1 Plan for lectures 24 and 25

- Padding oracle attack: a practical attack on SSL 3.0. At the core of this attack is the malleability of CBC-mode encryption. Given $(\mathsf{ct}_0, \mathsf{ct}_1) \leftarrow \mathsf{CBC\text{-}Enc}(m, k)$ for some $m, k \in \{0,1\}^n$, and any $\alpha \in \{0,1\}^n$, $\mathsf{CBC\text{-}Dec}((\mathsf{ct}_0 \oplus \alpha, \mathsf{ct}_1), k) = m \oplus \alpha$. Note that this means that anyone can modify the underlying message (without knowing anything about the message or the key) by simply tampering the ciphertext.
- Semantic security + plaintext integrity is not good enough for real-world applications, due to chosen ciphertext attacks
- Security against chosen ciphertext attacks (CCA security) handles malleability/tampering attacks (however, does not imply plaintext integrity).
- Semantic security + ciphertext integrity ensures CCA security. Therefore, semantic security + ciphertext integrity is the 'gold standard' definition for security of symmetric key encryption schemes.
- 'Encrypt-then-MAC' achieves both semantic security and ciphertext integrity.

2 Padding oracle attack

First, let us recall the MAC-then-encrypt based encryption scheme, where the underlying encryption scheme is CBC-mode encryption. The following construction is taken verbatim from Lecture 23.

Encryption used in SSL 3.0

The key consists of two keys, an AES key k_{ro} and a MAC key k_{mac}.

- $\mathsf{CBC\text{-}Enc}(m, (k_{\text{ro}}, k_{\text{mac}}))$: Here, m is an arbitrary sequence of bytes (since data is assumed to be a sequence of bytes).
  1. First, the scheme computes a signature on the input message. We assume our MAC scheme can handle arbitrary length messages, and outputs a 16 byte signture. Therefore, the scheme computes $\sigma = \mathsf{Sign}(m, k_{\text{mac}})$.
  2. Next, we need to compute an encryption of the message, concatenated with the signature. Let $\widetilde{m} = m \mid\mid \sigma$. The CBC-mode encryption can only handle messages whose length (in bytes) is a multiple of 16, hence we need to pad $\widetilde{m}$ appropriately.

     Padding the message to fit the block: We need the message length to be a multiple of 16 bytes. Therefore, in the last block, we add zeroes, and in the last byte of the last block, we include the number of 'padded bytes'. See the following examples, where the message is described as a sequence of bytes, and each byte expressed as a number in $[0, 255]$.
     - $\widetilde{m}$ = (21 22 00 92 00). This message is 5 bytes long, and after padding, the message that's actually encrypted is a 16-byte message m' = (21 22 00 92 00 00 00 00 00 00 00 00 00 00 00 11).
     - $\widetilde{m}$ = (00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 16). This message is 16 bytes long. Therefore, to avoid ambiguity, have a new 'padding block'. The padded message is a 32-byte message m' = (00 00 ... 00 16 00 00 ... 00 16)
  3. Let m' be the message obtained from padding $\widetilde{m}$. This is a message whose length (in terms of number of bytes) is a multiple of 16. Let $m' = (m'_1, \ldots, m'_\ell)$, where each block is 16 bits long. Choose a uniformly random x (16 bytes long), and set $\mathsf{ct}_0 = x$. For all $i > 1$, compute $\mathsf{ct}_i = F(m'_i \oplus \mathsf{ct}_{i-1}, k)$.
  4. Finally, output $(\mathsf{ct}_0, \ldots, \mathsf{ct}_\ell)$ as the ciphertext.

- $\mathsf{CBC\text{-}Dec}((\mathsf{ct}_0, \ldots, \mathsf{ct}_\ell), (k_{\mathrm{ro}}, k_{\mathrm{mac}}))$: At a high level, the decryption algorithm first decrypts the ciphertext to obtain a message m and a signature σ. Next, it checks if σ is a valid signature on m. However, the exact implementation details are important here. They are specified below.

