^r$ and $g_4 = g_2^{r'}$; as usual, with all but negligible probability we have $r \neq r'$. When $\mathcal{A}$ is given the challenge ciphertext, denoted by $(g_3, g_4, w^* = g_3^x g_4^y \cdot m_b, e^* = g_3^{a+\alpha a'} g_4^{b+\alpha b'})$, $\mathcal{A}$ additionally learns:

$$\log_{g_1} e^* = (a + \alpha a')r + (b + \alpha b')\gamma r'. \tag{10}$$

As in the previous proof, we want to show that, with all but negligible probability, any "bad" decryption queries made by $\mathcal{A}$ — i.e., decryption queries (u, v, w, e) with $\log_{g_1} u \neq \log_{g_2} v$ — will be rejected. Recall that $\mathcal{A}$ is not allowed to submit $(u, v, w, e) = (u^*, v^*, w^*, e^*)$. We look at three possible cases:

Case 1. If $(u, v, w) = (u^*, v^*, w^*)$ but $e \neq e^*$, it is easy to see that this query will always be rejected.

Case 2. If $(u, v, w) \neq (u^*, v^*, w^*)$ but $H(u, v, w) = H(u^*, v^*, w^*)$ then this means that $\mathcal{A}$ has found a collision in H. But since H is collision-resistant and $\mathcal{A}$ runs in polynomial time, we may assume that this happens with only negligible probability.

Case 3. If $H(u, v, w) \neq H(u^*, v^*, w^*)$, then let $\alpha' = H(u, v, w)$ (and, as in Equation (10), $\alpha = H(u^*, v^*, w^*)$). Let $\log_{g_1} u = \hat{r}$ and $\log_{g_2} v = \hat{r}'$, and recall that since we are considering "bad" queries we have $\hat{r} \neq \hat{r}'$. Looking at the first "bad" decryption query made by $\mathcal{A}$, we see that it is not rejected only if:

$$\log_{g_1} e = (a + \alpha' a')\hat{r} + (b + \alpha' b')\gamma \hat{r}'.$$

The key point is that this equation is *linearly independent* of Equations (8)–(10), where these are being viewed as four equations in the four unknowns a, b, a', b'. (Linear independence can be verified by "brute-force" calculation, which is worth doing at least once. In fact, if you work it out you will find that the equations are linearly independent exactly when $r \neq r'$ and $\hat{r} \neq \hat{r}'$ and $\alpha \neq \alpha'$, which are exactly the conditions we consider!)

As in the previous proof, this means that the first "bad" decryption query of $\mathcal{A}$ is rejected except with probability $1/q$. Continuing in the same way as in the previous proof, we see that *all* of the "bad" decryption queries of $\mathcal{A}$ are rejected, except with negligible probability (here, we use the fact that $\mathcal{A}$ may make only polynomially-many queries). This concludes the proof of the claim, and hence the proof of the theorem. ■ ■

References

[1] R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. Crypto '98. Full version available from http://eprint.iacr.org.

CMSC 858K — Advanced Topics in Cryptography March 2, 2004

Lecture 11

Lecturer: Jonathan Katz *Scribe(s): Rengarajan Aravamudhan, Nan Wang*

1 Review of the Cramer-Shoup Encryption Scheme

At the beginning of this lecture, we reviewed the proofs of security for both the Cramer-Shoup "lite" and the full Cramer-Shoup schemes. However, rather than repeating the proofs here, we instead refer the interested reader to the previous lecture notes.

2 NIZK Proof Systems

Previously in the course, we have seen the Naor-Yung and Dolev-Dwork-Naor encryption schemes, both of which rely on adaptively-secure non-interactive zero-knowledge (NIZK) proof systems. In the next few lectures, we will see how to construct such proof systems. Our discussion is drawn from the work of Feige-Lapidot-Shamir [7, 3, 2, 4] and is also based on [5, Section 4.10]. We define both "non-adaptive" NIZK and adaptively-secure NIZK. Although we need adaptively-secure NIZK for our applications, non-adaptive NIZK will be useful toward developing intuition for the problem.

Definition 1 ((Non-Adaptive) NIZK) A pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$ is an *NIZK proof system* for a language $L \in \mathcal{NP}$ if, for some polynomials p_1, p_2:

- **(Completeness)** For all $x \in L \cap \{0,1\}^{\leq p_1(k)}$ and witness w for x,

$$\Pr[r \leftarrow \{0,1\}^{p_2(k)}; \pi \leftarrow \mathcal{P}(1^k, r, x, w) : \mathcal{V}(1^k, r, x, \pi) = 1] = 1.$$

  We say π is *valid proof for* x (assuming k, r are clear from context) if $\mathcal{V}(1^k, r, x, \pi) = 1$.

- **(Soundness)** For all (even unbounded) $\mathcal{P}^*$ and all $x \notin L$, the following is negligible:

$$\Pr[r \leftarrow \{0,1\}^{p_2(k)}; \pi \leftarrow \mathcal{P}^*(r, x) : \mathcal{V}(1^k, r, x, \pi) = 1].$$

- **(Zero-Knowledge)** There exists a PPT simulator Sim such that the following two ensembles are computationally indistinguishable for all PPT A:

$$\{(x, w) \leftarrow A(1^k); r \leftarrow \{0,1\}^{p_2(k)}; \pi \leftarrow \mathcal{P}(1^k, r, x, w) : (r, x, \pi)\}$$

$$\{(x, w) \leftarrow A(1^k); (r, \pi) \leftarrow \mathsf{Sim}(1^k, x) : (r, x, \pi)\}.$$

  (We require that $A(1^k)$ output (x, w) with $x \in L \cap \{0,1\}^{\leq p_1(k)}$ and w a witness for x.)

$\diamondsuit$

The following definition strengthens the previous one in two ways: first, soundness holds even when the (cheating) $\mathcal{P}^*$ chooses $x \notin L$ *after* seeing the random string r (i.e.,

adaptively). Second, the zero-knowledge property holds even when the simulator learns x *after* fixing its simulated random string r (in particular, the zero-knowledge property holds even if an efficient adversary chooses x after seeing the simulated random string).

Definition 2 (Adaptively-Secure NIZK) A pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$ is an *adaptively-secure NIZK proof system* for a language $L \in \mathcal{NP}$ if, for some polynomials p_1, p_2:

- **(Completeness)** As before. Again, we say π is *a valid proof of* x (assuming k, r are clear from context) if $\mathcal{V}(1^k, r, x, \pi) = 1$.
- **(Adaptive Soundness)** For all (even unbounded) $\mathcal{P}^*$, the following is negligible:

$$\Pr[r \leftarrow \{0,1\}^{p_2(k)}; (x, \pi) \leftarrow \mathcal{P}^*(r) : \mathcal{V}(1^k, r, x, \pi) = 1 \wedge x \notin L].$$

- **(Adaptive Zero-Knowledge)** There exists a PPT simulator $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that for all PPT adversaries A the following are computationally indistinguishable:

$$\{r \leftarrow \{0,1\}^{p_2(k)}; (x, w) \leftarrow A(1^k, r); \pi \leftarrow \mathcal{P}(1^k, r, x, w) : (r, x, \pi)\}$$

$$\{(r, \mathsf{state}) \leftarrow \mathsf{Sim}_1(1^k); (x, w) \leftarrow A(1^k, r); \pi \leftarrow \mathsf{Sim}_2(1^k, x, \mathsf{state}) : (r, x, \pi)\},$$

  where we require that $A(1^k, \cdot)$ output a pair (x, w) with $x \in L \cap \{0,1\}^{\leq p_1(k)}$ and w a witness for x.

$\diamondsuit$

We note that it is easy to transform any NIZK proof system into one achieving adaptive *soundness* as follows: Let $(\mathcal{P}, \mathcal{V})$ be an NIZK proof system satisfying "non-adaptive" soundness. Assume for simplicity that $p_1(k) = k$, and let $\mathsf{Bad}_k = \bar{L} \cap \{0,1\}^{\leq k}$. Soundness implies that for any fixed $x \in \mathsf{Bad}_k$, the probability over random choice of r that there exists a valid proof π for x is negligible, and in particular less than $1/2$. (We assume that $\mathcal{V}$ simply rejects if the statement x is longer then $p_1(k)$). Consider the modified protocol $(\mathcal{P}', \mathcal{V}')$ using a random string of length $2k \cdot p_2(k)$ (where $p_2(\cdot)$ is the length of the random string used by $(\mathcal{P}, \mathcal{V})$). The prover $\mathcal{P}'(1^k, r', x, w)$ parses the given random string r' as $2k$ strings $r_1, \ldots, r_{2k}$ and runs $\mathcal{P}(1^k, r_i, x, w)$ using each of these strings to generate proofs $\pi_1, \ldots, \pi_{2k}$. The verifier $\mathcal{V}'$ accepts only if all of these proofs are valid (with respect to $\mathcal{V}$).

It is not hard to see that completeness and (non-adaptive) zero-knowledge are unaffected by this transformation. We now bound the probability, over random choice of common random string r', that there exists an $x \in \mathsf{Bad}_k$ and a valid proof $\pi = \pi_1, \ldots, \pi_n$ for x. For any fixed $x \in \mathsf{Bad}_k$, a simple probability calculation shows that the probability, over r', that there exists a valid proof for x is at most 2^{-2k}. In other words, for any given $x \in \mathsf{Bad}_k$ at most a fraction 2^{-2k} strings r' are "bad" for this x. Then summing over all 2^{k+1} strings in Bad_k shows that at most a fraction 2^{-k+1} strings r' are "bad" for *some* $x \in \mathsf{Bad}_k$. In other words, the probability of such a "bad" r' is at most 2^{-k+1}, which is negligible.

We remark that it is unknown how to convert an arbitrary NIZK proof system into one achieving adaptive zero-knowledge. However, we will see a specific construction that works. Let us briefly outline the next few lectures:

- We first introduce the "hidden-bits" model and show that any NIZK proof system in this model can be transformed to an NIZK proof system in the real model (i.e., where a common reference string is available) assuming the existence of trapdoor permutations.
- We then show how to construct an NIZK proof system in the "hidden-bits" model. Coupled with the previous result, this yields a construction of an NIZK proof system in the real model.
- Finally, we note that the construction above actually achieves *adaptively-secure* NIZK without any further modification.

3 The Hidden-Bits Model

Informally, an NIZK proof system in the hidden-bits model proceeds as follows: the prover is initially given some sequence of bits which are hidden from the verifier. In the course of proving that $x \in L$, the prover can choose to reveal some arbitrary set of these bits to the verifier. The verifier never learns the bits of the string that are *not* revealed to it by the prover, and the prover cannot cheat and change the values in the string it is given. Formally, we imagine that the prover is given a string r of length n and sends to the verifier (along with other information) a set of indices $I \subseteq [n]$ (where $[n] = \{1, \ldots, n\}$). The verifier is then given the bits $\{r_i\}_{i \in I}$, which we denote by r_I. We stress that the hidden-bits model is not meant to be a realistic, but is instead only a conceptual model useful as a step toward our ultimate goal. The full definition follows.

Definition 3 (NIZK in the Hidden-Bits Model) A pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$ is an *NIZK proof system in the hidden-bits model* if, for some polynomials p_1, p_2:

- **(Completeness)** For all $x \in L \cap \{0,1\}^{\leq p_1(k)}$ and witnesses w for x:

$$\Pr[r \leftarrow \{0,1\}^{p_2(k)}; (\pi, I) \leftarrow \mathcal{P}(1^k, r, x, w) : \mathcal{V}(1^k, r_I, x, \pi, I) = 1] = 1.$$

- **(Soundness)** For all (unbounded) $\mathcal{P}^*$, the following is negligible:

$$\Pr[r \leftarrow \{0,1\}^{p_2(k)}; (x, \pi, I) \leftarrow \mathcal{P}^*(r) : \mathcal{V}(1^k, r_I, x, \pi, I) = 1 \wedge x \notin L].$$

- **(Zero-Knowledge)** There exists a PPT simulator Sim such that the following are computationally indistinguishable for all PPT A:

$$\{(x, w) \leftarrow A(1^k); r \leftarrow \{0,1\}^{p_2(k)}; (\pi, I) \leftarrow \mathcal{P}(1^k, r, x, w) : (r_I, x, \pi, I)\};$$

$$\{(x, w) \leftarrow A(1^k); (r_I, \pi, I) \leftarrow \mathsf{Sim}(1^k, x) : (r_I, x, \pi, I)\}.$$

  (We require that $A(1^k)$ output (x, w) with $x \in L \cap \{0,1\}^{\leq p_1(k)}$ and w a witness for x.)

♢

We now show how to transform any NIZK proof system in the hidden-bits model to an NIZK proof system in the model where there is a common random string available to the players. The transformation is secure assuming the existence of any trapdoor permutation family. (See [6, Appendix C] for further details on necessary assumptions.)

Theorem 1 *Assuming the existence of trapdoor permutations and any NIZK proof system $(\mathcal{P}', \mathcal{V}')$ in the hidden-bits model, we may construct an NIZK proof system $(\mathcal{P}, \mathcal{V})$ in the common random string model.*

Proof We first show the construction, and then prove that the construction works as claimed. We use the following notation for our trapdoor permutation family[1]: algorithm $\mathsf{Gen}(1^k)$ is a randomized algorithm which outputs a pair of (efficiently computable) functions (f, f^{-1}) where f^{-1} is called the "trapdoor" for f. Furthermore, f is always a permutation over $\{0,1\}^k$, and $f^{-1}(f(x)) = x$ for all $x \in \{0,1\}^k$. We also assume that the set of "legal" f's (i.e., those that can possibly be output by Gen) is efficiently decidable.[2] Finally, we let h denote a hard-core bit for this trapdoor permutation family. Formally, this means that for all PPT algorithms A the following is neglible:

$$\left| \Pr\left[(f, f^{-1}) \leftarrow \mathsf{Gen}(1^k); x \leftarrow \{0,1\}^k; y = f(x) : A(1^k, f, y) = h(x) \right] - \frac{1}{2} \right|.$$

We also assume that $h(x)$, for randomly-chosen x, gives an perfectly unbiased bit (this is not essential, but it makes the proof slightly easier).

We make the following additional assumptions without loss of generality. First, we assume that the random string used by $\mathsf{Gen}(1^k)$ has length k, and hence the maximum number of different f's that can be output is 2^k. We also assume that the soundness error of $(\mathcal{P}', \mathcal{V}')$ (i.e., the probability that a cheating $\mathcal{P}^*$ succeeds in giving a valid proof for some $x \notin L$) is at most 2^{-2k}. Using the technique shown earlier in these notes, if $(\mathcal{P}', \mathcal{V}')$ does not satisfy this condition we may construct a new NIZK proof system (still in the hidden-bits model) that *does* satisfy this condition by running $2k$ copies of $(\mathcal{P}', \mathcal{V}')$ in parallel.

Let $n = p_2(k)$ refer to the length of the string r' given to $\mathcal{P}'$ in the hidden-bits model for security parameter k; we simply write n when k is clear from context. In the common random string model, we let the string r shared by $\mathcal{P}$ and $\mathcal{V}$ have length $k \cdot n$. Given a common string $r = r_1 | \cdots | r_n$, where each $r_i \in \{0,1\}^k$, the prover and verifier proceed as follows:

1. $\mathcal{P}(1^k, r, x, w)$ runs $\mathsf{Gen}(1^k)$ to obtain (f, f^{-1}). It then computes an n-bit string r' by setting $r'_i = h(f^{-1}(r_i))$ (here, r'_i simply denotes the i^{th} bit of r').
2. $\mathcal{P}$ then runs $\mathcal{P}'(1^k, r', x, w)$ to obtain π, I. Finally, $\mathcal{P}$ outputs f, π, I, and $\{f^{-1}(r_i)\}_{i \in I}$.
3. $\mathcal{V}(1^k, r, x, (f, \pi, I, \{z_i\}_{i \in I}))$ proceeds as follows: it first checks that f is valid (here is where we use the fact that the set of f's generated by Gen is efficiently decidable). For each $i \in I$, it checks that $f(z_i) = r_i$ (if any of these fail, then $\mathcal{V}$ outputs 0). Next, it sets $r'_i = h(z_i)$ for each $i \in I$. Finally, it outputs $\mathcal{V}'(1^k, r'_I, x, \pi, I)$.

The intuition is as follows: assume for a moment that the prover honestly generates (f, f^{-1}) at random, independent of r. Then the string r' constructed by the prover is uniformly distributed (to see this, note that once f^{-1} is fixed independent of r then $r'_i = h(f^{-1}(r_i))$

[1] Again, see [6, Appendix C] for more careful treatment of what assumptions are necessary.

[2] This assumption is necessary for the construction given below. However, it is possible to remove this assumption using a more complicated construction [1].

is an unbiased bit for each i). Having the prover send $z_i = f^{-1}(r_i)$ to the verifier has the effect of "revealing" the i^{th} bit of r' to the verifier; also once f^{-1} is fixed the prover cannot "cheat" by changing the value of r'_i (since the verifier will check that $f(z_i) = r_i$ before computing $r'_i = h(x_i)$, and the inverse of r_i under f is unique). Finally, at least informally, the bits of r' that are not revealed by the prover to the verifier remain "hidden" by the security of f^{-1} and its associated hard-core bit h.

It is immediate that $(\mathcal{P}, \mathcal{V})$ satisfies completeness. Next, we prove the soundness of $(\mathcal{P}, \mathcal{V})$. Say that $\mathcal{P}^*$ *cheats* if it outputs a valid proof for some $x \notin L$. First consider any fixed (f, f^{-1}). By what we have said above, the string r' generated using this pair will be uniformly distributed and hence the soundness of $(\mathcal{P}', \mathcal{V}')$ implies that for any cheating $\mathcal{P}^*$ we have

$$\Pr_r[\mathcal{P}^* \text{ can cheat using } f] \leq 2^{-2k}.$$

However, there is nothing to prevent $\mathcal{P}^*$ from generating and using some (f, f^{-1}) in a way which *depends* on r! (For example, it is easy to construct a cheating prover that always picks (f, f^{-1}) in such a way that $r'_1 = 0$, say.) We now use the fact that the number of possible (valid) f's is at most 2^k, and also that $\mathcal{P}^*$ cannot send an invalid f to $\mathcal{V}$ without being caught. Summing the above inequality over all possible f's shows that:

$$\Pr_r[\mathcal{P}^* \text{ can cheat using } \textit{any } f] \leq 2^k \cdot 2^{-2k} = 2^{-k},$$

which is negligible.

To complete the proof, we show a zero-knowledge simulator for $(\mathcal{P}, \mathcal{V})$. Let Sim' be the simulator for $(\mathcal{P}', \mathcal{V}')$. We construct simulator Sim for $(\mathcal{P}, \mathcal{V})$ as follows:

$\underline{\mathsf{Sim}(1^k, x)}$
$(r'_I, \pi, I) \leftarrow \mathsf{Sim}'(1^k, x)$;
$(f, f^{-1}) \leftarrow \mathsf{Gen}(1^k)$;
for $i \in I$:
  $z_i \leftarrow \{0,1\}^k$ s.t. $h(z_i) = r'_i$;
  $r_i = f(z_i)$;
for $i \notin I$:
  $r_i \leftarrow \{0,1\}^k$;
output $(r, f, \pi, I, \{z_i\}_{i \in I})$

Intuitively, there are two differences between real proofs (given by $\mathcal{P}$) and simulated proofs (given by Sim): first, the simulated proofs use the simulator Sim' for the original proof system rather than the actual prover $\mathcal{P}'$ for the original proof system. Second, the values $\{r_i\}_{i \notin I}$ now define completely random bits $\{r'_i\}_{i \notin I}$ in the underlying string r'; this is not necessarily so for real proofs. However, a hybrid argument will show that the above differences are inconsequential: the first due to the zero-knowledge of $(\mathcal{P}', \mathcal{V}')$ and the second due to the security of the trapdoor permutation family.

Formally, let A be a PPT algorithm. Our goal is to show that

$$\left\{(x, w) \leftarrow A(1^k); r \leftarrow \{0,1\}^{k \cdot n}; (f, \pi, I, \{z_i\}_{i \in I}) \leftarrow \mathcal{P}(1^k, r, x, w) : (r, x, f, \pi, I, \{z_i\}_{i \in I})\right\} \quad (1)$$

and

$$\left\{(x, w) \leftarrow A(1^k); (r, f, \pi, I, \{z_i\}_{i \in I}) \leftarrow \mathsf{Sim}(1^k, x) : (r, x, f, \pi, I, \{z_i\}_{i \in I})\right\} \quad (2)$$

are computationally indistinguishable (cf. Definition 1). We define an intermediate experiment via an algorithm Hybrid as follows:

$\underline{\mathsf{Hybrid}(1^k, x, w)}$
$r' \leftarrow \{0,1\}^n$;
$(\pi, I) \leftarrow \mathcal{P}'(1^k, r', x, w)$;
$(f, f^{-1}) \leftarrow \mathsf{Gen}(1^k)$;
for $i \in I$:
  $z_i \leftarrow \{0,1\}^k$ s.t. $h(z_i) = r'_i$;
  $r_i = f(z_i)$;
for $i \notin I$:
  $r_i \leftarrow \{0,1\}^k$;
output $(r, f, \pi, I, \{z_i\}_{i \in I})$

Claim 2 *Assuming that $(\mathcal{P}', \mathcal{V}')$ is an NIZK proof system in the hidden-bits model with simulation Sim', ensemble (2) is computationally indistinguishable from the following:*

$$\left\{(x,w) \leftarrow A(1^k); (r, f, \pi, I, \{z_i\}_{i \in I}) \leftarrow \mathsf{Hybrid}(1^k, x, w) : (r, x, f, \pi, I, \{z_i\}_{i \in I})\right\}. \tag{3}$$

Assume to the contrary that the claim is false. Then there is a PPT distinguisher D which can distinguish between the two ensembles with probability that is not negligible. We construct a D' which violates the claimed security of Sim' as a zero-knowledge simulator for the proof system $(\mathcal{P}', \mathcal{V}')$ in the hidden-bits model.

Let A output (x, w) as above. D' is then given a tuple (r'_I, x, π, I) (coming either from the real prover $\mathcal{P}'$ in the hidden-bits model, or from Sim') and runs as follows:

$\underline{D'(1^k, r'_I, x, \pi, I)}$
$(f, f^{-1}) \leftarrow \mathsf{Gen}(1^k)$;
for $i \in I$:
  $z_i \leftarrow \{0,1\}^k$ s.t. $h(z_i) = r'_i$;
  $r_i = f(z_i)$;
for $i \notin I$:
  $r_i \leftarrow \{0,1\}^k$;
output $D(r, x, f, \pi, I, \{z_i\}_{i \in I})$

One can check that if (r'_I, x, π, I) is distributed according to real proofs generated by $\mathcal{P}'$, then the input to D is distributed according to (3). On the other hand, if (r'_I, x, π, I) is distributed as the output of Sim', then the input to D is distributed according to (2). So the distinguishing advantage of D' (in distinguishing real proofs generated by $\mathcal{P}'$ from simulated proofs generated by Sim') is equal to the distinguishing advantage of D. But this contradicts the zero-knowledge property of $(\mathcal{P}', \mathcal{V}')$ with respect to Sim'. □

Claim 3 *Assuming that Gen defines a secure trapdoor permutation family, ensemble (1) is computationally indistinguishable from ensemble (3).*

Again, we prove this by contradiction. Assume to the contrary that there is a PPT distinguisher D which can distinguish between the two ensembles with a probability that is not

negligible. We then construct a D' that violates the security of the trapdoor permutation family. Before doing so, we note the following easy-to-prove fact. The security of Gen (and a standard hybrid argument) implies that no PPT algorithm D', given a randomly-generated f, outputting a sequence of bits $r'_1, \ldots, r'_\ell$, and receiving in return a sequence of k-bit values $r_1, \ldots, r_\ell$, can distinguish between the case when each r_i is randomly chosen in $\{0,1\}^k$ and the case when each r_i is randomly chosen in $\{0,1\}^k$ subject to $h(f^{-1}(r_i)) = r'_i$.

Define D' as follows:

$\underline{D'(1^k, f)}$
$(x, w) \leftarrow A(1^k)$;
$r' \leftarrow \{0,1\}^n$;
$(\pi, I) \leftarrow \mathcal{P}'(1^k, r', x, w)$;
for $i \in I$:
  $z_i \leftarrow \{0,1\}^k$ s.t. $h(z_i) = r'_i$;
  $r_i = f(z_i)$;
output $r'_{\bar{I}}$ and get back $r_{\bar{I}}$;
// note: for all i, $|r'_i| = 1$ and $|r_i| = k$
output $D(r, x, f, \pi, I, \{z_i\}_{i \in I})$

It is easy to see that in case the values $r_{\bar{I}}$ are randomly chosen in $\{0,1\}^k$, then the input to D is distributed according to (3). Though harder to see, it is also the case that when the values $r_{\bar{I}}$ are randomly chosen in $\{0,1\}^k$ subject to $h(f^{-1}(r_i)) = r'_i$ then the input to D is distributed according to (1). To see this, note that in (1) r and f are chosen (independently) at random and these are used to generate r' by setting $r'_i = h(f^{-1}(r_i))$; on the other hand, in the above experiment (when the case in question occurs) r' and f are chosen (independently) at random and then r is chosen randomly subject to $h(f^{-1}(r_i)) = r'_i$. Thus, the distributions on (r, f, r') are the same in both cases. Furthermore, these values determine the remaining values in each experiment in the same way.

The above shows that the distinguishing advantage of D' (in determining which case occurs) is equal to the distinguishing advantage of D. But, as we have noted above, this contradicts the security of the trapdoor permutation family. □

The above two claims complete the proof of the theorem via a standard hybrid argument. ■

References

[1] M. Bellare and M. Yung. Certifying Permutations: Non-Interactive Zero-Knowledge Based on any Trapdoor Permutation. *J. Crypto.* 9: 149–166, 1996.

[2] U. Feige. *Alternative Models for Zero-Knowledge Interactive Proofs.* PhD Thesis, Dept. of Computer Science and Applied Mathematics, Weizmann Institute of Science, 1990. Available from http://www.wisdom.weizmann.ac.il/~feige.

[3] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs Based on a Single Random String. *FOCS*, pp. 308–317, 1990.

[4] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs Under General Assumptions. *SIAM J. Computing*, Vol. 29, No. 1, pp. 1–28, 1999.

[5] O. Goldreich. *Foundations of Cryptography, vol. 1: Basic Tools*, Cambridge University Press, 2001.

[6] O. Goldreich. *Foundations of Cryptography, vol. 2*, to appear. Preliminary versions available from Goldreich's web page.

[7] D. Lapidot and A. Shamir. Publicly Verifiable Non-Interactive Zero-Knowledge Proofs. *Advances in Cryptology — CRYPTO '90*, pp. 353–365, 1990.

CMSC 858K — Advanced Topics in Cryptography March 4, 2004

Lecture 12

Lecturer: Jonathan Katz

Scribe(s): Omer Horvitz, Zhongchao Yu, John Trafton, Akhil Gupta

1 Introduction

Our goal is to construct an adaptively-secure non-interactive zero-knowledge (NIZK) proof system for any language in NP; we will do so in several steps. We first define the hidden-bits model, and show how to transform any NIZK proof system for a language L in the hidden-bits model into an NIZK proof system for L in the common random string model, using trapdoor permutations. We will then construct an NIZK proof system for any language in NP in the hidden-bits model.[1] Our exposition draws from the work of Feige, Lapidot, and Shamir [6, 2, 1, 3] and also the presentation of [4, Section 4.10].

1.1 From the Hidden-Bits Model to the CRS model

We begin with a quick review of the definitions at hand.

Definition 1 A pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$ is a *non-adaptive NIZK proof system* for a language $L \in$ NP in the *common random string* (*CRS*) model if:

1. Completeness: For all $x \in L$ where $|x| = k$ and all witnesses w for x,

$$\Pr[r \leftarrow \{0,1\}^{\mathrm{poly}(k)}; \Pi \leftarrow \mathcal{P}(r, x, w) : \mathcal{V}(r, x, \Pi) = 1] = 1.$$

2. (Adaptive) Soundness: For any (unbounded) algorithm $\mathcal{P}^*$, the following is negligible:

$$\Pr[r \leftarrow \{0,1\}^{\mathrm{poly}(k)}; (x, \Pi) \leftarrow \mathcal{P}^*(r) : \mathcal{V}(r, x, \Pi) = 1 \wedge x \notin L].$$

3. Zero-knowledge: There exists a PPT algorithm Sim such the following ensembles are computationally indistinguishable for all PPT A:

$$\begin{array}{ll} (1) & \{(x, w) \leftarrow A(1^k); r \leftarrow \{0,1\}^{\mathrm{poly}(k)}; \Pi \leftarrow \mathcal{P}(r, x, w) : (r, x, \Pi)\} \\ (2) & \{(x, w) \leftarrow A(1^k); (r, \Pi) \leftarrow \mathsf{Sim}(x) : (r, x, \Pi)\}, \end{array}$$

   where $x \in L$, $|x| = k$, and w is any witness for x.

♢

In the above, r is called the *common random string*.

[1] We focus on the case of *non-adaptive* NIZK. However, careful examination of the constructions show that we actually end up with *adaptively-secure* NIZK without any additional modifications.

Definition 2 A pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$ is a *non-adaptive NIZK proof system* for a language $L \in \text{NP}$ in the *"hidden-bits"* model if:

1. Completeness: For all $x \in L$ where $|x| = k$ and all witnesses w for x,

$$\Pr[b \leftarrow \{0,1\}^{\text{poly}(k)}; (\Pi, I) \leftarrow \mathcal{P}(b, x, w) : \mathcal{V}(\{b_i\}_{i \in I}, I, x, \Pi) = 1] = 1.$$

2. (Adaptive) Soundness: For any (unbounded) algorithm $\mathcal{P}^*$, the following is negligible:

$$\Pr[b \leftarrow \{0,1\}^{\text{poly}(k)}; (x, \Pi, I) \leftarrow \mathcal{P}^*(b) : \mathcal{V}(\{b_i\}_{i \in I}, I, x, \Pi) = 1 \wedge x \notin L].$$

3. Zero-knowledge: There exists a PPT algorithm Sim such the following ensembles are computationally indistinguishable for any PPT A:

$$(1) \quad \{(x, w) \leftarrow A(1^k); b \leftarrow \{0,1\}^{\text{poly}(k)}; (\Pi, I) \leftarrow \mathcal{P}(b, x, w) : (\{b_i\}_{i \in I}, I, x, \Pi)\}$$
$$(2) \quad \{(x, w) \leftarrow A(1^k); (\{b_i\}_{i \in I}, I, \Pi) \leftarrow \mathsf{Sim}(x) : (\{b_i\}_{i \in I}, I, x, \Pi)\},$$

where $x \in L$, $|x| = k$, and w is any witness for x.

$\diamondsuit$

In the above, b is called the *hidden-bits string* and the $\{b_i\}_{i \in I}$ are the *revealed bits*. We denote the latter by b_I for brevity.

Let $(\mathcal{P}'', \mathcal{V}'')$ be a non-adaptive NIZK proof system for $L \in NP$ in the hidden-bits model. First, we convert the system into one with a precise bound on the soundness error; this will be useful in the analysis of our main transformation. The idea is to run the given system enough times in parallel. Assume that on input x of length k, $(\mathcal{P}'', \mathcal{V}'')$ uses a hidden-bits string of length $p(k)$, for some polynomial p. Define $(\mathcal{P}', \mathcal{V}')$ as follows[2]:

$\mathcal{P}'(b = b_1 \cdots b_{2k}, x, w)$ // $b_j \in \{0,1\}^{p(k)}$
  For $j = 1$ to $2k$, do
    $(\Pi_j, I_j) \leftarrow \mathcal{P}''(b_j, x, w)$;
  Let $\Pi = \Pi_1 | \cdots | \Pi_{2k}$ and $I = \cup_{j=1}^{2k} I_j$
  Output Π, I.

$\mathcal{V}'(b_I, I, x, \Pi)$
  parse Π as $\Pi_1 | \cdots | \Pi_{2k}$ and I as $\cup_{j=1}^{2k} I_j$ (for simplicity, we assume this can be done easily, in some uniquely-specified way)
  If $\mathcal{V}''(b_{I_j}, I_j, x, \Pi_j) = 1$ for *all* $1 \le j \le 2k$ then
    output 1;
  else, output 0.

Claim 1 *If $(\mathcal{P}'', \mathcal{V}'')$ is a non-adaptive NIZK proof system for L in the hidden-bits model, then $(\mathcal{P}', \mathcal{V}')$ is a non-adaptive NIZK proof system for L in the hidden-bits model with soundness error at most 2^{-2k}.*

[2]We will slightly abuse the notation here, formatting the inputs and outputs of the prover and verifier in a manner that strays from the one specified in the definition, for clarity; this is purely syntactic.

In the previous lecture, we proved a substantially similar result; we therefore omit proof here.

We would now like to convert $(\mathcal{P}', \mathcal{V}')$ into a non-adaptive NIZK proof system for L in the CRS model. The idea is to use the CRS to "simulate" the hidden-bits string. This is done by treating the CRS as a sequence of images of a one-way trapdoor permutation, and setting the hidden-bits string to be the hard-core bits of the respective pre-images. By letting the prover have access to the trapdoor, he is able to "see" the hidden-bits and also to reveal bits in positions of his choosing.

As before, assume that $(\mathcal{P}', \mathcal{V}')$ uses a hidden-bits string of length $p(k)$ on security parameter k. Let algorithm Gen be a key-generation algorithm for a trapdoor permutation family which, on input 1^k, outputs permutations over $\{0,1\}^k$. Define $(\mathcal{P}, \mathcal{V})$ as follows:

$\mathcal{P}(r = r_0 | \cdots | r_{p(k)}, x, w)$ // $r_i \in \{0,1\}^k$
  $(f, f^{-1}) \leftarrow \mathsf{Gen}(1^k)$;
  For $i = 1$ to $p(k)$ do
    $b_i = r_0 \cdot f^{-1}(r_i)$; // "$\cdot$" denotes the dot product.
  $(\Pi, I) \leftarrow \mathcal{P}'(b_1 \ldots b_{p(k)}, x, w)$;
  Output $(\Pi, I, \{f^{-1}(r_i)\}_{i \in I}, f)$.

$\mathcal{V}(r, x, (\Pi, I, \{z_i\}_{i \in I}, f))$
  For all $i \in I$
    If $f(z_i) = r_i$ then
      let $b_i = r_0 \cdot z_i$;
    else stop and output 0;
  Output $\mathcal{V}'(\{b_i\}_{i \in I}, I, x, \Pi)$.

Note that b_i is computed as in the Goldreich-Levin construction [5], and is a hard-core bit for f. This particular hardcore-bit construction is used, as it guarantees that the "simulated" hidden bits are uniform with all but negligible probability (as opposed to just negligibly close to uniform when we use a general hardcore bit construction). This follows from that fact that $r_0 \cdot y = 0$ for precisely half of the strings $y \in \{0,1\}^k$, and from the fact that $f^{-1}(r_i)$ is uniform in that set, as r_i is uniform and f is a permutation. (Of course, this assumes $r_0 \neq \{0,1\}^k$, which occurs with all but negligible probability.)

Claim 2 *$(\mathcal{P}, \mathcal{V})$ is a non-adaptive NIZK proof system for L in the CRS model.*

Sketch of Proof (Informal) A full proof appears in the previous lecture, so we just remind the reader of the highlights here. Completeness of the transformed proof system is easy to see, as the prescribed $\mathcal{P}$ runs $\mathcal{P}'$ as a subroutine. For soundness, consider first a *fixed* trapdoor permutation (f, f^{-1}). As argued above, this (with all but negligible probability) results in a uniformly-random string b as seen by a cheating prover. So, soundness of the original proof system implies that a prover can only cheat, using this (f, f^{-1}), with probability at most 2^{-2k}. But a cheating prover can choose whatever (f, f^{-1}) he likes! However, summing over all 2^k possible choices of (f, f^{-1}) (we assume here **(a)** that legitimate output of Gen are easily decidable and **(b)** that Gen uses at most k random bits on security

parameter k; see last lecture for further discussion) shows that the probability of cheating (e.g., finding a "bad" (f, f^{-1}) that allows cheating) is at most 2^{-k} over the choice of r.

For zero-knowledge, let Sim' be the simulator for $(\mathcal{P}', \mathcal{V}')$. Define Sim as follows:

$\mathsf{Sim}(x)$
  $(\{b_i\}_{i\in I}, I, \Pi) \leftarrow \mathsf{Sim}'(x)$;
  $(f, f^{-1}) \leftarrow \mathsf{Gen}(1^k)$;
  $r_0 \leftarrow \{0,1\}^k$; // assume $r_0 \neq 0$
  For $i \in I$ do
    Pick $z_i \leftarrow \{0,1\}^k$ subject to $r_0 \cdot z_i = b_i$;
    Set $r_i = f(z_i)$;
  For $i \notin I$, $i \leq p(k)$ do
    Pick $r_i \leftarrow \{0,1\}^k$;
  Output $(r = r_0|\cdots|r_{p(k)}, (\Pi, I, \{z_i\}_{i\in I}, f))$.

Intuitively, Sim runs Sim', chooses f, then comes up with a CRS that is consistent with the b_i's that Sim' produced. Note that Sim does not know the actual distribution of values for the "hidden bits" at positions $i \notin I$; yet, informally, the security of the trapdoor permutation (and its hard-core bit) ensure that just choosing random r_i at those positions hides the underlying values at those positions anyway.

