University of Maryland
CMSC456 — Introduction to Cryptography
Professor Jonathan Katz

Lecture 39

1 The Random Oracle Model

At the end of the last lecture, we introduced the *random oracle model* and distinguished it from the *standard model* in which we have been working until now. We review and extend the discussion here.

- The random oracle model assumes a publicly-accessible oracle ("black box") that everyone can access. This oracle is assumed to implement a completely random function.
- In practice (random oracles do not exist in the real world!), we will instantiate the random oracle with a cryptographic hash function.
- From a theoretical point of view: no (efficiently-computable) hash function can possibly be a random function! (A true random function would have a description that is exponentially large!)
- From a practical point of view: better to have a construction which can be proven secure in the random oracle model than to have a construction with no proof at all.
- In practice: if an adversary can break a scheme which was proven secure in the random oracle model, then the hash function used to instantiate the random oracle must not be "good enough". So a proof of security in the random oracle model gives us confidence in the scheme, since it seems that the only way to break the scheme (in the real world) involves finding some weakness in the hash function used to instantiate the random oracle.
- In theory: no formal statement along these lines seems possible. In particular, it might be possible to break a scheme which was proven secure in the random oracle model without finding any specific weakness in the hash function used. Furthermore, there are (artificial) examples of schemes that can be proven secure in the random oracle model, but which are *insecure* when the random oracle is replaced by *any* specific hash function.

Hopefully, the above discussion captures some of the tension inherent in using the random oracle model (indeed, use of this model to prove security remains controversial among cryptographers). To summarize: a proof of security in the random oracle model is no *guarantee* of any security in the real world; on the other hand, a proof of security in the random oracle model is better than no proof at all, and the only efficient schemes we (currently) know how to construct are provably secure only in the random oracle model. So if we want efficient schemes with some evidence (but not guarantee!) of their security, it seems we must work in the random oracle model.

2 An Efficient Signature Scheme in the RO Model

As an example of how the random oracle model can be useful, we show here that the original suggestion of Diffie and Hellman for constructing signature schemes can be modified and proven secure in the random oracle model. Let $(\mathcal{K}, f, \mathsf{Inv})$ be a trapdoor permutation. (As usual, if td is the trapdoor associated with a particular key key, then we denote $\mathsf{Inv}_{\mathsf{td}}(\cdot)$ by $f_{\mathsf{key}}^{-1}(\cdot)$, where it is clear that f_{key}^{-1} can be efficiently evaluated *only* by someone who knows td.) We construct a signature scheme as follows:

- To generate a public/secret key, run $\mathcal{K}$ to generate $(\mathsf{key}, \mathsf{td})$. The public key is key and the secret key is td.
- Let $H : \{0,1\}^* \rightarrow \mathcal{D}_{\mathsf{key}}$ (recall $\mathcal{D}_{\mathsf{key}}$ is the domain of f_{key}) be a hash function modeled as a random oracle. To sign message m, output $\sigma = f_{\mathsf{key}}^{-1}(H(m))$.
- To verify signature σ on message m, check that $f_{\mathsf{key}}(\sigma) \stackrel{?}{=} H(m)$. (Remember that everyone is assumed to have access to the random oracle H).

The following theorem shows that this construction is indeed secure. Pay careful attention to the proof of security, which relies in an essential way on the fact that H is indeed a random oracle. (The proof simply fails when H is any concrete function.)

Theorem 1 *If $(\mathcal{K}, f, \mathsf{Inv})$ is a (t, ϵ)-secure trapdoor permutation, then the above signature scheme is $(t, q_h\epsilon)$-secure, where q_h represents the number of queries an adversary makes to the random oracle H.*

Proof Assume there is some adversary A which outputs a forgery for the above construction with probability δ. We use this adversary to construct an algorithm A' that inverts the trapdoor permutation. A' is given a key key and a random element y, and tries to compute x such that $f_{\mathsf{key}}(x) = y$. It proceeds as follows:

$A'(\mathsf{key}, y)$
 Set $PK = \mathsf{key}$
 Pick a random index $i^* \in \{1, \ldots, q_h\}$
 Run $A(PK)$
 Answer the i^{th} query of A to oracle H as follows (let m_i denote this i^{th} query):
  if $i = i^*$, return y
  otherwise, pick random $r_i \leftarrow \mathcal{D}_{\mathsf{key}}$, compute $\mathsf{ans}_i = f_{\mathsf{key}}(r)$, and return ans_i
 When A requests a signature on message m:
  find i such that $m = m_i$ (see discussion below)
  if $i = i^*$, abort; otherwise, return r_i as the signature
 When A outputs its forgery (m, σ)
  If $m = m_{i^*}$ then output σ; otherwise, abort

In the above description, we make a few assumptions about A: (1) we assume that before A ever asks for a signature on message m, it has already queried $H(m)$; (2) we assume that A never submits the same message to H more than once (indeed, there is no reason for it to do so, as the answer it would get back remains the same); (3) we assume that if A outputs a forgery on message m, then at some point it has queried $H(m)$. It is not hard to see that these assumptions may be made without loss of generality, since given any adversary A we can modify it to get an adversary $\tilde{A}$ that satisfies the above assumptions and has the same probability of forgery (although $\tilde{A}$ might now make at most $q_h + q_s + 1$ queries to H, where q_s is the number of signatures requested by A).

We note the following about the above simulation:

- As long as A' does not abort, the simulation for A is perfect. The outputs of oracle H are random and independently distributed, as required for a random oracle.

- A' sets things up so that it is able to answer all signature queries of A *unless* A' asks for a signature on m_{i^*}. Note that the response of A' to all other signature queries is indeed a correct signature.

- Since the output (m, σ) of A can only be a forgery if A never requested a signature on m, and since $m = m_j$ for some j (by assumption), it must be the case that there is at least one index j for which A never requests a signature on m_j. Since i^* is chosen at random, with probability at least $1/q_h$ we have $j = i^*$. When $j = i^*$ (and A outputs a valid forgery) then σ is indeed an inverse of y and A' succeeds in inverting the trapdoor permutation.

To summarize the above discussion, the probability that A' correctly outputs an inverse is at least $1/q_h$ times the probability that A outputs a forgery; hence, A' outputs the desired inverse with probability at least δ/q_h. Since the trapdoor permutation is assumed to be (t, ϵ)-secure we must have $\delta \leq q_h\epsilon$, completing the proof. ■

Again, we stress that it was vital to the proof that A' have the ability to "fix" the outputs of H. This is fine in the random oracle model (as long as A' fixes the outputs in an independent and uniform way) since the output is supposed to be random, anyway. But it should be clear that the proof fails once we commit to any *specific* function H (and A' loses the ability to fix outputs as it likes).