  1. Compute for each $i = 1$ to ℓ, $y_i = F^{-1}(\mathsf{ct}_i, k_{\mathrm{ro}}) \oplus \mathsf{ct}_{i-1}$. Note that each y_i is a sequence of 16 bytes.
  2. Check if y_ℓ is a valid padded string. That is, check that the last byte of y_ℓ is a number between 1 and 16. If the number is z, then check that the $z-1$ bytes before it are all 0. If any of these are violated, output **Error: bad padding**.
  3. Remove the last z bytes of y_ℓ, and let this truncated string be y'_ℓ. Let σ denote the last 16 bytes of $y_1 \mid\mid \ldots \mid\mid y_{\ell-1} \mid\mid y'_\ell$, and let m be the remaining bytes. Check if $\mathsf{Verify}(m, \sigma, k_{\mathrm{mac}}) = 1$. If this verification fails, output **Error: MAC verification failed**.
  4. If all the above checks pass, output m as the decrypted message.

Here are a few observations that will be useful for our attack. We will assume that AES is used as the block cipher, and hence the block size is 128 bits (16 bytes).

Observation 1: Let $m = (m_1, \ldots, m_\ell)$ be an ℓ-block message, and suppose $(\mathsf{ct}_0, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \leftarrow \mathsf{CBC\text{-}Enc}(m, k)$. Take any $i \in [\ell]$. Note that m_i is a 16 byte string. If this is a string with valid padding (that is, if the last byte is a number $p \in [16]$, and the previous $p-1$ bytes are all zeroes), then $\mathsf{CBC\text{-}Dec}((\mathsf{ct}_{i-1}, \mathsf{ct}_i), k)$ outputs that MAC verification failed (since the signature needs to be at least 16 bytes long, and with the valid padding removed, this cannot be a 16 byte string). If this is a string with invalid padding, then the decryption algorithm outputs that the padding is bad.

Observation 2: Let m be an ℓ-block message, and suppose $(\mathsf{ct}_0, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \leftarrow \mathsf{CBC\text{-}Enc}(m, k)$. Take any 16 byte string α. Then, for all i, let us consider $\mathsf{CBC\text{-}Dec}((\mathsf{ct}_{i-1} \oplus \alpha, \mathsf{ct}_i), k)$. If $m_i \oplus \alpha$ is a string with invalid padding, then the decryption outputs that the padding is invalid. Otherwise, as before, it outputs that the signature is invalid.

These two observations are sufficient for our attack. The attacker receives a ciphertext $\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$ from the client. It repeatedly tampers the ciphertext, sends it to the server, and analyses the error messages received.

- *Learning the padding length:*

  1. Suppose there are p bytes of padding in the last block. First, the adversary tries to find if $p = 1$. It takes a string α_1 which is 16 bytes long, and the second-last byte is non-zero (all other bytes are zeroes). It sends $(\mathsf{ct}_{\ell-1} \oplus \alpha_1, \mathsf{ct}_\ell)$ to the server. If it receives a bad signature error, then it concludes that $p = 1$. If it receives a bad padding error, it concludes that $p \geq 2$.

     Next, if $p \geq 2$ it picks another α_2 whose third-last byte is non-zero (and all other bytes are zeroes). It sends $(\mathsf{ct}_{\ell-1} \oplus \alpha_2, \mathsf{ct}_\ell)$ to the server. Again, if it receives a bad padding error, then it concludes that $p \geq 3$. If it receives a bad signature error, then it concludes that $p = 2$.

     Continuing this way, it learns the exact padding length. Convince yourself why this leads to a correct recovery of the padding length.

- *Learning the last non-padding byte of the message* The adversary now knows that there are p bytes of padding. It wishes to recover the last byte of the message.

  1. Consider a string α_0 which is zeroes in all bytes, except the last byte. In the last byte, the bits are such that the last byte of α_0, XORd with the binary representation of p, results in binary representation of $p+1$. The attacker sends $(\mathsf{ct}_{\ell-1} \oplus \alpha_0, \mathsf{ct}_\ell)$ to the server. If the server does not

give a bad padding error, then the adversary concludes that the last non-padding message byte is 0 (why is this correct?).