A complete proof was given in the previous lecture notes. □

2 NIZK for any $L \in NP$ in the Hidden-Bits Model

We now construct a non-adaptive NIZK proof system for a particular NP-Complete language L_0 in the hidden-bits model. Note that this implies a similar result for *any* $L \in NP$: to obtain a system for any $L \in NP$, simply reduce L to L_0 and proceed with the proof system shown below. Soundness, completeness, and zero-knowledge are all clearly preserved.

Specifically, the language L_0 we consider is Graph Hamiltonicity:

$$L_0 = \{G \mid G \text{ is a directed graph with a Hamiltonian cycle}\}$$

(recall that a Hamiltonian cycle in a graph is a sequence of edges that forms a cycle and passes through every vertex exactly once). In our construction, a graph with n vertices will be represented an an n by n boolean matrix, such that entry (i, j) in the matrix is 1 iff there is an edge from vertex i to vertext j (this is the standard *adjacency matrix* representation). In such representation, an n-vertex graph can be identified with a string of length n^2.

For now, we will make the assumption that the hidden-bits string is drawn from a *non-uniform* distribution: instead of being drawn uniformly over strings of length n^2, we assume it is drawn uniformly from strings of length n^2 representing "cycle graphs" (i.e., directed graphs consisting only of a single Hamiltonian cycle). We will show later how to remove this assumption. Given this assumption, define $(\mathcal{P}, \mathcal{V})$ as follows:

$\mathcal{P}(b, G, w)$ // b represents a (random) cycle graph; w is a Hamiltonian cycle in G
  Choose a permutation π on the vertices of G at random from those π that
    map w onto the directed edges of b;

(Imagine "overlaying" G onto b such that the cycle w in G lies on top of the cycle in b)
Let I be the set of positions in b corresponding (under π) to *non-edges* in G
Output π and I.

$\mathcal{V}(\{b_i\}_{i\in I}, I, G, \pi)$
Verify that π is a permutation, and that I contains all positions in b corresponding (under π) to non-edges in G
If all the revealed bits at those positions are 0, accept; otherwise, reject.

Claim 3 *$(\mathcal{P}, \mathcal{V})$ is a non-adaptive NIZK proof system for L_0 in the "hidden-bits" model.*

Sketch of Proof (Informal) Completeness clearly holds. We show that soundness holds with probability 1 (i.e., it is impossible for the prover to cheat). Let G be a graph and assume the verifier accepts. We know that the hidden-bits string b is guaranteed to be a cycle graph, by assumption on the distribution of b. If the verifier accepts, there must be a permutation π under which every non-edge of G corresponds to a non-edge (i.e., "0") in b. But this means, by contrapositive, that every edge ("1") in b corresponds to an edge in G. But since the edges in b form a cycle, this means there must be a cycle in G as well, and hence $G \in L_0$.

To prove zero-knowledge, define Sim as follows:

$\mathsf{Sim}(G)$
Pick a random permutation π on the vertices of G;
Let I be the set of positions corresponding (under π) to *non-edges* in G
Set the values of all "revealed bits" b_I to 0
Output π, b_I, and I

In fact, this gives a *perfect* simulation of $\mathcal{P}$ (although seeing this takes some thought). To see why, let $G \in L_0$ (recall that simulation only needs to work for statements in the language) and consider the distribution over (π, I, b_I) in the real-world. Since b is a random cycle graph, and π is a random permutation mapping the cycle in G to the cycle in b, this means that π is in fact a random permutation. I is a set of positions to which the non-edges of G are mapped under π. Finally, the b_I are all 0. But this is exactly the distribution produced by the simulator. □

References

[1] U. Feige. *Alternative Models for Zero-Knowledge Interactive Proofs*. PhD Thesis, Dept. of Computer Science and Applied Mathematics, Weizmann Institute of Science, 1990. Available from http://www.wisdom.weizmann.ac.il/~feige.

[2] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs Based on a Single Random String. In *FOCS*, pp. 308–317, 1990.

[3] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs Under General Assumptions. *SIAM Journal on Computing* 29(1): 1–28, 1999.

[4] O. Goldreich. *Foundations of Cryptography, vol. 1: Basic Tools*, Cambridge University Press, 2001.

[5] O. Goldreich and L. Levin. A hard-Core Predicate for all One-Way Functions. In *Symposium on the Theory of Computation*, 1989.

[6] D. Lapidot and A. Shamir. Publicly Verifiable Non-Interactive Zero-Knowledge Proofs. In *Advances in Cryptology - CRYPTO '90*, pp. 353-365, 1990.

CMSC 858K — Advanced Topics in Cryptography March 9, 2004

Lecture 13

Lecturer: Jonathan Katz *Scribe(s):* *Nagaraj Anthapadmanabhan*, *Minkyoung Cho*, *Ji Sun Shin*

1 Introduction

In the last few lectures, we introduced the hidden-bits model for non-interactive zero-knowledge (NIZK) and showed a conversion from any NIZK proof system in the hidden bits model to one in the real model, using trapdoor permutations. In this lecture, we complete the construction (which we had begin last lecture) of an NIZK proof system in the hidden-bits model. Putting these results together, we obtain our main goal: a construction of an NIZK proof system for any language in NP in the common random string model. As in the previous lecture, our presentation is based on [1, 2, 3, 4, 5] (As noted in the previous lecture, we focus on the case of non-adaptive NIZK but in fact the constructions given here achieve adaptively-secure NIZK as well.)

2 An NIZK Proof System in the Hidden-Bits Model

We begin by noting that in order to construct a proof system for an arbitrary language L in NP, it suffices to construct a proof system for a single NP-complete language.

Claim 1 *Given an NIZK proof system for an NP-Complete language L^*, we can construct an NIZK proof system for any language $L \in NP$.*

Proof Let $L \in NP$. Then there exists a polynomial-time function f such that

$$x \in L \Leftrightarrow f(x) \in L^*$$

(since L^* is NP-complete). So, given common input x, the prover and verifier can run the NIZK proof system for L^* on common input $f(x)$. Completeness and soundness clearly hold, and it is not too hard to see that zero-knowledge continues to hold as well. ■

For this reason, we now focus our attention on constructing an NIZK proof system for the NP-complete language of graphs containing Hamiltonian cycles (denoted HC).

2.1 Basic Construction

We review a basic construction of an NIZK proof system given in the last lecture. This proof system will be in a "modified" version of the hidden-bits model, where the hidden bits are chosen from a particular distribution which is not the uniform one. We then show how this "non-standard" distribution can be generated from a (long enough) sequence of uniformly-distributed bits — i.e., in the "actual" hidden-bits model.

Input: A directed graph $G = (V, E)$ with n vertices and containing a Hamiltonian cycle w which is known to the prover.

Hidden-bits string: Chosen uniformly from the set of strings representing n-vertex directed cycle graphs (using adjacency matrix representation). Call the given cycle graph C.

Prover: The prover chooses at random a permutation π on the vertices of G that lines up the Hamiltonian cycle w of G with the cycle in C. (This means that for every edge (i, j) in the cycle w, there is a corresponding edge $(\pi(i), \pi(j))$ in C.) The prover outputs π and also, for every non-edge in G, reveals a non-edge in the corresponding position (with respect to π) in C. Specifically, if (i, j) is a non-edge in G, then the prover reveals that entry $(\pi(i), \pi(j))$ in the adjacency matrix of C contains a "0" (i.e., is a non-edge).

Verifier: Verify that π is a permutation and also that for every non-edge in G, the prover has revealed the corresponding position in C (with respect to π), and this position contains a "0" (i.e., is a non-edge).

We now argue that this is indeed an NIZK proof system for graph Hamiltonicity:

Completeness: This follows by inspection. In particular, since an honest prover chooses a π mapping w to the cycle in C, and since C contains *only* this cycle (and no other edges), it will indeed be the case that all non-edges in G will map to non-edges in C.

Soundness: We show that the proof system as stated has *perfect* soundness (and so a prover has probability 0 of successfully proving a false statement). If the verifier accepts, this implies there is a permutation π and *some* cycle graph C (not necessarily known to the verifier) such that every non-edge in G corresponds to a non-edge in C. But then every edge in C corresponds to an edge in G. Since C is a cycle graph, this means that G contains a cycle. (Of course, this relies strongly on the fact that the hidden-bits string defines a cycle graph, but this is by assumption in the above proof system.)

Zero-knowledge: Let Sim be defined as follows:

$\underline{\mathsf{Sim}(G)}$
Pick a random permutation π on the vertices of G
Output π
For every non-edge in G, reveal a "0"

The output generated by Sim is *perfectly* indistinguishable from that generated by a real prover, assuming G is Hamiltonian. (Recall that indistinguishability is required to hold *only* in this case.) In particular: for a real prover, since C is a random cycle graph and π is randomly chosen from those mapping w onto the cycle in C, the permutation π is a random permutation (recall that the verifier never sees C). And both the real proof and the simulated proof reveal a "0" for all positions corresponding to non-edges in G.

2.2 Modified Construction

A problem with the previous construction is that we had assumed that the hidden-bits string was drawn uniformly from the set of (strings representing) cycle graphs. But in the actual hidden-bits model, the string is chosen uniformly at random (indeed, this property was used in the conversion from the hidden-bits model to the model in which a common

random string is available). So, we need to modify the previous construction; we do so by showing how to generate a random directed cycle graph from a uniformly-random string, with noticeable probability (here, we define "noticeable" as "inverse polynomial").

Toward this goal, we will work with *biased bits* which take on the value "1" with probability n^{-5} and "0" otherwise. (Here and in what follows, we let n denote the number of vertices in the graph, as well as the security parameter.) It is easy to obtain such biased bits from a uniform random string by simply parsing the original string in "chunks" of size $5 \log_2 n$, and calling a "chunk" a 1 only if all bits are 1, and a 0 otherwise.

We now define some terminology that will be useful in what follows.

Definition 1 A *permutation matrix* is a binary matrix in which each row and each column contain only a single "1". A *Hamiltonian matrix* is a permutation matrix that corresponds to a cycle; i.e., viewed as an adjacency matrix, it corresponds to a directed cycle graph. $\diamondsuit$

Consider the following procedure for generating a random Hamiltonian matrix from a string of biased bits of length n^6: View the string as an $n^3 \times n^3$ matrix M. We say that M is *useful* if it contains an $n \times n$ Hamiltonian sub-matrix and all other entries in M are 0.[1] We now show that this indeed generates a Hamiltonian matrix with noticeable probability.

Claim 2 $\Pr[M \textit{ is useful}] = \Theta(\frac{1}{n^2})$.

Proof We first claim that

$$\Pr[M \text{ contains exactly } n \text{ 1's}] = \Omega(\frac{1}{n}). \tag{1}$$

To see this, let X be the random variable denoting the number of 1's in M, and let $p = n^{-5}$. Then X follows a binomial distribution with expectation $pn^6 = n$ and variance $p(1-p)n^6 < n$. Applying Chebyshev's inequality (see the Appendix), we see that

$$\Pr\left[|X - n| > n\right] \leq \frac{n}{n^2} = \frac{1}{n}.$$

This implies:

$$\begin{aligned} \sum_{i=1}^{2n} \Pr\left[X = i\right] &= 1 - \Pr\left[|X - n| > n\right] \\ &> 1 - \frac{1}{n}. \end{aligned}$$

Since $\Pr[X = i]$ is maximal at $i = n$, we have:

$$\begin{aligned} \Pr[M \text{ contains exactly } n \text{ 1's}] = \Pr[X = n] &> \frac{\sum_{i=1}^{2n} \Pr[X = i]}{2n} \\ &> \frac{(1 - \frac{1}{n})}{2n} > \frac{1}{3n}, \end{aligned}$$

[1] For the construction, it suffices to find an $n \times n$ Hamiltonian sub-matrix regardless of the values of the other entries. However, it is not clear that one can find a Hamiltonian sub-matrix in polynomial time in the general case. Restricting consideration to matrices M having exactly n 1-entries makes the problem of finding a Hamiltonian sub-matrix easy!

proving Equation (1).

Given that M contains exactly n 1's, a "birthday paradox" argument shows that, with probability $\Omega(1)$, no row or column of M contains more than a single 1. This means that with probability $\Omega(\frac{1}{n})$, the matrix M contains a permutation sub-matrix. Now, there are $n!$ permutation matrices, and $(n-1)!$ Hamiltonian matrices (do you see why?). Thus, the probability that a random permutation matrix is a Hamiltonian matrix is $1/n$.

Let $\mathsf{n\text{-}ones}$ denote the event that M has exactly n 1's, let perm denote the event that M contains a permutation sub-matrix, and let Ham denote the event that M has a Hamiltonian sub-matrix. Putting everything together, the probability that M is useful is at least

$$\Pr[\mathsf{n\text{-}ones}] \cdot \Pr[\mathsf{perm} \mid \mathsf{n\text{-}ones}] \cdot \Pr[\mathsf{Ham} \mid \mathsf{perm}],$$

and we have shown that the above is $\Omega(\frac{1}{n}) \cdot \Omega(1) \cdot \Omega(\frac{1}{n}) = \Omega(\frac{1}{n^2})$. ■

Given the preceding claim, we now show the full construction of a proof system in the hidden-bits model.

Construction (A Proof System in the Hidden Bits Model for HC):

- Common input: A directed graph $G = (V, E)$ with $|V| = n$, where n is also the security parameter.
- Hidden-bits string : A uniformly-distributed string of length $n^3 \cdot (n^6 \cdot 5 \log_2 n)$. This is parsed as n^3 matrices, each containing $n^3 \times n^3$ biased bits as stated above. Denote these matrices by $M_1, \ldots, M_{n^3}$.
- For each M_i, check if M_i is useful.
  - If not, reveal all the entries of M_i.
  - Otherwise (M_i is useful), let C_i denote the $n \times n$ Hamiltonian sub-matrix of M_i. Reveal all $n^6 - n^2$ entries of M_i that are *not* in C_i. Also, use C_i to give a proof as described in Section 2.1.
- Verifier: (The verifier does not see the hidden string, but recall that it is given the positions of the bits revealed by the prover. So it makes sense to talk about the i^{th} matrix M_i even though the verifier does not necessarily see the entire matrix (i.e., besides what is revealed to it by the prover).) The verifier accepts only if the following are true for all n^3 matrices:
  - If the prover has revealed all of M_i, the verifier checks that M_i is not useful.
  - Otherwise, the prover checks that (i) the prover has revealed $n^6 - n^2$ entries in M_i that are 0, while the remaining n^2 entries of M_i form an $n \times n$ sub-matrix; and (ii) call the remaining $n \times n$ sub-matrix C_i. The verifier verifies the prover's proof with respect to C_i exactly as in Section 2.1.

We now argue that the above is an NIZK proof system for HC in the hidden-bits model:

Completeness is immediate. For any M_i that is not useful, the prover can easily convince the verifier by simply revealing all entries. When M_i is useful, the argument in Section 2.1 holds.

Soundness is no longer perfect, but instead holds with all but negligible probability. Following the argument given in Section 2.1, soundness holds with probability 1 whenever at least one of the M_i are useful. The probability that none of the M_i are useful is at most

$$(1 - \Omega(\frac{1}{n^2}))^{n^3} \leq e^{-\Omega(n)},$$

which is negligible.

To show *zero-knowledge* we simply need to modify the simulator given in Section 2.1. The simulator now proceeds in n^3 sequential iterations as follows: in the i^{th} iteration, it generates $n^6 \cdot 5 \log_2 n$ uniformly-random bits. If this defines a matrix M_i which is *not* useful, the simulator simply outputs these bits and moves to the next iteration. If this defines a *useful* matrix M_i with Hamiltonian sub-matrix C_i, the simulator outputs all $n^6 - n^2$ entries of M_i that are not in C_i (these entries are all 0), and then runs the simulator of Section 2.1. Note that this simulator will ignore C_i, and will instead just output a permutation π and "reveal" a 0 for every non-edge in G (as in Section 2.1).

References

[1] U. Feige. *Alternative Models for Zero-Knowledge Interactive Proofs*. PhD Thesis, Dept. of Computer Science and Applied Mathematics, Weizmann Institute of Science, 1990. Available from http://www.wisdom.weizmann.ac.il/~feige.

[2] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs Based on a Single Random String. In *FOCS*, pp. 308–317, 1990.

[3] U. Feige, D. Lapidot, and A. Shamir. Multiple Non-Interactive Zero-Knowledge Proofs Under General Assumptions. *SIAM Journal on Computing* 29(1): 1–28, 1999.

[4] O. Goldreich. *Foundations of Cryptography, vol. 1: Basic Tools*, Cambridge University Press, 2001.

[5] D. Lapidot and A. Shamir. Publicly Verifiable Non-Interactive Zero-Knowledge Proofs. In *Advances in Cryptology - CRYPTO '90*, pp. 353-365, 1990.

A Chebyshev's Inequality

Let X be a random variable with mean μ and variance σ^2. Chebyshev's inequality says that for any $k > 0$ we have

$$\Pr[|X - \mu| \geq k] \leq \frac{\sigma^2}{k^2}.$$

CMSC 858K — Advanced Topics in Cryptography March 11, 2004

Lecture 14

Lecturer: Jonathan Katz

Scribe(s): Alvaro A. Cardenas, Kavitha Swaminatha, Nicholas Sze

1 A Note on Adaptively-Secure NIZK

A close look at the constructions we have shown in class for NIZK shows that the given constructions *already* satisfy the adaptive zero-knowledge property. Interestingly, to the best of Prof. Katz's knowledge there is no known technique for converting an arbitrary *non-adaptive* NIZK proof system into an adaptive one. On the other hand, all the examples of NIZK proof systems of which he is aware happen to be adaptively-secure anyway.

2 The Random Oracle Model

In some sense, we have seen in class essentially all that is known about constructing CCA2-secure encryption schemes. There are two[1] high-level approaches to constructing such schemes: the first uses generic NIZK (an example of which is the scheme of Dolev-Dwork-Naor, although there are other examples as well) and the second relies on the techniques first introduced by Cramer and Shoup (the scheme we showed in class was based on the decisional Diffie-Hellman assumption, but constructions based on other number-theoretic assumptions are possible). The first approach currently does not give any practical schemes, since we do not currently have any examples of practical NIZK proofs (even for specific problems of interest). The second approach yields very efficient schemes, but is based on specific cryptographic assumptions. We remark further that even the Cramer-Shoup scheme (and its variants) is not as efficient as various CPA-secure schemes which are sometimes used in practice (e.g., El Gamal or (unproven) RSA-based schemes).

Assuming for a moment that only very efficient schemes will actually be used, what options do we have? One option is to simply resign ourselves to using encryption schemes satisfying weaker definitions of security (e.g., CPA-secure schemes). Another option is to use schemes which heuristically seem to protect against chosen-ciphertext attacks (but which are not provably secure). However, these options are unsatisfying: we have seen already that chosen-ciphertext attacks represent a real concern in many scenarios, and we also know that if we do not protect against these concerns in a provably-secure way then we leave ourselves open to the possibility of an attack.

A third approach is to introduce a new cryptographic model in which to prove schemes secure. (We will see below that this is *not* the same as introducing new cryptographic assumptions with which to build new, provably-secure schemes.) One very successful and widely-popular model is the *random oracle model*, first formalized by [1] (although it has

[1] Recently, a third approach was suggested [3].

been used previously; see, e.g., [4]). The random oracle model assumes the existence of a public oracle denoted H which implements a (truly) random function. That is:

1. The oracle is *public*: all parties, including the adversary, can submit queries x to the oracle and receive in return $H(x)$. However, queries to the oracle are private so that if an honest party queries $H(x)$, an external adversary does not see x.

2. The oracle implements a *truly random function* in the following sense. Say H maps ℓ-bit strings to n-bit strings. H will maintain a list of pairs $L = \{(x_i, y_i)\}$ such that $H(x_i) = y_i$; the list is initially empty. When H receives a query x, it searches through L for a tuple of the form (x, y): if it finds such a tuple, it returns y. Otherwise, H chooses a random string $y \in \{0,1\}^n$, returns the value y, and stores (x, y) in L. In this sense, H evaluates a function which is truly random (i.e., the value of $H(x)$ at a point x that has not yet been queried is truly random).

While the above is a fine theoretical model, it does not tell us what to do *in practice* with a scheme designed in the random oracle model (random oracles certainly do not exist, and even if we wanted to implement a [private] random function the space required would be prohibitive[2]). In practice, the random oracle will be instantiated with a particular cryptographic hash function based on, say, SHA-1 or MD5. Overall, then, we will design cryptographic schemes via the following, two-step process: First, design and prove the scheme secure in the random oracle model; then, instantiate the random oracle with a particular hash function H. The intuition is that if H is a "good" hash function, then it "acts" like a random oracle and thus the scheme should remain secure in the real world.

Is the above claim correct? Can we formally define what it means for a hash function to be "good" or to "act like a random oracle"? Unfortunately, these questions are still unresolved. On the one hand, there is no known way to instantiate the random oracle (in the theoretical model) by a cryptographic hash function (in the real world) in such a way that the resulting real-world scheme is guaranteed to be secure whenever the scheme is proven secure in theory. In fact, some negative results are known: for example, there are signature/encryption schemes which are secure in the random oracle model but which are insecure in the real world *regardless of how the random oracle is instantiated* [2]. Even worse, there are cryptographic tasks which can be achieved in the random oracle model but *cannot be achieved — by any scheme — in the real world*.

These negative results make the random oracle model somewhat controversial (in fact, proofs of security done without using the random oracle are said to be "in the standard model"), but it is worth considering the arguments in its favor: First, a scheme proven secure in the random oracle model can be said (very informally) to lack any "structural" flaws; thus, any attack on the scheme in the real world "must" arise due to some weakness in the hash function used to instantiate the random oracle, but does not represent a weakness of the scheme itself (and the hash function can then be replaced with a "better" one). Furthermore, it is certainly preferable to use a scheme which is provably-secure in the random oracle model than to use a scheme with no proof of security at all (of course, this assumes that, for reasons of efficiency, these are the only options...). Finally, schemes

[2]The space required to store a random function mapping ℓ-bit strings to m-bit strings is $m \cdot 2^\ell$ bits. Of course, in practice one could build the function dynamically as discussed in the text...

designed in the random oracle model are (currently, at least) much more efficient than schemes proven secure in the standard model: this is the ultimate reason for using the model, after all.

2.1 How the Random Oracle Model is Used

It might initially be surprising that introducing a "random function" can enable proofs of security that do not seem possible in the standard model. But we stress that proofs in the random oracle model rely on much more than the fact that H is "random-looking" or "unpredictable": they rely strongly on the fact that an adversary can only learn about the oracle by making *explicit* queries to an *external* oracle. This gives two advantages when proving security of a scheme: given an adversary A, another algorithm A' running A as a subroutine *can see the queries A makes to its random oracle* (this is because we imagine A as an oracle machine which outputs its queries to a special tape — which can be observed by A' — and expects to get back answers from a random oracle). Second, A' *can answer the random oracle queries of A any way it likes.* (In general, A' will have to answer these queries in a "random-looking" way so that A cannot distinguish whether it is interacting with a random oracle or with A', but this still gives A' an advantage.) We will see examples of both of these strategies in the next few lectures.

To get a feel for the difficulties that arise when translating this to the real world, note that if we instantiate a random oracle by, say, SHA-1, then neither of the above conditions hold (of course!). Furthermore, in the random oracle model a statement like the following makes sense: "no algorithm can distinguish $H(x)$ from an independent random string without explicitly querying the random oracle at point x" but in the real world the statement "no algorithm can distinguish SHA-1(x) from an independent random string without explicitly computing SHA-1(x)" is meaningless. (What does it mean to "compute" SHA-1? How do we tell whether an algorithm is computing SHA-1 or doing something different?)

It is also important to note that the random oracle model is not at all like the security model used for pseudorandom functions (PRFs), which we did not get to cover this semester. In that model, the adversary is provided with oracle access to a function $F_s(\cdot)$ *because the adversary is not supposed to be able to compute $F_s(\cdot)$* (since the seed s is secret). In contrast, in the random oracle model the adversary *is* supposed to be able to compute $H(\cdot)$ but we "force" the adversary (in the theoretical model) to compute it via oracle access only. Note further that $F_s(\cdot)$ is most definitely *not* pseudorandom (whatever that might mean) if s is revealed to the adversary; thus, one cannot simply replace a random oracle with a PRF.

3 Semantically-Secure Encryption in the RO Model

We now give a concrete example of how the random oracle model is used to design schemes, and how proofs of security in the random oracle model proceed. Before doing so, let us recall (from Lecture 2) the construction of semantically-secure encryption from trapdoor permutations in the standard model: Key generation involves choosing f, f^{-1}; the public key is f, while the secret key is (the trapdoor information needed to compute) f^{-1}. To encrypt a single bit b, the sender chooses a random domain element x and a random r and sends ciphertext $\langle f(x), (x \cdot r) \oplus b \rangle$ (i.e., we are using here the Goldreich-Levin hardcore

bit construction). One application of f is needed to encrypt a single bit — this is pretty inefficient, and one is not likely to do this in practice.

It is worth reminding ourselves also why more efficient approaches do not work. For example, why not encrypt a (longer) message m by choosing r of the appropriate length and sending $f(m \mid r)$? We noted that this would not be secure in general because $f(x)$ might leak the first 10 bits, say, of x. In fact, it is known that, in general, a trapdoor permutation is only "guaranteed" to have at least $O(\log k)$ bits (where k is the security parameter. So, the best we can hope to do (in some sense) is to encrypt $\log k$ bits per evaluation of f.

However, the above is all for the *standard model*. In the random oracle model, however, we can get an *unbounded* number of "hardcore" bits from any trapdoor permutation. We use the fact that the value of $H(r)$ is *truly random* if (1) H is a random oracle and (2) an adversary has not explicitly queried $H(r)$. (As noted at the end of the previous section, statement (1) clearly cannot be true for *any* concrete instantiation of H since, from an information-theoretic point of view, $f(r)$ completely determines r and thus $H(r)$, and so the entropy of $H(r)$ given $f(r)$ is 0! Also, $H(r)$ might be longer than r, implying that we are creating randomness out of thin air.) Furthermore, given $f(r)$ the adversary is not likely to query $H(r)$ because that would mean that the adversary had succeeded in inverting f. In short, we can encrypt a long message m by choosing random r and sending $\langle f(r), H(r) \oplus m \rangle$.

In more detail, let H map from the domain of the trapdoor permutation family to strings of length ℓ. Then we can encrypt ℓ-bit messages as follows (in the below, we assume for simplicity that the domain of the trapdoor permutation is $\{0,1\}^k$):

$\underline{\mathsf{Gen}(1^k)}$	$\underline{\mathcal{E}_{pk}(m)}$	$\underline{\mathcal{D}_{sk}(\langle y, c\rangle)}$
Generate f, f^{-1}	$r \leftarrow \{0,1\}^k$	$r = f^{-1}(y)$
$pk = f, sk = f^{-1}$	Output $\langle f(r), H(r) \oplus m\rangle$	Output $H(r) \oplus c$
Output pk, sk		

Claim 1 *The scheme above is semantically secure in the random oracle model if f is chosen from a trapdoor permutation family.*

Proof We need to prove that for every PPT A the following is negligible:

$$\mathsf{Adv}_A(k) \stackrel{\text{def}}{=} \left| \Pr\left[\begin{array}{c} (f, f^{-1}) \leftarrow \mathsf{Gen}(1^k); (m_0, m_1) \leftarrow A^H(f); b \leftarrow \{0,1\}; \\ r \leftarrow \{0,1\}^k; b' \leftarrow A^H(f, \langle f(r), H(r) \oplus m_b\rangle) \end{array} : b = b' \right] - \frac{1}{2} \right|.$$

Note that we give the adversary access to H, as we must in the random oracle model.

We observe the following: If A never queries $H(r)$ (where r is the random element chosen in the above experiment), then the value of $H(r)$ is truly random (at least from the point of view of A) and thus A has no information about the value of b. Let query be the event that A queries $H(r)$ at some point during the above experiment, and let Succ be the event that A correctly outputs $b' = b$. We then have

$$\begin{aligned} \mathsf{Adv}_A(k) &= \left| \Pr[\mathsf{Succ} \mid \mathsf{query}] \Pr[\mathsf{query}] + \Pr[\mathsf{Succ} \mid \overline{\mathsf{query}}] \Pr[\overline{\mathsf{query}}] - \frac{1}{2} \right| \\ &= \left| \Pr[\mathsf{Succ} \mid \mathsf{query}] \Pr[\mathsf{query}] + \frac{1}{2} \cdot \Pr[\overline{\mathsf{query}}] - \frac{1}{2} \right| \end{aligned}$$

$$
\begin{aligned}
&= \left| \Pr[\mathsf{Succ} \mid \mathsf{query}] \Pr[\mathsf{query}] - \frac{1}{2} \cdot \Pr[\mathsf{query}] \right| \\
&\leq \frac{1}{2} \Pr[\mathsf{query}].
\end{aligned}
$$

To complete the proof, we show that $\Pr[\mathsf{query}]$ is negligible.

Given any PPT adversary A as above, we construct an algorithm B that tries to invert f at a random point *in the real world*. Since B will not have access to any random oracle, it will have to *simulate* the random oracle for A. But this is easy to do: for every query x made by A to H, simply return a random answer if x was not queried before, or the same answer given previously if x *was* queried before (in fact, without loss of generality we may simply assume that A does not make the same query to H twice). We construct B as follows:

$\underline{B(f, y)}$
run $A^H(f)$ until it outputs m_0, m_1
  (answering queries to H as discussed below)
$c \leftarrow \{0,1\}^\ell$
run $A^H(f, \langle y, c \rangle)$ until it halts
for each query r_i made by A to H:
  if $f(r_i) = y$ output r_i and halt
  Otherwise, simply return a random ℓ-bit string

Let $r \stackrel{\text{def}}{=} f^{-1}(y)$ (of course, B does not know this value). Note that B provides a *perfect* simulation of the experiment for A up to the point (if any) that A queries $H(r)$. To see this, observe that f is randomly generated, $y = f(r)$ for a randomly-chosen r (recall the definition of inverting a trapdoor permutation from earlier lectures), and B faithfully simulates a random oracle on all points other than r. Now, in the above experiment the value c is a random string whereas in the real experiment we have $c = H(r) \oplus m_b$ where b is chosen at random. However, if A has not yet queried $H(r)$ then the value of $H(r)$ is *truly* random and thus choosing c as a random string results in a perfect simulation. This continues to be the case up to the point, if any, that A actually queries $H(r)$.

Finally, note that B succeeds in inverting f exactly when query occurs. By the above reasoning, query occurs in the above experiment with the exact same probability as in the real experiment. Thus, $\Pr[\mathsf{query}]$ must be negligible. ■

We remark that an extension of the above proof can be used to show that the scheme is secure against non-adaptive chosen-ciphertext attacks (this is left as an exercise). However the scheme is not secure against adaptive chosen-ciphertext attacks. Consider the algorithm which, upon receiving ciphertext $\langle y, c \rangle$ submits ciphertext $\langle y, c \oplus 1^\ell \rangle$ to the decryption oracle. By doing so, the adversary can recover $H(f^{-1}(y))$ (although it does not learn $f^{-1}(y)$ itself), and thereby figure out which message was encrypted.

4 Toward CCA2 Security in the Random Oracle Model

In the next lecture, we will construct a CCA2-secure encryption scheme in the random oracle model. In preparation, we first show how to construct an information-theoretically

secure message authentication code (MAC). Let $\mathbb{F}_q$ denote the field with q elements.

Claim 2 *If a, b are chosen at random from $\mathbb{F}_q$ and an adversary is given $(m, am+b)$ (for $m \in \mathbb{F}_q$ of the adversary's choice) the probability that the adversary can output (m', t') such that $t' = am' + b$ and $m' \neq m$ is at most $\frac{1}{q}$.*

Proof When the adversary is given (m, t), it knows that a, b are chosen uniformly at random subject to $t = am + b$. Note that, from the point of view of the adversary, a is uniformly distributed in $\mathbb{F}_q$ since for every $a \in \mathbb{F}_q$ there is exactly a single value of b (namely, $b = t - am$) such that $t = am + b$. Now, for any (m', t') with $m' \neq m$, we have $t' = am' + b$ iff $t' - t = a(m - m')$. Thus

$$\Pr[t' = am' + b] = \Pr[a = (t' - t)(m - m')^{-1}] = 1/q.$$

This concludes the proof. ■

This leads to a simple way to authenticate a single message: two parties share random (a, b) in advance, and to authenticate a message $m \in \mathbb{F}_q$ the sender computes $t = am+b$. An (all-powerful) adversary can "fool" the receiver into accepting some $m' \neq m$ with probability at most $1/q$. Choosing q large enough we get as much security as we like.

References

[1] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. ACM Conf. on Computer and Communications Security, 1993.

[2] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology Revisited. STOC '98.

[3] R. Canetti, S. Halevi, and J. Katz. Chosen-Ciphertext Security from Identity-Based Encryption. Eurocrypt 2004.

[4] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. Crypto '86.

CMSC 858K — Advanced Topics in Cryptography March 16, 2004

Lecture 15

Lecturer: Jonathan Katz *Scribe(s): Avi Dalal, Abheek Anand, Gelareh Taban*

1 Introduction

In the previous lecture, we introduced the notion of message authentication: Given message $m \in \mathbb{F}_q$, to authenticate it pick two random secrets $a, b \in \mathbb{F}_q$ and output $(m, am + b)$. The possibility of an attacker outputting (m', t') such that $(m' \neq m)$ and $(t = am' + b)$ is at most $1/q$. The security of this message authentication protocol is information-theoretic and does not rely on any computational assumptions (a proof was given last time). For future reference, we let $\mathsf{Mac}_{a,b}(m) \stackrel{\text{def}}{=} am + b$.

We will use this message authentication scheme to modify the encryption scheme given previously and make it secure against adaptive chosen-ciphertext attacks in the random oracle model. We also introduce OAEP$^+$ and prove its security.

2 The Modified Encryption Scheme

For simplicity, we assume that messages to be encrypted lie in some field $\mathbb{F}_q$ with $|q| = k$ (i.e., the security parameter), and also assume that H maps elements in the domain of the trapdoor permutation family to elements in $\mathbb{F}_q^3$. (If you like, you can think of messages as strings of length ℓ and set $q = 2^\ell$.)

$\underline{\mathsf{Gen}(1^k)}$	$\underline{\mathcal{E}_{pk}(m)}$	$\underline{\mathcal{D}_{sk}(\langle y, C, t\rangle)}$
Generate f, f^{-1}	$r \leftarrow \{0,1\}^k$	$r = f^{-1}(y)$
$pk = f, sk = f^{-1}$	let $H(r) = (a, b, c) \in \mathbb{F}_q^3$	$(a, b, c) = H(r)$
output pk, sk	$C = m + c$	if $aC + b \stackrel{?}{=} t$ then output $C + c$
	$t = \mathsf{Mac}_{a,b}(C)$	else output $\perp$
	output $\langle f(r), C, t\rangle$	

It is not hard to verify that the scheme gives correct decryption.

Theorem 1 *If f is chosen from a trapdoor permutation family, the above scheme is CCA2 secure in the random oracle model.*

Proof We assume the reader is familiar with the proof of semantic security for a related scheme that was given in Lecture 14. The proof here will be similar, but more complicated because we will now need to take into account the *decryption oracle* for an adversary attacking the scheme. Let A be an adversary attacking the scheme, and let r denote the random value used by the sender (i.e., encryption oracle) in constructing the challenge ciphertext $\langle y, C, t\rangle$ that is given to A. Let query be the event that A make the query

$H(r)$ at some point during the experiment, and let dec be the event that A submits a ciphertext $\langle y, C', t' \rangle$ with $(C', t') \neq (C, t)$ but where this ciphertext is decrypted properly (i.e., decryption does not result in $\perp$).

Since we are in the random oracle model, A can only gain any information about the encrypted message if either query or dec occur; thus, as in the proof given previously:

$$\mathsf{Adv}_A(k) \leq \frac{1}{2} \Pr[\mathsf{query} \vee \mathsf{dec}].$$

Define $H(r) = (a^*, b^*, c^*)$. Now, if query has not yet occurred then the only information A has about (a^*, b^*) is that $\mathsf{Mac}_{a^*,b^*}(C) = t$. But then the properties of the message authentication code imply that the probability that dec occurs in any particular query to the decryption oracle is at most $1/q$ (note that every "message" has a unique tag, so setting $C' = C$ will not help). Let query^{1st} denote the event that query occurs before dec (including the case when dec does not occur at all) and define dec^{1st} similarly. The above shows that if A makes at most q_d queries to the decryption oracle we have $\Pr[\mathsf{dec}^{1st}] \leq q_d/q$. Putting everything together we see:

$$\begin{aligned} \mathsf{Adv}_A(k) &\leq \frac{1}{2} \Pr[\mathsf{query} \vee \mathsf{dec}] \\ &= \frac{1}{2} \cdot \left(\Pr[\mathsf{query}^{1st}] + \Pr[\mathsf{dec}^{1st}] \right) \\ &\leq \frac{1}{2} \cdot (\Pr[\mathsf{query}^{1st}] + q_d/q). \end{aligned}$$

For A a PPT algorithm, q_d is polynomial and thus q_d/q is negligible (this is why we required $|q| = k$). To complete the proof, we show that $\Pr[\mathsf{query}^{1st}]$ is negligible.

