

# CPA and CCA security

---

## CPA & the IND-CPA game

You have an oracle, an adversary who is allowed to keep sending messages to the oracle at any point in time:  $m_1, m_2 \dots m_N$  and the oracle will send back  $E(m_1), E(m_2), \dots, E(m_N)$ .

The adversary will send  $m'_1, m'_2$  (they could be one of the  $m_1, m_2 \dots m_N$ ) and the oracle encrypts one of them (picks  $b$  randomly, 0 or 1) and sends the encryption to the adversary, who's supposed to tell whether  $m'_1$  or  $m'_2$  was encrypted. Therefore, no CPA scheme can be deterministic, since the adversary could already have the encryption of  $m'_1$  or  $m'_2$ .

$$\Pr[b' = b] \leq \frac{1}{2} + \varepsilon$$

$\varepsilon$  is the advantage here

## CCA & the IND-CCA game

The adversary also has access to decryption. He would always win since he would query the decryption oracle with the  $E(m'_b)$  challenge he received from the encryption oracle. Therefore, the attacker cannot query the decryption oracle with the challenges that he receives from the encryption oracle.

This is the **strongest model** of security for a cryptosystem.

## Example of CPA secure system that's not CCA secure

$$Enc_k(m) = \langle r, F_k(r) \oplus m \rangle, \text{ where } |m| = n, r \leftarrow U_n \text{ and } F_k \text{ is a PRF}$$

### A CCA attack

Adversary sends  $m_0 = 0^n$  and  $m_1 = 1^n$  to the encryption oracle.

He gets back  $Enc_k(m_b) = \langle r, F_k(r) \oplus m_b \rangle, b \leftarrow u$

$$- \quad Enc_k(m_b) = \begin{cases} Enc_k(m_0) = \langle r_0, F_k(r_0) \rangle, 50\% \text{ of the time} \\ Enc_k(m_1) = \langle r_1, \overline{F_k(r_1)} \rangle, 50\% \text{ of the time} \end{cases}$$

Since the decryption oracle will not accept the encryption of the challenge messages ( $m_0$  or  $m_1$ ) as input, the attacker will do the next best thing. He can flip the first bit in the encryption of  $m_b$ .

The decryption oracle will gladly decrypt the new ciphertext  $\langle r, c \oplus 10^{n-1} \rangle$ .

The adversary can now tell from what he gets back (either  $01^{n-1}$  or  $10^{n-1}$ ) whether  $m_0 = 0^n$  or  $m_1 = 1^n$  was encrypted.

## Constructing CCA-secure encryption schemes

**Common approach:** CPA-ENC + EU-MAC

Most (or maybe all) CCA attacks rely on the attacker modifying  $E_k(m'_b)$  and then feeding it to the decryption oracle whose answer will enable the attacker to tell which message was encrypted. If you add a MAC on top of the encryption, disallowing the attacker to fiddle with the ciphertext, then you essentially render the decryption oracle useless.

**Construction:** Let  $\pi = (Enc, Dec, Gen)$ . The generator now has to generate two keys, one for the MAC and one for the encryption scheme.

$$Gen(msg\ size) = \langle key_{enc}, key_{mac} \rangle$$

$$Enc(key_{enc}, key_{mac}, msg) = \langle ctxt, mac_{tag} \rangle$$

$$Dec(key_{enc}, key_{mac}, ctxt, tag) = plaintext\ or\ \perp$$

**Theorem:** If  $\pi_E$  is IND-CPA-secure and  $\pi_M$  is EU-MAC-secure then construction #1 is CCA-secure.

**Proof (idea):** Queries to the decryption oracle must be of the form  $\langle c, t \rangle$ , where:

$$c \leftarrow E_{k_1}(m) \text{ and } t \leftarrow Mac_{k_2}(c)$$

Since the MAC is secure the attacker has virtually no chance of modifying the challenge ciphertext without the decryption oracle knowing, effectively rendering the oracle useless.