2. If it receives a bad padding error, then that means that the last non-padding message byte is non-zero. It sets α_1, which is same as α_0, except in the $(16-p)^{\text{th}}$ byte, where it is 1. The attacker sends $(\mathsf{ct}_{\ell-1} \oplus \alpha_1, \mathsf{ct}_\ell)$ to the server. If the server does not give a bad padding error, then the adversary concludes that the last non-padding message byte is 1 (why is this a correct conclusion?).
3. Continuing this way, in 256 queries, it figures out the last non-padding byte. At some point, the decryption results in a MAC verification error.

This approach can be used to learn all the bytes of the last block.

- *Learning the last byte of the second last block* Let $m' = (m'_1, \ldots, m'_\ell)$ be the string underlying our ciphertext. We know that this is a string with valid padding. Let us focus our attention on just $m'_{\ell-1}$. If the last byte of $m'_{\ell-1}$ is 1, then $m'_{\ell-1}$ is a string with valid padding. Similarly, if the last byte of $m'_{\ell-1}$ is 2, and the second last byte is 0, then also $m'_{\ell-1}$ is a string with valid padding. In general, if the last byte of $m'_{\ell-1}$ is $p \leq 16$, and the $p-1$ bytes before it are all 0, then $m'_{\ell-1}$ is a string with valid padding.

  As a result, the adversary can first send $(\mathsf{ct}_{\ell-2}, \mathsf{ct}_{\ell-1})$ (without any modification). If it receives a bad padding error, then it knows that the last byte is not 1. Suppose all the bytes of $m_{\ell-2}$ are non-zero.[1] Then, there is exactly one value that, when added to the last byte, makes this a valid padded string. As a result, by altering the last byte of $\mathsf{ct}_{\ell-2}$ appropriately and repeatedly, it can figure out what the last byte of the second last block is. Once the last byte is figured out, it can apply the same approach as above, and learn the entire block, and hence the entire message.

This attack illustrates that just plaintext integrity and semantic security are not enough. There is a lot of information leakage from decryptions, and therefore, the adversary must be allowed to make decryption queries. This motivates *security against chosen ciphertext attacks*. This security game is very similar to the semantic security game, except that the adversary is now allowed to make decryption queries to the challenger. It should not query for decryption of the challenge ciphertext, but is allowed decryptions of any other ciphertexts of its choice.

3 Security against chosen ciphertext attacks

Security against chosen ciphertext attacks is captured via the following security game (defined in Figure 1).

Note that there is only one restriction on the queries: the post-challenge decryption queries should not be equal to the challenge ciphertext. Other than this, the adversary can request for decryptions of ciphertexts that it received from the challenger. It can also request for encryption of the same message, multiple times.

CCA security already takes care of a number of malleability/tampering attacks.

- Let $(\mathsf{Enc}, \mathsf{Dec})$ be a CCA secure encryption scheme. Then no p.p.t. adversary can tamper the encryption of m to produce an encryption of $m \oplus \alpha$. (Note, in this attack scenario, the adversary does not know m or the encryption/decryption key).

  Suppose an adversary could do this efficiently. Then we can use this adversary to break CCA security. The reduction algorithm sends two distinct messages m_0, m_1. It receives a challenge ciphertext ct^*. It then uses the adversary to alter ct^*, resulting in an encryption of either $m_0 \oplus \alpha$, or $m_1 \oplus \alpha$. Let ct' be the tampered ciphertext. The adversary sends this tampered ciphertext as a decryption query, and receives $m_b \oplus \alpha$. Given this information, the reduction algorithm now knows b.

[1] We can always add random values to the other bytes, thereby ensuring that they are non-zero.