Let A be a PPT adversary attacking the scheme who is given access both to the random oracle $H(\cdot)$ as well as a decryption oracle $D_{sk}(\cdot)$. We construct the following adversary B who will try to invert f on a given point chosen at random from the domain of f. As in the previous proof, B will simulate the experiment for A but this now includes simulating A's access to the decryption oracle (since B does not know $sk = f^{-1}$ we represent the decryption oracle by D). The oracle queries of A are answered in such a way as to ensure consistency between the answers given by the different oracles. This is done by storing two lists: list S_H contains tuples (r, a, b, c) such that $H(r) = (a, b, c)$ (as chosen by B), while list S_y contains tuples (y, a, b, c) such that $H(f^{-1}(y)) = (a, b, c)$ but the important point is that B may not know $f^{-1}(y)$. We now provide a complete description:

$\underline{B(f, y)}$
$S_H = \emptyset; S_y = \emptyset$
run $A^{D(\cdot), H(\cdot)}(f)$ until it outputs m_0, m_1
  (answering queries to D and H as discussed below)
$C, t \leftarrow \mathbb{F}_q$
run $A^{D(\cdot), H(\cdot)}(f, \langle y, C, t \rangle)$ until it halts
  (answering queries to D and H as discussed below)

To answer query $H(r_i)$:

if $f(r_i) = y$ output r_i and halt the experiment
if $r_i = r_j$ for some $(r_j, a_j, b_j, c_j) \in S_H$ then return (a_j, b_j, c_j)
if $f(r_i) = y_j$ for some $(y_j, a_j, b_j, c_j) \in S_y$ then return (a_j, b_j, c_j)
otherwise, choose $(a, b, c) \leftarrow \mathbb{F}_q^3$ and return (a, b, c)
store r_i and the returned values in S_H

To answer query $D(\langle y_i, C_i, t_i \rangle)$:

if $y_i = y$ return $\perp$
if $y_i = y_j$ for some $(y_j, a_j, b_j, c_j) \in S_y$ then decrypt using (a_j, b_j, c_j)
if $f(r_j) = y_i$ for some $(r_j, a_j, b_j, c_j) \in S_H$ then decrypt using (a_j, b_j, c_j)
otherwise, choose $(a, b, c) \leftarrow \mathbb{F}_q^3$ and decrypt using (a, b, c)
store y_i and the (a, b, c) values used in S_y

(Note: "decrypt $\langle y, C, t \rangle$ using (a, b, c)" simply means to return $C + c$ if $aC + b \stackrel{?}{=} t$, and $\perp$ otherwise.) Clearly, B runs in polynomial time when A does; also, it is easy to see that B succeeds in inverting f whenever query occurs and, in particular, if query^{1st} occurs. The above simulation is perfect unless event dec or query occurs. Since we are interested in the event query^{1st} — which occurs immediately if query occurs first and can no longer occur if dec occurs first — the probability of event query^{1st} is the same in the above experiment as in a real execution of A when attacking the encryption scheme. Thus, the security of the trapdoor permutation family implies that $\Pr[\mathsf{query}^{1st}]$ is negligible, as desired. ■

3 Optimal Asymmetric Encryption Padding (OAEP) and OAEP$^+$

A possible drawback of the above scheme is its ciphertext length. Given a trapdoor permutation f acting on k-bit strings, it would be nice to be able to send a ciphertext which is exactly k bits long. OAEP was designed to do this while allowing the message to be as long as possible (and while still being secure against chosen-ciphertext attacks).

OAEP was proposed by Bellare and Rogaway in 1994 [1] and is defined for any trapdoor permutation family. However, the proof was later found to have a subtle error and a number of fixes were proposed (see [4] for a good discussion of the flaw, and a counterexample which illustrates that the flaw is real). Fujisaki, et al. [3] and Shoup [4] show that OAEP is in fact secure when RSA is used as the underlying trapdoor permutation family; the proof of security relies on specific algebraic properties of RSA and does not hold for an arbitrary trapdoor permutation. Boneh [2] gave a simplified version of OAEP which is provably-secure when the RSA or Rabin trapdoor permutation families are used. Shoup [4] showed a way to modify OAEP so as to be secure for an arbitrary trapdoor permutation family. We will present this last scheme (called OAEP$^+$) here both because of its generality and also because it has what is (arguably) the simplest proof.

Let f be a one-way trapdoor permutation, acting on k-bit strings. Also let k_0, k_1 be two parameters such that $k_0 + k_1 < k$ and 2^{-k_0} and 2^{-k_1} are negligible. For example, in an asymptotic setting one could take $k_0 = k_1 = k/3$; more concretely, if RSA is used and

$k = 1024$, then we may set $k_0 = k_1 = 128$. The scheme encrypts messages $m \in \{0,1\}^n$ where $n = k - k_0 - k_1$. The scheme also makes use of three functions:

$$G : \{0,1\}^{k_0} \to \{0,1\}^n$$
$$H' : \{0,1\}^{n+k_0} \to \{0,1\}^{k_1}$$
$$H : \{0,1\}^{n+k_1} \to \{0,1\}^{k_0}.$$

These three functions will be modeled as independent random oracles in the security analysis. The scheme is defined as follows:

$\underline{\mathsf{Gen}(1^k)}$	$\underline{\mathcal{E}_{pk}(m)}$	$\underline{\mathcal{D}_{sk}(y)}$
Generate f, f^{-1}	$r \leftarrow \{0,1\}^{k_0}$	$s\|\|t = f^{-1}(y)$,
$pk = f, sk = f^{-1}$	$s = (G(r) \oplus m) \,\|\|\, H'(r\|\|m)$	(where $\|s\| = n + k_1$)
output pk, sk	$t = H(s) \oplus r$	$r = H(s) \oplus t$
	$y = f(s\|\|t)$	parse s as $s_1\|\|s_2$,
	output y	(where $\|s_1\| = n$; $\|s_2\| = k_1$)
		$m = G(r) \oplus s_1$
		if $(H'(r\|\|m) \stackrel{?}{=} s_2)$ output m
		else output $\perp$

The intuition is that this scheme is constructed such that an eventual *simulator*, who does not know sk, is able to answer the decryption queries of an adversary A based only on the oracle queries made by A.

Theorem 2 *If f is chosen from a trapdoor permutation family, the above scheme is CCA2 secure in the random oracle model.*

Proof The proof given here is organized a little differently from the proof given in [4], and the reader is advised to look there for much more detail. Let A be an adversary attacking the scheme. As usual, A will have access to the random oracles in addition to the decryption oracle (and the encryption oracle as well). We assume without loss of generality that whenever A makes a query $H'(r||m)$ it has previously made the query $G(r)$. Let S_G, S_H, and $S_{H'}$ be the set of points at which A has queried G, H, and H', respectively. (These sets grow dynamically each time A queries one of its oracles.) We begin by proving a claim regarding the decryption queries made by A. If a decryption query made by A results in response $\perp$, we say the query is *invalid*; queries which are not invalid are called *valid*. Note that any decryption query y made by A (implicitly) defines values s, t, r, and m (just by following the decryption process); we say a decryption query y is *likely to be invalid* if, at the time the query was made, either A had not yet queried $H'(r||m)$ or A had not yet queried $H(s)$ (for the r, m, s associated with y). Finally, we say a query is *exceptional* if it is likely to be invalid but is, in fact, valid. Then:

Claim 3 *Even if A is all-powerful (but can only make polynomially-many queries to its oracles), the probability that A makes an exceptional query is negligible.*

Proof (of Claim 3): Note that this is an information-theoretic argument based on A's lack of knowledge about the values of the random oracle on points it has not (yet) queried. Since A is all-powerful, we may as well dispense with f, f^{-1} and simply assume that when

A gets the challenge ciphertext y^* it immediately recovers $s^*||t^* = f^{-1}(y^*)$ and that when A submits a decryption query y it already knows $s||t = f^{-1}(y)$.[1] Let m^* be the message encrypted to give the challenge ciphertext (note that even an all-powerful A does not know m^* unless it queries $G(r^*)$), and let s_1^*, s_2^*, r^*, t^* be defined in the natural way based on y^*. We focus on a particular decryption query y that A makes *after* getting the challenge ciphertext (with s_1, s_2, r, t defined in the natural way), and show that the probability that y is exceptional is negligible. Since A makes at most polynomially-many decryption queries, this suffices to prove the claim.

Consider the query y where $s||t = f^{-1}(y)$, and assume that y is likely to be invalid (recall, this is either because A has not queried $H'(r||m)$ or because A has not queried $H(s)$). We show that y is invalid with all but negligible probability by considering the possible cases:

Case 1: A has not queried $H'(r||m)$ and $r = r^*$ and $m = m^*$. Since $(r, m) = (r^*, m^*)$, we also have $s_1 = s_1^*$. If the ciphertext is not invalid, then we must have $s_2 = s_2^*$ and hence $t = t^*$ as well. But this would imply that $y = y^*$, and A is prohibited from querying the decryption oracle with the challenge ciphertext.

Case 2: A has not queried $H'(r||m)$ and $r \neq r^*$. In this case, the value of $H'(r||m)$ is completely random given A's view of the experiment (note that $H'(r||m)$ was not queried during the course of constructing the challenge ciphertext, either). Thus, the probability that y is valid is the probability that $H'(r||m)$ is equal to s_2, which is $2^{-|s_2|} = 2^{-k_1}$ and hence negligible.

Case 3: A has not queried $H'(r||m)$ and $m \neq m^*$. The argument in this case is exactly as in the previous case, so we omit it.

Case 4: A has not queried $H(s)$ and $s = s^*$. Since we must have $y \neq y^*$, this implies that $t \neq t^*$ and hence $r \neq r^*$. The only way y can be valid is if $H'(r||m) = s_2 = s_2^*$, where $s_2^* = H'(r^*||m^*)$. Thus, y is valid only if A has managed to find a *different* input hashing to the same k_1-bit value s_2^*. Since A makes only polynomially-many queries to H', this occurs with only negligible probability.

Case 5: A has not queried $H(s)$ and $s \neq s^*$. In this case, the value of $H(s)$ is completely random from the point of view of A (note that $H(s)$ was not queried when the challenge ciphertext was constructed, either). Thus, the value of r is completely random from the point of view of A, and so the probability that A has queried $H'(r||m)$ is negligible. Assuming A has not queried $H'(r||m)$, we reduce to one of the cases considered previously. □

Given the above claim, we now prove the theorem in a manner similar to the proof of Theorem 1 (as well as the proof given in the previous lecture). We will be a little informal from now on, but the reader should be able to fill in the missing details (indeed, the difficult part of the proof is the above claim). Note that A has no information about the message that was encrypted to give the challenge ciphertext unless it queries $G(r^*)$. Also, the probability

[1] One may wonder why A needs to submit a decryption query if it is all powerful. The point is that in this claim we are interested in the probability a particular event which is independent of the security of the encryption scheme (indeed, if A is all-powerful than it can "break" the encryption scheme anyway). This claim will be used below to prove the actual security of the scheme for a PPT A.

that A queries $G(r^*)$ without first querying $H(s^*)$ is negligible (since A has no information about r^* until it queries $H(s^*)$, and A makes only polynomially-many queries to G). The preceding two statements are true even if A is all-powerful. So, letting query be the event that A queries both $H(s^*)$ and $G(r^*)$ we have:

$$\mathsf{Adv}_A(k) \leq \Pr[\mathsf{query}] + negl(k).$$

We show that $\Pr[\mathsf{query}]$ is negligible by giving an informal description of a PPT algorithm B which uses A as a subroutine and tries to invert f on a given point y^* chosen at random from the domain of f. B will simulate the random oracle queries of A in the natural way, and when A submits messages (m_0, m_1) to its encryption oracle, B returns the challenge ciphertext y^* to A. More interesting is B's simulation of the decryption oracle for A (recall that B does not know how to compute f^{-1}): upon receiving decryption query y, B searches through the list $S_{H'}$ of queries that A has made thus far to H'. For each $(r_i, m_i) \in S_{H'}$, B first computes

$$s_i = (G(r_i) \oplus m_i)||H'(r_i||m_i).$$

Next, if $s_i \notin S_H$ (i.e., A has not queried $H(s_i)$), B returns $\perp$. Otherwise, B computes $t_i = H(s_i) \oplus r_i$ and then checks whether $y \stackrel{?}{=} f(s_i||t_i)$ (note that B *can* evaluate f in the forward direction). If this test succeeds for a particular pair (r_i, m_i), then B returns m_i to A as the (correct) decryption of y. If the test fails for every i, B returns $\perp$.

At the end of the experiment, B looks through the lists S_H and S_G. For each $s_i \in S_H$ and $r_j \in S_G$, B computes $t_{i,j} = r_j \oplus H(s_i)$ and checks whether $f(s_i||t_{i,j}) \stackrel{?}{=} y^*$. If this is true for any pair, then B outputs $s_i||t_{i,j}$ as the (correct) answer.

The proof concludes using the following observations: (1) until query occurs, the only difference between the view of A in a real experiment and the view of A as simulated by B occurs when A makes an exceptional query (since, in this case, B returns $\perp$ but the decryption query was valid). However, by the claim proven earlier, this occurs with only negligible probability. Thus, (2) the probability of query in the experiment as simulated by B is negligibly close to $\Pr[\mathsf{query}]$ (i.e., the probability of query in the real experiment). Finally, (3) B succeeds in inverting y^* whenever query occurs. Since f is assumed to be a trapdoor permutation family, putting the above observations together shows that $\Pr[\mathsf{query}]$ is negligible. ■

References

[1] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption — How to Encrypt with RSA. Eurocrypt '94.

[2] D. Boneh. Simplified OAEP for the RSA and Rabin Functions. Crypto 2001.

[3] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is Secure Under the RSA Assumption. Crypto 2001.

[4] V. Shoup. OAEP Reconsidered. Crypto 2001.

CMSC 858K — Advanced Topics in Cryptography March 18, 2004

Lecture 16

Lecturer: Jonathan Katz

Scribe(s): Chiu Yuen Koo, Nikolai Yakovenko, Jeffrey Blank

1 Digital Signature Schemes

In this lecture, we introduce the notion of *digital signature schemes*, show a construction of a *one-time* signature scheme based on one-way functions in the standard model [4], and then cover the full-domain-hash (FDH) signature scheme based on trapdoor permutations in the random oracle model [1, 2]. We first define the semantics of a digital signature scheme.

Definition 1 A digital signature scheme consists of a triple of PPT algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ such that:

- Gen is a randomized algorithm which, on input security parameter 1^k, generates a pair of keys: a public (verification) key pk, and a secret (signing) key sk.
- Sign, which may be randomized, takes as input a secret key sk and a message m, and generates a signature σ. We write this as $\sigma \leftarrow \mathsf{Sign}_{sk}(m)$.
- Vrfy takes as input a public key, a message, and a (purported) signature; it outputs a single bit b with $b = 1$ indicating acceptance and $b = 0$ indicating rejection. (We assume for simplicity that Vrfy is deterministic.) We write this as $b = \mathsf{Vrfy}_{pk}(m, \sigma)$.

For correctness, we require that for all (pk, sk) output by $\mathsf{Gen}(1^k)$, for all messages m, and for all σ output by $\mathsf{Sign}_{sk}(m)$ we have $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$. ♢

Technically, one also has to specify a message space but this will be implicit in all the schemes we discuss. We sometimes say a signature σ is *valid* (for a particular message m and with respect to a particular public key pk) if $\mathsf{Vrfy}_{pk}(m, \sigma) = 1$.

We now give a notion of security for digital signatures, following the definition first given by [3]. The definition is a rather strong one: we allow an adversary (who is given the public key) to repeatedly ask for signatures on multiple messages of his choice (this is referred to as an "adaptive chosen-message attack"); the adversary succeeds if it can output a valid signature on any message of its choice which was not signed previously (this is called "existential forgery"). Security requires that the success probability of any polynomial-time adversary is negligible. This notion corresponds to *security against existential forgery under adaptive chosen-message attacks*, and is essentially the strongest considered in the literature.[1] One can imagine weakening this definition in several ways; except for introducing the notion of one-time signatures (below) we will not pursue this further here.

[1] Actually, a slightly stronger definition has recently been considered whereby the adversary succeeds even if it outputs a signature σ on a *previously-signed* message, such that σ is valid but not identical to one produced previously by the signer. Many schemes achieve this definition without further modification.

Definition 2 A signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ is *existentially unforgeable under an adaptive chosen-message attack* if for all PPT adversaries A, the following is negligible:

$$\Pr[(pk, sk) \leftarrow \mathsf{Gen}(1^k); (m, \sigma) \leftarrow A^{\mathsf{Sign}_{sk}(\cdot)}(pk) : \mathsf{Vrfy}(m, \sigma) = 1 \wedge m \notin M],$$

where M is the set of messages submitted by A to the Sign oracle. $\diamondsuit$

In other words, A is allowed to submit a polynomial number of message for signing. Even based on these signatures, A should not be able to generate a signature on any message not submitted to the oracle. We will also consider the following weaker definition of security whereby A is only allowed to submit a *single* message to its signing oracle. Formally:

Definition 3 A signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ is a secure *one-time signature scheme* if for all PPT adversaries A, the following is negligible:

$$\Pr[(pk, sk) \leftarrow \mathsf{Gen}(1^k); (m, \sigma) \leftarrow A^{\mathsf{Sign}_{sk}(\cdot)}(pk) : \mathsf{Vrfy}(m, \sigma) = 1 \wedge m \neq m'],$$

where m' is the single message that A submitted to its signing oracle. $\diamondsuit$

2 One-Way Functions

One-way functions are simply functions that are efficient to compute but hard to invert. They form the minimal "hardness" assumption necessary for most of cryptography, including symmetric-key encryption and digital signatures. We give a definition here tailored to a "concrete" security analysis rather than an asymptotic one.

Definition 4 A polynomial-time-computable function f over domain D_f is said to be (t, ε)-one-way if for all adversaries A running it time t we have:

$$\Pr[x \leftarrow D_f; y = f(x); x' \leftarrow A(y) : f(x') = y] \leq \varepsilon.$$

$\diamondsuit$

As an informal example, one may conjecture that 2048-bit RSA is currently (5 years, 2^{-60})-one-way (note that "RSA" does not quite fit into the above framework, but the definition can be easily modified to accommodate it). This means that no adversary running in five years can invert 2048-bit RSA (on a randomly-chosen challenge point, and for randomly generated modulus) with probability greater than 2^{-60}.

3 The Lamport One-Time Signature Scheme

We now show a one-time signature scheme based on one-way functions. Although not very practical, this scheme is important for several reasons: (1) it illustrates that signature schemes (at least weak ones) can be constructed from *one-way functions* in the *standard model* and do not require any sort of "trapdoor" as was initially believed; (2) the scheme is used as a building block in the construction of many other schemes, including "full-fledged" signature schemes secure against existential forgery against adaptive chosen-message attacks. The scheme shown here was suggested by Lamport [4], and we describe it for messages of length ℓ and using a one-way function f defined over domain $\{0,1\}^k$:

$\underline{\mathsf{Gen}(1^k)}$
for $i = 1$ to ℓ and $b \in \{0,1\}$:
  $x_{i,b} \leftarrow \{0,1\}^k$
  $y_{i,b} = f(x_{i,b})$
$pk \stackrel{\text{def}}{=} \begin{pmatrix} y_{1,0} \cdots y_{\ell,0} \\ y_{1,1} \cdots y_{\ell,1} \end{pmatrix}$
$sk \stackrel{\text{def}}{=} \begin{pmatrix} x_{1,0} \cdots x_{\ell,0} \\ x_{1,1} \cdots x_{\ell,1} \end{pmatrix}$
output (pk, sk)

$\underline{\mathsf{Sign}_{sk}(m)}$
let $m = m_1 \cdots m_\ell$
 with $m_i \in \{0,1\}$
let sk be as before
output $(x_{1,m_1}, \ldots, x_{\ell,m_\ell})$

$\underline{\mathsf{Vrfy}_{pk}(m, \sigma)}$
parse σ as $(x_1, \ldots, x_\ell)$
if $f(x_i) \stackrel{?}{=} y_{i,m_i}$ for $1 \le i \le \ell$
 output 1
else output 0

Theorem 1 *If f is (t, ε)-one-way (for some particular value of k) and requires time t_f to evaluate (in the forward direction), then no adversary running in time $O(t - 2\ell t_f)$ can "break" the one-time security of the scheme with probability better than $2\ell\varepsilon$.*

In particular, since ℓ and t_f are polynomial in k, this means that if f is (asymptotically) one-way then the Lamport scheme is (asymptotically) secure.

Proof Assume to the contrary that there exists an adversary A' running in time $t' = O(t - 2\ell t_f)$ and forging a signature with probability $\varepsilon' > 2\ell\varepsilon$. We construct an adversary A running in time t and inverting f with probability better than ε, a contradiction.

Define algorithm A (which gets a value y and tries to find an $x \in \{0,1\}^k$ such that $f(x) = y$) as follows:

$\underline{A(y)}$
$i^* \leftarrow \{1, \ldots, \ell\}$; $b^* \leftarrow \{0,1\}$
$y_{i^*,b^*} = y$
for all i, b with $1 \le i \le \ell$ and $b \in \{0,1\}$ and $(i,b) \ne (i^*, b^*)$:
  $x_{i,b} \leftarrow \{0,1\}^k$; $y_{i,b} = f(x_{i,b})$
$pk = \begin{pmatrix} y_{1,0} \cdots y_{\ell,0} \\ y_{1,1} \cdots y_{\ell,1} \end{pmatrix}$
run $A'(pk)$ until it requests a signature on $m = m_1 \cdots m_\ell$
if $m_{i^*} = b^*$, abort; otherwise, return the correct signature to A'
eventually, A' outputs a forged signature $(x_1, \ldots, x_\ell)$ on $m'_1 \cdots m'_\ell$
if $m'_{i^*} \ne b^*$ abort; otherwise, output x_{b^*}

In words, A does the following: it first chooses a random index i^* and a random bit b^*. This defines a position in the public key at which A will place the value y that it wants to invert. (Namely, A' sets $y_{i^*,b^*} = y$.) The remainder of the public key is generated honestly. This means that A *can output a correct signature for any message m such that $m_{i^*} \ne b^*$*. Then, A runs A' (giving A' the public key that A prepared) until A' requests a signature on message m. As noted, A can generate a perfectly valid signature as long as $m_{i^*} \ne b^*$. Otherwise, A simply aborts and gives up.

Assuming A has not aborted, it gives the signature thus computed to A' and continues running A' until A' returns a (supposed) forgery $(x_1, \ldots, x_\ell)$ on message $m'_1 \cdots m'_\ell$. Conditioned on the fact that A' has not aborted, this is a valid forgery with probability ε'. A valid forgery in particular means that $f(x_{i^*}) = y_{i^*,m'_{i^*}}$ and also that $m' \ne m$. In this case, if we also have $m'_{i^*} = b^*$ then A has found an inverse for y (since $y_{i^*,b^*} = y$).

We now analyze the probability that A finds a correct inverse. This occurs as long as the following three things happen: (1) A is able to return a correct signature to A' (i.e., $m_{i^*} \neq b^*$); (2) A' outputs a valid forgery; and (3) the forgery satisfies $m'_{i^*} = b^*$. The probability of event (1) is $1/2$ since b^* was chosen at random, and is independent of the view of A' up to and including the point when it requests a signature on message m (this follows since *all* entries $y_{i,b}$ [including y_{i^*,b^*}] of the public key are computed by choosing a random element $x_{i,b}$ from the domain of f and then setting $y_{i,b} = f(x_{i,b})$). Conditioned on the fact that event (1) occurs, the probability of event (2) is exactly ε', the assumed probability of forgery for A'. Finally, conditioned on the fact that events (1) and (2) both occur, we must have $m' \neq m$. So there exists at least one position i such that $m_i \neq m'_i$. If this i equals i^* then we are done (since event (1) occurred we know that $m_{i^*} \neq b^*$ so $m'_{i^*} = b^*$). Since m' is ℓ bits long and since the value of i^* is independent of the view of A' up to this point, i is equal to i^* with probability at least $1/\ell$.

Putting everything together, we see that A inverts f with probability $\frac{1}{2} \cdot \varepsilon' \cdot \frac{1}{\ell} = \varepsilon'/2\ell > \varepsilon$. Also, A runs in time (essentially) $t' + (2\ell - 1)t_f < t$. This contradicts the assumed security of f, proving that A' as described cannot exist. ■

4 Full Domain Hash (FDH)

As we have mentioned in passing previously, "full-fledged" signature schemes can be constructed from the minimal assumption of one-way functions (if you think about it, this is quite an amazing result!). However, constructions based on general assumptions such as one-way functions are not very practical. In fact, there is essentially only one known construction of a secure signature scheme in the standard model which is practical. Thus, we turn to the random oracle model to help us design efficient and provably-secure (albeit with all the caveats of the random oracle model) scheme based on trapdoor permutations. Before presenting the scheme, we define a concrete notion of security for the latter.

Definition 5 Let Gen_{td} represent a generation algorithm for a trapdoor permutation family. We say this family is (t, ε)-secure if for all adversaries A running in time at most t we have:

$$\Pr[(f, f^{-1}) \leftarrow \mathsf{Gen}_{td}; y \leftarrow D_f; x \leftarrow A(f, y) : f(x) = y] \leq \varepsilon,$$

where D_f is the domain/range of f (implicit in the description of f). ♢

We now describe the *full-domain hash* (FDH) signature scheme [1].

$\underline{\mathsf{Gen}(1^k)}$
$(f, f^{-1}) \leftarrow \mathsf{Gen}_{td}$
let $H : \{0,1\}^* \to D_f$
$pk = (f, H); sk = f^{-1}$
output (pk, sk)

$\underline{\mathsf{Sign}_{sk}(m)}$
output $\sigma = f^{-1}(H(m))$

$\underline{\mathsf{Vrfy}_{pk}(m, \sigma)}$
output 1 iff $f(\sigma) \stackrel{?}{=} H(m)$

It is not hard to see that correctness is satisfied. We now show that the scheme is secure if H is modeled as a random oracle.

Theorem 2 *If* Gen_{td} *is* (t, ε)*-secure and* f *requires time* t_f *to evaluate, then no adversary running in time* $O(t - q_h \cdot t_f)$ *can "break" FDH in the sense of existential unforgeability under adaptive chosen-message attack with probability better than* $q_h \cdot \varepsilon$ *in the random oracle model. Here,* q_h *is a bound on the number of hash queries made by the adversary.*

Again, since q_h and t_f are polynomial in the security parameter, this means that FDH is asymptotically secure as well.

Proof Assume to the contrary that there exists an adversary A' running in time $t' = O(t - q_h \cdot t_f)$ that succeeds in breaking the scheme with probability $\varepsilon' > q_h \cdot \varepsilon$. We construct an adversary A running in time t and inverting f with probability better than ε, a contradiction. We assume without loss of generality that (1) whenever A' asks a query $\mathsf{Sign}_{sk}(m)$, it has previously asked query $H(m)$; (2) if A' outputs alleged forgery (m, σ), it has previously queried $H(m)$ and has not previously queried $\mathsf{Sign}_{sk}(m)$; and (3) A' never queries the same value twice to H. Construct A (who tries to invert f at point y) as follows:

$\underline{A(f, y)}$
choose $i^* \leftarrow \{1, \ldots, q_h\}$
run $A'(f)$, answering queries to H and Sign_{sk} as follows:
 $\underline{\text{on the } i^{\text{th}} \text{ query } m_i \text{ to } H\text{:}}$
  if $i = i^*$ return y
  else, $x_i \leftarrow D_f$ and return $y_i = f(x_i)$
 $\underline{\text{on query } \mathsf{Sign}_{sk}(m)\text{:}}$
  let i be such that $m = m_i$
   (i.e., m was the i^{th} query to H)
  if $i = i^*$ abort
  otherwise, return x_i
when A' outputs (m^*, σ), find i such that $m^* = m_i$
if $i \neq i^*$ abort
else output σ

We make a number of observations about A. First, if A does not abort before A' outputs its (supposed) forgery, then the simulation provided for A' is perfect: all queries to the random oracle are answered with a point in D_f chosen independently at random (where we use the fact that f is a permutation, and also the fact that y is chosen at random) and all signing queries are answered correctly (since $f(x_i) = H(m_i)$ be construction). Second, if A does not abort during the course of the entire experiment that means $m^* = m_{i^*}$ and hence if A' has output a valid forgery we have $f(\sigma) = H(m_{i^*}) = y$, and thus A succeeds in inverting f at y. Finally, the running time of A is (essentially) $t' + (q_h - 1)t_f \leq t$.

It remains to analyze the probability that A does not abort. Note that A does not abort whenever $m^* = m_{i^*}$ (since in this case A' has also not queried $\mathsf{Sign}_{sk}(m_{i^*})$). Furthermore, the value of i^* is information-theoretically hidden from A' until such time (if any) that A aborts. Since $m^* = m_i$ for *some* $i \in \{1, \ldots, q_h\}$, the probability that $m^* = m_{i^*}$ is exactly $1/q_h$. The probability that A outputs a correct inverse is therefore $\varepsilon'/q_h > \varepsilon$, giving the desired contradiction. ■

4.1 An Improved Security Reduction Using RSA [2]

The proof in the previous section shows that FDH is asymptotically secure. In practice, however, the concrete security bound derived may not be "good enough" (or, put another way, achieving a reasonable level of security may not be "efficient enough"). For example, say we set $q_h \approx 2^{50}$ which simply means that an adversary evaluates SHA-1 on their own computer 2^{50} times (this is a large, but perfectly reasonably, number). If we use a trapdoor permutation which is (5 years, 2^{-60})-secure for sake of argument, say 2048-bit RSA), then the proof given previously shows that an adversary running for 3 years... cannot forge a signature in FDH with probability better than $2^{50} \cdot 2^{-60} = 2^{-10}$, which is not such a great guarantee! Of course, we can always "fix" this by using larger moduli; for example, if we assume that 4096-bit RSA is (5 years, 2^{-120})-secure then we achieve the acceptable probability of forgery $2^{50} \cdot 2^{-120} = 2^{-70}$. In this case, however, our scheme will be less efficient (since we are using a large modulus).

A natural question is: can we design a signature scheme with a better security reduction (say, where the probability of forgery is roughly equal to the probability of inverting the trapdoor permutation)? Or, can we improve our proof of security for the case of FDH? Both of these problems have been considered, and we will focus on the second one here. In particular, we will show that *for the particular case when RSA is used* as the trapdoor permutation for FDH, a better security reduction can be obtained. (The technique relies on some specific algebraic properties of RSA, and extends to other trapdoor permutations, but not to all trapdoor permutations.)

Theorem 3 *If the RSA trapdoor permutation (for some particular choice of key length) is (t, ε)-secure and takes time t_f to evaluate, then no adversary running in time $O(t - q_h \cdot t_f)$ can "break" RSA-FDH with probability better than $O(q_s \cdot \varepsilon)$ in the random oracle model. Here, q_s is a bound on the number of* signature *queries made by the adversary.*

Note that this offers much better security since $q_s \ll q_h$ (it is much harder to get a signer to "obliviously" sign something for you than to evaluate a hash function repeatedly).

Proof Here, we let $f = (N, e)$ be the RSA function (where the modulus N is generated at random, and e is relatively prime to $\varphi(N)$). Assume to the contrary that there exists an adversary A' running in time $t' = O(t - q_h \cdot t_f)$ that succeeds in breaking the scheme with probability $\varepsilon' > 3q_s \cdot \varepsilon$. We construct an adversary A running in time t and inverting f with probability better than ε, a contradiction. We assume without loss of generality that (1) whenever A' asks a query $\mathsf{Sign}_{sk}(m)$, it has previously asked query $H(m)$; (2) if A' outputs alleged forgery (m, σ), it has previously queried $H(m)$ and has not previously queried $\mathsf{Sign}_{sk}(m)$; and (3) A' never queries the same value twice to H or Sign_{sk}. We now construct A (who tries to find $y^{1/e} \bmod N$) as follows:

$\underline{A(N, e, y)}$
run $A'(pk = (N, e))$, answering queries to H and Sign_{sk} as follows:
 $\underline{\text{on the } i^{\text{th}} \text{ query } m_i \text{ to } H\text{:}}$
  $x_i \leftarrow \mathbb{Z}_N^*$
  with probability γ set $b_i = 0$ and return $x_i^e \bmod N$
  otherwise (i.e., with probability $1 - \gamma$) set $b_i = 1$ and return $x_i^e \cdot y \bmod N$

on query $\mathsf{Sign}_{sk}(m)$:
  let i be such that $m = m_i$
    (i.e., m was the i^{th} query to H)
  if $b_i = 1$ abort
  otherwise, return x_i
when A' outputs (m^*, σ), find i such that $m^* = m_i$
if $b_i = 0$ abort
else output $\sigma / x_i \bmod N$

Here, γ is a parameter we will fix later. For future reference, note that the running time of A is (essentially) $t' + (q_h - 1)t_f \leq t$ (since an RSA exponentiation dominates multiplications and other operations modulo N).

The hash queries of A' can be divided into two classes: "class 0" (with $b = 0$) consists of messages m_i for which A knows $x_i \stackrel{\text{def}}{=} H(m_i)^{1/e}$; thus, A can answer signing queries for messages in this class, but if A' forges a signature for a message in this class it does not help A (since it already knows a "forged signature" for this message anyway). "Class 1" (with $b = 1$) consists of messages for which A knows an x_i such that $x_i^e y = H(m_i)$; now, A cannot answer signing queries for such messages, but if A' forges a signature for a message in this class then A can invert y as follows: if $\sigma = H(m_i)^{1/e}$ then:

$$\sigma / x_i \bmod N = H(m_i)^{1/e} / (x_i^e)^{1/e} = (H(m_i)/x_i^e)^{1/e} = y^{1/e};$$

i.e., σ / x_i is the desired inverse of y.

Now, until such time (if any) that A aborts, the simulation provided to A' is perfect (in particular, all queries to the random oracle are answered with an independent and uniformly-random point in $\mathbb{Z}_N^*$), and furthermore A' has no information about which messages are in class 0 and which are in class 1. Since the probability that A does not abort is given by the product of the probabilities that (1) all the signing queries of A' are for "class 0" messages, and (2) the purported forgery of A' is for a "class 1" message, the probability that A does not abort is $\Pr[\text{no abort}] = \gamma^{q_s}(1 - \gamma)$. Choosing $\gamma = \frac{q_s}{q_s + 1}$ maximizes this expression and gives $\Pr[\text{no abort}] = e^{-1}/q_s$ (where $e \approx 2.72$ is the base of natural logarithms). Putting everything together, the probability that A outputs a correct inverse is $\varepsilon'/(e \cdot q_s) > \varepsilon$, giving the desired contradiction. ■

References

[1] M. Bellare and P. Rogaway. Random Oracles are Practical: a Paradigm for Designing Efficient Protocols. ACM Conference on Computer and Communications Security, 1993.

[2] J.-S. Coron. On the Exact Security of Full Domain Hash. Crypto 2000.

[3] S. Goldwasser, S. Micali, and R. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM J. Computing* 17(2): 281–308, 1988.

[4] L. Lamport. Constructing Digital Signatures from a One Way Function. SRI International Technical Report CSL-98 (October 1979).

CMSC 858K — Advanced Topics in Cryptography April 1, 2004

Lecture 17

Lecturer: Jonathan Katz

Scribe(s): Chiu Yuen Koo, Nikolai Yakovenko, Jeffrey Blank

1 "Limitations" of NIZK Proof Systems

In the NIZK proof systems we have seen, we have assumed that the prover and verifier share a common random string (CRS). Although in many applications of NIZK such an assumption presents no problem (e.g., in the application to chosen-ciphertext-secure public-key encryption, the receiver chooses a random string and includes it in her public key), in the general case it is not clear where this string comes from (if no trusted third-party is assumed). A natural question is whether NIZK can be achieved *without* a common random string. In fact, it is easy to see that this is impossible for any "non-trivial" language:

Lemma 1 *NIZK proofs are impossible without a CRS for any language $L \notin \mathcal{BPP}$.*

Proof (Sketch) Let (P, V) be an NIZK proof system for some language $L \in \mathcal{NP}$, with polynomial-time simulator Sim. For simplicity, we assume the protocol has perfect completeness and negligible soundness error, although these assumptions are not necessary. Our goal is to prove that $L \in \mathcal{BPP}$. By definition of NIZK, for any $x \in L$ the distribution of proofs π output by $\mathcal{P}(1^k, x, w)$ (where w is a witness for x) is indistinguishable from the distribution of proofs π output by $\mathsf{Sim}(1^k, x)$. In particular, we have $\mathcal{V}(\mathsf{Sim}(1^k, x), x) = 1$ with probability negligibly close to 1 for $x \in L$ (this follows from perfect completeness).

We show how to decide membership in L using a PPT algorithm A (by definition, this implies $L \in \mathcal{BPP}$). A works as follows: given x, it computes $\pi \leftarrow \mathsf{Sim}(1^k, x)$ and outputs 1 iff $\mathcal{V}(\pi, x) = 1$. By what we have said above, the probability that A outputs 1 when $x \in L$ is negligibly close to 1. On the other hand, *soundness* of the proof system implies that the probability that A outputs 1 when $x \notin L$ is negligible (since otherwise we have an easy way to "fool" $\mathcal{V}$ into accepting a proof of a false statement). □

It is instructive to see where the previous proof fails when a common random string r is available. In that case the simulator gets to choose r, so a simulator which outputs "valid-looking" proofs for false statements does not contradict the soundness of the NIZK proof system (since soundness holds with respect to the *fixed* string r that the cheating prover cannot change).

