Notes for Lecture 23

Scribed by Guoming Wang, posted May 4, 2009

Summary

Today we show how to construct an efficient CCA-secure public-key encryption scheme in the *random oracle model* using RSA.

As we discussed in the previous lecture, a cryptographic scheme defined in the *random oracle model* is allowed to use a random function $H : \{0,1\}^n \to \{0,1\}^m$ which is known to all the parties. In an implementation, usually a cryptographic hash function replaces the random oracle. In general, the fact that a scheme is proved secure in the random oracle model does not imply that it is secure when the random oracle is replaced by a hash function; the proof of security in the random oracle model gives, however, at least some heuristic confidence in the soundness of the design.

1 Hybrid Encryption with a Random Oracle

We describe a public-key encryption scheme $(\overline{G}, \overline{E}, \overline{D})$ which is based on: (1) a family of trapdoor permutations (for concreteness, we shall refer specifically to RSA below); (2) a CCA-secure *private-key* encryption scheme (E, D); (3) a random oracle H mapping elements in the domain and range of the trapdoor permutation into keys for the private-key encryption scheme (E, D).

1. Key generation: $\overline{G}$ picks an RSA public-key / private-key pair $(N, e), (N, d)$;
2. Encryption: given a public key N, e and a plaintext M, $\overline{E}$ picks a random $R \in \mathbb{Z}_N^*$, and outputs
$$(R^e \bmod N, E(H(R), M))$$
3. Decryption: given a private key N, d and a ciphertext C_1, C_2, $\overline{D}$ decrypts the plaintext by computing $R := C_1^d \bmod N$ and $M := D(H(R), C_2)$.

This is a hybrid encryption scheme in which RSA is used to encrypt a "session key" which is then used to encrypt the plaintext via a private-key scheme. The important difference from hybrid schemes we discussed before is that the random string encrypted with RSA is "hashed" with the random oracle before being used as a session key.

2 Security Analysis

The intuition behind the security of this scheme is as follows. For any adversary mounting a CCA attack on $(\overline{G}, \overline{E}, \overline{D})$, we distinguish between the case that it does not query R to the random oracle H and the case that it does. In the first case, the adversary learns nothing about $H(R)$, and so we reduce the CCA security of $(\overline{G}, \overline{E}, \overline{D})$ to the CCA security of the private-key encryption scheme (E, D). In the second case, the adversary has managed to invert RSA. Under the RSA trapdoor permutation assumption, this case can only happen with negligible probability.

This intuition is formalized in the proof of the following theorem.

Theorem 1 *Suppose that, for the key size used in $(\overline{G}, \overline{E}, \overline{D})$, RSA is a (t, ϵ)-secure family of trapdoor permutations, and that exponentiation can be computed in time $\leq r$; assume also that (E, D) is a (t, ϵ) CCA-secure private-key encryption scheme and that E, D can be computed in time $\leq r$.*

Then $(\overline{G}, \overline{E}, \overline{D})$ is $(t/O(r), 2\epsilon)$ CCA-secure in the random oracle model.

PROOF: Suppose A is a CCA attack on $(\overline{G}, \overline{E}, \overline{D})$ of complexity $t' \leq t/O(r)$ such that there are two messages M_0, M_1

$$|\,\mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_0)) = 1] - \mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_1)) = 1]| > 2\epsilon.$$

We will derive from A an algorithm A' running in time $O(t' \cdot r)$ which is a CCA attack on (E, D). If A' succeeds with probability at least ϵ in distinguishing between $E(M_0)$ and $E(M_1)$, then we have violated the assumption on the security of (E, D). But if A' succeeds with probability less than ϵ, then we can devise an algorithm A'', also of running time $O(t' \cdot r)$, which inverts RSA with probability at least ϵ.

The algorithm A' is designed to attack the private-key encryption scheme (E, D) by running A as a subroutine. Its idea is to convert the ciphertext $E(M)$ into the ciphertext $\overline{E}(M)$ and then use A to distinguish it. It needs to answer A's oracle queries to $\overline{E}, \overline{D}, H$, by accessing its own oracles E, D. In particular, in order to make its answers to the random oracle queries consistent, A' uses a data structure (a table) to remember the pairs $(R', H(R'))$ that have been queried so far. The formal definition of A' is as follows:

Algorithm $A'^{E,D}$:
Input: $C = E(K, M)$ $//K$ is unknown, $M \in \{M_0, M_1\}$

- pick RSA keys N, e, d.
- pick a random $R \in \mathbb{Z}_N^*$.
- pairs of strings $(\cdot, \cdot)$ are stored in a table which is initially empty.

- simulate $A^{\overline{E},\overline{D},H}(R^e \bmod N, C)$ as follows:
  - when A queries $H(R')$:
    - If there is an entry (R', K') in the table, return K' to A;
    - otherwise, pick a random K', store (R', K') in the table, return K' to A.
    - If $R' = R$, FAIL.
  - when A queries $\overline{E}(M')$:
    - pick a random $R' \in \mathbb{Z}_N^*$, compute $K' := H(R')$ as above, return $(R'^e \bmod N, E(K', M'))$ to A.
  - when A queries $\overline{D}(C_1, C_2)$:
    - compute $R' := C_1^d \bmod N$.
    - if $R' \neq R$, compute $K' := H(R')$ as above, return $D(K', C_2)$ to A;
    - if $R' = R$, query C_2 to the decryption oracle D and return the result to A.

A' needs time $O(r)$ to compute the answer of every oracle query from A. Given that A has complexity $t' \leq t/O(r)$, we get A' has complexity $O(t' \cdot r) \leq t$. Furthermore, A' never submits its challenge ciphertext C to its own decryption oracle D, because the only way this could happen would be if A submitted its challenge ciphertext $(R^e \bmod N, C)$ to its own decryption oracle $\overline{D}$, but this is not allowed.