Security against Chosen Ciphertext Attacks

1. **Setup:** Challenger chooses an encryption key $k \leftarrow \mathcal{K}$ and a bit $b \leftarrow \{0,1\}$.
2. **Pre-Challenge queries:** Adversary sends polynomially many queries (adaptively). Each query can either be an encryption query, or a decryption query.
   - The i^{th} enc. query consists of message m_i. The challenger sends $\mathsf{ct}_i = \mathsf{Enc}(m_i, k)$.
   - The i^{th} dec. query consists of ciphertext ct'_i. The challenger sends $y_i = \mathsf{Dec}(\mathsf{ct}'_i, k)$.

   Note that the key k was chosen during setup, and the same key is used for all queries. The bit b is not used here.
3. **Challenge:** After the encryption queries, the adversary sends two messages (m_0^*, m_1^*) and receives $\mathsf{ct}^* = \mathsf{Enc}(m_b^*, k)$.
   Note that the key k and bit b were chosen during setup phase.
4. **Post-Challenge Encryption queries:** Adversary sends polynomially many queries (adaptively). This is similar to the pre-challenge phase.
   - The i^{th} enc. query consists of message m_i. The challenger sends $\mathsf{ct}_i = \mathsf{Enc}(m_i, k)$.
   - The i^{th} dec. query consists of ciphertext ct'_i. If $\mathsf{ct}'_i \neq \mathsf{ct}^*$, the challenger sends $y_i = \mathsf{Dec}(\mathsf{ct}'_i, k)$.
5. **Guess:** The adversary sends its guess b', and wins the security game if $b = b'$.

Figure 1: Security against Chosen Ciphertext Attacks

Definition 24.01. An encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ is secure against chosen ciphertext attacks if, for any p.p.t. adversary $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$ such that for all n,

$$\Pr\left[\mathcal{A} \text{ wins in the CCA security game}\right] \leq 1/2 + \mu(n).$$

- Let $(\mathsf{Enc}, \mathsf{Dec})$ be a CCA secure encryption scheme. Then no p.p.t. adversary, given encryptions of m_1 and m_2, can produce an encryption of $m_1 \oplus m_2$. (Note, in this attack scenario, the adversary does not know m_1, m_2 or the encryption/decryption key).

  Suppose an adversary can do this efficiently. Then the reduction algorithm sends two distinct messages m_0, m_1 to the challenger, and recevies ct^* as the challenge ciphertext. It then queries for encryption of another message m' and receives ct_1. Next, it uses the adversary to produce either an encryption of $m_0 \oplus m'$ or $m_1 \oplus m'$. Let ct' denote this ciphertext. The reduction sends ct' to the CCA challenger, receives $m_b \oplus m'$, which reveals the bit b.

From the above examples, it appears that CCA security already addresses the issue of plaintext/ciphertext integrity. However, that's not true. There exist CCA schemes that do not satisfy plaintext/ciphertext integrity. See Exercise 9.12 in the textbook.

Qn: What is the relevance of pre and post-challenge queries?

Ans: We want to model real-world attacks, where the adversary might see some ciphertexts/query for some decryptions, then it finds a ciphertext that it wants to attack. After finding this ciphertext, it may make see more ciphertexts, or make more decryption queries before figuring out how to attack. In practice, the post-challenge queries are more useful.
From the viewpoint of a definition, the pre-challenge queries can help you to find the appropriate challenge messages, and the post-challenge queries can help you break the security.

4 Semantic security + ciphertext integrity implies CCA security

Let $(\mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme that satisfies both semantic security and ciphertext integrity. In this section, we will show that the scheme is also CCA secure. To argue CCA security, the reduction algorithm must handle decryption queries. If the decryption query is equal to one of the previous encryption queries, then the reduction clearly knows the decryption. On the other hand, if the adversary sends a fresh ciphertext for decryption, this ciphertext must be an invalid ciphertext (otherwise the scheme does not satisfy ciphertext integrity).

This is the base of the scheme.