The above demonstrates that *interaction* is necessary if we want to have zero-knowledge protocols which do not rely on any set-up assumptions (such as a common random string).

2 Zero-Knowledge (Interactive) Proof Systems

Note: Definitions of zero-knowledge are quite subtle and, although we will do our best to be accurate, we may omit some details for simplicity. The interested reader is referred to

other works [2, 3] for more details and a more rigorous and careful treatment.

We first define the notion of zero knowledge. Note that the definition is more complex than in the non-interactive case because in the latter case we only need to consider the proofs generated by a legitimate (honest) prover; in the interactive case, on the other hand, we must also consider the actions of a prover *when interacting with a dishonest verifier* who may not follow the protocol.

Definition 1 A pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$ is called a zero-knowledge (ZK) proof system for a language $L \in \mathcal{NP}$ (with $L_k \stackrel{\text{def}}{=} L \cup \{0,1\}^{\leq k}$) if it satisfies the following properties:

1. **(Completeness)** For all k, all $x \in L_k$, and all witnesses w for x,

$$\Pr[\mathcal{V}(1^k, x) \text{ accepts when interacting with } \mathcal{P}(1^k, x, w)] = 1.$$

2. **(Soundness)** For all $x \notin L_k$ and all (even all powerful) $\mathcal{P}^*$, the following is negligible:

$$\Pr[\mathcal{V}(1^k, x) \text{ accepts when interacting with } \mathcal{P}^*(x)].$$

3. **(Zero knowledge)** For all PPT (cheating verifiers) $\mathcal{V}^*$, there exists an (expected) polynomial time simulator Sim such that the following are computationally indistinguishable for any $x \in L_k$ and witness w for x:
   - the view of $\mathcal{V}^*(1^k, x)$ when interacting with $\mathcal{P}(1^k, x, w)$;
   - the output of $\mathsf{Sim}(1^k, x)$.

$\diamondsuit$

Note that the simulator is given exactly what the verifier knows. Thus, the intuition behind a ZK proof is that "anything a poly-time cheating verifier can learn from its interaction with $\mathcal{P}$ it could learn on its own, anyway (in essentially the same amount of time)". In the definition above, we allowed Sim to run in *expected* polynomial time rather than requiring it to run in (strict) polynomial time.[1] Also, the above definition considers only *uniform* poly-time cheating verifiers; the "standard" definition, however, requires the zero-knowledge condition to hold even for *non-uniform* poly-size cheating verifiers (i.e., poly-time verifiers with *auxiliary inputs*) for good reason. (We have chosen to ignore the issue for ease of presentation.) See [2, 3] for further details.

We first show a ZK proof system for the language of *graph isomorphism*.

Definition 2 Let $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ be two graphs, where V_i is the set of vertices and E_i is the set of edges of G_i. The graphs G_1 and G_2 are *isomorphic* (denoted by $G_1 \approx G_2$) if there exists a permutation $\varphi : V_1 \to V_2$, such that $(u, v) \in E_1 \Leftrightarrow (\varphi(u), \varphi(v)) \in E_2$. Such a φ is called an *isomorphism* from G_1 to G_2. We also let $\varphi(G_1)$ denote the graph obtained by "permuting" the vertices of G_1 according to φ and preserving edges; clearly, $G_2 = \varphi(G_1)$ (hopefully, using φ to refer to a permutation of the vertex set of G_1 as well as to a permutation of the graph G_1 will not cause confusion). $\diamondsuit$

[1] Allowing the simulator to run in expected poly-time is somewhat "controversial", and certainly strict polynomial time simulation is preferable. However, the only currently-known way to achieve "reasonably-efficient" ZK protocols is to allow expected poly-time simulation. For more discussion, see [2] or [1].

Define the language *graph isomorphism* by $L \stackrel{\text{def}}{=} \{(G_1, G_2) \mid G_1 \approx G_2\}$. Clearly, we have $L \in \mathcal{NP}$ (since it is easy to check that a given isomorphism is valid). Consider the following proof system for L: the prover begins with G_1, G_2, and an isomorphism φ from G_1 to G_2; the verifier knows only G_1, G_2. The protocol proceeds as follows:

1. $\mathcal{P}$ chooses a random permutation φ', and sends $H = \varphi'(G_1)$ to $\mathcal{V}$.
2. The verifier chooses a random $b \in \{1, 2\}$ and sends b to $\mathcal{P}$.
3. We assume the verifier sends a $b \in \{1, 2\}$ (any other response can be taken, by default, to represent a "1"), and the prover will send an isomorphism from G_b to H. In particular, if $b = 1$ then the prover sends φ'; if $b = 2$, the prover sends $\varphi' \circ \varphi^{-1}$ (i.e., the composed permutation).
4. The verifier receives a permutation φ'' and accepts iff $\varphi''(G_b) \stackrel{?}{=} H$.

Theorem 2 *The above is a zero-knowledge protocol (with soundness error 1/2) for graph isomorphism.*

Proof Clearly, $\mathcal{P}$ and $\mathcal{V}$ can be implemented in polynomial time. We now argue completeness. Assume the prover acts honestly, and so $H = \varphi'(G_1)$ for some permutation φ'. If $b = 1$ then the verifier clearly accepts. If $b = 2$, then the verifier also accepts since in this case $\varphi''(G_2) = \varphi' \circ \varphi^{-1}(G_2) = \varphi'(G_1) = H$.

We now show that the above protocol achieves soundness error 1/2 (meaning that if G_1, G_2 are *not* isomorphic, then no prover can make a verifier accept with probability better than 1/2). Assume $G_1 \not\approx G_2$, and let H be the graph sent by the prover in the first round. Note that at most one of $G_1 \approx H$ or $G_2 \approx H$ can be true (if they are both true, then by transitivity we have $G_1 \approx G_2$ which we know is not the case). So the prover can respond correctly to at most one of the verifier's possible challenges sent in the third round, and thus will succeed in making the verifier accept with probability at most 1/2. (The soundness error is not negligible, as required by Definition 1; however, we discuss below an easy way to modify the protocol to achieve negligible soundness error.)

The most interesting property to consider is the zero knowledge of the above protocol. To that end, we show the following simulator for any cheating verifier $\mathcal{V}^*$ (recall from the definition that the simulator is allowed to depend — in fact, must depend — on the verifier):

$\underline{\mathsf{Sim}(1^k, (G_1, G_2))}$
fix random coins ω for V^*
for $i = 1$ to k:
  choose a random permutation φ'
  $b \leftarrow \{1, 2\}$
  $H = \varphi'(G_b)$
  run $\mathcal{V}^*(1^k, (G_1, G_2), H; \omega)$ to obtain its response b'
  if $b = b'$ output view (ω, H, b, φ') and stop
if none of the above iterations have succeeded, output $\perp$

Fix $(G_1, G_2) \in L$ (recall that Definition 2 only requires simulation in this case). We first analyze the probability that Sim outputs $\perp$. In any given iteration of the inner loop and

for any value of ω, note that the value of b chosen by the simulator is *perfectly independent* of H: the graph H is a random isomorphic copy of G_b, but that has the same distribution as a random isomorphic copy of $G_{\bar{b}}$ (since $G_1 \approx G_2$). That means that the value of b is independent of the view of V^* in line 4 of the inner loop. Therefore, the probability that $b' = b$ is exactly $1/2$. Overall, then, the probability that we *never* have $b = b'$ is 2^{-k}, which is negligible. So, Sim outputs $\perp$ with only negligible probability.

We next claim that, conditioned on Sim not outputting $\perp$, the output of Sim is identically distributed to the view of $\mathcal{V}^*$. To see this, let us consider the elements of the view (ω, H, b, φ') one-by-one: clearly, ω is a random string having the same distribution as ω in a real interaction of $\mathcal{V}^*$ with $\mathcal{P}$. The next component, H, is a random isomorphic copy of G_b for some b, but we have already noted above that this is distributed identically to a random isomorphic copy of G_1 (which is what $\mathcal{P}$ sends in the real protocol). The challenge bit b is then a deterministic function of ω and H (since we have already fixed the coins ω of $\mathcal{V}^*$), and in both the real and simulated experiments is equal to $\mathcal{V}^*(1^k, (G_1, G_2), H)$. Finally, in the simulation φ' is uniformly distributed among isomorphisms mapping G_b to H; again, this is exactly as in a real execution of the protocol.

We remark that the above simulator runs in *strict* polynomial time. ■

The proof system presented above has soundness error $1/2$, but it is easy to make the soundness error as small as desired (and, in particular, negligible): simply repeat the above protocol ℓ times to achieve soundness error $2^{-\ell}$. Setting $\ell = \omega(\log k)$, the protocol can still be implemented in polynomial time and the soundness error becomes negligible.

We need to be careful, however, that in changing the protocol we do not destroy its zero-knowledge property! There are two natural ways to implement the ℓ-fold repetition suggested above: the first is to run the ℓ instances of the protocol *in parallel*: in round 1 the prover sends ℓ different graphs $H_1, \ldots, H_\ell$ (each computed using an independent isomorphism); in round 2 the verifier replies with ℓ challenges $b_1, \ldots, b_\ell$, and in round 3 the prover responds to each challenge as in the original protocol. The second natural possibility is to run ℓ instances of the protocol *sequentially* in the obvious way. A drawback of the latter method is that the round complexity becomes $O(\ell)$, whereas when using the first method the round complexity remains the same (i.e., three rounds) as in the original protocol.

Unfortunately, *running the above protocol* **in parallel** *for $\ell = \omega(\log k)$ is not known to result in a zero knowledge protocol.* (It is not known definitively that the protocol is *not*[2] zero knowledge, but there is no known way of proving that the protocol *is* zero knowledge, either.) To see the difficulties that arise, imagine adapting the simulator given earlier for the case of ℓ-fold parallel repetition. Now, the simulator will "guess" in advance the challenge bits $b_1, \ldots, b_\ell$, construct graphs $H_1, \ldots, H_\ell$ in the appropriate way, and then hope that the output $b'_1, \ldots, b'_\ell$ of $\mathcal{V}^*$ satisfies $b'_i = b_i$ for all i. But the probability that this occurs is $2^{-\ell}$ (note that we cannot assume that $\mathcal{V}^*$ uses the same challenge bits every time — for all we know, $\mathcal{V}^*$ may choose its challenge bits based on all the graphs $H_1, \ldots, H_\ell$ sent by $\mathcal{P}$ in the first round). If $\ell = O(\log k)$ then $2^{-\ell}$ is inverse polynomial, and so by repeating this process some sufficiently-large polynomial number of times the simulator will succeed with all but negligible probability (as in the case of the original simulator). On the other hand, if $\ell = \omega(\log k)$ then $2^{-\ell}$ is negligible and so any simulator of this sort running in (expected)

[2] Although there are reasons to believe that constructing a simulator for this protocol will be "difficult" [4].

polynomial time will only succeed in outputting a valid view with negligible probability. But we need $\ell = \omega(\log k)$ to obtain negligible soundness error...

The good news is that ℓ-fold *sequential* repetition of the above protocol *does* result in a zero-knowledge protocol for any $\ell = \text{poly}(k)$ (note that we must have $\ell = \text{poly}(k)$ in order for the protocol to run in polynomial time).[3] The simulator for this protocol will essentially run the simulator given earlier, ℓ times sequentially. For completeness, we describe the simulator here but leave the analysis to the reader:

$\underline{\mathsf{Sim}(1^k, (G_1, G_2))}$
fix random coins ω for V^*
let $\mathsf{view}_0 = \omega$
for $i = 1$ to ℓ:
  for $j = 1$ to k:
    choose a random permutation φ'
    $b \leftarrow \{1, 2\}$
    $H = \varphi'(G_b)$
    run $\mathcal{V}^*(1^k, (G_1, G_2), \mathsf{view}_{j-1}, H)$ to obtain its response b'
    if $b = b'$, let $\mathsf{view}_j = \mathsf{view}_{j-1}||(H, b, \varphi')$ and exit loop
  if none of the above iterations have succeeded, output $\perp$
output view_ℓ

3 Honest-Verifier Zero Knowledge

A weaker definition of zero knowledge which is often useful is that of *honest-verifier zero knowledge* (HVZK). In this case, we only require simulation for a verifier who honestly follows the protocol, but may later try to learn some information (that it wasn't supposed to) from the transcript. This models so-called "honest-but-curious" adversaries who act in this way, and also models adversaries who eavesdrop on an honest execution of a protocol (between an honest verifier and the prover).

Definition 3 A pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$ is called a zero-knowledge (ZK) proof system for a language $L \in \mathcal{NP}$ (with $L_k \stackrel{\text{def}}{=} L \cup \{0,1\}^{\leq k}$) if it satisfies the following properties:

1. **(Completeness and soundness)** As in Definition 1.
2. **Honest-verifier zero knowledge** There exists a polynomial time simulator Sim such that the following are computationally indistinguishable for any $x \in L_k$ and witness w for x:
   - the view of $\mathcal{V}(1^k, x)$ when interacting with $\mathcal{P}(1^k, x, w)$;
   - the output of $\mathsf{Sim}(1^k, x)$.

$\diamondsuit$

We do not allow for expected poly-time simulation since it is not needed to get efficient protocols (since the zero-knowledge property was weakened).

[3] In fact, *auxiliary-input* zero-knowledge (mentioned earlier but not formally defined in these notes) is always preserved under sequential composition; see [2, 3].

We now show that ℓ-fold parallel repetition of the previously-shown protocol for graph isomorphism is honest-verifier zero knowledge (for any polynomial ℓ):

$\underline{\mathsf{Sim}(1^k, (G_1, G_2))}$
choose random permutations $\varphi'_1, \ldots, \varphi'_\ell$
choose random $b_1, \ldots, b_\ell \leftarrow \{1, 2\}$
for all $i \in \{1, \ldots, \ell\}$: $H_i = \varphi'_i(G_{b_1})$
output $(H_1, \ldots, H_\ell; b_1, \ldots, b_\ell; \varphi'_1, \ldots, \varphi'_\ell)$

Note that the simulator's job here is much easier — because we are only concerned with simulating the view of an honest verifier, we may assume that the challenge bits $b_1, \ldots, b_\ell$ are chosen uniformly and independently of the graphs $H_1, \ldots, H_\ell$ sent in the first round.

References

[1] B. Barak and Y. Lindell. Strict Polynomial Time in Simulation and Extraction. STOC 2002. Full version available at http: eprint.iacr.org.

[2] O. Goldreich. *Foundations of Cryptography, vol. 1: Basic Tools*. Cambridge University Press, 2001.

[3] O. Goldreich. Tutorial on Zero Knowledge. Available at http://www.wisdom.weizmann.ac.il/~oded/zk-tut02.html

[4] O. Goldreich and H. Krawczyk. On the Composition of Zero-Knowledge Proof Systems. *SIAM J. Computing* 25(1): 169–192 (1996).

CMSC 858K — Advanced Topics in Cryptography April 6, 2004

Lecture 18

Lecturer: William Gasarch *Scribe(s): Avinash J. Dalal, Julie Staub*

1 Summary — What is Private Information Retrieval?

In this set of lecture notes we begin a brief discussion of **Private Information Retrieval** (PIR). We begin by defining what PIR is about. We then present two PIR protocols, and give simple proofs that these two protocols are indeed correct. Finally, we discuss some generalizations of these basic schemes.

Let us begin by introducing the problem. Suppose we have a database $DB = (x_1, \ldots, x_n)$, viewed as an n-bit string (i.e., the database has n entries, with each entry being a single bit). Suppose further that a user wants to learn the i^{th} bit of the database *without leaking to the database administrator*[1] *any information about* i (in an information-theoretic sense).

Informally, this is known as the PIR problem. Note that there does exist a solution: simply send to the user the contents of the entire database. The user can discard all the entries other than the one it is interested in, and the database clearly learns no information about which entry this is. The communication complexity of this solution, however, is $O(n)$. The PIR problem asks: can we do better?

It is actually not too difficult to show that it is impossible to do better in the above scenario if we require information-theoretic privacy for the user (i.e., even an all-powerful database learns nothing about the bit i the user is interested in) [1]. Thus, we must relax the scenario a bit to allow for more efficient solutions. One possibility that we explore here is to assume that there are multiple (identical) *copies* of the database, and the administrators of these databases cannot communicate with each other. We see that this allows for much more efficient solutions (in terms of their communication complexity).

2 A First Approach

We discuss here an approach yielding schemes with communication complexity $\sqrt[d]{n}$ when there are 2^d copies of the databases. The scheme will be illustrated first for the cases of $d = 2$ and $d = 3$ and we then discuss its extension to the general case.

A 4-Database PIR Protocol. Suppose we have 4 identical copies of the data in databases DB_1, DB_2, DB_3, DB_4. We show a PIR protocol with $O(\sqrt{n})$ communication complexity in this setting. View the data (which is an n-bit string) as a $\sqrt{n} \times \sqrt{n}$ array $\{x_{i,j}\}_{1 \le i,j \le \sqrt{n}}$ (we assume for simplicity that n is a square). When the user wants to learn bit x_{i^*,j^*} at position (i^*, j^*) he proceeds as follows:

1. Choose two random strings S, T each of length $\sqrt{n}$, and view these as subsets of $\{1, \ldots, \sqrt{n}\}$ in the natural way.

[1] From now on, "the database" will refer to the data itself as well as to the "administrator" of this data.

2. Let $S' = S \oplus \{i^*\}$ and $T' = T \oplus \{j^*\}$. Here, we let

$$S \oplus \{s\} \stackrel{\text{def}}{=} \begin{cases} S \cup \{s\} & s \notin S \\ S \setminus \{s\} & s \in S \end{cases}.$$

   We now have four subsets (equivalently, $\sqrt{n}$-bit strings) S, T, S', T'.

3. Send:

   - (S, T) to DB_1;
   - (S, T') to DB_2;
   - (S', T) to DB_3; and
   - (S', T') to DB_4.

4. Database k receives a pair of subsets $(\hat{S}_k, \hat{T}_k)$, and sends to the user the single bit:

$$X_k \stackrel{\text{def}}{=} \bigoplus_{i \in \hat{S}_k, j \in \hat{T}_k} x_{i,j}.$$

   Here, of course, $\oplus$ represents the bit-wise xor operation.

5. Upon receiving the responses from the databases, the user computes

$$x_{i^*,j^*} = X_1 \oplus X_2 \oplus X_3 \oplus X_4.$$

We first show that the above protocol is *correct*; i.e., that the user correctly recovers the bit of interest. To see this, note that:

$$X_1 \oplus X_2 \oplus X_3 \oplus X_4 = \bigoplus_{i \in S, j \in T} x_{i,j} \oplus \bigoplus_{i \in S', j \in T} x_{i,j} \oplus \bigoplus_{i \in S, j \in T'} x_{i,j} \oplus \bigoplus_{i \in S', j \in T'} x_{i,j}.$$

Now, we claim that for each $(i, j) \neq (i^*, j*)$, the value $x_{i,j}$ appears an even number of times in the above sum; in particular:

- If $i \neq i^*$ and $j \neq j^*$ then (i, j) is in either zero or all of the sets $S \times T$, $S \times T'$, $S' \times T$, $S' \times T'$.
- If $i = i^*$ but $j \neq j^*$ (or vice versa) then (i, j) is in either zero or exactly two of the sets $S \times T$, $S \times T'$, $S' \times T$, $S' \times T'$.

Thus, all of these contributions cancel out. On the other hand, the value x_{i^*,j^*} appears an odd number of times in the above sum, and hence $X_1 \oplus X_2 \oplus X_3 \oplus X_4 = x_{i^*,j^*}$, as desired.

We next argue that the above protocol is *private*; this is straightforward since each database simply receives a pair of uniformly-distributed $\sqrt{n}$-bit strings. (It is easy to see that S, T are uniformly distributed. For the case of, e.g., S', note that $S' \oplus \{i^*\}$ is uniformly distributed since S is uniformly distributed; the set S is acting as a "one-time pad" of sorts.)

An 8-Database PIR Protocol. As Dr. Gasarch would ask: "Can we do better?". In particular, what if we had more copies of the database? In fact, we can generalize the above protocol to one with communication complexity $O(\sqrt[3]{n})$ when there are 8 databases $DB_1, \ldots, DB_8$ available. Here, we view the data as a $\sqrt[3]{n} \times \sqrt[3]{n} \times \sqrt[3]{n}$ cube $\{x_{i,j,k}\}_{1 \leq i,j,k \leq \sqrt[3]{n}}$. When the user wants to learn the bit x_{i^*,j^*,k^*} at position (i^*, j^*, k^*) he proceeds as follows:

1. Choose three random strings R, S, T each of length $\sqrt[3]{n}$, and view these as subsets of $\{1, \ldots, \sqrt[3]{n}\}$ in the natural way.

2. Let $R' = R \oplus \{i^*\}$, $S' = S \oplus \{j^*\}$, and $T' = T \oplus \{k^*\}$, where the notation is as in the previous section. We now have eight subsets (equivalently, $\sqrt[3]{n}$-bit strings) R, R', S, T, S', T'.

3. Send:

$$\begin{array}{ll} (R,S,T) \text{ to } DB_1 & (R',S,T) \text{ to } DB_5 \\ (R,S,T') \text{ to } DB_2 & (R',S,T') \text{ to } DB_6 \\ (R,S',T) \text{ to } DB_3 & (R',S',T) \text{ to } DB_7 \\ (R,S',T') \text{ to } DB_4 & (R',S',T') \text{ to } DB_8 \end{array}$$

4. Database w receives subsets $(\hat{R}_w, \hat{S}_w, \hat{T}_w)$, and sends to the user the single bit:

$$X_w \stackrel{\text{def}}{=} \bigoplus_{i \in \hat{R}_w, j \in \hat{S}_w, k \in \hat{T}_w} x_{i,j,k}.$$

5. Upon receiving the responses from the databases, the user xor's them all together to compute the desired bit.

Proofs of correctness and privacy are exactly as before, and are therefore omitted.

2.1 Extending the Scheme

It should be clear that the above approach generalizes to give a 2^d-database scheme with communication complexity $\sqrt[d]{n}$, for any integer $d \geq 1$. The database is viewed as a d-dimensional hypercube $(\sqrt[d]{n})^d$, and then the approach above is applied. For details, see [1].

3 Improving the Previous Approach

Here, we show how the last scheme can be improved: we show a scheme with the same communication complexity but using only *two* databases. The basic intuition leading to this improvement is to balance the communication between the user and the databases. In the previous scheme, the user sends $O(\sqrt[3]{n})$ bits to each database, while each database responds with a single bit. By shifting more communication onto the databases, we are able to reduce the number of databases needed.

Recall in the previous scheme (of Section 2), the user chooses initial random sets R, S, T and then sends a triple of sets (based on these sets as well as the index of interest) to each of the eight databases. We show how the information sent by these eight databases can in fact be sent by two databases... but still without leaking any information to either of these databases about the index the user is interested in.

Let R, S, T, R', S', T' be as in Section 2. The user will send R, S, T to the first database, and R', S', T' to the second. (Note that these leak no information to either database about the index the user is interested in.) We would like the databases to now "simulate" the actions of the eight databases in the original scheme. Clearly, it is easy for the first database

to simulate DB_1 from before and equally easy for the second database to simulate DB_8. What about the other databases? Consider how the first database might simulate the actions of DB_2. To do this, it seemingly needs to know T'; on the other hand, if it knew T' then (using T) it would be able to determine k^* and this would leak information about the user's query. Instead, what it will do is *try all possible values for* T', and simulate DB_2 for each possibility. Since T and T' differ in only one position, there are only $\sqrt[3]{n}$ possibilities for T'. So, simulating the response of DB_2 for each of these possibilities only requires an additional $\sqrt[3]{n}$ bits (one bit per possibility).

In exactly this way, the first database can simulate the responses of DB_3 and DB_4. The total communication it sends to the user is therefore $3\sqrt[3]{n}+1$ bits.

The second database simulates DB_5, DB_6, and DB_7 in a similar manner (and acts exactly as DB_8 would). So it also sends $3\sqrt[3]{n}+1$ bits to the user. At this point, it is clear that the user can recover the desired answer (it just picks out the appropriate bits from the two responses, and then xor's them together as in the previous scheme), and the total communication complexity is $O(\sqrt[3]{n})$ bits.

3.1 Extending this Approach for More Databases

Let us try to generalize the approach of the previous section. Abstractly, we can view the eight triples of sets that the user sends to the eight databases in the original scheme as binary strings of length three: the tuple (R, S, T) corresponds to $(0, 0, 0)$, (R, S, T') corresponds to $(0, 0, 1)$, etc. In the improved scheme, we send the "sets" $(0, 0, 0)$ to the first database and let it simulate $(0, 0, 1)$, $(0, 1, 0)$, and $(1, 0, 0)$. We also send the "sets" $(1, 1, 1)$ to the second database and let it simulate $(1, 1, 0)$, $(1, 0, 1)$, and $(0, 1, 1)$. The key point is that each database simulates queries of Hamming distance 1 from the query it receives. This is what allows the total communication complexity to remain $O(\sqrt[3]{n})$.

Consider the 16-database, $O(\sqrt[4]{n})$-bit PIR protocol which is the extension of the schemes in Section 2. Here, the user sends "queries" in the form of 4-bit strings (i.e., the "query" $(0, 0, 0, 0)$ represents four random subsets (R, S, T, W) of $\{1, \ldots, \sqrt[4]{n}\}$). If we try to apply the improvement above, we see that we cannot do it using only two databases: if we send $(0, 0, 0, 0)$ to the first database and ask it to simulate query $(1, 1, 0, 0)$, for example, then it will have to try $\sqrt[4]{n} \times \sqrt[4]{n}$ different possibilities (namely, $\sqrt[4]{n}$ possibilities for each of R' and S') and the communication complexity will be too high. A little thought shows that in order to implement this improvement we need *four* databases: the user will send $(0, 0, 0, 0)$ to the first, $(1, 1, 0, 0)$ to the second, $(0, 0, 1, 1)$ to the third, and $(1, 1, 1, 1)$ to the fourth and now every 4-bit string in within Hamming distance 1 of at least one of these representatives.

For further information, see [1] or the extensive PIR references available at [2].

References

[1] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private Information Retrieval. *Journal of the ACM* 45(6): 965–981, 1998.

[2] W. Gasarch. http://www.cs.umd.edu/~gasarch/pir

CMSC 858K — Advanced Topics in Cryptography April 8, 2004

Lecture 19

Lecturer: Jonathan Katz *Scribe(s): Nikolai Yakovenko, Jeffrey Blank*

1 Introduction and Preliminaries

In a previous lecture, we showed a zero-knowledge (ZK) proof system for the language of graph isomorphism. Our goal here is to show a ZK proof system for any language in $\mathcal{NP}$. To do so, it suffices to show a ZK proof system Π for any $\mathcal{NP}$-complete language L (note that graph isomorphism is not believed to be $\mathcal{NP}$-complete); given such a Π and any $L' \in \mathcal{NP}$, we then obtain a ZK proof for L' by (1) reducing the common input x' (which is supposedly in L') to a string x such that $x' \in L' \Leftrightarrow x \in L$; and then (2) running the original proof system Π on common input x. (Actually, if we want this to work for a poly-time prover then we need the reduction from L' to L to also preserve witnesses; i.e., there should be poly-time computable functions f_1, f_2 such that $x' \in L' \Leftrightarrow f_1(x') \in L$ and if w' is a witness for $x' \in L'$ then $f_2(w')$ should be a witness that $f_1(x') \in L$.)

In this lecture, we show a ZK proof system for the language of 3-colorability, which is $\mathcal{NP}$-complete. Before doing so, we will first define the notion of a *commitment scheme*.

1.1 Commitment Schemes

Informally, a commitment scheme provides a way for a *sender* to commit to a value without revealing it to a *receiver*. At some later point, however, the sender can reveal his committed value and the receiver will be convinced that the sender did not "change his mind". A good analogy is the following commitment scheme which works when the parties are sitting at a table together: the sender writes his value on a piece of paper, places it in an envelope, seals the envelope, and places the envelope on the table. Assuming normal paper and ink, the sender certainly cannot change the value inside the envelope (i.e., he is *committed* to that value) yet the receiver cannot learn the value while the envelope remains unopened. Unfortunately, this only works when the parties are in the room together, but does not lead to a protocol that can be run over the Internet!

We refer to the properties sketched above as *hiding* and *binding*: The *hiding property* refers to the receiver's inability to learn the value after the sender has committed, but before he has revealed his commitment. The *binding property* refers to the sender's inability to change the value after committing to it.

If we try to formally define these notions, it turns out that there are two different "flavors" of commitment schemes one can consider: the first ensures that the binding property holds even for an all-powerful sender, but the hiding property "only" holds for a computationally-bounded[1] receiver. (We may also say that binding holds information-

[1] While one choice might be to equate "computationally-bounded" with PPT, for the application to ZK proofs we will need hiding to hold with respect to polynomial-size *circuits* (i.e., we need a non-uniform hardness assumption).

theoretically, while hiding holds only computationally.) Commitment schemes of this sort are called *standard*. The second type of commitment satisfies the hiding property even for an all-powerful receiver, but now the binding property only holds for a computationally-bounded sender. (I.e., this scheme achieves information-theoretic hiding, but only computational binding.) Such commitment schemes are termed *perfect*.[2] The scheme one uses will depend on the application, as we will see.

For now, we define only a standard commitment scheme. In general, a commitment scheme may be interactive, but for simplicity we give a definition only for the case of non-interactive commitment. Here, the sender outputs a pair $(\mathsf{com}, \mathsf{dec})$ consisting of a commitment and a decommitment: sending com to the receiver constitutes the commitment phase and sending dec to the receiver constitutes the decommitment phase.

Definition 1 A *standard commitment scheme* consists of a pair of PPT algorithms $(\mathcal{S}, \mathcal{R})$ satisfying the following:

Correctness For all k and all $b \in \{0,1\}$:

$$\Pr[(\mathsf{com}, \mathsf{dec}) \leftarrow \mathcal{S}(1^k, b) : \mathcal{R}(1^k, \mathsf{com}, \mathsf{dec}) = b] = 1.$$

Binding The following is negligible even for an all-powerful $\mathcal{S}^*$:

$$\Pr[(\mathsf{com}, \mathsf{dec}, \mathsf{dec}') := \mathcal{S}^*(1^k) : \mathcal{R}(1^k, \mathsf{com}, \mathsf{dec}) = 0 \wedge \mathcal{R}(1^k, \mathsf{com}, \mathsf{dec}') = 1].$$

Hiding The following is negligible for any family $\{R_k^*\}$ of polynomial-size circuits (see footnote 1):

$$\left| \Pr[b \leftarrow \{0,1\}; (\mathsf{com}, \mathsf{dec}) \leftarrow \mathcal{S}(1^k, b) : \mathcal{R}_k^*(\mathsf{com}) = b] - \frac{1}{2} \right|.$$

$\diamondsuit$

We remark that it is easy to extend the above definition to *string commitment* rather than just *bit commitment*. Furthermore, it is easy to construct a string commitment scheme from any bit commitment scheme: just commit to the bits of the string one-by-one. Here, the hiding definition may be more easily thought of in terms of an "indistinguishability-type" game as in the case of encryption: an adversary submits two strings m_0, m_1 to a "commitment oracle" which returns a commitment of m_b for random b; the adversary succeeds if it guesses the value of b, and we say the commitment scheme is secure if every poly-size family of circuits succeeds with probability negligibly close to half.

We do not pursue constructions of commitment schemes here, and instead defer that to another lecture. Here, we will instead be more interested in using a commitment scheme (as a black box) to construct a ZK proof system.

[2] In case you are wondering: it is not too difficult to show the *impossibility* of simultaneously achieving information-theoretic binding and hiding. At the other extreme, schemes achieving both computational binding and hiding may be suitable for some applications, but since we can (for the most part) achieve the stronger notions of security anyway, this case is not so interesting.

2 A ZK Proof System for 3-Colorability

The language of 3-colorability is the set of graphs which can be "3-colored"; i.e., graphs for which the colors "read", "blue", and "green" can be assigned to its vertices such that no two adjacent vertices (vertices sharing an edge) have the same color. Formally, a coloring of a graph G can be viewed as function ϕ from the vertices of G to the set $\{r, b, g\}$ such that if (u, v) is an edge in G, then $\phi(u) \neq \phi(v)$. Deciding 3-colorability is known to be an $\mathcal{NP}$-complete problem.

We now show a ZK proof for 3-colorability: At the beginning of the protocol, both the prover and verifier know the same graph G with n vertices, and the prover also knows a 3-coloring ϕ for this graph (we let ϕ_i denote $\phi(i)$; i.e., the color assigned to vertex i).

- First, the prover chooses a random permutation φ over the set $\{r, b, g\}$. He then commits to the (permuted) coloring vertex-by-vertex, and sends the n commitments $\boxed{\varphi(\phi_1)} \cdots \boxed{\varphi(\phi_n)}$.
- The verifier chooses a random edge (i, j) in G, and sends (i, j) to the prover.
- The prover sends decommitments to the i^{th} and j^{th} commitments that it sent in the first round.
- The verifier recovers the decommitted values, denoted φ_i and φ_j. The verifier accepts iff $\varphi_i, \varphi_j \in \{r, b, g\}$ and $\varphi_i \neq \varphi_j$.

Note that the proof system satisfies completeness, since an honest prover using a valid coloring will always cause the verifier to accept. Furthermore, (weak) soundness holds if the commitment scheme is binding. To see this, assume for a moment that the commitments are "perfect" (i.e., sealed envelopes) and let P^* be a cheating prover with G a graph that is not 3-colorable. Then after the first round, P^* is committed to *some* assignment of vertices to colors (we may assume that if a particular commitment is invalid in any way, then we arbitrarily assign it the color "red"). Since the graph is not 3-colorable, there must then be at least one edge (u, v) for which u and v are assigned the same color. So if the verifier chooses this edge, he will reject the proof. The probability that the verifier chooses such an edge is (at least) $1/|E| \geq 1/n^2$, where $|E|$ is the number of edges in G (it is at least this probability because there might be more than one edge whose vertices are not colored correctly). So, the prover fails to convince the verifier with probability at least $1/n^2$, which is inverse polynomial (in the size of the graph). Of course, we need to also take into account the fact that these are not "perfect" commitments; however, the probability that the prover can open any of these commitments in more than one way is negligible (by the binding property) so this decreases the probability that the verifier will accept by only a negligible probability.

As usual, repeating the protocol sufficiently-many (but polynomially-many) times (sequentially if we want to preserve the ZK property[3]) yields a proof system with negligible soundness error.

In the next two sections, we show that this proof system is *zero knowledge*.

[3]We rely here on the fact that the above proof system satisfies the stronger definition of *auxiliary-input zero knowledge*; see Lecture 17 and [1].

2.1 Simulation for an Honest Verifier

First, we informally discuss why the above protocol is *honest-verifier* zero knowledge. (We do not give a formal proof, since one will follow anyway from the stronger result we show in the following section.) Imagine the following simulator, which receives only the graph G (but no coloring); as usual, the simulator "guesses in advance" the challenge of the verifier:

- Choose a random edge (i, j) in G.
- Choose φ_i at random from $\{r, b, g\}$ and φ_j at random from $\{r, b, g\} \setminus \{\varphi_i\}$. For all $k \in \{1, \ldots, n\}$, $k \neq i, j$, set $\varphi_k = r$. Generate commitments $\boxed{\varphi_1} \cdots \boxed{\varphi_n}$ to these n values.
- Output the transcript $\boxed{\varphi_1} \cdots \boxed{\varphi_n}$; (i, j); φ_i, φ_j. (Actually, the last round should include decommitments to φ_i, φ_j.)

Now, note that the "only" difference between the distribution on transcripts output by the simulator, and the distribution on transcripts resulting from a real execution of the protocol are that, in the former, all commitments other than φ_i, φ_j are to "r" while, in the latter, all the commitments are to some valid 3-coloring. However, by the hiding property of the commitment scheme, these two distributions are computationally indistinguishable.[4]

2.2 Simulation for a Dishonest Verifier

We now show, more formally, a simulator for an arbitrary PPT verifier V^*.

$\underline{\mathsf{Sim}(1^k, G)}$
fix random tape ω for V^*
for $i = 1$ to $|E|^2$:
  choose random edge (u, v)
  generate vector of commitments com as in previous section
  run $V^*(\mathsf{com}; \omega)$ to obtain challenge (u^*, v^*)
  if $(u^*, v^*) = (u, v)$ output transcript as in previous section
if all previous iterations have failed, output $\perp$

We now want to claim that, for all G, ϕ, the output distribution defined by Sim is computationally indistinguishable from the distribution over real executions of the protocol. The intuition is exactly as in the case of the ZK proof of graph isomorphism: we want to claim that Sim outputs $\perp$ with only negligible probability, and that conditioned on *not* outputting $\perp$ the transcripts looks "the same". However, two differences arise here which did not arise previously:

1. First, it is not immediately clear that Sim outputs $\perp$ with only negligible probability. To argue this, we would like to claim that in any iteration of the loop the probability that $(u^*, v^*) = (u, v)$ is $1/|E|$ (similar to the case of graph isomorphism). In the

[4] We remark that this is in contrast to the ZK proof system for graph isomorphism, where the simulated transcripts were *perfectly* indistinguishable from real transcripts in the case of HVZK, and *statistically* indistinguishable from real transcripts in the case of ZK.

case of graph isomorphism, however, the view of V^* was independent of the challenge guessed by the simulator; here, this is *no longer true* since the vector of commitments given to V^* *does* reveal the guess of Sim (in an information-theoretic sense). On the other hand, since V^* runs in polynomial-time and the commitments are hiding we can show that this does not make "too much difference"; this requires formal proof.

