1 Plan for today's lecture

Today, we will discuss the 'gold-standard' security notion for encryption schemes — one that guarantees both confidentiality and integrity. In the first half of the semester, we introduced encryption schemes that are secure against 'read-only' attacks (these ensure confidentiality of the message), and we discussed message authentication codes (which ensure integrity of the message). Today, we will look for meaningful combinations of these two primitives that allow us confidentiality as well as integrity.

2 The building blocks

There are two main building blocks for our 'gold-standard' encryption scheme. The first is an encryption scheme that handles unbounded length messages, and is semantically secure. We will use the CBC-mode encryption scheme, since it is a widely used encryption mode in practice and you have already proven its security in one of the previous assignments (moreover, looking ahead, it has an important role to play in our story). Let us recall the construction here.

CBC-mode encryption

The construction uses a PRP $F : \mathcal{X} \times \mathcal{K} \to \mathcal{X}$. In practice, $\mathcal{X} = \mathcal{K} = \{0,1\}^{128}$.

- $\mathsf{Enc}(m = (m_1, \ldots, m_\ell), k)$: Choose a random $x \leftarrow \mathcal{X}$, set $\mathsf{ct}_0 = x$. For all $i > 0$, compute $\mathsf{ct}_i = F(m_i \oplus \mathsf{ct}_{i-1}, k)$. The final ciphertext is $(\mathsf{ct}_0, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)$.
- $\mathsf{Dec}(\mathsf{ct} = (\mathsf{ct}_0, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell), k)$: For all $i > 0$, compute $y_i = F^{-1}(\mathsf{ct}_i, k) \oplus \mathsf{ct}_{i-1}$.

Strictly speaking, this construction only handles messages whose length is a multiple of 128. We did not discuss how to handle arbitrary length messages. This will be important for today's lecture, and we will come back to it later in the lecture. For now, assume that there exists some way to transform a message to one whose length is a multiple of the block size.

The second building block is a strongly unforgeable MAC scheme that can handle unbounded length messages. We can use the encrypted CBC-MAC here, and I'm including the construction below for completeness.

Encrypted CBC-MAC

The scheme uses a PRF $F : \mathcal{X} \times \mathcal{K} \to \mathcal{X}$, and can handle unbounded length messages. Let $m = (m_1, \ldots, m_\ell)$ be the message to be signed. The secret key consists of two PRF keys k_1, k_2.

- $\mathsf{Sign}(m, (k_1, k_2))$: Let $y_1 = F(m_1, k_1)$. For $i \in [2, \ell]$, the signing algorithm computes $y_i = F(m_i \oplus y_{i-1}, k_1)$. Finally, it outputs $\sigma = F(y_\ell, k_2)$.
- $\mathsf{Verify}(m, \sigma, (k_1, k_2))$: Let $y_1 = F(m_1, k_1)$. For $i \in [2, \ell]$, the verification algorithm computes $y_i = F(m_i \oplus y_{i-1}, k_1)$. Finally, it outputs 1 iff $\sigma = F(y_\ell, k_2)$.

Again, we have conveniently assumed that the message length is a multiple of the block size. Assignment 3 discusses how to handle arbitrary length messages.

3 Combining semantically secure encryption and secure MAC for ensuring confidentiality and integrity

Let $\mathcal{E}_{\text{r.o.}} = (\mathsf{Enc}_{\text{ro}}, \mathsf{Dec}_{\text{ro}})$ be a semantically secure encryption scheme with key space $\mathcal{K}_{\text{enc,ro}}$ (such as CBC-mode encryption). Here the 'ro' subscript denotes 'read-only'. Let $\mathsf{MAC} = (\mathsf{Sign}, \mathsf{Verify})$ be a strongly unforgeable MAC with key space $\mathcal{K}_{\text{mac}}$. [1] Our goal is to build an encryption scheme $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$, with appropriate key space, that offers both integrity and confidentiality.

There are a few different ways to put together these building blocks.