Theorem 24.01. Let $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme that satisfies both semantic security and ciphertext integrity. Then, for every p.p.t. adversary $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$ such that for all n, $\Pr\left[\mathcal{E} \text{ wins the CCA security game against } \mathcal{E}\right] \leq \mu(n)$.

Proof. As usual, we will start with the CCA security game, and alter it gradually, until we reach a game where the adversary has negligible advantage.

- **Game 0:** This is the CCA game against $\mathcal{E}$.
  - *Setup phase* Challenger chooses a key k and a bit b.
  - *Pre-challenge query phase* Adversary can make polynomially many queries.
    For each encryption query m_i, the challenger sends $\mathsf{ct}_i \leftarrow \mathsf{Enc}(m_i, k)$.
    For each decryption query ct'_i, the challenger sends $y_i = \mathsf{Dec}(\mathsf{ct}'_i, k)$.
  - *Challenge phase* Adversary sends two messages (m_0^*, m_1^*). Challenger sends $\mathsf{ct}^* = \mathsf{Enc}(m_b^*, k)$.
  - *Post-challenge query phase* Adversary can make polynomially many queries.
    For each encryption query m_i, the challenger sends $\mathsf{ct}_i \leftarrow \mathsf{Enc}(m_i, k)$.
    For each decryption query ct'_i $(\neq \mathsf{ct}^*)$, the challenger sends $y_i = \mathsf{Dec}(\mathsf{ct}'_i, k)$.
  - *Guess* Adversary sends its guess b' and wins if $b = b'$.

- **Game 1:** This game is similar to the previous one, except that the challenger does not use the secret key for decryption. Instead, it maintains a list of all encryption queries made. If a decryption query is in the list, the challenger sends the corresponding message, else it sends $\perp$.
  - *Setup phase* Challenger chooses a key k and a bit b.
    The challenger also maintains a list $\mathcal{T}$ of (message, ciphertext) pairs which is initially empty.
  - *Pre-challenge query phase* Adversary can make polynomially many queries.
    For each encryption query m_i, the challenger sends $\mathsf{ct}_i \leftarrow \mathsf{Enc}(m_i, k)$ and adds (m_i, ct_i) to $\mathcal{T}$.
    For each decryption query ct'_i, the challenger checks if $\exists\ (m', \mathsf{ct}'_i) \in \mathcal{T}$ for some m'.
    If so, it sends m', else it sends $\perp$.
  - *Challenge phase* Adversary sends two messages (m_0^*, m_1^*). Challenger sends $\mathsf{ct}^* = \mathsf{Enc}(m_b^*, k)$.
  - *Post-challenge query phase* Adversary can make polynomially many queries.
    For each encryption query m_i, the challenger sends $\mathsf{ct}_i \leftarrow \mathsf{Enc}(m_i, k)$ and adds (m_i, ct_i) to $\mathcal{T}$.
    For each decryption query ct'_i $(\neq \mathsf{ct}^*)$, the challenger checks if $\exists\ (m', \mathsf{ct}'_i) \in \mathcal{T}$ for some m'.
    If so, it sends m', else it sends $\perp$.
  - *Guess* Adversary sends its guess b' and wins if $b = b'$.

Analysis: Let p_b be the winning probability in Game b.

Claim 24.01. If there exists a p.p.t. adversary $\mathcal{A}$ that makes Q decryption queries, and $p_0 - p_1$ is non-negligible, then there exists a p.p.t. algorithm $\mathcal{B}$ that wins the ciphertext integrity game with probability $(p_0 - p_1)/Q$.

Proof Sketch: The only difference in the two games is if one of the decryption queries is a fresh ciphertext that decrypts to a non-$\perp$ value. The reduction algorithm can guess the first location i^* of such a decryption query. This guess is correct with probability $1/Q$. For all the encryption queries before this one, the reduction simply forwards them to the ciphertext integrity challenger. For all decryption queries before this one, it either uses the list $\mathcal{T}$, or sends $\perp$. Finally, on receiving the $i^{*\text{th}}$ decryption query, it forwards this query to the ciphertext integrity challenger.