2. Second, in the case of graph isomorphism the transcripts were identically distributed (conditioned on not outputting $\perp$); here, though, the transcripts will (only) be computationally indistinguishable.

We now give a (sketch of a) formal proof which will (we hope!) provide the interested reader with all the necessary elements to construct a full proof. (The reader is also invited to see [1].) First, consider the following modified "simulation" which is not really a simulation at all since it will use the coloring ϕ used by the real prover.

$\underline{\mathsf{Sim}'(1^k, G, \phi)}$
fix random tape ω for V^*
for $i = 1$ to $|E|^2$:
  choose random edge (u, v)
  Using ϕ, generate a vector of commitments com exactly like the honest prover
  run $V^*(\mathsf{com}; \omega)$ to obtain challenge (u^*, v^*)
  if $(u^*, v^*) = (u, v)$ output transcript as in previous section
if all previous iterations have failed, output $\perp$

We claim that (for all G, ϕ) the output distribution generated by Sim' is *statistically-close* to the distribution of real executions of the protocol. The argument here is *exactly* as in the case of graph isomorphism: note that now the "guess" of the simulator is information-theoretically hidden from V^* (since the vector of commitments is for a *valid* 3-coloring, so there is no way to tell which edge was guessed by Sim') and so the probability of outputting $\perp$ is negligible; furthermore, conditioned on not outputting $\perp$ the distributions are identical. (We stress that the above does *not* constitute a valid simulation, however, since Sim' is given ϕ. Instead, it is just a "mental experiment".)

We next claim that (for all G, ϕ) the output distribution generated by Sim' is computationally indistinguishable from the output distribution generated by Sim. (By a hybrid argument, this shows that the distribution generated by Sim is computationally indistinguishable from the real distribution, and completes the proof.) To see this, assume the contrary. Then there is a poly-time distinguisher D^* that can distinguish between the two distributions with probability that is not negligible. But then we can create a poly-time distinguisher[5] D that violates the hiding property of the commitment scheme as follows (we use the "indistinguishability-based" characterization of the hiding property, as discussed

[5] In fact, the distinguisher we construct will be a poly-size circuit (and this is why we need to commitment scheme to satisfy a non-uniform definition of security) because we will have to incorporate the graph G and the coloring ϕ. Such subtleties are glossed over in this write-up.

earlier):

$\underline{D(1^k, G, \phi)}$
fix random tape ω for V^*
for $i = 1$ to $|E|^2$:
1. choose random edge (u, v)
2. Choose random, different colors φ_u for u and φ_v for v and commit to these
3. for all other vertices, generate two vectors of length $n - 2$:
   one in which every vertex (i.e., other than u, v) is colored red,
   and one in which the vertices are colored using a random permutation φ of ϕ
   subject to $\varphi(u) = \varphi_u$ and $\varphi(v) = \varphi_v$
3. submit these messages to the commitment oracle and get back
   a vector of $n - 2$ commitments
   let com represent these commitments along with
   the commitments to φ_u, φ_v (all in the correct order)
4. run $V^*(\mathsf{com}; \omega)$ to obtain challenge (u^*, v^*)
if $(u^*, v^*) = (u, v)$ output $\langle \mathsf{com}; (u, v); \varphi_u, \varphi_v \rangle$ as the transcript
if all previous iterations have failed, output $\perp$
run D^* on the resulting transcript, and output whatever D^* outputs

The proof concludes by making the following observations: (1) if the commitments returned by the "commitment oracle" are of the first type (where vertices other than u, v are colored red) then the transcript given to D^* is distributed exactly according to the transcripts output by Sim; (2) if the commitments returned by the "commitment oracle" are of the second type (where they form a commitment to a valid 3-coloring) then the transcript given to D^* is distributed exactly according to the transcripts output by Sim'. Thus, (3) if D^* can distinguish between these, then D can distinguish what kind of commitments are being given to it by its oracle. Since D runs in polynomial time, this is a contradiction.

References

[1] O. Goldreich. *Foundation of Cryptography, vol. 1: Basic Tools*, Cambridge University Press, 2001.

CMSC 858K — Advanced Topics in Cryptography	April 13, 2004

Lecture 20

Lecturer: Bill Gasarch	*Scribe(s):*	*Rengarajan Aravamudhan* *Julie Staub* *Nan Wang*

1 Last Lecture Summary

In lecture 18, we showed a technique that can be used to achieve k-database PIR with communication complexity $\mathcal{O}\left(n^{1/(\log k+\log\log k)}\right)$ (we did not prove this bound in class, but the method we showed yields this bound). Today, we show an improved approach due to Ambainis [1] that achieves k-database PIR with $\mathcal{O}(n^{1/2k-1})$ communication complexity.

2 A 3-Database PIR Protocol with $\mathcal{O}(n^{1/5})$ Communication

We start by illustrating the technique for the case of 3 databases, using the notation as in lecture 18 (in fact, this scheme builds on the previous schemes, and we assume the reader is familiar with lecture 18). Here, we model the n-bit database as an $n^{1/5} \times n^{1/5} \times n^{1/5} \times n^{1/5} \times n^{1/5}$ table $\{x_{i_1,i_2,i_3,i_4,i_5}\}$, with the desired bit indexed as $(i_1^*, i_2^*, i_3^*, i_4^*, i_5^*)$. The three databases are DB_1, DB_2, and DB_3. We first discuss the intuition, and then describe the actual protocol.

As in the previous schemes we have seen, the user begins by choosing five random sets $S_1^0, S_2^0, S_3^0, S_4^0, S_5^0 \subseteq \{1, 2, ..., n^{1/5}\}$, and computing $S_1^1 = S_1^0 \oplus \{i_1^*\}$, ..., $S_5^1 = S_5 \oplus \{i_5^*\}$. The user will send $S_1^0, S_2^0, S_3^0, S_4^0, S_5^0$ to DB_1 and DB_2, and send $S_1^1, S_2^1, S_3^1, S_4^1, S_5^1$ to DB_3. Let

$$X_{b_1b_2b_3b_4b_5} \stackrel{\text{def}}{=} \bigoplus_{i_1 \in S_1^{b_1}} \bigoplus_{i_2 \in S_2^{b_2}} \bigoplus_{i_3 \in S_3^{b_3}} \bigoplus_{i_4 \in S_4^{b_4}} \bigoplus_{i_5 \in S_5^{b_5}} x_{i_1,i_2,i_3,i_4,i_5}.$$

Recall (from the schemes we have seen in lecture 18) that the user would like to obtain all 32 of the values $X_{00000}, \ldots, X_{11111}$, and then xor these together to recover the desired bit. Now, DB_1 (and DB_2) can easily compute X_{00000}, while DB_3 can easily compute X_{11111}. Furthermore, since the user is already sending $O(n^{1/5})$ communication to the databases we may as well let the databases send this much communication back. We saw (in the improved scheme from lecture 18) that this can help because we may then have, for example, DB_3 compute X_{11110} *for all* $n^{1/5}$ *possible values of* S_5^0, and then send these $n^{1/5}$ bits back to the user (who selects and uses the one he is interested in). Adopting this approach, we see that DB_3 can send back (enough information for the user to compute) $X_{11110}, X_{11101}, X_{11011}, X_{10111}$, and X_{01111}. Similarly, either of DB_1 or DB_2 can send back (enough information for the user to compute) $X_{00001}, X_{00010}, X_{00100}, X_{01000}$, and X_{10000}. This can be all done while maintaining communication complexity $\mathcal{O}(n^{1/5})$.

Unfortunately, we are not yet done because the user will still be missing 20 of the values he needs to recover the desired bit. Note that we cannot extend the above approach in

the trivial way to allow the user to recover the necessary values without exceeding $\mathcal{O}(n^{1/5})$ communication complexity: if we have DB_1 send back X_{11000} for all possible values of S_1^1 *and* S_2^1, this will require DB_1 to send $n^{2/5}$ bits.

Instead, what we do is to apply a 2-database PIR protocol as a subroutine. Namely, *both* DB_1 and DB_2 will compute each of the remaining values for all possibilities of each of the unknown sets. (For example, DB_1 and DB_2 will compute X_{11100} for all possible values of S_1^1, S_2^1, and S_3^1.) This results in each of these databases holding the same copy of 20 strings, each of length at most $n^{3/5}$. The key point is that *the user only needs one bit from each of these strings* (i.e., the bit corresponding to the actual value of $S_1^1, \ldots, S_5^1$) and this bit is known by the user in step 1. Since DB_1 and DB_2 hold identical copies of these strings, they and the user can use multiple invocations of a 2-database PIR protocol with communication complexity $\mathcal{O}(N^{1/3})$ (we saw such a scheme last time) to allow the user to obtain the desired bits from each of these strings. Since $N \leq n^{3/5}$ in our case, this will require communication complexity $\mathcal{O}(n^{1/5})$ meaning that the overall communication complexity of the entire protocol remains $\mathcal{O}(n^{1/5})$.

Using this intuition, we obtain the following protocol:

1. The user begins by choosing five random sets $S_1^0, S_2^0, S_3^0, S_4^0, S_5^0 \subseteq \{1, 2, ..., n^{1/5}\}$, and computing $S_1^1 = S_1^0 \oplus \{i_1^*\}$, $\ldots$, $S_5^1 = S_5 \oplus \{i_5^*\}$. The user sends $S_1^0, S_2^0, S_3^0, S_4^0, S_5^0$ to DB_1 and DB_2, and sends $S_1^1, S_2^1, S_3^1, S_4^1, S_5^1$ to DB_3.

2. The user also sends to DB_1 and DB_2 20 queries for any 2-database PIR protocol with $\mathcal{O}(n^{1/3})$ comm. complexity. These queries are determined based on the sets that the user generated in the previous step.

3. DB_1 and DB_3 send back X_{00000} and X_{11111}, respectively. Also, DB_1 sends back $X_{00001}, \ldots, X_{10000}$ for all $n^{1/5}$ possible values for each of $S_1^1, \ldots, S_5^1$. Similarly, DB_3 sends $X_{11110}, \ldots, X_{01111}$ for all $n^{1/5}$ possible values for each of $S_1^0, \ldots, S_5^0$.

4. (We use X_{11100} as an example, but exactly the same computation is carried out for each of the remaining values.) DB_1 and DB_2 both generate X_{11100} for all $n^{3/5}$ possibilities of S_1^1, S_2^1, S_3^1. This results in each of these databases holding identical copies of a string of length $n^{3/5}$. Using the appropriate query that was sent by the user, each database computes a response using the underlying 2-database PIR protocol.

5. The user obtains X_{00000} and X_{11111} immediately, and can easily select the values for $X_{00001}, \ldots, X_{10000}$ and $X_{11110}, \ldots, X_{01111}$ from the data sent back by the databases. For the remaining values, the user runs the underlying PIR protocol using the appropriate replies sent back by DB_1 and DB_2 to recover all remaining values. The desired bit of the original data is recovered as:

$$\bigoplus_{b_1=0}^{1} \bigoplus_{b_2=0}^{1} \bigoplus_{b_3=0}^{1} \bigoplus_{b_4=0}^{1} \bigoplus_{b_5=0}^{1} X_{b_1 b_2 b_3 b_4 b_5}.$$

3 Extending the Scheme for k Databases

We generalize the scheme of the previous section and prove the following theorem:

Theorem 1 *For all $k \geq 2$, there exists a k-database PIR scheme with $\mathcal{O}(n^{1/2k-1})$ communication complexity.*

Proof We will prove this by induction. For $k = 2, 3$, we have already shown that the theorem holds. So, assume the theorem is true for $k-1$, and there exists a $(k-1)$-database PIR scheme with $\mathcal{O}(n^{1/2k-3})$ communication complexity. We show how to construct a k-database scheme as claimed by the theorem.

We view the n-bit database as an $n^{1/2k-1} \times n^{1/2k-1} \times \cdots \times n^{1/2k-1}$ array with the desired bit indexed as $(i_1^*, i_2^*, \ldots, i_{2k-1}^*)$. We use the notation from the previous section. The protocol is defined as follows:

1. The user chooses $2k-1$ random sets $S_1^0, \ldots, S_{2k-1}^0 \subseteq \{1, \ldots, n^{1/(2k-1)}\}$, and computes $S_\ell^1 = S_\ell^0 \oplus \{i_\ell^*\}$. The user sends $S_1^0, \ldots, S_{2k-1}^0$ to each of $DB_1, \ldots, DB_{k-1}$ and sends $S_1^1, \ldots, S_{2k-1}^1$ to DB_k.

2. DB_1 will compute and send $X_{0^{2k-1}}$ as well as the $n^{1/(2k-1)}$ possibilities for each of $\{X_{0^i 1 0^{2k-2-i}}\}_{i=0}^{2k-2}$. Similarly, DB_k will compute and send $X_{1^{2k-1}}$ as well as the $n^{1/(2k-1)}$ possibilities for each of $\{X_{1^i 0 1^{2k-2-i}}\}_{i=0}^{2k-2}$.

3. In addition, each of the databases $DB_1, \ldots, DB_{k-1}$ will compute all possible values for X_w for all $(2k-1)$-bit strings w having at least 2 and at most $2k-3$ ones. Instead of sending these directly to the user, the databases will execute the $(k-1)$-database PIR protocol (that exists by assumption) on these strings, using queries sent by the user in the first stage.

The total communication complexity can be computed as follows:

- Sending the sets from the user to all k databases requires $k \cdot (2k-1) \cdot n^{1/(2k-1)}$ bits.
- Steps 2 and 3 each require only $(2k-2) \cdot n^{1/(2k-1)}$ bits to be sent by databases DB_1 and DB_k.
- Step 4 involves running a $(k-1)$-database PIR protocol on fewer than 2^{2k-1} strings, each of length at most $n^{(2k-3)/(2k-1)}$. Since the underlying PIR protocol has communication complexity $\mathcal{O}(N^{1/(2k-3)})$, this requires total communication (including the communication from the user in the first round) $\mathcal{O}(n^{1/(2k-1)})$.

The total communication complexity is therefore $\mathcal{O}(n^{1/(2k-1)})$, as desired. ■

When including the dependence on k, the communication complexity is $\mathcal{O}(2^{k^2} n^{1/(2k-1)})$. This is quite high even for moderate values of k! However, k-database PIR schemes with communication complexity $\mathcal{O}(k^3 n^{1/2k-1})$ and $\mathcal{O}(n^{(\log\log k)/(k \log k)})$ (ignoring the constant which depends on k) are known. See http://www.cs.umd.edu/~gasarch/pir.

References

[1] A. Ambainis. Upper Bound on the Communication Complexity of Private Information Retrieval. ICALP, 1997.

CMSC 858K — Advanced Topics in Cryptography		April 15, 2004
Lecture 21		
Lecturer: Jonathan Katz	*Scribe(s):*	*Omer Horvitz* *Zhongchao Yu* *John Trafton* *Akhil Gupta*

1 Introduction

In a previous lecture, we saw how to construct a three-round *zero-knowledge* (ZK) proof system for *graph 3-colorability* with soundness error $1 - 1/|E|$ on a common input $G = (V, E)$. The soundness error can be made negligible, while maintaining zero knowledge, by repeating the protocol $|E| \cdot \omega(\log k)$ times sequentially (where k is the security parameter); unfortunately, this increases the round complexity of the protocol tremendously and in particular does not result in a constant-round protocol. On the other hand, repeating the protocol many times in parallel is not known to result in a zero-knowledge protocol (i.e., we do not know how to show a simulator for the resulting protocol). The resulting protocol, however, can be shown to satisfy honest-verifier zero knowledge, a weaker variant of zero knowledge.

In this lecture, we consider another weakening of the zero-knowledge property known as *witness indistinguishability* [1]. This notion is useful in its own right, and also leads to constructions of constant-round ZK proof systems (with negligible soundness error) as we will see in a later lecture.

2 Witness Indistinguishability

In general, an $\mathcal{NP}$ statement may have multiple witnesses. For example, a Hamiltonian graph may have multiple Hamiltonian cycles; a 3-colorable graph may have multiple (non-isomorphic) 3-colorings; etc. We are interested in proof systems (for languages in $\mathcal{NP}$) that do not leak information about which witness the prover is using, even to a malicious verifier. In the following, we let $\langle A(y), B(z)\rangle(x)$ denote the view (i.e., inputs, internal coin tosses, incoming messages) of B when interacting with A on common input x, when A has auxiliary input y and B has auxiliary input z.

Definition 1 Let $L \in \mathcal{NP}$ and let $(\mathcal{P}, \mathcal{V})$ be an interactive proof system for L with perfect completeness. We say that $(\mathcal{P}, \mathcal{V})$ is *witness-indistinguishable* (WI) if for every PPT algorithm $\mathcal{V}^*$ and every two sequences $\{w_x^1\}_{x\in L}$ and $\{w_x^2\}_{x\in L}$ such that w_x^1 and w_x^2 are both witnesses for x, the following ensembles are computationally indistinguishable:

1. $\left\{\left\langle \mathcal{P}(w_x^1), \mathcal{V}^*(z)\right\rangle(x)\right\}_{x\in L, z\in\{0,1\}^*}$
2. $\left\{\left\langle \mathcal{P}(w_x^2), \mathcal{V}^*(z)\right\rangle(x)\right\}_{x\in L, z\in\{0,1\}^*}$.

(Note: when the security parameter is not written explicitly, we simply take $|x| = k$.) In particular, we may have $z = (w_x^1, w_x^2)$. ♢

Remark WI is defined with respect to *auxiliary-input* verifiers (where the auxiliary input can include the witnesses which the prover might use). Although we have not done so before, zero knowledge can also be defined with respect to auxiliary-input verifiers (this is called *auxiliary-input zero knowledge*). In this lecture, when referring to a ZK proof system we will mean auxiliary-input zero knowledge by default.

Witness indistinguishability is clearly a weaker notion than zero-knowledge, and in particular there exists protocols which are trivially WI and definitely *not* zero-knowledge. For example, whenever there is only a *single* witness for a particular statement (e.g., a single Hamiltonian cycle in a graph) then a protocol in which the prover sends the witness to the verifier is WI but not (in general) ZK! To flesh this example out a bit, assume the existence of a length-preserving one-way permutation f and define the language $L_0 \stackrel{\text{def}}{=} \{y \mid \text{the first bit of } f^{-1}(y) \text{ is } 0\}$. (Note that every $y \in L_0$ has the unique witness $x = f^{-1}(y)$.) An interactive proof in which the prover sends $f^{-1}(y)$ on common input y is witness indistinguishable (trivially) but not zero knowledge (assuming f is indeed one-way).

On the other hand, one can show that zero knowledge is strictly stronger than witness indistinguishability:

Theorem 1 (ZK implies WI) *If an interactive proof system $(\mathcal{P}, \mathcal{V})$ for a language L is zero-knowledge, then it is also witness indistinguishable.*

Proof (Sketch) Let $\mathcal{V}^*$ be a PPT algorithm, and let Sim be the simulator guaranteed by the zero-knowledge property of $(\mathcal{P}, \mathcal{V})$. Then for any $x \in L$ and any witnesses w_1, w_2 for x, we have that:

- $\{\mathsf{Sim}(x,z)\}_{x\in L, z\in\{0,1\}^*} \stackrel{c}{\approx} \{\langle \mathcal{P}(w_1), \mathcal{V}^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*}$
  and
- $\{\mathsf{Sim}(x,z)\}_{x\in L, z\in\{0,1\}^*} \stackrel{c}{\approx} \{\langle \mathcal{P}(w_2), \mathcal{V}^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*}$.

By the transitivity of computational indistinguishability, it follows that:

$$\{\langle \mathcal{P}(w_1), \mathcal{V}^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*} \stackrel{c}{\approx} \{\langle \mathcal{P}(w_2), \mathcal{V}^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*},$$

as required. □

Next, we show that — in contrast to zero knowledge — witness indistinguishability is preserved under parallel composition (i.e., parallel repetition of the protocol). Before doing so, we first formally define parallel composition and, in the process, establish some notation.

Definition 2 Let $(\mathcal{P}, \mathcal{V})$ be an interactive proof system for a language L, and let $\ell = \ell(k)$. We define the ℓ-fold parallel composition of $(\mathcal{P}, \mathcal{V})$, denoted $(\mathcal{P}_\ell^{\|}, \mathcal{V}_\ell^{\|})$, as the protocol obtained by running ℓ *independent* executions of $(\mathcal{P}, \mathcal{V})$ in parallel. Namely:

- $\mathcal{P}_\ell^{\|}$, on input $1^k, x, w$ with $x \in L$ and w a witness for x, generates $\ell = \ell(k)$ independent random tapes $\omega_1, \ldots, \omega_\ell$. It then runs $\mathcal{P}(1^k, x, w; \omega_1), \ldots, \mathcal{P}(1^k, x, w; \omega_\ell)$ to generate $\vec{m}_1 \stackrel{\text{def}}{=} m_1^1, \ldots, m_1^\ell$. It sends $\vec{m}_1$ to the verifier as its first message.

- $\mathcal{V}_\ell^{\|}$, on (common) input x, chooses ℓ independent random tapes $\omega_1', \ldots, \omega_\ell'$. After receiving message $\vec{m}_1 = m_1^1, \ldots, m_1^\ell$ from the prover, it runs $\mathcal{V}(1^k, x, m_1^1; \omega_1')$, $\ldots$, $\mathcal{V}(1^k, x, m_1^\ell; \omega_\ell')$ to generate $\vec{m}_2 \stackrel{\text{def}}{=} m_2^1, \ldots, m_2^\ell$. It sends $\vec{m}_2$ to the prover.
- The execution continues in this way. It is stressed that each of the ℓ executions is "oblivious" to the other $\ell - 1$ executions, and all random coins are chosen independently.
- At the end of the protocol, $\mathcal{V}_\ell^{\|}$ accepts only if all ℓ invocations of $\mathcal{V}$ have accepted.

$\diamondsuit$

It is not difficult to see that perfect completeness is preserved under parallel composition, and that if $(\mathcal{P}, \mathcal{V})$ has soundness error $\varepsilon(k)$ (against an all-powerful prover — i.e., this is a *proof* system) then $(\mathcal{P}_\ell^{\|}, \mathcal{V}_\ell^{\|})$ has soundness error $\varepsilon(k)^{\ell(k)}$.[1] We have also mentioned already that zero knowledge is *not* necessarily preserved under parallel composition. However:

Theorem 2 (WI is preserved under parallel composition) *For any polynomial $\ell(\cdot)$, if $(\mathcal{P}, \mathcal{V})$ is witness indistinguishable then so is $(\mathcal{P}_\ell^{\|}, \mathcal{V}_\ell^{\|})$.*

Proof At its core, the proof is via a hybrid argument but some things are slightly more subtle here because we are dealing with an interactive process (also, contrary to intuition(?), ZK is not preserved under parallel composition and so we must check the details and not rely on our intuition).

We prove the theorem for $\ell = 2$ (and write $\mathcal{P}^{\|}$ instead of $\mathcal{P}_2^{\|}$); the proof extends for any polynomial ℓ. Assume to the contrary that there exists a PPT algorithm $\mathcal{V}^*$, an infinite sequence $\{(x_i, w_i^1, w_i^2, z_i\}_{i \in \mathbb{N}}$, and a PPT distinguisher D such that the following is not negligible:

$$\Big| \Pr\left[\mathsf{view} \leftarrow \left\langle \mathcal{P}^{\|}(w_k^1), \mathcal{V}^*(z_k) \right\rangle (x_k) : D(1^k, \mathsf{view}) = 1 \right] - \Pr\left[\mathsf{view} \leftarrow \left\langle \mathcal{P}^{\|}(w_k^2), \mathcal{V}^*(z_k) \right\rangle (x_k) : D(1^k, \mathsf{view}) = 1 \right] \Big| .$$

Let $\hat{\mathcal{P}}^{\|}(w, w')$ denote a prover who runs two parallel executions of $\mathcal{P}$ but uses witness w in the first execution and w' in the second (we assume these are both witnesses for the same x). Using this notation, $\hat{\mathcal{P}}^{\|}(w, w) = \mathcal{P}^{\|}(w)$. Thus, a standard hybrid argument shows that at least one of

$$\Big| \Pr\left[\mathsf{view} \leftarrow \left\langle \hat{\mathcal{P}}^{\|}(w_k^1, w_k^1), \mathcal{V}^*(z_k) \right\rangle (x_k) : D(1^k, \mathsf{view}) = 1 \right] - \Pr\left[\mathsf{view} \leftarrow \left\langle \hat{\mathcal{P}}^{\|}(w_k^1, w_k^2), \mathcal{V}^*(z_k) \right\rangle (x_k) : D(1^k, \mathsf{view}) = 1 \right] \Big|$$

or

$$\Big| \Pr\left[\mathsf{view} \leftarrow \left\langle \hat{\mathcal{P}}^{\|}(w_k^1, w_k^2), \mathcal{V}^*(z_k) \right\rangle (x_k) : D(1^k, \mathsf{view}) = 1 \right] - \Pr\left[\mathsf{view} \leftarrow \left\langle \hat{\mathcal{P}}^{\|}(w_k^2, w_k^2), \mathcal{V}^*(z_k) \right\rangle (x_k) : D(1^k, \mathsf{view}) = 1 \right] \Big|$$

[1] Interestingly, this is not necessarily true when soundness is defined for computationally-bounded provers (i.e., an *argument* system, which will be defined in a later lecture).

is not negligible. Without loss of generality, assume it is the former. We show that this contradicts the witness indistinguishability of the original proof system $(\mathcal{P}, \mathcal{V})$.

Construct the following PPT verifier $\mathcal{V}^{**}$ who gets auxiliary input (z_k, w_k^1) and interacts with $\mathcal{P}$:

$\underline{\mathcal{V}^{**}(1^k, x, z_k, w_k^1)}$
run $\mathcal{V}^*(1^k, x, z_k)$ as follows:
 for the second of the two parallel executions of $\mathcal{P}$ that $\mathcal{V}^*$ expects to see
  forward the appropriate messages to and from the *external* prover $\mathcal{P}$
 for the first of the two parallel executions
  run $\mathcal{P}$ *internally*, using the witness w_k^1
when done, output the view of $\mathcal{V}^*$ (which is easy to reconstruct)

Note that

$$\left\langle \mathcal{P}(w_k^1), \mathcal{V}^{**}(z_k, w_k^1) \right\rangle (x_k) \equiv \left\langle \hat{\mathcal{P}}^{\|}(w_k^1, w_k^1), \mathcal{V}^*(z_k) \right\rangle (x_k)$$

(that is, the distributions are identical), and

$$\left\langle \mathcal{P}(w_k^2), \mathcal{V}^{**}(z_k, w_k^1) \right\rangle (x_k) \equiv \left\langle \hat{\mathcal{P}}^{\|}(w_k^1, w_k^2), \mathcal{V}^*(z_k) \right\rangle (x_k).$$

But then D distinguishes between the two distributions on the left-hand sides of the equations above, contradicting the witness indistinguishability of $(\mathcal{P}, \mathcal{V})$. ■

Recall the three-round, zero-knowledge protocol for graph 3-colorability from a previous lecture. Theorem 1 implies that this protocol is witness indistinguishable, and Theorem 2 implies that its witness indistinguishability is preserved under parallel composition. We thus obtain:

Corollary 3 *There exists a three-round, witness indistinguishable proof system with negligible soundness error for the NP-complete language graph 3-colorability, and hence for any language in NP.*

We will see applications of witness indistinguishability, both on its own and also as a tool to construct zero-knowledge protocols, in the upcoming lectures.

3 Commitment Schemes

As mentioned previously, two "flavors" of commitment schemes can be considered:

- *Perfect* (or *computationally binding*) *commitments*, protecting against all-powerful receivers and polynomial-time senders.
- *Standard* (or *computationally hiding*) *commitments*, protecting against polynomial-time receivers and all-powerful senders.

In the basic, three-round proof system for graph 3-colorability, we need soundness to hold even for all-powerful provers. Thus, we need the prover to use a standard commitment scheme to commit to the coloring of the graph in the first round.

We now show an easy construction of a standard commitment scheme: Let $(\mathsf{Gen}, \mathcal{E}, \mathcal{D})$ be a public-key encryption scheme with perfect decryption (i.e., a decryption error never occurs). Consider the following, two-phase protocol between a sender $\mathcal{S}$ on input $b \in \{0,1\}$ and a receiver $\mathcal{R}$. In the first phase, $\mathcal{S}$ runs $(pk, sk) \leftarrow \mathsf{Gen}(1^k; r_{\mathsf{Gen}})$, computes $c \leftarrow \mathcal{E}_{pk}(b; r_{\mathcal{E}})$ and sends (pk, c) to $\mathcal{R}$. In the second phase, $\mathcal{S}$ sends $(b, r_{\mathsf{Gen}}, r_{\mathcal{E}})$ to the receiver.

Claim 4 *The above protocol constitutes a standard commitment scheme.*

Proof (Sketch) Hiding follows directly from the hiding property of the encryption scheme (if the sender is honest, then r_{Gen} and $r_{\mathcal{E}}$ are chosen uniformly at random, and hence b is hidden; a formal proof is left to the reader). Perfect binding follows directly from the perfect decryption property of the encryption scheme. It is essential here that we force the sender to send all its randomness in the second round, since otherwise it may be possible for the sender to "cheat" (for example, to send an invalid public key that could not possibly be output by Gen). □

Public-key encryption schemes with perfect decryption can be based on trapdoor permutations, and so standard commitments can be based on that assumption. Specifically, for a trapdoor-permutation generator Gen, the sender commits to a bit b by computing $(f, f^{-1}) \leftarrow \mathsf{Gen}(1^k; r_{\mathsf{Gen}})$, selecting $r \leftarrow \{0,1\}^n$, computing $y = f(r)$, and sending $(f, y, h(r) \oplus b)$ to $\mathcal{R}$, where h is hard-core for f. The sender decommits by sending (r_{Gen}, r).

In the next lecture, we will see how to construct standard commitment schemes from the much weaker assumption of the existence of one-way functions.

References

[1] U. Feige and A. Shamir. Witness Indistinguishable and Witness Hiding Protocols. In *22nd ACM Symposium on Theory of Computing*, pages 416–426, 1990.

CMSC 858K — Advanced Topics in Cryptography	April 20, 2004
Lecture 22	
Lecturer: Jonathan Katz	*Scribe(s): Nagaraj Anthapadmanabhan, Ji Sun Shin*

1 Introduction to These Notes

In the previous lectures, we saw the construction of a zero-knowledge proof system for the language of 3-colorable graphs (and hence for all of NP) using a *commitment scheme* as a tool. In this lecture we will discuss the two different notions of commitment schemes and show how these types of commitment schemes can be constructed.

Recall that a commitment scheme consists of two phases: a *commitment phase* after which the sender is supposed to be committed to a value, and a *decommitment phase* during which the committed value is revealed to the receiver. Two types of commitment schemes can be defined, one protecting against an all-powerful *sender* and one protecting against an all-powerful *receiver*:

- A **standard commitment scheme** protects against a PPT receiver and an all-powerful sender. In particular, a PPT receiver cannot determine any information about the value committed to by the sender (this can be defined in a way similar to indistinguishability for encryption) in the commitment phase, and an all-powerful sender is bound to only one value (with all but negligible probability) following the commitment phase. We say such a scheme is *computationally hiding* and *information-theoretically binding*.
- A **perfect commitment scheme** protects against a PPT sender and an all-powerful receiver. Namely, even an all-powerful receiver cannot determine any information about the value committed to by the sender (except possibly with negligible probability) in the commitment phase, and a PPT sender is bound to only one value following the commitment phase. We say it is *computationally binding* and *information-theoretically hiding*.

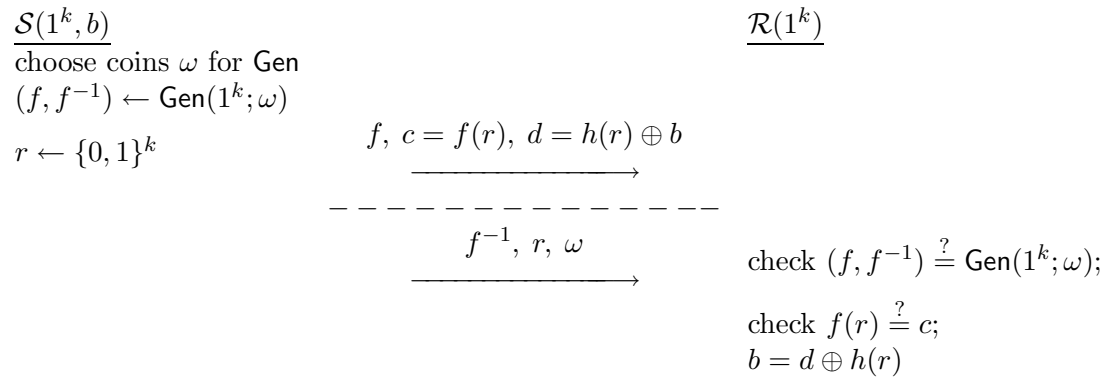
It can be proved that commitment schemes which are both information-theoretically hiding and information-theoretically binding do not exist.

2 Standard Commitment Schemes

We first consider some constructions of standard commitment schemes.

2.1 Constructions Based on One-Way (Trapdoor) Permutations

One possible construction is obtained by using a *trapdoor permutation family*. Here we show how this can be used to allow a sender $\mathcal{S}$ to commit to a bit b:



To commit to a bit b, the sender $\mathcal{S}$ generates a trapdoor permutation (f, f^{-1}), chooses a random element r in the domain of f (here, the domain of f is assumed to be $\{0,1\}^k$), and sends a description of f along with $f(r)$ and $h(r) \oplus b$ (here, $h(\cdot)$ is a hard-core bit of f). To decommit, $\mathcal{S}$ sends the random string ω (used to prove that f is indeed a permutation), as well as r. The receiver $\mathcal{R}$ does the verification in the obvious manner and recovers b.

The *correctness* property is obviously satisfied as the receiver $\mathcal{R}$ "accepts" if $\mathcal{S}$ acts honestly. (This will be the case for all constructions in this lecture.) We now sketch why the other two properties hold:

Binding: We need to prove binding even for an all-powerful sender. Notice that the f sent in the first round must be a permutation, since the sender has to reveal the randomness ω used to generate f in the decommitment phase (and we assume that Gen always outputs a permutation). Thus, c uniquely determines a value $r = f^{-1}(c)$, and this in turn uniquely determines $b = h(r) \oplus d$.

Hiding: Since f is one-way, a PPT receiver cannot distinguish $h(r)$ from a random bit (we use here the fact that (f, f^{-1}) are randomly-generated, and that r is chosen uniformly at random). Thus, $h(r)$ computationally hides the value of b. A formal proof is left as an exercise for the reader.

The reader may notice that we never use the trapdoor f^{-1} in the above construction. Thus, we may in fact base the above construction on the (potentially) weaker assumption of a one-way (*not* necessarily trapdoor) permutation.