## Three ways of implementing EU-MAC and CCA-ENC

### Encrypt and authenticate independently

This method computes the MAC of the message, not of the encryption. The ciphertext looks like this:

$$\langle c, t \rangle, \text{ where } c \leftarrow E_{k_{enc}}(m) \text{ and } t \leftarrow Mac_{key_{mac}}(m)$$

**Counterexample:**

$$\pi_e \text{ is IND-CPA, } c \leftarrow E_{k_1}(m)$$

$$\pi_M \text{ is EU-MAC, } t \leftarrow (m, Mac_{k_2}(m))$$

This is bad, because the tag  $t$  exposes the message  $m$ .

$$\langle c, t \rangle = \langle E_{k_{enc}}(m), \langle m, Mac_{key_{mac}}(m) \rangle \rangle$$

### Authenticate then encrypt

This method also computes the MAC of the message, but this time the MAC is concatenated with the message  $m$  and together they are encrypted to form the ciphertext.

$$\langle c, t \rangle, \text{ where } t \leftarrow Mac_{k_2}(m) \text{ and } c \leftarrow E_{k_1}(m \parallel t)$$

**Counterexample:**

Let  $Tr(m) =$  replace each bit  $b$  of  $m$  with  $F(b)$ .

$F(b)$  works very simply. If  $b = 0$ , then  $F(b) = 00$  and if  $b = 1$ , then  $F(b) = 01$  or  $10$  with equal probability.

$$F(b) = \begin{cases} F(0) = 00, \text{ with } p = 1.0 \\ F(1) = 01, \text{ with } p = 0.5 \\ F(1) = 10, \text{ with } p = 0.5 \end{cases}$$

Let  $Enc_k$  be a CTR-mode block-cipher. Let our encryption algorithm be  $Enc_k(m) = CTR(Tr(m \parallel t))$ .

Then, our decryption algorithm will be  $Dec_k(\langle c, t \rangle) = Tr^{-1}(CTR^{-1}(c))$

If the first bit of the message  $m$  is 1 then, after encryption,  $\langle c, t \rangle = \begin{cases} CTR(01 \dots) \\ CTR(10 \dots) \end{cases}$

If the first bit of the message  $m$  is 0 then, after encryption,  $\langle c, t \rangle = CTR(00 \dots)$

Challenge starts, attacker sends  $m_0 = 0^n$  and  $m_1 = 1^n$  to the encryption oracle which sends the encryption  $\langle c, t \rangle = \langle Enc_k(m_b), t \rangle$ , where  $b \leftarrow u$  of either  $m_0$  or  $m_1$ .

The attacker will use the following strategy:

He will **flip the first two bits** in the ciphertext  $\langle c, t \rangle$ , and send the modified query to the decryption oracle. The decryption oracle will first run CTR mode in reverse, obtaining  $Tr(m \parallel t)$ . It will then run  $Tr^{-1}(Tr(m \parallel t))$ . There are two cases now. (Let  $m = m_b$ ).

1. The first bit of the message  $m$  was 1, then flipping the first two bits in the ciphertext will either change 01 to 10, or 10 to 01 in the transformation  $Tr(m \parallel t)$  of the original message  $m$  (remember XOR and its properties and the fact that  $Enc_k(m) = CTR(Tr(m \parallel t))$ ).
  - a. Either way, while decrypting,  $Tr^{-1}(Tr(m \parallel t))$  will run successfully returning  $m \parallel t$ .
  - b. The decryption oracle will successfully check the integrity of  $m$  using the tag  $t$ , since there was no overall change in the original message, even though we flipped bits, and it will send  $m$  to the attacker.
  - c. The attacker now knows  $m$ , so he knows which message was encrypted, which breaks the CCA security.
2. The first bit of the message  $m$  was 0, then flipping the first two bits in the ciphertext will change 00 to 11 in the transformation  $Tr(m \parallel t)$  of the original message  $m$ .
  - a. This is bad, since there is no reverse mapping from 11 in  $F(b)$ , which means  $Tr^{-1}$  will give an error.
    - i. Remember, how  $F(b)$  works, it only maps bits to 00, 01 and 10. It doesn't map anything to 11.
  - b. Therefore, when this happens the decryption oracle will return  $\perp$ , indicating it couldn't decrypt the ciphertext.
  - c. The attacker now knows that the first bit of the message  $m$  was 0, since he knows the only error that was introduced was the flipping of the first two bits in the ciphertext which resulted in the first bits of  $Tr(m \parallel t)$  to be set to 11, making  $Tr^{-1}$  fail. This breaks the CCA security since the attacker will know  $m_0$  was the encrypted message, since  $m_1$  was all one's.

### Encrypt then authenticate

Our first example proved this construction was CCA-secure.

Alin Tomescu, CSE408

Tuesday, March 3<sup>rd</sup>, Lecture #10

## **Public-key cryptography**

Key distribution is a problem. Key revocation is a problem, if you want to change someone's key, you do that by changing everyone else's key to a new key. Public-key cryptography is here to solve these problems.