Claim 24.02. There exists a p.p.t. algorithm $\mathcal{B}$ that wins the semantic security game with probability p_1.

Proof sketch: In Game 1, the decryption queries are handled without the secret key. Therefore, the reduction algorithm can interact with the query-based semantic security challenger and the CCA adversary. For the encryption queries, it forwards them to the challenger, and stores the (message, ciphertext) pairs in a list $\mathcal{T}$. For the decryption queries, it uses the list $\mathcal{T}$, and if a query is not present in the list, it outputs $\perp$.

From Claim 24.02, it follows that p_1 is at most $1/2 + \mathsf{negl}$. From Claim 24.01, it follows that p_0 is negligibly close to p_1, and hence at most $1/2 + \mathsf{negl}$.

□

5 Authenticated Encryption: the 'gold-standard' security definition for symmetric key encryption

The previous section shows that semantic security + ciphertext integrity implies security against chosen ciphertext attacks. Note that ciphertext integrity also implies plaintext integrity. Therefore, the combination of semantic security and ciphertext integrity is taken as the gold-standard definition for security against active attacks. It is commonly referred to as 'authenticated encryption'.

Definition 24.02. An encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ is said to be an authenticated encryption scheme if it satisfies semantic security and ciphertext integrity.

We saw that an authenticated encryption scheme satisfies CCA security. Does the converse also hold true? No, because CCA security doesn't necessarily imply (plaintext/ciphertext) integrity. However, CCA security, together with plaintext integrity, implies ciphertext integrity (and hence is equivalent to authenticated encryption).

6 'Encrypt-then-MAC' is an authenticated encryption scheme

Let $\mathcal{E}_{\text{r.o.}} = (\mathsf{Enc}_{\text{ro}}, \mathsf{Dec}_{\text{ro}})$ be a semantically secure encryption scheme, and $(\mathsf{Sign}, \mathsf{Verify})$ a MAC scheme. Recall the following encryption scheme, where we first encrypt the message, then compute a MAC on the ciphertext.

Encrypt-then-MAC

- $\mathsf{Enc}(m, (k_{\text{ro}}, k_{\text{mac}}))$: The encryption algorithm first computes $\mathsf{ct}_{\text{ro}} \leftarrow \mathsf{Enc}_{\text{ro}}(m, k_{\text{ro}})$. Next, it computes $\sigma \leftarrow \mathsf{Sign}(\mathsf{ct}_{\text{ro}}, k_{\text{mac}})$. The final ciphertext is $(\mathsf{ct}_{\text{ro}}, \sigma)$.
- $\mathsf{Dec}((\sigma, \mathsf{ct}_{\text{ro}}), (k_{\text{ro}}, k_{\text{mac}}))$: To decrypt a ciphertext $(\mathsf{ct}_{\text{ro}}, \sigma)$, first check the signature; that is, check if $\mathsf{Verify}(\mathsf{ct}_{\text{ro}}, \sigma, k_{\text{mac}}) = 1$. If this check passes, output $\mathsf{Dec}_{\text{ro}}(\mathsf{ct}_{\text{ro}}, k_{\text{ro}})$.

Claim 24.03. Assuming $\mathcal{E}_{\text{r.o.}}$ satisfies semantic security, the above construction satisfies semantic security.

Proof sketch: The reduction algorithm, on receiving a message pair from the attacker, forwards it to the challenger for $\mathcal{E}_{\text{r.o.}}$. It receives a challenge ciphertext from the challenger, computes a signature on the ciphertext, and forwards the signature together with the ciphertext.

Claim 24.04. Assuming MAC is a strongly unforgeable MAC scheme, the above construction satisfies ciphertext integrity.

Proof sketch: The reduction algorithm interacts with the MAC challenger. It first chooses an encryption key k_{ro}. On receiving an encryption query m_i from the adversary, the reduction computes an encryption $\mathsf{ct}_{\text{ro},i}$ of m_i using k_{ro}. It then forwards $\mathsf{ct}_{\text{ro},i}$ to the signing algorithm, and receives a signature σ_i. It sends $(\mathsf{ct}_{\text{ro},i}, \sigma_i)$ to the adversary.