2.2 A Construction Based on a One-Way Function

Actually, we can do even better and show a construction based on the minimal assumption of a one-way function (the proof that this assumption is indeed minimal is left as another exercise!). Here, we use the following "hard" theorem that was mentioned, but not proved, in class:

Theorem 1 *The existence of a one-way function implies the existence of a length-tripling pseudorandom generator (PRG).*

Recall the definition of a (length-tripling) PRG:

Definition 1 $G = \{G_k : \{0,1\}^k \to \{0,1\}^{3k}\}_{k\in\mathbb{N}}$ is a *pseudorandom generator (PRG)* if the following is negligible for any PPT distinguisher D:

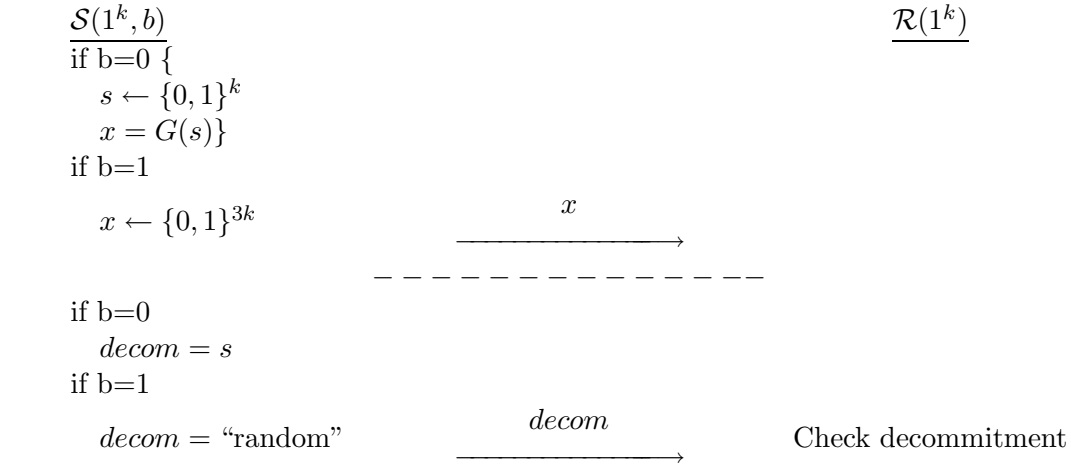
$$\left|\Pr\left[x \leftarrow \{0,1\}^{3k} : D(x) = 1\right] - \Pr\left[s \leftarrow \{0,1\}^k; x = G_k(s) : D(x) = 1\right]\right|. \tag{1}$$

In other words,

$$U_{3k} \approx G(U_k) \tag{2}$$

where U_k denotes the uniform distribution on k-bit strings. ♢

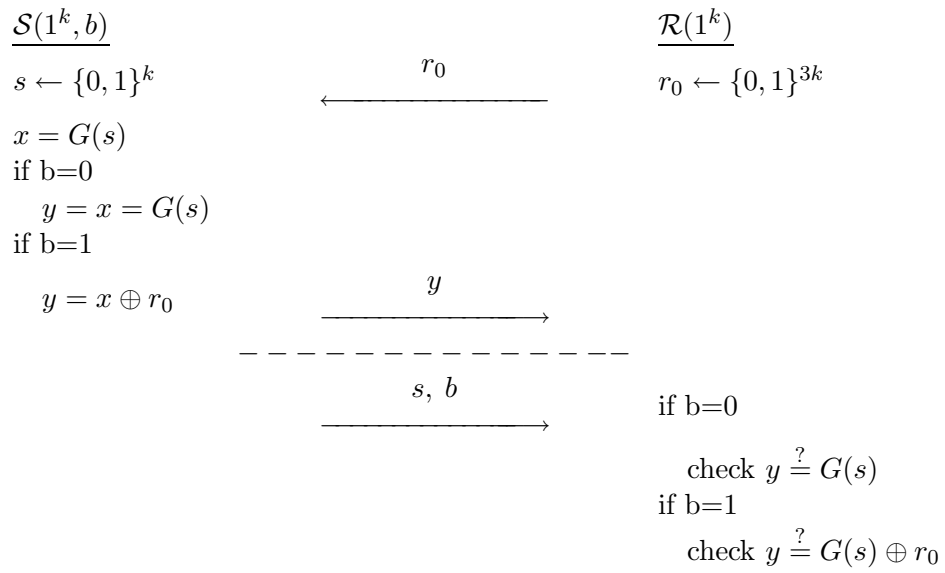
We would like to use a PRG to construct a standard commitment scheme. We will first see a naïve idea that does not work.



I.e., $\mathcal{S}$ sends a pseudorandom string to commit to "0", and a random string to commit to "1". To decommit, $\mathcal{S}$ sends the seed if x was pseudorandom (i.e., $b = 0$) or just says that x is "random" otherwise (i.e., $b = 1$).

It is easy to see that the above scheme satisfies hiding: this follows readily from the definition of a PRG. Unfortunately, the scheme is not binding, even for a polynomial-time sender. This is because a cheating $\mathcal{S}^*$ can send a pseudorandom string $x = G(s)$ and later decommit this to either a 0 (by sending s) or a 1 (by claiming that x is a random string). Note that the properties of the PRG imply that a PPT $\mathcal{R}$ cannot even tell that $\mathcal{S}^*$ is cheating!

Informally, the problem arises from the fact that the sender is not required to use any "secret" when committing to $b = 1$. So, we fix this by making the sender use a secret for either value of b. The following construction is due to Naor [1]:



The receiver first sends a random $3k$-bit string r_0. The sender chooses a random seed s and computes $x = G(s)$. The sender simply sends x to commit to "0", and sends $x \oplus r_0$ to commit to "1". To decommit, $\mathcal{S}$ sends both s and the committed value (and the receiver verifies these in the obvious way). Clearly, this construction satisfies the correctness property. We claim that this construction also satisfies the hiding and binding requirements:

Hiding: The following claim proves the hiding property:

Claim 2 *If a* PPT *cheating receiver $\mathcal{R}^*$ can distinguish a commitment to 0 from a commitment to 1 (before the decommitment phase), then we can use $\mathcal{R}^*$ to "break" the PRG.*

Proof Let $\mathcal{R}^*$ be a PPT receiver that distinguishes a commitment to 0 from a commitment to 1 with probability $\varepsilon(k)$ (namely, the difference between the probability that $\mathcal{R}^*$ outputs 1 when interacting with a sender committing to a "0" and the probability that $\mathcal{R}^*$ outputs 1 when interacting with a sender committing to a "1" is $\varepsilon(k)$). We want to show that $\varepsilon(k)$ is negligible. To that end, use $\mathcal{R}^*$ to construct a PPT distinguisher D trying to distinguish whether its input x is random or pseudorandom:

$D(x)$
$b \leftarrow \{0,1\}$
run the commitment phase of the above scheme with $\mathcal{R}^*$ using bit b; i.e.:
  get r_0 from $\mathcal{R}^*$
  if $b = 0$ send x to $\mathcal{R}^*$
  if $b = 1$ send $x \oplus r_0$ to $\mathcal{R}^*$
if $\mathcal{R}^*$ guesses b correctly, then D outputs 1 (i.e., "PRG")
otherwise, D outputs 0 (i.e., "random"

Note that when x is pseudorandom, D acts as a completely honest sender and so the probability that D outputs 1 is given by:

$$\frac{1}{2} \cdot (\Pr[\mathcal{R}^* \text{ outputs } 0 \mid \mathcal{S} \text{ commits to } 0] + \Pr[\mathcal{R}^* \text{ outputs } 1 \mid \mathcal{S} \text{ commits to } 1])$$

$$= \frac{1}{2} \cdot (1 - \Pr[\mathcal{R}^* \text{ outputs } 1 \mid \mathcal{S} \text{ commits to } 0] + \Pr[\mathcal{R}^* \text{ outputs } 1 \mid \mathcal{S} \text{ commits to } 1])$$
$$= 1/2 + \varepsilon(k)/2.$$

On the other hand, when x is random then the view of $\mathcal{R}^*$ is independent of b (since either one of x or $x \oplus r_0$, for any r_0, is just a uniformly-distributed string) and so the probability that D outputs 1 is exactly 1/2. The security of the PRG thus implies that $1/2 + \varepsilon(k)/2 - 1/2 = \varepsilon(k)/2$ is negligible, completing the proof. ■

Binding: Proving the binding property is more interesting. We show that an all-powerful $\mathcal{S}^*$ can decommit to two different values with only negligible probability (assuming $\mathcal{R}$ is honest). Note that $\mathcal{S}^*$ can only potentially cheat if $\mathcal{R}$ sends an r_0 for which there exist y, s, s' such that: $y = G(s)$ and $r_0 = y \oplus G(s')$ or, in other words, if there exist s, s' such that $G(s) \oplus G(s') = r_0$. Call r_0 with this property "bad". Because s and s' are k-bit strings, there are at most 2^{2k} possible "bad" values r_0. Since r_0 is a uniformly-random $3k$-bit string, the probability of $\mathcal{R}$ choosing a "bad" r_0 is at most $2^{2k}/2^{3k} = 2^{-k}$, which is negligible.

3 Perfect Commitment Schemes

Perfect commitment schemes seem much more difficult to construct. In particular, a construction of a perfect commitment scheme based on one-way *permutations* is known, but it is an open question to construct such a scheme based on one-way functions. Here, we show two constructions based on number-theoretic assumptions.

3.1 A Construction Based on the Discrete Logarithm Assumption

The following scheme allows a sender to commit to a string $x \in \{0,1\}^{k-1}$, not just a bit:

$\underline{\mathcal{S}(1^k, x)}$		$\underline{\mathcal{R}(1^k)}$
		generate a cyclic group G of prime order q with $\lvert q\rvert = k$
	$\xleftarrow{g,\ h,\ G,\ q}$	pick random generators $g, h \in G$
verify q prime; $\lvert q\rvert = k$ verify g, h generators of G view x as a value in $\mathbb{Z}_q$ $r \leftarrow \mathbb{Z}_q$		
$c = g^x h^r$	$\xrightarrow{c}$	
	$- - - - - - - - - - - -$	
	$\xrightarrow{x,\ r}$	verify $c \stackrel{?}{=} g^x h^r$

Note that in a cyclic group of prime order, every element except the identity is a generator.

We now show that the above constitutes a perfect commitment scheme:

Hiding: We claim that c is a uniformly-distributed group element, independent of the committed value x. To see this, note that the sender verifies that q is prime and that g, h are generators of G (also, $\mathcal{S}$ implicitly verifies that G is a cyclic group of prime order). Fixing x and considering any element $\mu \in G$, we calculate the probability that $c = \mu$ (where the probability is over the sender's choice of r). Let $\alpha = \log_g \mu$ and $\beta = \log_g h$ (these are well-defined, since g is a generator and G is cyclic). Then $c = \mu$ if and only if $\alpha = \log_g \mu = \log_g(g^x h^r) = x + r\beta \bmod q$ or, equivalently, if $r = (\alpha - x)\beta^{-1} \bmod q$ (note that $\beta^{-1} \bmod q$ is defined, since h is a generator so $\log_g h \neq 0$). But the probability that r takes on this value is exactly $1/|\mathbb{Z}_q| = 1/q = 1/|G|$. This implies that the value of x is perfectly hidden.

Another way to view this ("in reverse") is to note that for any value c there are exactly q possible pairs (x, r) satisfying $g^x h^r = c$, one for each possible value of $x \in \mathbb{Z}_q$. So, even an all-powerful $\mathcal{R}^*$ cannot tell which value of x is the one committed to by $\mathcal{S}$.

Binding: We show that if a PPT sender $\mathcal{S}^*$ can decommit to two different values, then we can use $\mathcal{S}^*$ to break the discrete logarithm assumption in G. On input G, q, g, h, algorithm A (with the goal of computing $\log_g h$) sends G, q, g, h to $\mathcal{S}^*$ who responds with a value $c \in G$. If $\mathcal{S}^*$ is now able to decommit to (x, r) *and* (x', r'), with $x \neq x'$ and $g^x h^r = c = g^{x'} h^{r'}$, then A can compute the desired discrete logarithm by noting that:

$$g^x h^r = g^{x'} h^{r'} \iff g^{x-x'} = h^{r'-r} \iff g^{(x-x')/(r'-r)} = h,$$

or $\log_g h = (x - x')(r' - r)^{-1} \bmod q$. (Note that $(r' - r)$ is non-zero as long as h is a generator — if not, then it is easy for A to compute $\log_g h$!). Clearly, A runs in polynomial time if $\mathcal{S}^*$ does; also, if $\mathcal{S}^*$ decommits to two different values with probability $\varepsilon(k)$ then A correctly computes $\log_g h$ with the same probability. Since this must be negligible under the discrete logarithm assumption, the binding property follows.

3.2 A Construction Based on the RSA Assumption

The next construction is based on the RSA assumption (for large prime public exponents), and again allows the sender to commit to a string $x \in \{0,1\}^k$.

$\underline{\mathcal{S}(1^k, x)}$		$\underline{\mathcal{R}(1^k)}$
		generate k-bit primes p, q set $N = pq$ choose a prime e s.t. $e > N$
verify e prime; $e > N$; $\mu \in \mathbb{Z}_N^*$	$\xleftarrow{N,\ e,\ \mu}$	$\mu \leftarrow \mathbb{Z}_N^*$
view x as an element of $\mathbb{Z}_e$ $r \leftarrow \mathbb{Z}_N^*$		
$c = \mu^x r^e \bmod N$	$\xrightarrow{c}$	
	- - - - - - - - - - - -	
	$\xrightarrow{x,\ r}$	verify $x \in \mathbb{Z}_e$ and $c \stackrel{?}{=} \mu^x r^e \bmod N$

We now prove both hiding and binding:

Hiding: Before sending the commitment, the sender verifies that e is prime and $e > N$; this guarantees that $\gcd(e, \phi(N)) = 1$. (We require $e > N$ because the receiver might try to cheat and send a modulus N which is not a product of two primes.) Thus, the function $f : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ defined by $f(x) = x^e \bmod N$ is a permutation. Since r is chosen uniformly at random in $\mathbb{Z}_N^*$, the value $f(r) = r^e \bmod N$ is uniformly distributed in $\mathbb{Z}_N^*$. Since $\mu \in \mathbb{Z}_N^*$ and hence $\mu^x \in \mathbb{Z}_N^*$, the product $c = \mu^x r^e$ is uniformly distributed in $\mathbb{Z}_N^*$ and reveals no information about x.

Another way to understand this is that for any commitment $c \in \mathbb{Z}_N^*$ and for any possible $x \in \mathbb{Z}_e$, there exists an $r \in \mathbb{Z}_N^*$ such that $c = \mu^x r^e \bmod N$. So no information about x is revealed.

Binding: We show that if a cheating sender $\mathcal{S}^*$ can decommit to two different values, then we can use it to break the RSA assumption (for large public exponents). Namely, given (N, e, μ) with N a randomly-generated product of two primes, $e > N$ a prime, and μ chosen at random from $\mathbb{Z}_N^*$, we show how to use $\mathcal{S}^*$ to compute $\mu^{1/e}$. Simply send (N, e, μ) to $\mathcal{S}^*$ and assume it sends back $c \in \mathbb{Z}_N^*$ and can decommit to (x, r) and (x', r') such that $x \neq x'$, $\mu^x r^e = c = \mu^{x'}(r')^e \bmod N$, and both $x, x' \in \mathbb{Z}_e$. Without loss of generality assume $x > x'$. We know that:

$$\mu^x r^e = \mu^{x'}(r')^e \bmod N \iff \mu^{x-x'} = (r'/r)^e \bmod N.$$

Let $\Delta \stackrel{\text{def}}{=} x - x'$. Using the fact that $x, x' \in \mathbb{Z}_e$ and e is prime, we see that $\Delta < e$ and hence $\gcd(\Delta, e) = 1$. Using the extended Euclidean algorithm, we can compute in polynomial time integers A, B such that $A\Delta + Be = 1$ (over the integers). Then:

$$\begin{aligned} \mu^1 = \mu^{A\Delta+Be} &= \left(\mu^{\Delta}\right)^A \mu^{Be} \bmod N \\ &= \left((r'/r)^e\right)^A (\mu^B)^e \bmod N \\ &= \left((r'/r)^A \mu^B\right)^e \bmod N, \end{aligned}$$

and so

$$\mu^{1/e} = (r'/r)^A \mu^B \bmod N.$$

This algorithm runs in polynomial time if $\mathcal{S}^*$ does, and outputs the desired inverse with the same probability that $\mathcal{S}^*$ can decommit to two different values. Since the former probability is negligible under the RSA assumption (for large prime exponents), the binding property follows.

References

[1] M. Naor. Bit Commitment Using Pseudorandomness. *Journal of Cryptology* 4(2): 151–158 (1991).

CMSC 858K — Advanced Topics in Cryptography April 22, 2004

Lecture 23

Lecturer: Jonathan Katz

Scribe(s): Nicholas Sze, Ji Sun Shin, Kavitha Swaminathan

1 Introduction

We showed previously a zero-knowledge proof system for 3-colorability. Unfortunately, to achieve negligible soundness error while maintaining zero knowledge it was required to repeat the basic, 3-round protocol *sequentially* polynomially-many times (giving a protocol with polynomial round complexity). Here, we show a *constant-round* zero-knowledge proof system for $\mathcal{NP}$. We will also discuss the notion of *proofs of knowledge* and show a (non-constant-round) zero-knowledge proof of knowledge for languages in $\mathcal{NP}$.

1.1 A Brief Review

At a very high level, we review why our previous techniques did not suffice to give a constant-round zero-knowledge proof system (refer to previous lectures for more details). Recall the basic, 3-round protocol for 3-colorability: the prover sends commitments to the colorings of the vertices in the graph; the verifier sends a "challenge" (i, j) (where this is an edge in the graph); and the prover responds by "opening" the commitments to the colors for vertices i and j. The verifier accepts only if these colors are different.

In proving the zero-knowledge property of this basic protocol, we relied on the fact that a simulator could "guess" the verifier's challenge (i, j) in advance with noticeable (i.e., inverse polynomial) probability. So, having the simulator "rewind" polynomially-many times would be sufficient to allow the simulator to "guess correctly" at least once. But this very property also allows a cheating prover to guess the verifier's challenge in advance with noticeable probability, meaning that the soundness will not be negligible.

We can decrease the soundness error by repeating the basic protocol many times. But if we schedule to repetitions in parallel, the simulator has only negligible probability of guessing all the verifier's queries (simultaneously) in advance. On the other hand, if we schedule the repetitions sequentially then the simulator can guess each challenge (one-by-one) with noticeable probability.[1]

2 A Constant-Round Zero-Knowledge Proof for $\mathcal{NP}$

Goldreich and Kahan [3] suggested the first constant-round ZK proof system for $\mathcal{NP}$. The intuition behind their scheme is to force the verifier to commit to its challenges in advance. Then, once the simulator has learned the verifier's challenges, it can rewind and commit to a set of colorings that will allow it to answer the challenges correctly.

[1] Note that the simulator has more power than a cheating prover since it can rewind the verifier.

We now describe the protocol in detail:

Initialization The prover and verifier each have a graph $\mathcal{G}$. The prover also knows a 3-coloring of this graph. Let the common security parameter be k (this might be the number of vertices in $\mathcal{G}$, but it could also be independent of $\mathcal{G}$).

First stage The verifier chooses k edges $(i_1, j_1), \ldots, (i_k, j_k)$ uniformly at random from (the edge set of) $\mathcal{G}$. It then commits to these edges using a perfect commitment scheme, and sends these commitments to the prover. We saw in the last lecture that there are two-round protocols for perfect commitment based on some number-theoretic assumptions, so for convenience we will assume that the first stage is carried out in rounds 1 and 2.

Rounds 3–5 The prover and verifier now execute k *parallel* executions of the basic, 3-round protocol for graph 3-colorability. Sketching this in a bit more detail (but still assuming the reader is familiar with the basic protocol from a previous lecture):

- **Round 3** The prover commits to k different colorings of $\mathcal{G}$ using independent randomness for each of these k iterations, and where the colors in each iteration are committed to vertex-by-vertex (as usual). It is stressed that independent randomness is used in each of the k iterations, so in particular each of the k colorings is a random permutation of the coloring the prover started with.
- **Round 4** The verifier decommits the challenges that it committed to in the first stage. This results in a sequence of k edges (and the corresponding decommitments) that are sent to the prover.
- **Round 5** The prover first checks to make sure that the verifier opened his commitments correctly. If not, then the verifier is cheating so the prover aborts. Otherwise, the prover responds to the challenges as usual: in iteration ℓ, if the challenge is (i_ℓ, j_ℓ) then the prover reveals the colors of vertices i and j in the ℓ^{th} iteration.

Acceptance The verifier accepts only if all k iterations were successfully completed.

Note that the verifier uses a *perfect* commitment scheme to commit to its challenges in the first phase, while the prover uses a *standard* commitment scheme to commit to the colorings in round 3. The is necessary because a proof system requires soundness to hold against an *all-powerful* (cheating) prover. If the verifier used a standard commitment scheme, then an all-powerful prover would be able to figure out the verifier's challenges before round 3, and could then fool the verifier into accepting even if $\mathcal{G}$ were not a 3-colorable graph. Similarly, if the prover's commitments were not information-theoretically binding then it would be able to "change" its answers depending on the challenges of the verifier.

Given the above discussion, it is easy to see that the above scheme satisfies correctness and has negligible soundness error even for an all-powerful prover. The difficult part is to show that the protocol is zero knowledge. In fact, a full proof is quite involved and we will not give one here (see [3]). Instead, we will only give some of the intuition for the proof by considering the case of a verifier who *always opens the commitments correctly in round 4*. Also, we will be relatively informal here (since we are not giving a complete proof anyway)

but the interested reader will be able to derive a proof for this restricted case from the proof given earlier for the basic, 3-round protocol.

A simulator for the type of verifier considered here proceeds as follows:

1. For the first phase, the simulator runs the perfect commitment scheme normally and obtains a sequence of commitments from the verifier.

2. Simulating round 3, the simulator sends k "garbage" commitments to colorings of $\mathcal{G}$. Namely: for each of k iterations, commit to "red" for each vertex of the graph. (The simulator must commit to "garbage" because it does not know a coloring. But by indistinguishability of the commitments, a poly-timer verifier can't distinguish these "garbage" commitments from commitments that would be sent by a real prover.)

3. The verifier then decommits to the challenges that it committed to in the first stage. (Recall we assume that the verifier always decommits properly.) Denote these challenges by $(i_1, j_1), \ldots, (i_k, j_k)$.

4. Now, the simulator "rewinds" the verifier and sends k commitments to colorings of $\mathcal{G}$ *for which it can answer the challenges of the verifier*. That is: for the ℓ^{th} iteration, the simulator chooses random, distinct colors for vertices i_ℓ and j_ℓ, commits to these colors for i_ℓ and j_ℓ, and commits to "red" for all other vertices (in that iteration). Denote these commitments by $\mathsf{com}_1, \ldots, \mathsf{com}_k$ (each com_ℓ is composed of commitments for each vertex of $\mathcal{G}$).

5. The verifier again decommits to the challenges that it committed to in the first stage. Although we assume that the verifier always decommits properly, we do not necessarily assume that the decommitted values now are the same as they were before! However, they *do* in fact have to be the same *with all but negligible probability*; this follows from the (computational) binding of the commitment scheme used in the first phase.

6. Assuming the commitments were again opened in the same way, the simulator can easily decommit the relevant vertices correctly.

7. The final "view" output by the simulator includes the verifier's random coins, the messages sent to the verifier during the first stage, the second sequence of commitments $\mathsf{com}_1, \ldots, \mathsf{com}_k$, and the decommitments for the appropriate vertices.

Informally, the simulated transcript is indistinguishable from a real transcript because of the hiding property of the commitment scheme used by the prover in the 3^{rd} round. For a careful proof in the general case and much more discussion and details, see [3].

3 Proofs of Knowledge

Proofs of knowledge may be viewed as formalizing an even stronger notion of soundness. Very informally, a proof system may be viewed as demonstrating that a particular statement is true; a proof of knowledge may be viewed as demonstrating that the prover "knows" why the statement is true. Although it is fair to say that the notion of a proof of knowledge was introduced for (very important) technical reasons, there are some practical examples of

why proofs of knowledge are necessary. As an example of the latter, let G be a finite cyclic group of order q in which the discrete logarithm assumption is believed to hold, and let g be a generator of G. Consider the language $L_G = \{h \mid \exists x \in \mathbb{Z}_q \text{ s.t. } g^x = h\}$. On the one hand, L_G is just G itself since for every element $h \in G$ we know that $g^{\log_g h} = h$. So a proof that $h \in L_G$ is trivial (assuming that deciding membership in G is trivial). On the other hand, a proof *of knowledge* that $h \in L_G$ implies that the prover "knows" the value of $\log_g h$, something that is not implies by a proof alone. Similarly, if f is a one-way permutation and we define $L_f = \{y \mid \exists x \text{ s.t. } f(x) = y\}$, then a proof for L_f is trivial (since L_f contains all strings) but a proof of knowledge that $y \in L_f$ is not (as it implies that the prover "knows" $f^{-1}(y)$).

Of course, this leaves us with a vague sense of discomfort: what does it mean for a machine to "know" something? We define this in terms of the ability to *extract* the knowledge from the machine: i.e., a machine M "knows" something if there is a poly-time process by which we can extract this knowledge from M. We do not give a formal definition here (see [2] or [1] instead), but give the following informal definitions instead:

Definition 1 A *relation* R is a set of pairs of strings. A relation is said to be "polynomial-time" if: (1) there exists a polynomial $p(\cdot)$ such that $(x, y) \in R$ implies $|y| \leq p(|x|)$, and (2) given a pair (x, y), one can decide in polynomial time (in $|x|$) whether $(x, y) \in R$. $\diamondsuit$

Any relation R defines a language $L_R \stackrel{\text{def}}{=} \{x \mid \exists y \text{ s.t. } (x, y) \in R\}$. Furthermore, if R is polynomial time, then $L_R \in \mathcal{NP}$. Finally, any language $L \in \mathcal{NP}$ defines a relation $R_L \stackrel{\text{def}}{=} \{(x, y) \mid x \in L \wedge y \text{ is a witness for } x\}$.

Definition 2 Let R be a polynomial-time relation, and L_R be as above. A proof system $(\mathcal{P}, \mathcal{V})$ for L_R is a *proof of knowledge for L_R with soundness error $\varepsilon(k)$* if the following holds for all x and all cheating provers $\mathcal{P}^*$: Let $\mathsf{succ}_{\mathcal{P}^*}(k) = \Pr[\langle \mathcal{P}^*, \mathcal{V} \rangle(1^k, x) = 1]$. If $\mathsf{succ}_{\mathcal{P}^*}(k) > \varepsilon(k)$ then with probability negligibly close to $\mathsf{succ}_{\mathcal{P}^*}(k)$ one can extract a value y from $\mathcal{P}^*$ in polynomial time[2] such that $(x, y) \in R$. A *proof of knowledge for L_R* is one having negligible soundness error. $\diamondsuit$

This definition will hopefully become more clear after we show an example in the following section.

Note that the zero-knowledge (ZK) requirement is orthogonal to the proof of knowledge (PoK) requirement. The former protects the prover from a malicious verifier, while the latter protects (in some sense) a verifier from a malicious prover. On the other hand, without any additional requirements it is trivial to construct a PoK for any polynomial-time relation: on common input x, the prover simply sends y such that $(x, y) \in R$. So, we will be interested in witness indistinguishable PoKs (WI-PoKs) or ZK-PoKs.

3.1 A Proof of Knowledge for Hamiltonian Cycle

Here, we show a basic 3-round ZK PoK (with non-negligible soundness error) for the language HAM of Hamiltonian cycles (this is the set of graphs containing a Hamiltonian

[2] Paralleling the case of zero-knowledge, extraction in expected polynomial-time are often allowed.

cycle — i.e., a cycle that includes each vertex of the graph exactly once). This language is $\mathcal{NP}$-complete, so this gives[3] a PoK for all of $\mathcal{NP}$.

The protocol proceeds as follows:

$\underline{\mathcal{P}(\mathcal{G}, \text{Ham-cycle})}$		$\underline{\mathcal{V}(\mathcal{G})}$
pick a random permutation Π of the graph $\mathcal{G}$; commit to the adjacency matrix of $\Pi(\mathcal{G})$	$\left(\begin{array}{ccc} \mathsf{com}_{11} & \cdots & \mathsf{com}_{1n} \\ \vdots & \ddots & \vdots \\ \mathsf{com}_{n1} & \cdots & \mathsf{com}_{nn} \end{array}\right)$ $\longrightarrow$	
	$\longleftarrow$ c	$c \leftarrow \{0,1\}$
if $c = 0$, open all commitments and send Π if $c = 1$, open commitments on the Hamiltonian cycle		
	open according to c value $\longrightarrow$	
		verify commitments if $c = 0$ check that the matrix is equal to $\Pi(\mathcal{G})$ if $c = 1$ check that a cycle was revealed

Claim 1 *This is a proof of knowledge with soundness error* 1/2.

Proof Note that for any cheating prover and any graph $\mathcal{G}$, this prover either succeeds with probability 0, probability 1/2, or probability 1. All we need to prove (cf. the definition of proofs of knowledge) is that if $\mathcal{P}^*$ succeeds with probability 1, then we can extract from $\mathcal{P}^*$ a Hamiltonian cycle in $\mathcal{G}$ with probability negligible close to 1 (in fact, we will extract with probability 1). Succeeding with probability 1 simply means that it answers both possible challenges correctly.

Given such a $\mathcal{G}$ and $\mathcal{P}^*$ who convinces the verifier with probability 1, we simply let the prover send its initial message; send challenge "0" and get the response; then rewind the prover and send challenge "1" and get the response. By assumption, both responses of $\mathcal{P}^*$ would cause the honest verifier to accept. So, from the $c = 0$ response, we have a graph $\mathcal{G}'$ (that $\mathcal{P}^*$ committed to in the first round) and a permutation Π such that $\Pi(\mathcal{G}) = \mathcal{G}'$. From the $c = 1$ response, we have a Hamiltonian cycle in $\mathcal{G}'$. It is now easy to recover a Hamiltonian cycle in the original graph $\mathcal{G}$. ■

[3]There are some additional subtleties here: we need it to be the case that for any $L \in \mathcal{NP}$ there exist poly-time computable functions f_1, f_2 such that: $x \in L \Leftrightarrow f_1(x) \in HAM$ and also $(f_1(x), y) \in R_{HAM} \Leftrightarrow (x, f_2(y)) \in R_L$.

The proof system is also zero knowledge. A simulator Sim can be constructed as follows:

	$\underline{\mathsf{Sim}(G)}$
Repeat k times:	
1	Guess $c' \leftarrow \{0,1\}$
$2a$	If $c' = 0$, commit to a random permutation Π of $\mathcal{G}$
$2b$	If $c' = 1$, commit to a random cycle graph
3	Send the commitments to the verifier, who responds with c
4	If $c = c'$, output a transcript including the correct response

We do not prove that this simulator "works", but leave this as an exercise for the reader.

As usual, by repeating the protocol multiple times we can decrease the soundness error. We know that by repeating the protocol sequentially we retain the zero-knowledge property (at the expense of high round complexity); what about the proof of knowledge property?

Claim 2 *Running the above protocol k times sequentially results in a proof of knowledge with soundness error $1/2^k$.*

Proof (Sketch) Assume a graph $\mathcal{G}$ and a prover $\mathcal{P}^*$ who convinces $\mathcal{V}$ with probability strictly greater than $1/2^k$. We can view the execution of $\mathcal{P}^*$ with $\mathcal{V}$ as a binary tree of height k, where the root corresponds to the beginning of the protocol and a node at level i (with the root at level 0) has two children corresponding to the two possible challenges that can be sent at round $i+1$. Call a leaf of the tree *accepting* only if the prover answers correctly to all challenges on the path from the root to this leaf. Since $\mathcal{P}^*$ convinces $\mathcal{V}$ with probability strictly greater than $1/2^k$, there are at least two accepting leaves. Intuitively, the paths from these two leaves to the root must have at least one node in common; at this node, $\mathcal{P}^*$ answers correctly for *both* possible challenges, and we can then extract as in the previous claim. We now show how to do this efficiently:

for $i = 1, \cdots, n$
 run $\mathcal{P}^*$ for round i
 by rewinding, send both $c = 0$ and $c = 1$
 if $\mathcal{P}^*$ answers correctly both times then extract a witness (as before)
 otherwise, increment i and continue along the path for which $\mathcal{P}^*$ answered correctly

(In the last step, there must be a value of the challenge for which $\mathcal{P}^*$ answers correctly since otherwise $\mathcal{P}^*$ convinces the verifier with probability 0.) Eventually, the above algorithm finds a node where two paths from the root to accepting nodes diverge (drawing a picture and following the execution of the above algorithm should convince you of this). □

If we want to construct a protocol with better round complexity, we can do so by running the basic, 3-round protocol in parallel. We know that this will not preserve the zero-knowledge property, but it will preserve the (weaker) property of witness indistinguishability. What about the proof of knowledge property?

Claim 3 *Running the basic protocol k times in parallel results in a proof of knowledge with soundness error $1/2^k$. (However, here extraction requires expected polynomial time.)*

Proof Note that we have a 3-round protocol where the prover begins by sending a vector of k commitments (to adjacency matrices); the verifier sends a k-bit challenge vector; and the prover then responds to each of the k 1-bit challenges individually, as in the basic protocol. If we can find two *different* vectors $\vec{c}, \vec{c}^*$ for which the prover responds correctly, then we can extract a witness as before: simply find an index i where $c_i \neq c_i^*$ (such an index must exist since $\vec{c} \neq \vec{c}^*$) and then extract using the i^{th} adjacency matrix sent by the prover in the first round and the i^{th} response given by the prover in the last round.

Assume, then, that we have a $\mathcal{G}$ and a prover $\mathcal{P}^*$. Consider the following algorithm to extract a witness:

$\mathcal{P}^*$ sends its vector of commitments
$\vec{c} \leftarrow \{0,1\}^k$
run $\mathcal{P}^*$ using challenge $\vec{c}$
if $\mathcal{P}^*$ fails to respond correctly, halt
otherwise:
for $i = 0$ to $2^k - 1$:
 $\vec{c}^* \leftarrow \{0,1\}^k$
 run $\mathcal{P}^*$ using challenge $\vec{c}^*$
 if $\mathcal{P}^*$ responds correctly and $\vec{c} \neq \vec{c}^*$, extract a witness and halt
 run $\mathcal{P}^*$ using challenge $\langle i \rangle$
 if $\mathcal{P}^*$ responds correctly and $\vec{c} \neq \langle i \rangle$, extract a witness and halt

(In the above, $\langle i \rangle$ represents a standard k-bit binary encoding of the number i.)

Let $\varepsilon(k)$ denote the probability that $\mathcal{P}^*$ answers correctly. We need to show two things: (1) if $\varepsilon(k) > 1/2^k$, then we extract a witness with (negligibly close to) the same probability; (2) the algorithm above runs in expected polynomial time *regardless* of ε.

If $\varepsilon > 1/2^k$ then there are at least 2 difference challenges for which $\mathcal{P}^*$ answers correctly. The algorithm above enters the loop with probability exactly ε; once it enters the loop, it is guaranteed to eventually find a second, different challenge for which $\mathcal{P}^*$ answers correctly. Since it extracts a witness in this case, we have that it extracts a witness overall with probability exactly ε. (Actually, we have ignored the negligible probability with which $\mathcal{P}^*$ might be able to break the binding property of the commitment scheme. But the basic argument remains the same or we can use a commitment scheme with perfect binding.)

We also need to also argue that extraction runs in expected polynomial time (in k); this seems worrisome since the inner loop potentially counts up to $2^k - 1$. Consider the following cases: if $\varepsilon = 0$ then clearly the algorithm runs in polynomial time (since it never enters the loop). If $\varepsilon = 2^{-k}$ then we enter the loop with probability ε and then run $2^k - 1$ iterations of the inner loop. So the expected number of loop iterations is:

$$2^{-k}(2^k - 1) + (1 - 2^{-k}) \cdot 0 < 1,$$

and the algorithm runs in *expected* polynomial time (we do not extract a witness in this case, but that is ok). Finally, if $\varepsilon > 2^{-k}$ then consider what happens if $\mathcal{P}^*$ answers correctly in the initial stage for a vector $\vec{c}$. In this case, the probability of choosing $\vec{c}^* \neq \vec{c}$ for which $\mathcal{P}^*$ answers correctly is exactly $\frac{2^k\varepsilon - 1}{2^k} \geq \varepsilon/2$. So the expected number of loop iterations until such a $\vec{c}^*$ is found is at most $2/\varepsilon$. Since the probability of entering the loop in the first

place is ε, the expected number of loop iterations overall is:

$$\varepsilon \cdot \frac{2}{\varepsilon} + (1 - \varepsilon) \cdot 0 < 2,$$

and the algorithm runs in expected polynomial time. ■

References

[1] M. Bellare and O. Goldreich. *On Defining Proofs of Knowledge*. Crypto '92.

[2] O.Goldreich. *Foundations of Cryptography, vol 1: Basic Tools*. Cambridge University Press, 2001.

[3] O. Goldreich and A. Kahan. *How to Construct Constant-Round Zero-Knowledge Proof Systems for NP*. Journal of Cryptology, 1996.

CMSC 858K — Advanced Topics in Cryptography April 27, 2004

Lecture 24

Lecturer: Jonathan Katz *Scribe(s): A. Anand, G. Taban, M. Cho*

1 Introduction and Review

In the previous classes, we have discussed proofs of knowledge (PoKs) and zero-knowledge (ZK) proofs. We briefly review these notions here:

ZK proofs. Zero-knowledge proofs involve a prover P trying to prove a statement to a verifier V without revealing any knowledge beyond the fact that the statement is true. For example, consider the problem of proving membership in an NP language L, (e.g., graph Hamiltonicity, 3-coloring, etc.). A ZK proof protects against a cheating prover, in the sense that if a prover tries to give a proof for an $x \notin L$ the verifier will reject the proof with all but negligible probability. Further, a ZK proof protects against a cheating verifier, in the sense that it ensures that the verifier (informally) does not learn anything from a proof that $x \in L$ other than the fact that $x \in L$.

A ZK proof system requires the existence of a simulator who can simulate a transcript of the protocol execution without knowing the witness to the statement. As we have seen, a simulator typically does this by rewinding the verifier to a prior state and then trying to continue the simulation until it comes up with a valid transcript.

Proofs of knowledge. Proofs of knowledge are protocols in which the prover actually proves that he "knows" a *witness*. In addition to the formal sense in which this holds (i.e., via the additional requirement that there exists a *knowledge extractor* who can extract a valid witness from any prover who succeeds in giving a correct proof with high-enough probability), there are also examples where membership is "easy" to determine but proving knowledge of a witness might be hard. The classic example is the case of a cyclic group G with generator g in which the discrete logarithm is hard. Here, for a given $h \in G$ it might be easy to determine that, in fact, h is an element of G and therefore there *exists* an x such that $g^x = h$; however, we may additionally want a prover to prove that he *knows* this x.