- **Encrypt and MAC**: In this approach, the key space is $\mathcal{K}_{\text{enc,ro}} \times \mathcal{K}_{\text{mac}}$. To encrypt a message m using key $(k_{\text{ro}}, k_{\text{mac}})$, compute $\mathsf{ct}_{\text{ro}} \leftarrow \mathsf{Enc}_{\text{ro}}(m, k_{\text{ro}})$ and $\sigma \leftarrow \mathsf{Sign}(m, k_{\text{mac}})$ and output $(\mathsf{ct}_{\text{ro}}, \sigma)$ as the final ciphertext.

  As discussed on Piazza, this seems like a bad idea since the signature may reveal some information about the message. (Recall, signatures don't offer any confidentiality of the message).

- **MAC then encrypt**: In this approach, the encryption algorithm first computes a signature on the message, then it encrypts the message together with the signature. The key space is again $\mathcal{K}_{\text{enc,ro}} \times \mathcal{K}_{\text{mac}}$. To encrypt a message m using key $(k_{\text{ro}}, k_{\text{mac}})$, first compute $\sigma \leftarrow \mathsf{Sign}(m, k_{\text{mac}})$. Next, compute $\mathsf{ct} \leftarrow \mathsf{Enc}_{\text{ro}}(m \mid\mid \sigma, k_{\text{ro}})$ and output it as the final ciphertext.

  Decryption works as follows: given a ciphertext ct, first compute $(m \mid\mid \sigma) = \mathsf{Dec}_{\text{ro}}(\mathsf{ct}, k_{\text{ro}})$. Next, check if $\mathsf{Verify}(m, \sigma, k_{\text{mac}}) = 1$. If verification passes, then output the message m.

- **Encrypt then MAC**: Here, 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 $(\mathsf{ct}_{\text{ro}}, \sigma)$.

  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 $\mathsf{Dec}_{\text{ro}}(\mathsf{ct}_{\text{ro}}, k)$.

In order to judge which of these is secure against tampering attacks, we will need a security definition.

4 Defining security against tampering attacks

Recall, we want both confidentiality and integrity. Confidentiality is ensured by the semantic security game. What would be an appropriate security game for integrity? The following two games were proposed in class,[2] both are analogous to the MAC security game (with variations in the winning condition). In all these games, the adversary can query for encryptions of messages of its choice, and must finally exhibit an 'integrity attack'. All these games are with respect to an encryption scheme $\mathcal{E} = (\mathsf{Enc}, \mathsf{Dec})$. We assume that decryption outputs a special symbol $\perp$ whenever decryption fails.

- **Attempt 1:**

PTXT-INT

- (Setup Phase) Challenger chooses an encryption key k.
- (Encryption Queries) Adversary sends polynomially many encryption queries. For the i^{th} query message m_i, it receives $\mathsf{ct}_i \leftarrow \mathsf{Enc}(m_i, k)$.
- (Integrity Attack) Adversary finally outputs a ciphertext ct^* and wins if $m^* = \mathsf{Dec}(\mathsf{ct}^*, k) \neq \perp$, and $m^* \notin \{m_i\}_i$.

Figure 1: Attempt 1 for defining integrity attacks

[1] I am skipping the key generation algorithm for both these primitives.

[2] In class, we also saw a third security game, but I am not listing it below, since that might be a slight distraction at this point.

What is the above security game capturing? Suppose an encryption scheme is such that no polynomial time adversary can succeed in the above security game. That means an adversary can see the interaction between Alice and Bob, but it cannot produce an encryption of a **new** message. It can still produce a different encryption of some old message though. Is this good enough for practice? Depends on the application.

- **Attempt 2:**

CTXT-INT

- (Setup Phase) Challenger chooses an encryption key k.
- (Encryption Queries) Adversary sends polynomially many encryption queries. For the i^{th} query message m_i, it receives $\mathsf{ct}_i \leftarrow \mathsf{Enc}(m_i, k)$.
- (Integrity Attack) Adversary finally outputs a message and ciphertext (m^*, ct^*) and wins if $m^* = \mathsf{Dec}(\mathsf{ct}^*, k)$, and $m^* \notin \{m_i\}_i$.

Figure 2: Attempt 2 for defining integrity attacks

What is the above security game capturing? This one seems stronger than the first attempt. If no p.p.t. adversary can win in this game, then the adversary can't even produce a new ciphertext for an old message! Are there any real world scenarios where this stronger definition would be needed? As it turns out, this is the definition of integrity that we should use in practice. To understand why Attempt 2 should be preferred over Attempt 1, let us consider a real-world encryption scheme deployment that satisfies Attempt 1, but is completely broken in practice.

