1 Review of Lecture 31: the Random Oracle Heuristic

Let us review the random oracle heuristic using MACs. In a heuristic MAC scheme, both the signing and verification algorithm use a deterministic function H. As a result, if Alice and Bob are using this function, both know the full description of H. In fact, this is no secret information, everyone knows the implementation of H. However, when we prove security, we 'assume' that the adversary does not know the implementation of H. Instead, if the adversary needs to know $H(x)$, it must query the challenger. The challenger, at the start of the game, samples a random function f, which will be used in place of H. That is, the signing and verification algorithms will use f. The adversary can make polynomially many random oracle queries. For each query x, it receives $f(x)$. Note that the oracle must remain consistent. Last lecture, we saw a secure MAC scheme, and two insecure ones in the random oracle model. If a scheme is insecure in the ROM, then it is surely insecure in the standard model. If it is secure in ROM, then that does not translate to any formal guarantees in the standard model. However, if we use a sufficiently complex function such as SHA, then a ROM proof gives us some confidence in the real-world implementations.

Here are the key points to remember:

- any crypto primitive has a security definition (that captures real-world attacks). If we are using the random oracle heuristic, we first need to define security in the random oracle model. In most cases, this is straightforward: just allow the adversary to make random oracle queries.
- the various algorithms of a crypto primitive can use the function H. In the security game, when these algorithms are used, we need to make appropriate calls to the random oracle. For instance, in the MAC construction from last class, every MAC query results in one random oracle call.
- last class, we used one key property of random oracles: given a new input, the random oracle output is completely random (and independent of previous responses). In today's lecture, we will see a few more useful ways of using random oracles.
- interpreting security in the random oracle model: while security in ROM doesn't give any formal guarantees for the standard model, it serves as a good heuristic. Prove something to be secure in the ROM, then use a hash function like SHA, and it is reasonable to expect this scheme resists all attacks in the standard model.

Plan for today's lecture: In the last lecture, we saw a few warm-up security proofs in the ROM. These were not very interesting, because we have already seen efficient constructions for these primitives that are also provably secure in the standard model. Today, we will analyse the security of RSA-based encryption schemes in the random oracle model.

2 RSA-based heuristic encryption scheme

Construction 3: A practically efficient, heuristic scheme based on RSA

Message space is $\{0,1\}^*$. This construction uses a det. function $H : \mathbb{Z}_N \to \{0,1\}^n$. The scheme also uses a symmetric key encryption scheme $\mathcal{E}_{\text{sym}} = (\mathsf{Enc}_{\text{sym}}, \mathsf{Dec}_{\text{sym}})$.

- $\mathsf{Enc}(m,(N,e))$: Choose $x \leftarrow Z_N$, compute $\mathsf{ct}_1 = x^e \bmod N$, $k = H(x)$, $\mathsf{ct}_2 = \mathsf{Enc}_{\text{sym}}(m,k)$.
- $\mathsf{Dec}((\mathsf{ct}_1,\mathsf{ct}_2),(N,d))$: Compute $x = \mathsf{ct}_1^d \bmod N$, $k = H(x)$. Output $\mathsf{Dec}_{\text{sym}}(\mathsf{ct}_2,k)$.

Correctness follows immediately using the correctness of $\mathcal{E}_{\text{sym}}$.

Again, note that collision resistance does not suffice here. In fact, for the above construction, we don't even need H to be a compressing function.

Suppose an adversary can win the semantic security game against this scheme in the random oracle model. We will start with the semantic security game in the random oracle model, then gradually modify this game so that eventually, the adversary's advantage in the last game is negligible.

Sequence of hybrid games

Game 0: This is the original semantic security game in the random oracle model.

How to share this key d

- (Setup phase) The challenger chooses $\mathsf{pk} = (N, e)$, $\mathsf{sk} = (N, d)$. It chooses the random oracle 'on-the-fly'. Initially, it creates an empty table $\mathcal{T}$.
- (Random oracle queries) The adversary sends $x_i \in \mathbb{Z}_N$ as a random oracle query. If $(x_i, y_i) \in \mathcal{T}$, the challenger sends y_i in response. Else, it chooses a uniformly random n bit string y_i, adds (x_i, y_i) to the table, and sends y_i to the adversary.

Note that multiple msg queries are not used here. Think why...

- (Challenge phase) The adversary sends two challenge messages m_0, m_1.