Thinking about it a bit, the ZK property and the PoK property seem to be at odds: a ZK proof requires a simulator who (typically) rewinds a cheating verifier to simulate a proof without knowing a witness, while a PoK requires a knowledge extractor who (typically) rewinds a cheating prover to extract a witness. And indeed, we will see in what follows that the approach to constructing a constant-round ZK proof from the previous lecture (namely, forcing the verifier to commit to its challenges in advance) seemingly destroys the PoK property. To obtain a constant-round protocol which is *both* zero-knowledge *and* admits a knowledge extractor we will consider a relaxation of proofs called *arguments* in which the soundness condition is only required to hold only with respect to *polynomial-time* cheating provers (recall that *proofs* require the soundness condition to hold even for *all-powerful* provers).[1]

[1] In fact, constant-round zero-knowledge *proofs* of *knowledge* for all of NP are possible, but we will not see an example in this course.

2 Review: A ZK PoK Protocol

In this section, we review a ZK PoK protocol for graph Hamiltonicity from the previous lecture; see Figure 1. In the figure, G is a graph and C represents a Hamilton cycle in G. In the previous lecture, we showed that this protocol is zero-knowledge and a proof of knowledge with soundness error 1/2. We now informally recall the proofs of these properties:

- To prove the zero-knowledge property, we considered the following simulator: it guesses in advance a bit $\tilde{c}$. If $\tilde{c} = 0$, it commits to a (random permutation of) the adjacency matrix for G; if $\tilde{c} = 1$, it commits to a (randomly-permuted) cycle graph. It sends the resulting commitment to the verifier who responds with a challenge c. If $\tilde{c} = c$ then the prover responds in the natural way and is done. Otherwise, the simulator rewinds and tries again. Since $\tilde{c} = c$ with probability 1/2 each time, if the simulator rewinds k times it will succeed at least once with all but negligible probability. (The rest of the proof is then devoted to showing that the simulation is computationally indistinguishable from a real execution.)
- To show the knowledge extraction property, assume we have a prover who gives a correct proof with probability better than 1/2. This implies that the prover responds correctly for both possible values of c. So we simply rewind the prover, send him both possible challenges, and use the two (correct) responses to compute a Hamiltonian cycle in G.

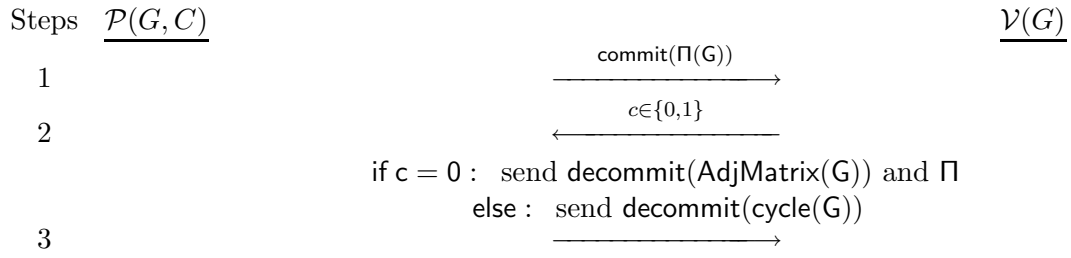


Figure 1: A ZK-PoK protocol.

We also noted that we could run the above protocol k times *sequentially* to reduce the soundness error to 2^{-k}. Doing so maintains the ZK property of the construction, and we showed that the resulting protocol was also a PoK with the claimed soundness error.

2.1 A Parallel Execution of the Protocol

What happens if we run the original protocol k times *in parallel*? The PoK property remains intact:

Claim 1 *Running the above protocol k times in parallel results in a PoK with soundness error 2^{-k}.*

Proof To prove the above claim, we consider a knowledge extractor E which works as follows.

- Obtain a first message from the prover P.
- Pick a random challenge $c_1 \in \{0,1\}^k$, and send this challenge to the prover. If the prover does not answer correctly, stop.

- Otherwise, repeatedly choose a random $c_2 \in \{0,1\}^k$, rewind the prover, and send c_2 to the prover until the prover answers correctly a second time and $c_2 \neq c_1$. In parallel, perform an exhaustive search for a Hamiltonian cycle in G and stop once one is found or when it is determined that no such cycle exists.

We now show two facts: (1) E runs in expected polynomial time (this assumes that P runs in expected polynomial time), and (2) if the probability p that P gives a successful proof is greater than 2^{-k} then E succeeds in computing a Hamiltonian cycle in G with probability p.

To prove the first statement, note that when $p = 0$ then E clearly runs in (strict) polynomial time. So consider the case that $p > 2^{-k}$. Let $n > 1$ be the number of challenges for which P answers correctly (i.e., $p = \frac{n}{2^k}$). When P does not respond correctly to the first challenge c_1 (with happens with probability $1-p$), then E runs in (strict) polynomial time. When P responds correctly to c_1, then the expected number of times E rewinds P until it finds a second (different) c_2 for which P answers correctly is $(\frac{n-1}{2^k})^{-1}$. Overall, then, the expected running time of E is given by:

$$(1-p) \cdot \mathsf{poly}(k) + \frac{n}{2^k} \cdot \frac{2^k}{n-1} \cdot \mathsf{poly}(k) = \mathsf{poly}(k)$$

(we use $\mathsf{poly}(k)$ here to refer to an arbitrary polynomial). The last case remaining is when $p = 2^{-k}$ (i.e., P responds correctly to exactly one challenge). As before, when P does not respond correctly to c_1 then E runs in strict polynomial time. When P responds correctly to c_1, then E will *never* find a $c_2 \neq c_1$ for which P answers correctly again. But, P is also running an exhaustive search for a Hamiltonian cycle in G and we assume this takes at most $2^k \cdot \mathsf{poly}(k)$ steps.[2] So, the expected running time of E is given by:

$$(1-2^{-k}) \cdot \mathsf{poly}(k) = 2^{-k} \cdot 2^k \cdot \mathsf{poly}(k) = \mathsf{poly}(k).$$

We now move on to a proof of the second statement. Note that P responds correctly to challenge c_1 with probability exactly p. We claim that as long as $p > 2^{-k}$, then E *always* computes a Hamiltonian cycle whenever P responds correctly to c_1. To see this, note first that $p > 2^{-k}$ implies that G has a Hamiltonian cycle. (We assume here that the commitments sent in the first round are perfectly binding.) When P responds correctly to c_1, then E stops its execution when either (1) it finds a $c_2 \neq c_2$ for which P also responds correctly, or (2) it completes its exhaustive search for a cycle. In either of these cases, E can then compute the desired Hamiltonian cycle. ■

Unfortunately, k-fold parallel repetition of the protocol seems to destroy the zero-knowledge property. At a minimum, the type of simulator we considered before does not work, and no simulator is known which would prove the ZK property. In particular, if we consider the simulation strategy as before then we would have a simulator who tries to guess $\tilde{c} = c$ in advance: but now $c \in \{0,1\}^k$ and so the probability of guessing correctly is negligible! (And so even repeatedly guessing polynomially-many times will not help.)

2.2 Further Modifications?

We can try to recover the ZK property (for the protocol obtained via k-fold parallel repetition of the original, 3-round protocol) by using the Goldreich-Kahan technique, in which the verifier is forced to commit (using a perfectly-hiding commitment scheme) to their challenge vector c *in*

[2] If one is unhappy with this assumption, note that exhaustive search takes at most $k! \cdot \mathsf{poly}(k)$ time and so by running the protocol $\log k! = O(k \log k)$ times in parallel the proof goes through.

advance. For future reference, let us call the round in which the verifier commits to c "round 0". This modification will indeed result in a zero-knowledge protocol... but the modified protocol no longer appears to be a proof of knowledge! Indeed, the very fact that the verifier is forced to commit in advance to c means that the knowledge extraction strategy outlined earlier will no longer work: even E cannot break the commitment scheme, and so it cannot decommit to $c_2 \neq c_1$ unless it rewinds all the way back to round 0 and sends a new set of commitments, in which case P might change its round-1 message!

Somewhat paradoxically(?), it is possible to design a constant-round ZK-PoK. Instead of showing this, however, we consider a relaxation of the notion of a "proof" and show how to achieve both knowledge extraction and zero-knowledge subject to this relaxation.

3 Zero-Knowledge *Arguments* of Knowledge

As discussed in the introduction, an *argument* requires soundness to hold only for provers running in polynomial time (whereas a *proof* requires soundness to hold even for *all-powerful* provers). (An *argument of knowledge* is defined similarly, such that a knowledge extractor is only required to extract a witness from provers running in polynomial time.) We will now show a construction of a constant-round zero-knowledge argument of knowledge due to Feige and Shamir. Our discussion will be somewhat informal and "high-level"; the reader is referred to [2, 1] for further details.

Let f be a one-way function. The basic protocol proceeds in 6 rounds (it is possible to "collapse" this to a 4-round protocol, but the proof is less intuitive in this case so we do not present this extension). Let L be an NP language; we describe the protocol assuming the honest prover is proving that $x \in L$ given some witness w.

Rounds 1–3: The verifier chooses x_1, x_2 at random, computes $y_1 = f(x_1)$ and $y_2 = f(x_2)$, and sends y_1, y_2 to the prover. The verifier then gives a 3-round witness-indistinguishable (WI) proof of knowledge (with negligible soundness error) of "$f^{-1}(y_1)$ or $f^{-1}(y_2)$". Note that the verifier can do this efficiently, since it knows witnesses x_1, x_2 (in fact, only one witness is needed). We comment briefly below on the existence of 3-round WI proofs of knowledge.

Rounds 4–6: If the proof given by the verifier fails, the prover simply aborts. Otherwise, the prover gives a 3-round witness-indistinguishable proof of knowledge of "$f^{-1}(y_1)$ or $f^{-1}(y_2)$ or $x \in L$". Note that the prover can do this efficiently since it has a witness that $x \in L$.

In the previous lecture we have already shown a 3-round WI proof of knowledge with negligible soundness error: the k-fold parallel repetition of the basic, 3-round protocol (with soundness error 1/2). We proved explicitly last time that this protocol is a proof of knowledge with negligible soundness error. The fact that it is witness indistinguishable follows from the facts that: (1) as proved last time, the basic 3-round protocol is zero-knowledge; (2) zero-knowledge implies witness indistinguishability, and hence the basic, 3-round protocol is witness indistinguishable; finally (3) we saw in an earlier lecture that witness indistinguishability is preserved under parallel repetition.

We now discuss, informally, why this protocol is both zero-knowledge and an argument of knowledge. (Note that it is certainly not a proof, since an all-powerful prover can invert f and then give a successful proof even when $x \notin L$.)

Zero-knowledge. We show a simulator demonstrating that the protocol is zero knowledge (although no formal proof will be given). The simulator, on input $x \in L$ but *without* a witness, proceeds as follows:

Rounds 1–3: Interact with the (possibly cheating verifier) V^* to obtain values y_1, y_2 and a transcript T of the first three rounds. If V^* does not successfully complete its proof of knowledge, then the simulator can abort (just like the real prover would) and the simulation is done by simply outputting T. Otherwise, if V^* does give a successful proof of knowledge, the simulator runs the knowledge extractor for this 3-round proof to obtain a witness for "$f^{-1}(y_1)$ or $f^{-1}(y_2)$". (If this extraction fails, then the entire simulation is aborted.) Note that, assuming a witness is extracted, this gives an x such that either $f(x) = y_1$ or $f(x) = y_2$.

Rounds 4–6: Continuing in an execution with V^* with initial transcript T, the simulator now simply gives a WI proof of knowledge of "$f^{-1}(y_1)$ or $f^{-1}(y_2)$ or $x \in L$". The key point is that the simulator can do this without any further rewinding since it does indeed know a witness for this statement.

A proof that this results in a simulation which is computationally-indistinguishable from a real execution is relatively straightforward given all the machinery at our disposal. The initial portion of the transcript (i.e., the transcript T of the first 3 rounds) is identically distributed to the first 3 rounds in a real execution of the protocol. If V^* gives a successful proof in rounds 1–3, knowledge extraction will succeed with all but negligible probability. Assuming this to be the case, then the last 3 rounds in the simulation consist of a WI proof using the witness x extracted in the previous phase; in a real execution, these last 3 rounds would be a WI proof using a witness for $x \in L$ (the same statement is being proved in either case). But witness indistinguishability of the proof system used in rounds 4–6 implies that these two resulting transcripts are indeed computationally indistinguishable.

Argument of knowledge. We next argue that the protocol is an argument of knowledge, which will imply soundness (for poly-time provers). The knowledge extractor E is the obvious one: simply run the knowledge extractor for the WI proof of knowledge given by the prover in rounds 4–6. The analysis of E is the tricky part. A proof that E extracts a witness for $x \in L$ follows from two claims along the following lines:

Claim 2 *(Informal) E extracts a witness for the statement "$f^{-1}(y_1)$ or "$f^{-1}(y_2)$ or $x \in L$" with sufficiently-high probability ("sufficiently-high probability" here simply refers to the probability required by the definition of an argument/proof of knowledge).*

This claim follows immediately from the fact that the proof given in rounds 4–6 is a proof of knowledge. Next:

Claim 3 *E extracts a witness for "$f^{-1}(y_1)$ of $f^{-1}(y_2)$" with only negligible probability.*

Thus, whenever E extracts a witness for "$f^{-1}(y_1)$ or "$f^{-1}(y_2)$ or $x \in L$" (which is does sufficiently-often, by the previous claim) it in fact extracts a witness, as desired, for $x \in L$ except with negligible probability.

The proof of the above claim is more difficult, and proceeds in the following steps:

1. Say the probability of extracting a witness for "$f^{-1}(y_1)$ of $f^{-1}(y_2)$" is p. Let p_b denote the probability of extracting a witness for $f^{-1}(y_b)$. Clearly, either $p_1 \geq p/2$ or $p_2 \geq p/2$; assume the former without loss of generality.

2. The above refers to the probability of extraction when, in rounds 1–3, the verifier (i.e., the knowledge extractor playing the role of the verifier) gives a WI proof using witnesses for both

$f^{-1}(y_1)$ and $f^{-1}(y_2)$. In fact, it is enough to use only a witness for $f^{-1}(y_2)$ when giving this proof. Witness indistinguishability of the proof system in rounds 1–3 can be used to show that the probability of extracting a witness for $f^{-1}(x_1)$ is not affected by more than a negligible amount, and so the probability of extracting a witness for $f^{-1}(x_1)$ in this case is negligibly close to $p_1 \geq p/2$.

3. We show that if p_1 is non-negligible then we can use the cheating prover to invert the one-way function f as follows: Given a point y_1, choose a random x_2, compute $y_2 = f(x_2)$, and then run the above experiment (giving the appropriate WI proof in rounds 1–3 and then extracting in rounds 4–6). By what we have said above, the probability that we extract a witness for $f^{-1}(y_1)$ is negligibly-close to p_1. But extracting a witness for $f^{-1}(y_1)$ exactly means that we have computed $f^{-1}(y_1)$! Since this cannot occur with more than negligible probability, we conclude that p_1 (and hence p) is negligible.

We remark that the use of *two* values y_1, y_2 in the argument system is essential to the above proof. If we had used only a single value y_1, then in order to reach the extraction phase (i.e., rounds 4–5) we would need to successfully complete the proof in rounds 1–3. . . but since this is not a ZK proof we actually need a witness to do so. But if we have the witness $f^{-1}(y_1)$ in rounds 1–3, then extracting this witness in rounds 4–6 is not a contradiction! The nice feature of the Feige-Shamir system is that it allows the extractor to "switch" witnesses in rounds 1–3, and hence derive the desired contradiction.

Very detailed and formal proofs of the above properties (as opposed to the hand-waving proof sketches above) are given in [1].

4 Proof of Knowledge for Number Theoretic Arguments

The preceding ends our discussion (for now, anyway) of generic proofs/arguments for languages in NP. We now focus in *efficient* proofs of knowledge for *specific* number-theoretic relations. In particular, we show a proof of knowledge of discrete logarithms. Let G be a finite, cyclic group of prime order q. Let g be a generator for g, and let $h \in G$ be arbitrary. Consider the following protocol in which a prover P tries to convince a verifier that P indeed known the discrete logarithm $\log_g h$ (i.e., an x such that $g^x = h$):

Steps	$P(g, x, h = g^x)$		$V(g, h)$
1	$r \leftarrow \mathbb{Z}_q$		
2		$\xrightarrow{A = g^r}$	
3		$\xleftarrow{c}$	$c \leftarrow \mathbb{Z}_q$
4	$\gamma = cx + r \bmod q$	$\xrightarrow{\gamma}$	verify $g^\gamma = h^c \cdot A$

Figure 2: A PoK for discrete logarithms.

It is not hard to see that the above protocol is complete: if the prover is honest then

$$g^\gamma = g^{cx+r} = g^{cx} \cdot g^r = h^c \cdot A\,.$$

We now sketch why this protocol is a proof of knowledge. The knowledge extractor E will be similar to the one defined earlier: if P responds correctly to an initial challenge c then E will

repeatedly rewind P until it finds a different challenge c' for which P also responds correctly.[3] If E can find two such challenges c, c', we claim that it can then compute the desired discrete logarithm. Indeed, this means that E has values $A, c, c', \gamma, \gamma'$ such that $g^\gamma = h^c A$ and $g^{\gamma'} = h^{c'} A$. We claim that $\log_g h = (\gamma - \gamma')(c - c')^{-1} \bmod q$ (which can be computed easily; note that $c - c' \neq 0$ by construction). Indeed:

$$\begin{aligned} g^{\frac{\gamma-\gamma'}{c-c'}} &= \left(g^{\gamma-\gamma'}\right)^{1/(c-c')} \\ &= \left(g^\gamma g^{-\gamma'}\right)^{1/(c-c')} \\ &= \left(\frac{h^c A}{h^{c'} A}\right)^{1/(c-c')} \\ &= \left(h^{c-c'}\right)^{1/(c-c')} = h\,, \end{aligned}$$

as claimed.

References

[1] U. Feige. Alternative Models for Zero Knowledge Interactive Proofs. PhD thesis, Weizmann Institute of Science, 1990. Available at http://www.wisdom.weizmann.ac.il/~feige.

[2] U. Feige and A. Shamir. Zero Knowledge Proofs of Knowledge in Two Rounds. Crypto '89.

[3] O. Goldreich. *Foundations of Cryptography, Vol 1: Basic Tools*. Cambridge University Press, 2001.

[4] O. Goldreich and A. Kahan. How to Construct Constant-Round Zero-Knowledge Proof Systems for NP. *Journal of Cryptology*, 1996.

[3] As previously, E will also have to perform an exhaustive search for $\log_g h$ in parallel so that it doesn't run "too long". Details omitted.

CMSC 858K — Advanced Topics in Cryptography	April 29, 2004
Lecture 25	
Lecturer: Jonathan Katz	*Scribe(s): (none)*

1 Public-Key Identification Schemes

In this lecture, we discuss concrete applications of zero-knowledge proof systems to the task of constructing (public-key[1]) identification schemes. We begin with an informal definition of what these are.

Definition 1 A (public-key) identification scheme consists of a prover P who has generated some public-/private-key pair (PK, SK), and wants to prove his identity to a verifier V who knows PK. ♢

In terms of security, we say a scheme is secure against a *passive eavesdropper* if an adversary cannot impersonate the prover even after passively monitoring multiple executions of the protocol between the (real) prover and the (honest) verifier. We say a scheme is secure against an *active adversary* if an adversary cannot impersonate the prover (to an honest verifier) even after the adversary — acting in the role of the verifier — has interacted with the prover multiple times. We stress that in the latter case the adversary is not restricted to running the honest verification protocol, but may instead deviate arbitrarily from the prescribed actions. We also remark that security against an active adversary implies security against passive eavesdropping, since an active adversary can simulate passive eavesdropping by simply acting as an honest verifier.

It is simple to construct a public-key identification scheme from any signature scheme:

$\underline{P(SK)}$		$\underline{V(PK)}$
	$\xleftarrow{r}$	$r \leftarrow \{0,1\}^k$
$\sigma \leftarrow \text{sign}_{SK}(r)$	$\xrightarrow{\sigma}$	accept if $\text{Verify}_{PK}(r, \sigma)$=1

This scheme is secure against an active adversary. As an informal proof, imagine an adversary who — playing the role of the verifier — interacts with the honest prover, say, n times. In doing so, the adversary may send whatever values $r_1, \ldots, r_n$ it likes to the prover, who responds by signing these values. Then, the adversary interacts with an honest verifier who sends some value $\tilde{r}$. There are two possibilities:

1. $\tilde{r} \in \{r_1, \ldots r_n\}$. This happens only with probability $n/2^k$. Since n is polynomial in the security parameter k, this probability is negligible.

2. $\tilde{r} \notin \{r_1, \ldots r_n\}$. In this case, impersonation of the prover is equivalent to forging a signature on a "new" (i.e., previously-unsigned) message. By security of the signature scheme, this is infeasible.

[1] Although it is possible to consider the analogous case of private-key identification, the public-key setting is the one more often considered in this context. Thus, when we say "identification scheme" we mean the public-key setting by default.

It is also possible to construct an identification scheme based on any CCA1-secure public-key encryption scheme:

$\underline{P(SK)}$		$\underline{V(PK)}$
	$\xleftarrow{\varepsilon_{PK}(r)}$	$r \leftarrow \{0,1\}^k$
	$\xrightarrow{r'}$	Accept if $r = r'$

A proof of security for this protocol (against active attacks) is slightly more difficult, but not much more so.

Although these schemes are conceptually simple and achieve security against active attacks, there are at least two drawbacks that have motivated the search for other identification schemes: (1) the above schemes leave an undeniable trace that the prover executed the identification protocol. One may want to avoid leaving behind such "evidence". (2) Generally speaking, signature schemes and CCA1-secure encryption schemes are difficult to construct (especially if we limit ourselves to the standard model). Thus, it is natural to wonder whether we can construct identification schemes without relying on these tools. Indeed, as we will briefly mention below, we can go in the reverse direction and build efficient signature schemes (in the random oracle model) from identification schemes of a certain form. In fact, it is fair to say that identification protocols have almost no practical application as identification protocols *per se*. (The reason is that one typically wants identification to be coupled with key exchange, so that communication between the parties can be secured once they have identified each other.) However, identification protocols have found extensive application as building blocks for other primitives, with signature schemes being the primary example.

2 General Paradigms for Constructing Identification Schemes

We show now some general techniques for constructing identification schemes based on proof systems satisfying various additional properties (many of these ideas are due to Feige, Fiat, and Shamir [1]). One immediate idea is to use a *zero-knowledge (ZK) proof of knowledge (PoK)* for a "hard" language. In more detail, consider the following identification protocol based on a one-way function f:

- The public and secret keys are generated as follows: the prover chooses a random $x \in \{0,1\}^k$ (where k is the security parameter) and sets $y = f(x)$. The public key is y and the secret key is x.
- To identify himself to a verifier holding his public key y, the prover gives a ZK PoK (with negligible soundness error) of x s.t. $f(x) = y$. The verifier accepts iff the proof is successful.

We remark that, if we don't care about round complexity, the ZK PoK can be based on the one-way function f and so we get an identification protocol based on any one-way function.

It is not too hard to see that the above gives an identification scheme secure against active attacks. We do not give a formal proof, but instead describe informally how such a proof would proceed.

Theorem 1 *The above identification scheme is secure against active adversaries.*

Proof (Sketch) Assume we have an adversary A attacking the above identification scheme via an active attack, and succeeding with probability $\varepsilon(k)$. We will prove that $\varepsilon(k)$ is negligible. We can view the adversary's attack as follows: first, the adversary receives a public key y, where $y = f(x)$ for random $x \in \{0,1\}^k$. Next, the adversary — playing the role of a verifier who may act in an arbitrary manner — interacts with the honest prover as many times as the adversary likes. We call this stage 1. Finally, the adversary interacts with an honest verifier (holding public key y); we call this stage 2. The adversary succeeds in impersonating the prover (in stage 2) with probability $\varepsilon(k)$, where this probability is taken over y, random coins used by the adversary and the honest prover in stage 1, and the random coins of the adversary and the honest vefifier in stage 2.

Now, consider modifying the above experiment in the following way: in stage 1, instead of having the honest prover P perform a real execution of the protocol with the adversary A in stage 1, we instead run the zero-knowledge simulator Sim (guaranteed for the proof system) in stage 1. Let $\varepsilon'(k)$ denote the probability that A succeeds in impersonating P in stage 2 in this modified experiment. By the properties of the ZK simulator, we can show that the difference $|\varepsilon(k) - \varepsilon'(k)|$ is negligible.

In the previous experiment, we no longer need to know a pre-image of y (i.e., x from the original experiment). This means we can construct an adversary A' (who will internally run both A and Sim) who obtains a value y and then succeeds in giving a valid ZK PoK for this y with probability $\varepsilon'(k)$ (in particular, phase 1 is no longer relevant since A' is simulating all of phase 1 internally and there is no longer any need to interact with an outside prover).

But now we claim that we can use such an A' to invert the one-way function f on a randomly-generated output point y (i.e., $y = f(x)$ for random x) with probability negligibly close to $\varepsilon'(k)$ in case this latter quantity is non-negligible. How do we do this? We simply run the *knowledge extractor* for this proof system with adversarial prover A'. This extractor guarantees (informally) that if $\varepsilon'(k)$ is non-negligible — and, in particular, larger than the soundness error of the proof system — then an inverse of y will be extracted with probability negligibly close to $\varepsilon'(k)$.

In summary, if $\varepsilon(k)$ is non-negligible, we end up with an efficient algorithm (but see the remark below) inverting f with non-negligible probability. Since f is a one-way function, this is a contradiction. Thus, we must have $\varepsilon(k)$ negligible, as desired.

Technical remark: Actually, the algorithm we construct to invert f runs in *expected polynomial time* rather than *strict polynomial time* (this is so since both the ZK simulator and the PoK knowledge extractor may run in expected polymomial time). However, one can show that the existence of an expected polynomial-time algorithm A_1 which inverts f with non-negligible probability implies the existence of a strict polynomial-time algorithm A_2 which inverts f with non-negligible probability (without giving the details, A_2 simply runs A_1 but aborts if it runs for "too long"). Thus, the above proof does indeed lead to a contradiction. □

The above construction can be simplified if we are content to achieve security against a passive adversary. In this case, we only need a proof system which is *honest verifier* zero-knowledge (as well as being a proof of knowledge). Since in the case of a passive attack the

adversary (by definition) is limited to eavesdropping on executions of the protocol between the prover and an *honest* verifier, honest-verifier zero-knowledge suffices in this case.

2.1 On Using Witness Indistinguishability

A natural question is whether we can further simplify things by using a *witness indistinguishable* (WI) proof of knowledge (in the case of an active adversary). As we will see, the answer is *yes*; however, it requires some changes to the protocol as stated above. In particular, recall that a WI proof guarantees nothing in case there is only one witness (since then witness indistinguishability is trivial to achieve); so, we need to modify the protocol so that two or more witnesses are available. The idea is similar to that used by Feige-Shamir (cf. the previous lecture) in constructing their constant-round ZK argument of knowledge.

Let f be a one-way function. The protocol is as follows:

- The public and secret keys are generated by having the prover chooses *two* random points $x_1, x_2 \in \{0,1\}^k$ and setting $y_1 = f(x_1)$ and $y_2 = f(x_2)$. The public key is (y_1, y_2) and the secret key is x_1. (As we will see, x_2 is not needed.)
- To identify himself to a verifier holding his public key (y_1, y_2), the prover gives a WI PoK (with negligible soundness error) of x such that either $f(x) = y_1$ or $f(x) = y_2$.

As before, we provide only a sketch of security for the above protocol.

Theorem 2 *The above identification scheme is secure against active adversaries.*

Proof (Sketch) Assume we have an adversary A attacking the above identification scheme via an active attack, and succeeding with probability $\varepsilon(k)$. We will prove that $\varepsilon(k)$ is negligible. We can view the adversary's attack as follows: first, the adversary receives a public key (y_1, y_2), where $y_i = f(x_i)$ for random $x_i \in \{0,1\}^k$. Next, the adversary — playing the role of a verifier who may act in an arbitrary manner — interacts with the honest prover as many times as the adversary likes. We call this stage 1. Finally, the adversary interacts with an honest verifier (holding public key (y_1, y_2)); we call this stage 2. The adversary succeeds in impersonating the prover (in stage 2) with probability $\varepsilon(k)$, where this probability is taken over (y_1, y_2), random coins used by the adversary and the honest prover in stage 1, and the random coins of the adversary and the honest vefifier in stage 2.

If $\varepsilon(k)$ is not negligible, then by negligible soundness error of the PoK we can extract a witness from the adversary in stage 2 of its attack with probability negligibly close to $\varepsilon(k)$. In particular, we extract either a "type 1 witness" (i.e., an x such that $f(x) = y_1$) or a "type 2 witness" (i.e., an x such that $f(x) = y_2$) with non-negligible probability. Let $\varepsilon_1(k)$ denote the probability that we extract a type 1 witness, and similarly for $\varepsilon_2(k)$. The above shows that either $\varepsilon_1(k)$ or $\varepsilon_2(k)$ is not negligible (or possibly both are). We show that either of these lead to an efficient procedure for inverting f with non-negligible probability, and hence a contradiction.

If $\varepsilon_2(k)$ is not negligible, we can almost immediately derive our desired contradiction. Given a point y which is equal to $f(x)$ for a random x, simply choose $x_1 \in \{0,1\}^k$ at random and set the public key equal to (y_1, y). We can now interact with the adversary exactly as in the real experiment (recall that in the real protocol, the prover does not use

x_2 anyway) and extract a type 2 witness with probability exactly $\varepsilon_2(k)$. (Note that this will exactly be an inverse of y.) Since $\varepsilon_2(k)$ is non-negligible by assumption, we have the desired contradiction.

In case $\varepsilon_1(k)$ is non-negligible, the only thing we need to do is to "switch" from using x_1 to using x_2. In particular, we now construct the public key (y_1, y_2) in the same way as before, but throw away x_1 instead of x_2. Also, when interacting (as a prover) with the adversary A (acting as a verifier) we use witness x_2 instead of witness x_1. (We extract a witness from the adversary in stage 2 as before.) By witness indistinguishability of the proof system, one can show that the probability $\varepsilon_2'(k)$ of extracting a type 2 witness is negligibly-close to the probability of extracting a type 2 witness in the original experiment (i.e., $|\varepsilon_1(k) - \varepsilon_2(k)|$ is negligible). Thus, $\varepsilon_2'(k)$ is non-negligible. That this leads to a contradiction can be proven in exactly the same way as above.

Technical remark: The same issue arises here as in the previous proof, in that the adversary we construct to invert f runs in expected polynomial time rather than strict polynomial time (due to the possible expected-polynomial running time of the knowledge extractor). We may deal with this in exactly the same way as before. □

In a later lecture, we will show how to *efficiently* instantiate the protocols of this section and the previous section based on a specific number-theoretic assumption.

2.2 On Avoiding Proofs of Knowledge

The constructions in the previous sections rely in an essential way on the use of proofs of knowledge (examining the previous proofs, we see that we need this property in order to be able to extract the desired inverse of f). We show here that it is possible to base identification schemes on zero-knowledge *proofs*.[2] We will only sketch the idea, and leave the proof to the reader. As far as we are aware, this idea originates in [2].

The concept is quite simple: instead of proving knowledge of some hard-to-compute function (namely, $f^{-1}(y)$ as in the previous sections), we prove membership in some hard-to-decide language. To be specific, let G be a pseudorandom generator (PRG) stretching its input by k bits (we know that this can be constructed based on any one-way function). Then consider the following identification protocol:

- The public and secret keys are generated by choosing a random $x \in \{0,1\}^k$ and then setting $y = G(x)$. The public key is y and the secret key is x.
- To identify himself to a verifier holder his public key y, the prover gives a ZK proof that $y = G(x)$ for some x (i.e., that y is in the image of G, or that y is pseudorandom).

One can show that the above identification scheme is secure against active adversaries.

[2] We believe it is possible to also base identification schemes on witness indistinguishable proofs; however, the idea of the previous section applied to the protocol of the present section will not work, and a more complex construction is required.

References

[1] U. Feige, A. Fiat, and A. Shamir. Zero-Knowledge Proofs of Identity. *J. Crypto* 1(2): 77–94 (1988).

[2] J. Katz and N. Wang. Efficiency Improvements for Signature Schemes with Tight Security Reductions. ACM CCCS 2003.

CMSC 858K — Advanced Topics in Cryptography	May 4, 2004
Lecture 26	
Lecturer: Chiu Yuen Koo	*Scribe(s): (none)*

1 Introduction

In this lecture, we study the Byzantine Agreement problem, defined as follows: consider a network of n processors, where each pair of processors can communicate (this is the so-called "point-to-point" model). Furthermore, at most t processors within this network may be faulty; a faulty processor may exhibit arbitrary behavior. (We also assume that the behavior of these faulty processors may be coordinated by a single adversary, and sometimes do not place any computational restrictions on this adversary.) Initially, each processor has an input value p_i; this group of processors then runs a protocol which results in each (non-faulty) processor deciding on a value p_i^*. Besides requiring that the protocol terminate, a *Byzantine agreement* protocol also satisfies the following (as long as no more than t processors are faulty):

Agreement All non-faulty processors decide on the same value. I.e., if i, j are non-faulty then $p_i^* = p_j^*$.

Validity If the initial inputs of all non-faulty players are identical, then all non-faulty players decide on that value. I.e., if $p_i = p^*$ for all non-faulty players i, then $p_i^* = p^*$ for all non-faulty players i.

We remark that either one of these properties is trivial to achieve on their own (agreement by having all non-faulty processors always output "0", and validity by having each processor i output $p_i^* = p_i$); the tricky part is to guarantee that they both hold in tandem.

The Byzantine agreement problem was first formulated by Lamport, Pease, and Shostak [4, 2], and was motivated by fault-tolerant systems where a set of independent processors are required to come to an exact agreement in presence of faulty processors. Examples include synchronization of internal clocks or agreement on sensor readings. From a cryptographic perspective, Byzantine agreement is a central primitive used within multi-party computation protocols, where now a certain fraction of adversarial processors may be actively trying to prevent agreement among the honest processors.

1.1 Broadcast

A broadcast channel is (as one might expect) a channel to which any player may send a value which is then received by all other players (we also assume that players can tell which player sent the value). Note that if a broadcast channel is available, then the Byzantine agreement problem is trivial (for $t < n/2$): each player simply broadcasts their value p_i and then honest players decide on the majority value. Unfortunately, most real-world systems (at best) only guarantee "point-to-point" communication as we have described above.

However, it is worthwhile to consider broadcast as a *functionality* (rather than as simply an atomic primitive); doing so, we come up with the following definition: Assume an n-party network with point-to-point channels, as above. We now assume a distinguished party s within this network, called *the sender*, who initially holds an input value p_s. As before, we will allow up to t parties within the network (the dealer possibly included) to be faulty. The processors in the network then run a protocol which results in each non-faulty player deciding on a value p_i^*. In addition to requiring that the protocol terminate, a *broadcast* protocol also satisfies the following (as long as no more than t players in total are faulty):

Agreement All non-faulty players decide on the same value; i.e., $p_i^* = p_j^*$ for all non-faulty i, j.

Correctness If the dealer is honest, then all non-faulty players decide on p_s. I.e., if the dealer is non-faulty then $p_i^* = p_s$ for all non-faulty players i.

(Note that broadcast may now be implemented by a *protocol*, rather than only by an atomic "broadcast channel".) We have noted above that if we can achieve broadcast and $t < n/2$, then we can achieve Byzantine agreement. In fact, the opposite direction also holds (for any t, although as defined above Byzantine agreement is not meaningful when $t \geq n/2$), as the following protocol shows: first, the dealer sends his value p_s to each other player (using a point-to-point channel). Next, the players run a Byzantine agreement protocol and decide on the result. It is not hard to see that agreement and correctness both hold.

Because of this equivalence, we are free to focus on either Byzantine agreement or broadcast. In the remainder of this lecture, we focus on broadcast. A key results in this area is the following, which we will prove in this and the next lecture:

Theorem 1 *Broadcast (or Byzantine agreement) is possible iff $t < n/3$ (or $t \geq n - 1$, which is a trivial case). The possibility result holds even for computationally-unbounded adversaries who coordinate the actions of all faulty players. The impossibility result holds even for computationally-bounded adversaries (as long as the adversary is allowed to run for essentially the same amount of time as honest processors), assuming no prior set-up phase is allowed.*

This result was first proved in [4]. A few comments are in order:

1. We have not yet said anything about protocol efficiency, and indeed the original protocol for Byzantine agreement (when $t < n/3$) ran in exponential time. Of course, we ultimately want a protocol that runs in polynomial time. Since the initial work of [4, 2], much further work has focused on constructing polynomial-time protocols, and then optimizing the number of rounds/messages/etc.

2. The remark at the end of the theorem will be discussed later in the lecture. As a preview, note that one possibility for a set-up phase is to assume a PKI such that each player i (including the faulty ones) has established a public/secret key pair (PK_i, SK_i) for a digital signature scheme, with the same value PK_i known to all other players.

3. Other work in this area has focused on what can be achieved within different communication models (say, where a channel does not necessarily exist between every pair of players, etc.).