Let $QUERY$ denote the event that A queries R to the random oracle H during its execution. Note that in A''s strategy of answering A's decryption queries, A' has implicitly set $H(R) = K$. If $QUERY$ happens, i.e. A queries $H(R)$, A' returns a random $\hat{K}$ to A in response to this query, but $\hat{K}$ might not be equal to the unknown K, and this will cause an inconsistency in A''s answers. However, if $QUERY$ does not happen, the answers A' gives to A in response to its oracle queries to $\overline{E}, \overline{D}, H$ are always consistent and distributed identically to the answers obtained from the corresponding real oracles. So in this case, the behavior of A remains unchanged. Thus we get

$$\mathbb{P}[A'^{E,D}(E(M)) = 1] = \mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M)) = 1 \wedge \neg QUERY],$$

and hence

$$\begin{aligned} & |\,\mathbb{P}[A'^{E,D}(E(M_0)) = 1] - \mathbb{P}[A'^{E,D}(E(M_1)) = 1]| \\ = \; & |\,\mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_0)) = 1 \wedge \neg QUERY] \\ & - \mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_1)) = 1 \wedge \neg QUERY]|. \end{aligned}$$

Then by the triangle inequality we obtain

$$\begin{aligned}
2\epsilon &< |\,\mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_0)) = 1] - \mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_1)) = 1]| \\
&\leq |\,\mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_0)) = 1 \wedge QUERY] \\
&\quad - \mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_1)) = 1 \wedge QUERY]| \\
&\quad + |\,\mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_0)) = 1 \wedge \neg QUERY] \\
&\quad - \mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M_1)) = 1 \wedge \neg QUERY]| \\
&\leq \max_{M \in \{M_0, M_1\}} \mathbb{P}[A^{\overline{E},\overline{D},H}(\overline{E}(M)) = 1 \wedge QUERY] \qquad (1) \\
&\quad + |\,\mathbb{P}[A'^{E,D}(E(M_0)) = 1] - \mathbb{P}[A'^{E,D}(E(M_1)) = 1]| \qquad (2)
\end{aligned}$$

So at least one of (1) and (2) must be greater than ϵ.

If (2) $> \epsilon$, then A' breaks the CCA-security of (E, D), contradicting our assumption. So we should have (1) $> \epsilon$. But in this case, we can devise an algorithm A'' of complexity $O(t' \cdot r)$ such that A'' solves RSA with probability $> \epsilon$.

The idea of A'' is also to simulate A by providing it with appropriate input and oracle query answers. Similarly, A'' also needs a data structure (a table) to record the answers $(R', H(R'))$ it has given so far. However, now A'' only knows the public RSA keys N, e, but does not know the private key d. Consequently, when decrypting a ciphertext (C_1, C_2), it cannot compute $R' := C_1^d \bmod N$ (without factoring N). Then how can we get $H(R')$ without knowing R'? In order to overcome this obstacle, we store triples of strings $(R', R'^e \bmod N, H(R'))$, instead of pairs $(R', H(R'))$, in the table. If R' is unknown yet, we use a special symbol $\perp$ to represent it. Now we no longer check whether the first item of each entry in the table is equal to R', but check whether the second item of each entry is equal to $R'^e \bmod N$. Because there is a one-to-one correspondence between R' and $R'^e \bmod N$ (given N, e), by checking the second item, we can get $H(R')$ without knowing R'.

The formal definition of A'' is as follows:

Algorithm A'':
Input: N,e, $C = R^e \bmod N$ // R is unknown

- pick a random K.
- triples of strings $(\cdot, \cdot, \cdot)$ are stored in a table which initially contains only $(\perp, C, K)$.
- simulate $A^{\overline{E},\overline{D},H}(C, E(K, M))$ as follows: //M is the message for which (1)$> \epsilon$
  - when A queries $H(R')$:
    - if $R'^e \bmod N == C$, output R', halt;

    * if there is an entry $(R', R'^e \bmod N, K')$ in the table, return K' to A;
    * otherwise, if there is an entry $(\perp, R'^e \bmod N, K')$ in the table, replace it with $(R', R'^e \bmod N, K')$, return K' to A;
    * otherwise, pick a random K', store $(R', R'^e \bmod N, K')$ in the table, return K' to A.
  - when A queries $\overline{E}(M')$:
    * pick a random $R' \in \mathbb{Z}_N^*$, compute $K' := H(R')$ as above, return $(R'^e \bmod N, E(K', M'))$ to A.
  - when A queries $\overline{D}(C_1, C_2)$:
    * if there is an entry (R', C_1, K') or $(\perp, C_1, K')$ in the table, return $D(K', C_2)$ to A;
    * otherwise, pick a random K', store $(\perp, C_1, K')$ in the table, return $D(K', C_2)$ to A.

A'' also needs time $O(r)$ to compute the answer of every oracle query from A. Given that A has complexity $t' \le t/O(r)$, we get A'' has complexity $O(t' \cdot r) \le t$. Moreover, the answers A'' gives to A in response to its oracle queries to $\overline{E}, \overline{D}, H$ are always consistent and distributed identically to the answers obtained from the corresponding real oracles. Thus, the behavior of A remains unchanged. Especially, the event $QUERY$ remains unchanged. Furthermore, A'' correctly solves the given RSA instance whenever $QUERY$ occurs. So we have

$$\mathbb{P}[A''(N, e, R^e \bmod N) = R] = \mathbb{P}[QUERY] \ge (1) > \epsilon,$$

which contradicts with the assumption that RSA is (t, ϵ)-secure. This concludes the proof of the theorem. □