1 Review: CCA Security

Last class, we introduced CCA security for public key encryption schemes. In this security game, the adversary is allowed decryption queries before/after the challenge ciphertext is sent. The only constraint is that none of the post-challenge decryption queries should be equal to the challenge ciphertext.

CCA security game

- **Setup phase** Challenger chooses $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$ and sends pk to $\mathcal{A}$.
- **Pre-challenge decryption queries** Adversary sends decryption query ct_i, and receives $y_i = \mathsf{Dec}(\mathsf{ct}_i, \mathsf{sk})$ in response.
- **Challenge phase** Adversary sends two messages m_0, m_1.
  Challenger picks $b \leftarrow \{0,1\}$, sends $\mathsf{Enc}(m_b, \mathsf{pk})$ to $\mathcal{A}$.
- **Post-challenge decryption queries** Adversary sends decryption query ct_i. If $\mathsf{ct}_i = \mathsf{ct}^*$, challenger sends $\perp$. Else it sends $y_i = \mathsf{Dec}(\mathsf{ct}_i, \mathsf{sk})$ in response.
- **Guess** $\mathcal{A}$ sends its guess b' and wins if $b = b'$.

Figure 1: Security against Chosen Ciphertext Attacks

2 CCA security in the random oracle model: RSA-based Heuristic Scheme

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

Heuristic scheme based on RSA

- $\mathsf{Enc}(m, (N, e))$: 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 is immediate, let us consider the CCA security proof.

Sequence of hybrid games

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

- (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 $\mathcal{T}$.
- (Pre-challenge queries)
  - (Pre-challenge 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.
  - (Pre-challenge Decryption queries) The adversary sends ciphertext $\mathsf{ct}_i = (\mathsf{ct}_{i,1}, \mathsf{ct}_{i,2})$ as a decryption query. The challenger checks if there exists an entry $(\mathsf{ct}_{i,1}^{1/e}, y) \in \mathcal{T}$. If so, it sends $\mathsf{Dec}_{\text{sym}}(\mathsf{ct}_{i,2}, y)$. Else it samples a fresh $y \leftarrow \{0,1\}^n$, adds $(\mathsf{ct}_{i,1}^{1/e}, y)$ to $\mathcal{T}$, and sends $\mathsf{Dec}_{\text{sym}}(\mathsf{ct}_{i,2}, y)$.
- (Challenge phase) The adversary sends m_0^*, m_1^*. Challenger chooses $x^* \leftarrow \mathbb{Z}_N$ and $b \leftarrow \{0,1\}$. Next, it checks if $(x^*, y) \in \mathcal{T}$. If so, it sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2 = \mathsf{Enc}_{\text{sym}}(m_b^*, y)$.
  If no $(x^*, y) \in \mathcal{T}$, it chooses $y \leftarrow \{0,1\}^n$, adds (x^*, y) to $\mathcal{T}$, sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2^* = \mathsf{Enc}_{\text{sym}}(m_b^*, y)$.
  It sends $(\mathsf{ct}_1^*, \mathsf{ct}_2^*)$ to $\mathcal{A}$.

- (Post-challenge queries)
  - (Post-challenge Random Oracle queries) Same as pre-challenge random oracle queries
  - (Post-challenge Decryption queries) Same as pre-challenge decryption queries, except that the challenger outputs $\perp$ if the decryption query is equal to the challenge ciphertext.

Game 1: This is similar to the previous game, except that the challenger stores (x^e, y) in $\mathcal{T}$ instead of (x, y). Note that since everything else is identical, the game is identical in the adversary's view. However, this change is important because instead of computing the e^{th} root (which requires the secret key component $d = 1/e$), the challenger only computes the e^{th} power (which is an easy operation).

- (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 $\mathcal{T}$.
- (Pre-challenge queries)
  - (Pre-challenge Random oracle queries) The adversary sends $x_i \in \mathbb{Z}_N$ as a random oracle query. If $(x_i^e, y_i) \in \mathcal{T}$, the challenger sends y_i in response. Else, it chooses a uniformly random n bit string y_i, adds (x_i^e, y_i) to the table, and sends y_i to the adversary.
  - (Pre-challenge Decryption queries) The adversary sends ciphertext $\mathsf{ct}_i = (\mathsf{ct}_{i,1}, \mathsf{ct}_{i,2})$ as a decryption query. The challenger checks if there exists an entry $(\mathsf{ct}_{i,1}, y) \in \mathcal{T}$. If so, it sends $\mathsf{Dec}_{\text{sym}}(\mathsf{ct}_{i,2}, y)$. Else it samples a fresh $y \leftarrow \{0,1\}^n$, adds $(\mathsf{ct}_{i,1}, y)$ to $\mathcal{T}$, and sends $\mathsf{Dec}_{\text{sym}}(\mathsf{ct}_{i,2}, y)$.
- (Challenge phase) The adversary sends m_0^*, m_1^*. Challenger chooses $x^* \leftarrow \mathbb{Z}_N$ and $b \leftarrow \{0,1\}$. Next, it checks if $((x^*)^e, y) \in \mathcal{T}$. If so, it sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2 = \mathsf{Enc}_{\text{sym}}(m_b^*, y)$.