  The challenger picks a uniformly random number $x \leftarrow \mathbb{Z}_N$. If there exists no entry corresponding to x in $\mathcal{T}$, it chooses k and adds (x, k) to $\mathcal{T}$ (if $(x, k) \in \mathcal{T}$, it uses this k for the remaining computation). It sets $\mathsf{ct}_1 = x^e \bmod N$, $\mathsf{ct}_2 = \mathsf{Enc}_{\text{sym}}(m_b, k)$ (where $b \leftarrow \{0, 1\}$) and sends $(\mathsf{ct}_1, \mathsf{ct}_2)$ to the adversary.
- (Random oracle queries) The adversary is allowed random oracle queries after seeing the challenge ciphertext. Again, on receiving $x_i \in \mathbb{Z}_N$ as a random oracle query, the challenger checks if $(x_i, y_i) \in \mathcal{T}$ for some y_i. If so, the challenger sends y_i in response. Else, it chooses a uniformly random n bit string y_i, adds (x_i, y_i) to the table, and sends y_i to the adversary.
- Finally, the adversary sends its guess b'.

Game 1: In this game, the challenger does not use the table for the challenge ciphertext. It just samples a random key k and uses it for the challenge ciphertext, but does not add (x, k) to the table $\mathcal{T}$.

- (Setup phase) The challenger chooses $\mathsf{pk} = (N, e)$, $\mathsf{sk} = (N, d)$. It chooses the random oracle 'on-the-fly'. Initially, it creates an empty table $\mathcal{T}$.
- (Random oracle queries) The adversary sends $x_i \in \mathbb{Z}_N$ as a random oracle query. If $(x_i, y_i) \in \mathcal{T}$, the challenger sends y_i in response. Else, it chooses a uniformly random n bit string y_i, adds (x_i, y_i) to the table, and sends y_i to the adversary.
- (Challenge phase) The adversary sends two challenge messages m_0, m_1.

  The challenger picks a uniformly random number $x \leftarrow \mathbb{Z}_N$.

  It chooses $k \leftarrow \{0, 1\}^n$, sends $\mathsf{ct}_1 = x^e \bmod N$, $\mathsf{ct}_2 = \mathsf{Enc}_{\text{sym}}(m_b, k)$ (where $b \leftarrow \{0, 1\}$). It does not search for $(x, k) \in \mathcal{T}$, and neither does it add (x, k) to $\mathcal{T}$.

Note that the adversary here chooses the key k randomly and not H(x). This is important for proving that the probability of winning in Game 1 is negl.

- (Random oracle queries) The adversary is allowed random oracle queries after seeing the challenge ciphertext. Again, on receiving $x_i \in \mathbb{Z}_N$ as a random oracle query, the challenger checks if $(x_i, y_i) \in \mathcal{T}$ for some y_i. If so, the challenger sends y_i in response. Else, it chooses a uniformly random n bit string y_i, adds (x_i, y_i) to the table, and sends y_i to the adversary.
- Finally, the adversary sends its guess b'.

Qn: Why is it important to use a table $\mathcal{T}$ in the games above? Both games could have been defined by g a uniformly random function.

ns: This is important for the reduction algorithms to be polynomial time (see reductions below).

Analysis: First, let us observe the differences between Game 0 and Game 1. There are two main differences:

- The adversary made a pre-challenge random oracle query corresponding to x^*, receives y^*. During the challenge phase, the challenger chose $x = x^*$. In this scenario, the behaviour changes in Game 0 and Game 1. In Game 0, the challenger uses the string y^* for computing encryption of the challenge message. In Game 1, the challenger picks a uniformly random key k and uses it for computing the challenge ciphertext.

- The challenger chooses $x = x^*$ for the challenge ciphertext, and uses string k for computing the challenge ciphertext. Later, in the post-challenge query phase, the adversary makes a random oracle query for x^*. In Game 0, it receives k (the same string that was used in the challenge phase). However, in Game 1, the pair (x^*, k) was not added to $\mathcal{T}$. Therefore, the challenger picks a uniformly random string y^* and sends it to the adversary.

Nice claim

Intuitively, the first event will happen with very low probability (since x^* is chosen at random during the challenge phase). The second one will also happen with low probability, and this argument relies on the hardness of RSA problem. If an adversary has significant difference in winning probabilities in the two games, then it must break the RSA assumption.

Formal proof: Let p_b denote the probability of adversary $\mathcal{A}$ winning in Game b.

Claim 32.01. Suppose there exists a p.p.t. adversary $\mathcal{A}$ such that $p_0 - p_1 = \epsilon$. Then there exists a p.p.t. algorithm $\mathcal{B}$ that solves the RSA problem with probability ϵ.