5 Mac-then-encrypt in real-world protocols

The Secure Sockets Layer (SSL 3.0) protocol was a widely deployed protocol for securing internet traffic. At its core is the following encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$. At a high level, the scheme employs the mac-then-encrypt approach, with the hope of guaranteeing data hiding and integrity. There are lots of details here, the reader is encouraged to go over them carefully. It uses AES with key space and input/output space being $\{0,1\}^{128}$ for encrypting, and a MAC scheme $\mathsf{MAC}_{\text{bdd}} = (\mathsf{Sign}, \mathsf{Verify})$ with key space and signature space $\{0,1\}^{128}$.

Encryption used in SSL 3.0

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

- $\mathsf{Enc}(m, (k_{\text{ro}}, k_{\text{mac}}))$: 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. 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 bits) is a multiple of 128, 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 128. 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 $[0, 255]$.

     - $m = (21\ 22\ 00\ 92\ 00)$. This message is 5 bytes long, and after padding, the message that's actually encrypted is a 128-bit message $m' = $ (21 22 00 92 00 00 00 00 00 00 00 00 00 00 00 11).

  - $m =$ (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 256-bit 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 bits) is a multiple of 128. Let $m' = (m'_1, \ldots, m'_\ell)$, where each block is 128 bits long. Choose a uniformly random $x \leftarrow \{0,1\}^{128}$, 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.

- $\mathsf{Dec}((\mathsf{ct}_0, \ldots, \mathsf{ct}_\ell), (k_{\mathrm{ro}}, k_{\mathrm{mac}}))$: 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.

One can easily show that the above encryption scheme satisfies semantic security. With a little effort, one can also show that no p.p.t. adversary can win the PTXT-INT game against this encryption scheme. And this scheme was deployed in practice (this was part of the SSL 3.0 protocol). In the deployed prootcol, suppose a client wants to send an encryption of message m to a server. The server behaves as follows: if it is a valid ciphertext, it proceeds with some computation (and sends no error message). However, if decryption fails (either due to the bad padding, or due to failed verification), the server sends this error message.

There was a devastating real-world attack against this scheme that totally recovers the underlying message. Here is how the attack works:

- the attacker intercepts the ciphertext sent by the client.
- it tampers the ciphertext and sends it to the challenger. On most occasions, the tampering results in 'bad padding' error, and when the attacker sends the tampered ciphertext to the server, it receives a 'bad padding' error.
- sometimes, the ciphertext is a valid ciphertext, in which case the server sends nothing, and the attacker uses this to learn something about the message. With enough iterations, it learns the entire message.

As a first step, try the following: suppose you are the person-in-the-middle, and you intercept a ciphertext sent from a client to the server. This ciphertext is the encryption of some message, and you can see the number of blocks in the ciphertext. Your goal is to find the exact length of the message encrypted (in terms of number of bytes in the message). You are allowed to send at most 20 ciphertexts to the server, and observe its behaviour.

Also, think about the following: Mac-then-encrypt is a semantically secure encryption scheme, and it also satisfies plaintext integrity (that is, no p.p.t. adversary can win the game described in Figure 1). We will prove both in one of the following lectures. Then why is the scheme described in Construction 5 broken?

6 Lecture summary, plan for next lecture, additional resources

Summary: We saw three constructions for achieving tamper resilience. One of them did not guarantee message confidentiality. The other two can be potentially secure. We discussed two security games for integrity. The first one guarantees integrity of the message, the second one guarantees integrity of ciphertext. Finally, we started discussing the attack on SSL protocol.

Next lecture: We will complete our description of the attack. Next, we will show that Encrypt-then-MAC satisfies both semantic security and ciphertext integrity.

Relevant sections from textbook [Boneh-Shoup]: Section 9.1 of the book introduces the definitions, and the various combinations of $\mathsf{MAC}_{\mathrm{bdd}}$ and $\mathcal{E}_{\mathrm{r.o.}}$ are discussed in Section 9.4.