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

$$\begin{aligned} \mathsf{Adv}_A(k) &= \left| \Pr[\mathsf{Succ} \mid \mathsf{query}] \Pr[\mathsf{query}] + \Pr[\mathsf{Succ} \mid \overline{\mathsf{query}}] \Pr[\overline{\mathsf{query}}] - \frac{1}{2} \right| \\ &= \left| \Pr[\mathsf{Succ} \mid \mathsf{query}] \Pr[\mathsf{query}] + \frac{1}{2} \cdot \Pr[\overline{\mathsf{query}}] - \frac{1}{2} \right| \end{aligned}$$

$$= \left| \Pr[\mathsf{Succ} \mid \mathsf{query}] \Pr[\mathsf{query}] - \frac{1}{2} \cdot \Pr[\mathsf{query}] \right|$$
$$\leq \frac{1}{2} \Pr[\mathsf{query}].$$

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.