Proof. The reduction algorithm receives (N, e, y) from the RSA challenger. It sends $\mathsf{pk} = (N, e)$ to the adversary. The reduction also maintains a table $\mathcal{T}$, which is initially empty.

Next, the adversary makes pre-challenge random oracle queries. For each RO query x_i, the reduction algorithm does the following:

- If there exists an entry (x_i, y_i) in $\mathcal{T}$, then the reduction algorithm sends y_i.
- If there exists no entry corresponding to x_i in $\mathcal{T}$, the reduction algorithm samples $y_i \leftarrow \{0,1\}^n$, adds (x_i, y_i) to $\mathcal{T}$ and sends y_i to $\mathcal{A}$. It also checks if $x_i^e = y$. If so, it sends x_i to the RSA challenger (and wins the RSA game).

For the challenge phase, the reduction algorithm receives challenge messages m_0^*, m_1^* from the adversary. It picks a uniformly random string $k \leftarrow \{0,1\}^n$, sets $\mathsf{ct}_1^* = y$. Next, it picks a random bit $b \leftarrow \{0,1\}$, computes $\mathsf{ct}_2^* = \mathsf{Enc}_{\mathrm{sym}}(m_b^*, k)$ and sends $(\mathsf{ct}_1^*, \mathsf{ct}_2^*)$ to $\mathcal{A}$.

Note: If the e^{th} root of y was queried in one of the random oracle queries, then the reduction has already won the RSA game. If it was not queried, then Game 0 and Game 1 are identical up to the challenge phase (and the reduction perfectly simulates the two games up to this point).

The adversary then makes post-challenge random oracle queries. For each RO query x_i, the reduction algorithm checks if $x_i^e = y$. If so, it sends x_i to the RSA challenger. Else, it checks if there exists an entry $(x_i, y_i) \in \mathcal{T}$. If so, it sends y_i to $\mathcal{A}$. If there is no such entry, then it samples $y_i \leftarrow \{0,1\}^n$, sends y_i to $\mathcal{A}$ and adds (x_i, y_i) to $\mathcal{T}$.

Similar to what is mentioned above, if the adversary does not make a random oracle query on input $y^{(1/e)}$, then Game 0 and Game 1 are identical. Therefore, if the winning probabilities are different in the two games, then it must query the random oracle on input $y^{(1/e)}$, in which case the reduction wins the RSA game.

□

Claim 32.02. Suppose there exists a p.p.t. adversary $\mathcal{A}$ that wins in Game 1 with probability ϵ. Then there exists a reduction algorithm $\mathcal{B}$ that wins the no-query semantic-security game against $\mathcal{E}_{\mathrm{sym}}$ with probability ϵ.

Proof. The proof follows from the simple observation that in Game 1, string k is only used in the challenge phase.

The reduction algorithm chooses (N, e) and sends it to the adversary. The reduction maintains a table $\mathcal{T}$ which is initially empty. On receiving a random oracle query x_i, the reduction checks if there exists (x_i, y_i) in the table. If so, it sends y_i. Else, it chooses $y_i \leftarrow \{0,1\}^n$, adds (x_i, y_i) to $\mathcal{T}$ and sends y_i to $\mathcal{A}$.

When the adversary sends (m_0, m_1) as the challenge messages, the reduction sends (m_0, m_1) to the $\mathcal{E}_{\text{sym}}$ challenger. It receives ct from the challenger. The reduction algorithm chooses $x \leftarrow \mathbb{Z}_N$, sets $\mathsf{ct}_1 = x^e \bmod N$, $\mathsf{ct}_2 = \mathsf{ct}$ and sends $(\mathsf{ct}_1, \mathsf{ct}_2)$ to $\mathcal{A}$.

The post-challenge random-oracle queries are handled similar to the pre-challenge ones.

CRUX

Note: In Game 1's challenge phase, the challenger chooses $x \leftarrow \mathbb{Z}_N$, $k \leftarrow \{0,1\}^n$, but does not add (x, k) to $\mathcal{T}$. As a result, if the adversary queries the random oracle on this x, it receives a fresh random string y. This is important for our reduction here.

Finally, the adversary sends its guess b', which the reduction forwards to the challenger. The winning probability of $\mathcal{A}$ in Game 1 is equal to the winning probability of the reduction against $\mathcal{E}_{\text{sym}}$ (why?). □

3 Lecture summary, plan for next lecture, additional resources

Summary: We proved security of the RSA-based heuristic construction in the random oracle model.

Next lecture: We will start with security against chosen-ciphertext attacks, and discuss a few heuristic constructions for the same. We will prove security for one of them.

Additional resources The Katz-Lindell chapter also contains a proof of the heuristic construction.