2 The Impossibility Result

In this section, we prove the "only if" part of the theorem; i.e., we prove:

Theorem 2 *Broadcast (and hence Byzantine agreement) is not achievable when the number of faulty player t satisfies $n - 1 > t \geq n/3$. (Note that $t \geq n - 1$ implies that the number of honest players is at most 1, in which case the whole problem becomes trivial.)*

Proof We first focus on the case $n = 3$, $t = 1$, and then prove the case for general n. Call the sender s and label the two other players "1" and "2". In the real network, each of these players is connected with the other two. But imagine now that we make copies $s', 1', 2'$ of each of these players and arrange them in a hexagon as follows:

$$\begin{array}{ccccc} s & \leftrightarrow & 1 & \leftrightarrow & 2 \\ \updownarrow & & & & \updownarrow \\ 2' & \leftrightarrow & 1' & \leftrightarrow & s' \end{array}$$

Now, assume toward a contradiction that we have some (possibly randomized) protocol P for broadcast. This protocol completely specifies what each player $s, 1, 2$ should do given its initial input (in the case of s) and the messages it receives from the other parties. Consider the case when the players above execute the protocol honestly, communicating as in the diagram, *but where s has input 0 and s' has input 1.* We claim the following:

1. Let us look at things from the point of view of players 1 and 2. The key is to realize that, from their perspective, the combined actions of s and s' represent possible adversarial activity of the sender s in the original network. (In particular, the "real" sender s in the original, 3-player network can simulate the actions of $1'$, $2'$, s with input 0, and s' with input 1.) Furthermore, from this viewpoint players 1 and 2 are acting completely honestly. Since, by assumption, P is a protocol for broadcast, we must then have that 1 and 2 output the same value. Letting p_1 (resp., p_2) represent the output of player 1 (resp., 2), we then have $p_1 = p_2$.

2. Now, look at things from the point of view of players s and 1. Here, the key is to realize that, from their perspective, the combined actions of players 2 and $2'$ represent possible malicious activity of player 2 in the original network. (Again, a malicious player 2 in the "real", 3-player network can simulate the actions of $2, 2', s, s'$.) But, again, players s and 1 are acting completely honestly. Since P is a broadcast protocol, it must be the case that player 1 outputs the initial input value of s. That is, $p_1 = 0$.

3. However, an exactly symmetric argument with respect to $s', 2$ shows that player 2 must output the initial input of s'; i.e., $p_2 = 1$.

The above give the desired contradiction (namely, we require $p_1 = p_2$ but $p_1 = 0$ while $p_2 = 1$), showing that the claimed broadcast protocol cannot exist.

We now prove the claim for the case of n a multiple of 3 (although a small modification of what we say extends to give a proof for arbitrary n). We do so by reducing in to the case $n = 3$. Namely, we show that if there exists a broadcast protocol for $n \leq 3t$ then we can construct a broadcast protocol for $n = 3$, $t = 1$.

So, assume we have a broadcast protocol P for n players (n a multiple of 3) and secure against t malicious parties, with $t \geq n/3$. We assume for simplicity that player 1 is the sender. Construct broadcast protocol P' for $n' = 3$ players $1', 2', 3'$ as follows (again, player $1'$ will be the sender): The basic idea is that player $1'$ will simulate players 1 through $n/3$; player $2'$ will simulate players $n/3 + 1$ through $2n/3$; and player $3'$ will simulate players $2n/3 + 1$ through n. In more detail, focusing on player $1'$ (actions of players $2'$ and $3'$ are defined similarly): If player $1'$ has input b, it runs player 1 (internally) with input b. Whenever players $i, j \in [1, n/3]$ in protocol P want to send a message to each other, player $1'$ simply simulates this internally. When player $i \in [1, n/3]$ wants to send a message m to player $j \in [n/3 + 1, 2n/3]$, player $1'$ sends the message (i, j, m) to player $2'$. When $j \in [2n/3+1, n]$, player $1'$ sends a similar message to player $3'$. When player $1'$ receives a message (j, i, m) from player $2'$ with $j \in [2n/3, n]$, player $1'$ internally passes message m to (internal) player i "from" player j. When such a message comes from player $3'$, player $1'$ acts appropriately. Players $2'$ and $3'$ output whatever is output (internally) by any of the players they are simulating.

Assume P is secure for $t \geq n/3$. We then claim that P' is secure for $t = 1$. To see this, note that the actions of any one compromised player in protocol P' can be simulated by the compromise of at most $n/3$ players in protocol P (namely, the $n/3$ players in P assigned to the corrupted player in P'). Consider the case when $1'$ is compromised in P', and hence players 1 through $n/3$ are compromised in P. Since agreement holds for P, we know that players $n/3 + 1$ through n all output the same value, and so players $2'$ and $3'$ will output the same value and agreement holds for P'. Next, consider the case when $2'$ is compromised in P', corresponding to compromise of players $n/3 + 1$ through $2n/3$ in P (the situation is analogous when player $3'$ is compromised). Since correctness holds for P, we know that players $2n/3+1$ through n output the initial input value of player 1. Since this latter value is the same as the initial input value of player $1'$, we have that player $3'$ outputs the initial input value of player $1'$ and hence correctness holds for P'. ■

In the next lecture, we will show a protocol for Byzantine agreement/broadcast for n players and $t < n/3$ faulty players.

3 "Authenticated" Broadcast

We now return to the remark about circumventing the impossibility result proved above if we allow an initial "set-up" phase. In particular, we will assume a PKI has been established such that each player i has a public-/secret-key pair (PK_i, SK_i) and (i, PK_i) is known to all other players. (We stress that *every* player holds the same PK_i for player i — this is crucial.) We will also assume a computationally-bounded adversary who cannot forge a signature (with non-negligible probability) on any previously-unsigned message with respect to the public key of any of the honest players.

First, it is worthwhile to see where the proof of the impossibility result from the previous

section breaks down in this case. Looking at the case $n = 3$, $t = 1$, we see that a crucial element in the proof is the ability of corrupted players to simulate the actions of non-corrupted players (for example, we required that a malicious s can simulate the actions of "honest" $1'$ and $2'$). But when the players might sign messages (and we assume a computationally-bounded adversary) *it is no longer necessarily possible for corrupted players to simulate the actions of honest players* and the proof break down.

We now show a protocol for "authenticated" broadcast when a PKI is established, as described above. Our description of the protocol is based on [1]. Again, we will assume that player 1 is the sender. We first introduce some notation: a message is called (v, i)-*authentic for* j if it has the form $(v, p_1, \sigma_{p_1}, \ldots, p_i, \sigma_{p_i})$ where $v \in \{0, 1\}$, $p_1 = 1$, all p_i are distinct, $j \notin \{p_1, \ldots, p_i\}$, and, for all i, the signature σ_{p_i} is a valid signature with respect to PK_{p_i} of the string $(v, p_1, \sigma_1, \ldots, p_{i-1}, \sigma_{i-1})$. When it is obvious which player j we are talking about, we will simply call this a (v, i)-authentic message. The protocol proceeds as follows:

Round 1: Party 1 signs its input v and sends $(v, 1, \sigma_1)$ to all parties. (Note that when player 1 is honest, this message is $(v, 1)$-authentic for all $j \neq 1$.)

Rounds 2 through $n - 1$**:** Each player j acts as follows in round i: for each $v \in \{0, 1\}$, if player j has received a $(v, i-1)$-authentic message $(v, p_1, \sigma_{p_1}, \ldots, p_{i-1}, \sigma_{i-1})$ in the previous round, then it signs this message and sends $(v, p_1, \sigma_{p_1}, \ldots, p_{i-1}, \sigma_{i-1}, j, \sigma_j)$ to all other players.

Note that each player sends at most 2 messages per round to all players (one for each possible value of v), even if it has received many different $(v, i-1)$-authentic messages.

Conclusion: Each party j decides on its output as follows:

If party j has ever received both a $(0, i)$-authentic message and a $(1, i')$-authentic message (where possibly $i = i'$), then it outputs the default value 0. (Note that when this occurs the sender must be cheating [assuming the security of the signature scheme], since it has issued signatures on two different values.)

If player j has ever received a (v, i)-authentic message, but never received a $(\bar{v}, i')$-authentic message, then it outputs v.

If player j has never received a (v, i)-authentic message for any value of v, then it outputs the default value 0. (Again, this situation implies that the sender must be dishonest, since it was supposed to send out an authentic message in round 1.)

We claim that the above is a secure broadcast protocol. Correctness is easy to see: if the sender is honest and has initial input v, then every player receives a $(v, 1)$-authentic message in the first round; assuming the security of the signature scheme, players will never receive a $(\bar{v}, i)$-authentic message (since, in particular, this would require forging the signature of player 1) and so all honest players will output v.

Agreement is a bit more difficult to verify. The first thing we may notice is that if any honest player j receives a (v, i)-authentic message with $i < n - 1$, then every honest player j' will receive a $(v, i + 1)$-authentic message in the following round (because j will send such a message to everyone). It only remains to see what happens when an honest

player receives a $(v, n-1)$-authentic message in the last round. In this case, the fact that this message is $(v, n-1)$-authentic means that every other player has signed this message, and so every honest player has observed a (v, i)-authentic message in some previous round (again, assuming the security of the signature scheme so that the signatures of honest players cannot be forged). Putting this together, we have that if any honest player has seen a (v, i)-authentic message by the end of the protocol, then *every* honest player has seen a (v, i')-authentic message by the end of the protocol. It is thus clear that agreement holds.

References

[1] O. Goldreich. *Foundations of Cryptography, vol. 2: Basic Applications*. Cambridge University Press, 2004.

[2] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3): 382–401 (1982).

[3] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., 1996.

[4] M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *J. ACM*, 27(2): 228–234 (1980).

CMSC 858K — Advanced Topics in Cryptography		May 6, 2004
Lecture 27		
Lecturer: Chiu Yuen Koo	*Scribe(s):*	*Omer Horvitz* *Zhongchao Yu* *John Trafton* *Kavitha Swaminathan*

1 Introduction

In a previous lecture, we defined the Byzantine agreement/broadcast problems and showed that there is no protocol solving these problems when the fraction of corrupted players is 1/3 or larger. Today, we prove the converse by showing a protocol for broadcast (and hence Byzantine agreement; cf. the previous lecture) when the fraction of corrupted players is less than 1/3. The protocol we will show is called the *Exponential Information Gathering (EIG)* protocol, and was essentially the first known protocol for this task. (See [2, 4, 1, 3].) We stress at the outset that the protocol is not very efficient — as the name suggests, its complexity is exponential in the number of players — but it forms an important feasibility result. Since the protocol was introduced, much work has focused on improving various parameters of the protocol, and fully polynomial Byzantine agreement/broadcast protocols with optimal resilience are now known.

2 The EIG Protocol

Let n be the number of players, and let t be the upper-bound on the number of malicious players (for simplicity, let the number of faults be exactly t). We show how to achieve broadcast when $t < n/3$. To gain intuition, we describe how the protocol works for the case of four players, when only one is assumed to be faulty. Let the players be denoted A, B, C, D, and assume the sender A holds a value v. A begins by sending v to B, C, D, who proceed to send the value they received from A to each other. At this point, each player has a value that it received from A and values that the other two players report to have received from A. Each player maintains this information in a tree, as depicted in Fig. 1. Each player $i \in \{B, C, D\}$ stores the value it received from A in the root of its tree, and the value that node $j \in \{B, C, D\} \setminus \{i\}$ says it received from A in node Aj. Each player $i \in \{B, C, D\}$ decides on the majority value of the latter three values (i.e., the values stored at the leaves of the tree). If no majority value exists, then the players decide on some default value v_0.

We proceed with the analysis: if A is non-faulty then one of B, C, D *is* faulty; without loss of generality, assume it is B. Looking at the trees maintained by C and D, we see that in each of their trees the value stored at leaves AC and AD is exactly v, the initial input of the sender A. Thus, regardless of what B sends to C and D (i.e., regardless of what appears at leaf AB in C's tree and lead AB in D's tree), C and D will decide on v, ad required.

On the other hand, if A is faulty then B, C, and D are not faulty. Let v_b, v_c, v_d denote the values that A sent to B, C, D, respectively. Looking at the trees maintained by B, C, and D, we see that each will have the value v_b stored at leaf AB, the value v_c stored at leaf

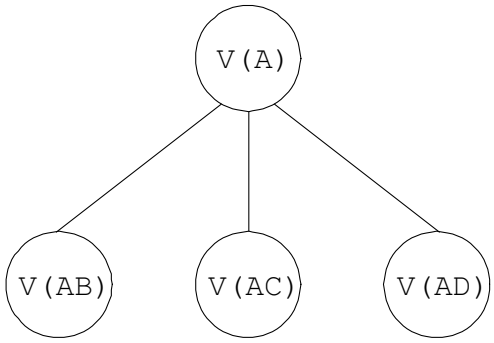


Figure 1: Information Gathering Tree for $n = 4, k = 1$.

AC, and the value v_d stored at leaf AD. Thus, they will all decide on some common value (whatever that value may be) as required.

More generally, assume the network consists of n players $A, B, C, \ldots$, of which $t < n/3$ are faulty (equivalently, $n > 3t$). The EIG algorithm, described below, involves the maintenance of a tree of height t (i.e., having $t+1$ levels) by each player. The root of the tree is labeled with the sender A. Every internal node labeled ℓ (where ℓ is a string) has one child for each player s that does not appear in ℓ; the label of this child is ℓs, the concatenation of the label of the parent with the name of the aforementioned player. The node labeled ℓs is said to *correspond* to player s. For example, if we have players A, B, C, D, E, F, G, then the root has children AB, AC, AD, AE, AF, AG; the node labeled AE has children AEB, AEC, AED, AEF, AEG; and nodes AEF, AEG are said to correspond to F, G, respectively.

The protocol proceeds in $t + 1$ rounds. In the first round, A sends its input v to all other players. Each player stores the value it received from A in the root of its tree. In each subsequent round, each player (except A, who no longer needs to take part in the protocol) broadcasts the most-recently-filled level of its tree. Upon receiving these messages, each player P fills the next level of its tree by storing at node ℓX the value that player X claims to have stored at node ℓ in its own tree.[1] Intuitively, player P stores in node $A \cdots Y X$ the value that "X says that Y says ... that the sender A said". We refer to the value stored at node ℓ in P's tree as $v_P(\ell)$. A generic information-gathering tree is depicted in Fig 2 (the identity of the particular player maintaining the tree is omitted).

After completion of the $t + 1$ rounds, we define a *reduced value* for each node. For a player P and a node labeled σ, the reduced value $v'_P(\sigma)$ is defined as follows:

- If σ is a leaf, then $v'_P(\sigma) = v_P(\sigma)$.
- If σ is not a leaf, then $v'_P(\sigma)$ is the majority value of the *reduced values* of its children (in P's tree). If no majority exists, $v'_P(\sigma)$ is assigned the default value v_0.

Each player computes and decides on the reduced value of the root of its tree.

[1] *Note*: this includes messages that player P sends "to itself". Of course, no such message is actually sent but P can certainly simulate the sending of such a message.

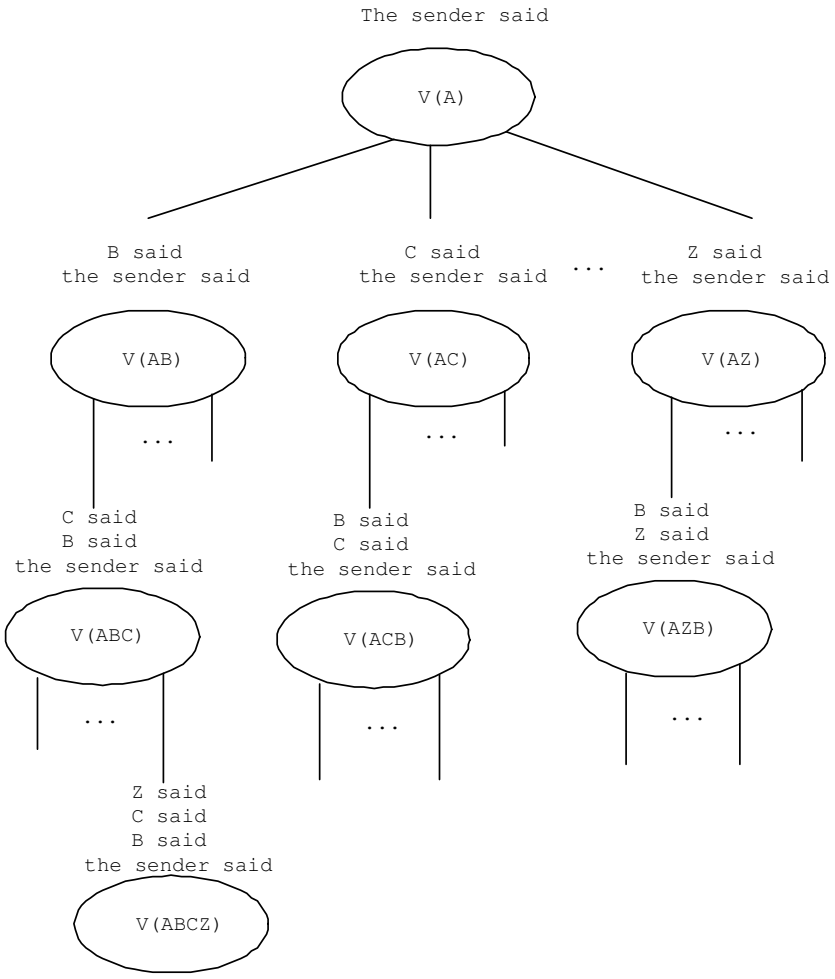


Figure 2: A Generic Information Gathering Tree.

We now prove the correctness of this protocol. Each (honest) player maintains a tree with $t+1$ levels (numbered 1 through $t+1$), and each node at level k has $n-k$ children. It follows that in the tree maintained by each player, every internal node has at least $n-t \geq 2t+1$ children (of which at most t are faulty). The correctness of the protocol follows from the following lemmas:

Lemma 1 *If a node σ corresponds to a non-faulty player, then $v_P(\sigma) = v_Q(\sigma)$ for all non-faulty players P, Q.*

Proof Let $\sigma = \sigma' R$, where R is a non-faulty player. The lemma follows easily from the fact that R sends the same value $v_R(\sigma')$ to all other players. (Note that the lemma holds even when $R \in \{P, Q\}$.) ■

Lemma 2 *If a node σ corresponds to a non-faulty player, then there is a value v such that $v'_P(\sigma) = v_P(\sigma) = v$ for any non-faulty player P.*

Proof By backward induction on the level of σ in the information-gathering tree.

Base case: Here, σ is a leaf. Then $v'_P(\sigma) = v_P(\sigma)$ for any honest P by the definition of reduced values. Moreover, Lemma 1 shows that for any non-faulty player Q we have $v_P(\sigma) = v_Q(\sigma)$, and hence $v'_Q(\sigma) = v'_P(\sigma)$ as well.

Inductive step: Assume the claim holds for all nodes at level $k+1$, and consider a node σ at level k. Lemma 1 shows that all non-faulty processes P have the same value $v_P(\sigma)$. Call this value v. Each non-faulty process will send v to all other processes in round $k+1$, so $v_Q(\sigma P) = v$ for all non-faulty P, Q. The inductive hypothesis now implies that $v'_Q(\sigma P) = v_Q(\sigma P) = v$ for all non-faulty P, Q. Since a majority of the children of σ are non-faulty (by the argument above), this implies that $v'_Q(\sigma) = v = v_Q(\sigma)$ for all non-faulty Q.

■

If A is honest, the above lemma immediately implies that all non-faulty players decide on the sender's initial value. All that is left to be shown is that all non-faulty players reach agreement when the sender is faulty. We prove a slightly stronger lemma:

Lemma 3 *If all paths from a node σ to a leaf contain at least one node corresponding to a non-faulty player, then $v'_P(\sigma) = v'_Q(\sigma)$ for all non-faulty players P, Q.*

Proof By backward induction on the level of σ in the information-gathering tree.

Base case: σ is a leaf. The claim reduces to Lemma 2.

Inductive step: Assume the claim holds for all nodes at level $i+1$, and consider a node σ at level i.

- If σ corresponds to a non-faulty node, the claim reduces to Lemma 2.
- If σ corresponds to a faulty node, consider a child σ' of σ. It must be the case that all paths from σ' to a leaf contain at least one node corresponding to a non-faulty player. By the induction hypothesis, $v'_P(\sigma') = v'_Q(\sigma')$ for all non-faulty players P, Q. By the definition of the reduced value, it follows that $v'_P(\sigma) = v'_Q(\sigma)$ too.

■

Notice that every path from the root of the information-gathering tree to a leaf contains a node corresponding to a non-faulty player, because the length of any such path is $t+1$ and there are at most t faulty players. Applying Lemma 3 to the root completes our proof.

We sum up the result with the following theorem:

Theorem 4 ($n > 3t$ is sufficient for Byzantine agreement) *There exists a protocol that achieves Byzantine agreement when $n > 3t$.*

References

[1] A. Bar-Noy, D. Dolev, C. Dwork and R. Strong. Shifting Gears: Changing Algorithms on the Fly to Expedite Byzantine Agreement. In *Proc. 6th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 42–51, 1987.

[2] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* 4(3): 382–401 (1982).

[3] Nancy Lynch. Distributed Algorithms. Morgan Kaufmann, 1996.

[4] M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *J. ACM* 27(2): 228–234 (1980).

CMSC 858K — Advanced Topics in Cryptography May 11, 2004

Lecture 28

Lecturer: Jonathan Katz *Scribe(s): Nagaraj Anthapadmanabhan, Alvaro Cardenas*

1 Introduction

In a previous class (Lecture 25), we showed how to construct an identification scheme which is secure against a *passive adversary* using an *Honest-Verifier Zero-Knowledge Proof of Knowledge* (HVZK-PoK). We also showed that it is possible to construct an Identification Scheme secure against an *active adversary* using a *Witness Indistinguishable Proof of Knowledge* (WI-PoK). In this lecture, we will construct efficient proof systems with these properties, and thus efficient identification schemes, based on the discrete logarithm assumption. We will also see how the resulting identification schemes can be converted into signature schemes.

2 Security Against Passive Adversaries

We refer to Lecture 25 for the definitions of identification schemes and their security against passive adversaries. We also refer there for a proof that an identification scheme can be constructed using any one-way function f and an HVZK-PoK of a value x such that $f(x) = y$ (where y is included in the prover's public key). Here, we merely show a specific (efficient) HVZK-PoK for the particular case when the one-way function f is the discrete exponentiation function (and the hardness of inverting f is therefore equivalent to the discrete logarithm assumption that we have seen previously). To establish some notation, let $\mathbb{G}$ be a cyclic group of prime order q and let g be a fixed generator of $\mathbb{G}$. We will assume that $\mathbb{G}, g$, and q are publicly known. If $y = g^x$ then we let $x \stackrel{\text{def}}{=} \log_g y$.

The setup is as follows (cf. Lecture 25): the verifier knows the prover's public key y, while the prover has a secret key x such that $y = g^x$. The following protocol due to Schnorr [4] allows the prover to prove to the verifier that he indeed knows x:

$$\begin{array}{lcl} & \mathbb{G}, q, g, y & \\ \underline{\mathcal{P}(x)} & & \underline{\mathcal{V}} \\ r \leftarrow \mathbb{Z}_q & A = g^r \longrightarrow & \\ & \longleftarrow\ c & c \leftarrow \mathbb{Z}_q \\ & b = (cx + r) \bmod q \longrightarrow & y^c A \stackrel{?}{=} g^b \end{array}$$

Let us first show that the above scheme is correct. If both players act honestly and $y = g^x$, then we have:

$$g^b = g^{cx+r} = g^{cx} g^r = y^c A,$$

where we use the fact that the order of the group is q, and so exponents are reduced modulo q (i.e., $g^{cx+r} = g^{cx+r \bmod q}$). We now prove that the above protocol satisfies the desired properties.

Theorem 1 *The protocol above is both honest-verifier zero-knowledge as well as a proof of knowledge.*

Proof We prove that the protocol is honest-verifier zero-knowledge by showing a simulator. The simulator proceeds as follows: it first chooses random $c, b \in \mathbb{Z}_q$ and then sets $A = g^b y^{-c}$. It outputs the transcript (A, c, b). We claim that the distribution of transcripts as output by this simulator is *identical* to the distribution of transcripts of real executions of the protocol (with an honest verifier). To see this, note that c is uniform in $\mathbb{Z}_q$ in both cases. Furthermore, the distributions of both real and simulated transcripts have A uniform in the group, independent of c. (This is clear for real transcripts since g is a generator and r is chosen uniformly at random from $\mathbb{Z}_q$, independent of c. For simulated transcripts, this is true since the value g^b is uniform in $\mathbb{G}$ and hence — for any c — the value $g^b y^{-c}$ is uniform in $\mathbb{G}$.) Finally, in both cases b is completely determined by A and c as the unique value satisfying $b = (c \cdot \log_g y + r) \bmod q$ (note that $\log_g y$ is well-defined, even if we cannot compute it efficiently). This completes this part of the proof.

To show that the protocol is a proof of knowledge, one needs to show a knowledge extractor satisfying the technical conditions hinted at (but not defined formally) in earlier lectures. Assume an adversarial prover who has some probability λ of successfully executing the protocol for a particular value of y. Note that this probability is only over the random challenge c sent in the second round. Before describing the extractor, we define some terminology: say the adversary *succeeds* on challenge c (assuming A is already fixed) if the adversary responds to this challenge with a b such that $y^c A = g^b$. Otherwise, say the adversary *fails*. We construct the extractor as follows:

> Receive some A from the adversary
> choose $c_1 \leftarrow \mathbb{Z}_q$ and send c_1 to the adversary
> if the adversary fails on c_1, halt
> otherwise, for $i = 1$ to q do:
> choose $c_2 \leftarrow \mathbb{Z}_q \setminus \{c_1\}$
> if the adversary succeeds on c_2, compute $\log_g y$ as described below and halt
> if $g^i = y$, output i and halt

To complete the description, we describe how to compute $\log_g y$ when the extractor finds c_1, c_2 such that the adversary succeeds for both. In this case, we have A, c_1, c_2, b_1, b_2 such that $y^{c_1} A = g^{b_1}$ and $y^{c_2} A = g^{b_2}$. Dividing, we obtain:

$$y^{c_1 - c_2} = g^{b_1 - b_2},$$

and hence $\log_g y = (b_1 - b_2)(c_1 - c_2)^{-1} \bmod q$. Note that we can efficiently compute the latter since we have b_1, b_2, c_1, c_2, q, and $c_1 \neq c_2$ so $(c_1 - c_2)^{-1}$ exists.

Analyzing the extractor in its totality, we see that it computes the correct value for $\log_g y$ whenever the adversary succeeds on c_1. By assumption, this occurs with probability exactly λ. The only thing left to argue is that the extractor runs in expected polynomial

time. To see this, we consider two possibilities: $\lambda > 1/q$ and $\lambda = 1/q$ (if $\lambda < 1/q$ then $\lambda = 0$ and the proof is easy). In the first case, say $\lambda = t/q$ for some integer $t > 1$. Now, the probability that the extractor enters the loop (i.e., the last four lines) is $\lambda = t/q$. The expected number of iterations of the loop, once reached, is $(\frac{t-1}{q})^{-1} = q/(t-1)$. So the expected total running time of the extractor is

$$\mathsf{poly} + \mathsf{poly} \cdot \frac{t}{q} \cdot \frac{q}{t-1} = \mathsf{poly} + \mathsf{poly}.$$

(Where poly in the above refers to some arbitrary polynomial in some implicit security parameter which determined the size of q.) In case $\lambda = 1/q$ the expected number of iterations of the loop is (at worst) q; however, the probability of entering the loop in the first place is $1/q$ and so the expected total running time is:

$$\mathsf{poly} + \mathsf{poly} \cdot \frac{1}{q} \cdot q = \mathsf{poly} + \mathsf{poly}.$$

In either case, then, the extractor runs in expected polynomial time. ■

3 Security Against Active Adversaries

As discussed in Lecture 25, to obtain security against active adversaries we can use a *witness-indistinguishable* proof of knowledge. Recall also that this only helps if there is more than one witness to speak of; for this reason we will now let the prover's secret be a *representation* of some value y with respect to *two* generators g, h. In more detail, if $g, h \in \mathbb{G}$ are generators we say that (x_1, x_2) is a representation of y if $g^{x_1} h^{x_2} = y$. Note that for any $y \in \mathbb{G}$ and any $x_1 \in \mathbb{Z}_q$ there is a unique $x_2 \in \mathbb{Z}_q$ such that (x_1, x_2) is a representation of y. In other words, there are q possible different "witnesses" or representations.

Whereas Schnorr's protocol is a proof of knowledge of the discrete logarithm of y (with respect to g), the following protocol due to Okamoto [3] is a proof of knowledge of a representation of y (with respect to g, h).

$\underline{\mathcal{P}(x_1, x_2)}$	$\mathbb{G}, q, g, h, y$	$\underline{\mathcal{V}}$
$r_1, r_2 \leftarrow \mathbb{Z}_q$	$A = g^{r_1} h^{r_2}$ $\longrightarrow$	
	$\longleftarrow$ c	$c \leftarrow \mathbb{Z}_q$
$b_1 = (cx_1 + r_1) \bmod q$		
$b_2 = (cx_2 + r_2) \bmod q$	$b_1,\ b_2$ $\longrightarrow$	$Ay^c \stackrel{?}{=} g^{b_1} h^{b_2}$

Let us first verify correctness. If $g^{x_1} h^{x_2} = y$ then we have:

$$Ay^c = g^{r_1} h^{r_2} (g^{x_1} h^{x_2})^c = g^{r_1 + cx_1} h^{r_2 + cx_2} = g^{b_1} h^{b_2}.$$

We now prove the desired properties of this protocol

Theorem 2 *The protocol above is a witness indistinguishable proof of knowledge.*

Proof The proof of knowledge property is easy to show, along the lines of the proof for the case of Schnorr's protocol. Without going through the details again, we simply show how to extract a representation from two accepting transcripts sharing the same value of A. Thus, assume we have values $A, c, c', b_1, b_2, b_1', b_2'$ such that $Ay^c = g^{b_1}h^{b_2}$ and $Ay^{c'} = g^{b_1'}h^{b_2'}$ but $c \neq c'$. Dividing, we obtain $y^{c-c'} = g^{b_1-b_1'}h^{b_2-b_2'}$ and so $(\frac{b_1-b_1'}{c-c'}, \frac{b_2-b_2'}{c-c})$ is a representation of y (again, we use the fact that $c - c' \neq 0$ so that the necessary inverse exists).

It remains to argue that the protocol is witness indistinguishable. To prove this, we show that for any adversarial verifier $\mathcal{V}^*$ and any two representations (x_1, x_2), (x_1', x_2'), the distribution on the transcripts of an execution of the protocol when the prover uses the first representation is *identical* to the distribution on the transcripts of an execution of the protocol when the prover uses the second representation. First note that the distribution over the first message A sent by the prover is independent of the representation being used. Next, the challenge c may be viewed as a deterministic function of A (since we can imagine fixing the random coins of the dishonest verifier — note that c may be chosen using some arbitrary (poly-time) function, since the adversary may not be following the honest verification procedure), and so the distribution on c is each case will be identical.

It only remains to argue about the final message b_1, b_2. Conditioned on some fixed values of A, c, when the prover is using the first representation this message is distributed according to:

$$\begin{aligned} b_1 &= cx_1 + r_1 \\ b_2 &= cx_2 + r_2, \end{aligned}$$

where r_1, r_2 are uniformly distributed *over representations of* A. (Another way to view this distribution is one in which r_1 is chosen uniformly from $\mathbb{Z}_q$ and then r_2 is the unique value such that (r_1, r_2) is a representation of A.) It is not hard to see that this is equivalent to saying that b_1, b_2 are uniformly distributed over representations of Ay^c. (Namely, b_1 is uniform in $\mathbb{Z}_q$ and then b_2 is the unique value such that (b_1, b_2) is a representation of Ay^c.) Conditioned on the same values of A, c, when the prover is using the *second* representation the final message is distributed according to:

$$\begin{aligned} b_1 &= cx_1' + r_1 \\ b_2 &= cx_2' + r_2, \end{aligned}$$

where r_1, r_2 are uniformly distributed *over representations of* A. But then b_1, b_2 are again distributed uniformly over representations of Ay^c.

The above discussion shows that the protocol is perfectly witness indistinguishable. ■

The above theorem does not quite allow us to directly apply the results from Lecture 25 and claim that Okamoto's scheme is an identification scheme secure against active adversaries. (If we were directly following the paradigm of Lecture 25, then we would have the prover's public key be y_1, y_2 and the prover would give a witness-indistinguishable proof of knowledge of either $\log_g y_1$ or $\log_g y_2$.) However, the same ideas behind the proof of Theorem 2, Lecture 25 can be used to prove this result. We assume the reader is familiar with that proof, and just sketch an outline of the proof here.

Theorem 3 *The protocol above gives an identification scheme secure against active adversaries.*

Proof (Sketch) Given an active adversary $\mathcal{A}$ attacking the scheme, we will use this adversary to compute $\log_g h$ (*not* $\log_g y$ or something similar as in the case of Schnorr's protocol!). We do this as follows:

1. Given input g, h we choose random x_1, x_2 and set $y = g^{x_1} h^{x_2}$. The public key y is given to $\mathcal{A}$ and the secret key is (x_1, x_2). Note that we know a perfectly valid secret key for y!

2. We then interact with $\mathcal{A}$, who will be acting as a dishonest verifier. Note that we can easily simulate the actions of a prover (without any rewinding or any difficulty) since we know a valid representation of y.

3. Once $\mathcal{A}$ is done with the previous stage, it then tries to impersonate the prover. If it succeeds, we run the knowledge extractor for the proof of knowledge to extract a representation (x_1', x_2') for y.

4. Assuming we have extracted some (x_1', x_2') as above, the key claim is that with all but negligible probability we have $(x_1', x_2') \neq (x_1, x_2)$. Why should this be the case? Well, since the protocol is perfectly witness indistinguishable, $\mathcal{A}$ has no idea (in an information-theoretic sense) which representation of y we know, and all valid representations are equally likely from $\mathcal{A}$'s point of view. Since there are q possible representations, $(x_1', x_2') = (x_1, x_2)$ with probability $1/q$, which is negligible.

5. Assuming we have extracted a representation (x_1', x_2') different from (x_1, x_2), we can compute $\log_g h$ by noting that $g^{x_1} h^{x_2} = g^{x_1'} h^{x_2'}$ and so

$$g^{x_1 - x_1'} = h^{x_2' - x_2}.$$

   Thus, $\log_g h = (x_1 - x_1')(x_2' - x_2)^{-1} \bmod q$. (Note that since the representations are different we must have $x_2' \neq x_2 \bmod q$.)

6. Putting everything together, if $\mathcal{A}$ succeeds with non-negligible probability, then we compute $\log_g h$ with non-negligible probability.

■

4 From Identification Schemes to Signature Schemes

In this section, we show how any identification scheme of a certain form can be transformed into a signature scheme in the random oracle model. Assume a 3-round identification scheme in which the challenge sent by an honest verifier in the second round is generated by picking an element uniformly at random from some space. (The technique extends for multi-round protocols but we will deal with the 3-round case here since this is most common and leads to the most efficient signature schemes.) Let A, c, b denote the messages sent in the first,

second, and third rounds, respectively. We transform this protocol into a signature scheme in the following way: the signer's public key is the public key of the identification scheme and the secret key is the secret key of the identification scheme. To sign message m, the signer begins by generating an initial message A just as in the identification scheme. The signer then computes $c = H(A, m)$ where H is a cryptographic hash function modeled as a random oracle. Finally, the signer computes the correct response b to this "challenge" c (using the secret key and its knowledge of how A was generated) and outputs the signature (A, b). Anyone can verify this signature on message m by computing $c = H(A, m)$ and then checking whether (A, c, b) is a valid transcript of the identification protocol with respect to the given public key. Applied to the Schnorr identification protocol, we obtain:

$$
\begin{array}{lcl}
 & PK = (\mathbb{G}, q, g, y) & \\
\underline{\mathsf{Sign}_{SK}(m) \text{ (where } SK = x = \log_g y)} & & \underline{\mathcal{V}(PK, m)} \\
r \leftarrow \mathbb{Z}_q;\ A = g^r & & \\
c = H(A, m);\ b = cx + r & \xrightarrow{A,\ b} & \\
 & & c = H(A, m) \\
 & & y^c A \stackrel{?}{=} g^b
\end{array}
$$

The general transformation described above (i.e., for an arbitrary identification scheme) was first proposed by Fiat and Shamir [2] and is known as the *Fiat-Shamir transformation*. It has since been analyzed rigorously in numerous works. We state the following without proof, and refer the interested reader to [1] for more details and a full proof.

Theorem 4 *When the Fiat-Shamir transformation is applied to any identification scheme (of the appropriate form) which is secure against passive attacks, the resulting signature scheme is existentially unforgeable under adaptive chosen-message attacks in the random oracle model.*

The above theorem is quite nice, in that it shows that an identification protocol secure against a very *weak* form of attack (i.e., a passive attack) suffices to give a signature scheme which is secure in (essentially) the *strongest* sense.

References

[1] M. Abdalla, J. H. An, M. Bellare, C. Namprempre. From Identification to Signatures via the Fiat-Shamir Transform: Minimizing Assumptions for Security and Forward-Security. Eurocrypt 2002.

[2] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. Crypto '86.

[3] T. Okamoto. Provably-Secure and Practical Identification Schemes and Corresponding Signature Schemes. Crypto '92.

[4] C.-P. Schnorr. Efficient Identification and Signatures for Smart Cards. Crypto '89.