  If no $((x^*)^e, y) \in \mathcal{T}$, it chooses $y \leftarrow \{0,1\}^n$, adds $((x^*)^e, y)$ to $\mathcal{T}$, sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2^* = \mathsf{Enc}_{\text{sym}}(m_b^*, y)$. It sends $(\mathsf{ct}_1^*, \mathsf{ct}_2^*)$ to $\mathcal{A}$.
- (Post-challenge queries)
  - (Post-challenge Random Oracle queries) Same as pre-challenge random oracle queries
  - (Post-challenge Decryption queries) Same as pre-challenge decryption queries, except that the challenger outputs $\perp$ if the decryption query is equal to the challenge ciphertext.

Game 2: 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 $\mathcal{T}$.
- (Pre-challenge queries)
  - (Pre-challenge Random oracle queries) The adversary sends $x_i \in \mathbb{Z}_N$ as a random oracle query. If $(x_i^e, y_i) \in \mathcal{T}$, the challenger sends y_i in response. Else, it chooses a uniformly random n bit string y_i, adds (x_i^e, y_i) to the table, and sends y_i to the adversary.
  - (Pre-challenge Decryption queries) The adversary sends ciphertext $\mathsf{ct}_i = (\mathsf{ct}_{i,1}, \mathsf{ct}_{i,2})$ as a decryption query. The challenger checks if there exists an entry $(\mathsf{ct}_{i,1}, y) \in \mathcal{T}$. If so, it sends $\mathsf{Dec}_{\text{sym}}(\mathsf{ct}_{i,2}, y)$. Else it samples a fresh $y \leftarrow \{0,1\}^n$, adds $(\mathsf{ct}_{i,1}, y)$ to $\mathcal{T}$, and sends $\mathsf{Dec}_{\text{sym}}(\mathsf{ct}_{i,2}, y)$.

- (Challenge phase) The adversary sends m_0^*, m_1^*. Challenger chooses $x^* \leftarrow \mathbb{Z}_N$ and $b \leftarrow \{0,1\}$.

  ~~Next, it checks if $((x^*)^e, y) \in \mathcal{T}$. If so, it sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2^* = \mathsf{Enc}_{\mathrm{sym}}(m_b^*, y)$.~~

  It chooses $k \leftarrow \{0,1\}^n$, ~~adds $((x^*)^e, y)$ to $\mathcal{T}$,~~ sets $\mathsf{ct}_1^* = (x^*)^e$, $\mathsf{ct}_2^* = \mathsf{Enc}_{\mathrm{sym}}(m_b^*, k)$.

  It sends $(\mathsf{ct}_1^*, \mathsf{ct}_2^*)$ to $\mathcal{A}$.

- (Post-challenge queries)

  - (Post-challenge Random Oracle queries) Same as pre-challenge random oracle queries
  - (Post-challenge Decryption queries) Let ct_i be the decryption query. If $\mathsf{ct}_i = \mathsf{ct}^*$, then output $\perp$.

    If $\mathsf{ct}_{i,1} = \mathsf{ct}_1^*$ but $\mathsf{ct}_{i,2} \neq \mathsf{ct}_2^*$, the challenger sends $\mathsf{Dec}_{\mathrm{sym}}(\mathsf{ct}_{i,2}, k)$.[1]

    If $\mathsf{ct}_{i,1} \neq \mathsf{ct}_1^*$, it proceeds similar to the pre-challenge decryption queries. It first checks if $(\mathsf{ct}_{i,1}, y)$ is present in $\mathcal{T}$. If so, it sends $\mathsf{Dec}_{\mathrm{sym}}(\mathsf{ct}_{i,2}, y)$. Else it samples $y \leftarrow \{0,1\}^n$, adds $(\mathsf{ct}_{i,1}, y)$ to $\mathcal{T}$ and sends $\mathsf{Dec}_{\mathrm{sym}}(\mathsf{ct}_{i,2}, y)$.

Analysis: First, observe that Game 0 and Game 1 are equivalent from the adversary's perspective. In one case, the challenger adds (x, y) to the table, in the other case it adds (x^e, y) to the table. Therefore, if an adversary wins with non-negligible advantage in Game 0, then it wins with non-negligible advantage in Game 1.

Next, let us consider the differences between Game 1 and Game 2.

- The first difference is in the challenge phase. In Game 1, the challenger first samples $x^* \leftarrow \mathbb{Z}_N$, then checks if $((x^*)^e, y) \in T$. In Game 2, the challenger simply picks a random k and uses it for computing ct_2^*. Since $x \leftarrow \mathbb{Z}_N$,
- The second difference is in the post-challenge queries. Note, in Game 1, after the challenge phase, there exists a tuple $((x^*)^e, y) \in \mathcal{T}$. In Game 2, the challenger sends $((x^*)^e, \mathsf{Enc}_{\mathrm{sym}}(m_b, k)$, 'stores' k separately, but does not add $((x^*)^e, k)$ to $\mathcal{T}$.

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).

[1] This is needed because we have not added (ct_1^*, k) to $\mathcal{T}$.

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 $\mathcal{E}_{\mathrm{sym}}$ 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.

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 $\mathcal{E}_{\mathrm{sym}}$ (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.