After polynomially many ciphertext queries, the adversary outputs a fresh ciphertext $\mathsf{ct}^* = (\mathsf{ct}^*_{\text{ro}}, \sigma^*)$ such that it is a valid ciphertext, and is not equal to any $(\mathsf{ct}_{\text{ro},i}, \sigma_i)$. This means that σ^* is a valid signature on $\mathsf{ct}^*_{\text{ro}}$, and since $(\mathsf{ct}^*_{\text{ro}}, \sigma^*) \notin \{(\mathsf{ct}_{\text{ro},i}, \sigma_i)\}_i$, this is a valid signature forgery.

6.1 Bad implementations of 'encrypt-then-MAC'

- The above scheme is secure only if the entire ciphertext is signed by the MAC signing algorithm. For some reason, a common implementation flaw is to ignore part of the ciphertext while signing. Take CBC-mode encryption for example. There have been some implementations where the first ciphertext component (the random string) was not signed by the MAC signing algorithm. The resuting scheme is not CCA-secure.
- When $\mathcal{K}_{\text{enc,ro}} = \mathcal{K}_{\text{mac}}$, sometimes implementations use the same key for both signing and encryption. This may also lead to vulnerabilities. See exercise 9.8 in the textbook.

7 Lecture summary, plan for next lecture, additional resources

Summary: In these two lectures, we saw the padding oracle attack, which shows that semantic security + message integrity are not enough; we need a stronger definition that can handle information leakage from decryption queries. This is captured by security against chosen ciphertext attacks. Next, we showed that semantic security + ciphertext integrity implies security against chosen ciphertext attacks. An encryption scheme is said to be a secure authenticated encryption scheme if it satisfies both semantic security and ciphertext integrity.

Next lecture: This lecture concludes our discussion of private/symmetric key cryptography. Next lecture, we will start with public key cryptography.

Relevant sections from textbook [Boneh-Shoup]: Sections 9.1 to 9.4

8 Questions

Question 24.01. Show that the MAC-then-Encrypt approach gives an encryption scheme that satisfies both semantic security and plaintext integrity.

Question 24.02. Let $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme secure against chosen ciphertext attacks. Let $\mathsf{MAC} = (\mathsf{Sign}, \mathsf{Verify})$ be a strongly unforgeable message authentication code. Consider the following encryption scheme:

- $\mathsf{Enc}'(m, (k_1, k_2, k_3))$: First compute $\mathsf{ct}_1 \leftarrow \mathsf{Enc}(m, k_1)$. Next, compute $\mathsf{ct}_2 \leftarrow \mathsf{Enc}(m, k_2)$ and $\sigma \leftarrow \mathsf{Sign}(\mathsf{ct}_2, k_3)$. The encryption scheme outputs $(\mathsf{ct}_1, \mathsf{ct}_2, \sigma)$.
- $\mathsf{Dec}'((\mathsf{ct}_1, \mathsf{ct}_2, \sigma), (k_1, k_2, k_3))$: The decryption algorithm first verifies the signature by computing $\mathsf{Verify}(\mathsf{ct}_2, \sigma, k_3)$. Next, it computes $m_1 = \mathsf{Dec}(\mathsf{ct}_2, k_2)$ and $m_2 = \mathsf{Dec}(\mathsf{ct}_1, k_1)$. If $m_1 \neq m_2$, it outputs $\perp$, else it outputs m_1.

Show that the encryption scheme $(\mathsf{Enc}', \mathsf{Dec}')$ is not secure against chosen ciphertext attacks.

Question 24.03. Consider the encrypt-then-MAC approach, but the MAC used is only weakly unforgeable. That is, there exists a p.p.t. adversary that, given a signature on some string z, produces a new signature on z. Show that the resulting encrypt-then-MAC scheme is not secure against chosen ciphertext attacks.