Authenticated Encryption

6.1600 Course Staff

Fall 2023

Disclaimer: This set of notes is a work in progress. It may have errors and be missing citations. It is certainly incomplete. Please let the staff know of any errors that you find.

We have just constructed an encryption scheme with weak (CPA) security: one that provided security given that the adversary could see *encryptions* of messages of her choice. The "gold standard" security notion for encryption schemes allows the attacker to receive both encryptions of messages of its choice *and* decryptions of ciphertexts of its choice. Our security notion then says that even an attacker with this power should not be able to distinguish which of two chosen plaintext message a given ciphertexts encrypts. This new and strong notion of security for encryption schemes is called *security against adaoptive chosen-ciphertext attacks*.

Motivation: Chosen-ciphertext security It is not clear why chosen-ciphertext security is the right security notion to consider. In real-life applications, why would we ever allow an attacker to obtain *decryptions* of ciphertexts of its choosing? It turns out that, in many settings, attackers can indeed trick honest parties into decrypting adversarially ciphertexts and revealing their contents.

As a simple example, imagine a server that receives encrypted requests from the network, decrypts them, and either:

- returns an error if the decrypted request is malformed, or
- silently processes the request otherwise.

In this case, an attacker can send ciphertexts to the server and learn information about their decryptions by noticing whether or not the server returned an error message.

If the server returns an error when the decrypted message is illformed, we often call it a *padding oracle*. Many many real-world protocols using non-chosen-ciphertext-secure encryption schemes fall victim to this sort of attack.

CPA-secure encryption schemes provide *no* security guarantees in this setting. Even if the server leaks a single bit to the attacker about the decrypted value (such as whether the decrypted ciphertext is a well-formed request or not), the attacker could potentially learn the entire secret key!

As a concrete application-level example, consider what could happen if SSH (the secure shell protocol) used the counter-mode encryption scheme, as described in the previous lecture, without any authentication. A user might send the command `echo secret > secret-file` to write their secret string, `secret`, to their own private file `secret-file`. The encryption of this command, sent over the network, would be the XOR of the command and the PRF-generated pseudorandom bytes. Suppose that the adversary knows the user

is running this command, but doesn't know the contents of the 6-byte secret string. Consider what happens if the adversary XORs the encrypted message with the hexadecimal bytes `00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 5c 11 0e 02 4a 04 58 04 05 05 06`. The leading zeroes mean that the first part of the command, `echo secret > `, will remain unchanged. When the server decrypts the latter part of the command (originally `secret-file`), however, it will obtain the XOR of `secret-file` and the bytes that the adversary XOR'ed in, which happens to turn into the string `/tmp/public`. As a result, if the server now runs this command, the user's secret data will be written to a file `/tmp/public`, which might be available to an adversary that also has an account on that same server and can look at files in `/tmp`.

Chosen-ciphertext security guarantees that such attacks are ineffective.

1 *Defining Authenticated Encryption*

Authenticated-encryption schemes simultaneously provide message integrity (as a MAC does) and confidentiality (as CPA-secure encryption does). Additionally, authenticated-encryption schemes remain secure even when an attacker can see encryptions and decrypts of ciphertexts of her choice. Perhaps unsurprisingly, the standard way to construct an authenticated-encryption scheme is to combine a CPA-secure encryption scheme with a MAC in a careful way.

We now formally define our strong security notion: *indistinguishability under chosen ciphertext attacks*, also known as IND-CCA2 security or CCA security.

Definition 1.1 (CCA security for encryption (strong))**.** An encryption scheme is *secure against adaptive chosen-ciphertext attacks* if every efficient adversary wins the following game with probability at most $\frac{1}{2}$ + "negligible":

- The challenger samples $b \xleftarrow{R} \{0,1\}$ and $k \xleftarrow{R} \mathcal{K}$.
- The adversary can make either of the following queries to the challenger repeatedly:
  - *Chosen-plaintext queries*
    - The adversary sends the challenger a message $m_i \in \mathcal{M}$
    - The challenger replies with $c_{m_i} \leftarrow \mathsf{Enc}(k, m_i)$.
  - *Chosen-ciphertext queries*
    - The adversary sends the challenger a *ciphertext* $c_j \notin \{c_{m_0}, \ldots, c_{m_i}\}$
    - The challenger replies with $m_{c_j} \leftarrow \mathsf{Dec}(k, c_j)$.
- The adversary then sends two messages $(m_0^*, m_1^*) \in \mathcal{M}^2$ to the challenger, where $|m_0^*| = |m_1^*|$.

- The challenger replies with $c^* \leftarrow \mathsf{Enc}(k, m_b^*)$.
- The adversary can make more chosen-plaintext queries and more chosen-ciphertext queries. (The adversary may not make a chosen-ciphertext query on the challenge ciphertext c^*.)
- The adversary outputs a value $b' \in \{0,1\}$.
- The adversary wins if $b = b'$.

1.1 Encrypt then MAC

We typically achieve CCA security using the "encrypt-then-MAC" construction:

- First, encrypt the message using a CPA-secure encryption scheme on key k_{Enc}.
- Next, MAC the *ciphertext* using a secure MAC scheme and an independent key k_{MAC}.
- Output the ciphertext and the MAC tag.

As we discuss below, it is possible to derive both keys k_{Enc} and k_{MAC} from a single key k using a pseudorandom function.

The decryption routine first checks the MAC tag, then decrypts the ciphertext.

Using independent keys $(k_{\mathsf{Enc}}, k_{\mathsf{MAC}})$ is important in encrypt-then-MAC, as in many other cryptographic constructions. For example, a CPA-secure encryption scheme using n-bit keys can reveal the low order $n/2$ bits of its secret key in the ciphertext. And a secure MAC scheme using n-bit keys can reveal the high-order $n/2$ bits of its secret key in each MAC tag. Used independently, the encryption scheme and the MAC scheme are both secure. Used together with the same key k, the attacker learns all n bits of the key n and can break both primitives!

So, in general, you should always use independent keys for different primitives. To reduce the amount of keying material parties need to store, it is actually sufficient to store a single secret key k for a pseudorandom function F and derive all subsequent keys from the pseudorandom strings $F(k,0), F(k,1), \ldots$.

Theorem 1.2 (Informal)**.** *The Encrypt-then-MAC construction yields a CCA-secure encryption scheme, provided that: the underlying encryption scheme is CPA-secure and the underlying MAC scheme is secure (existentially unforgeable against adaptive chosen message attacks).*

Reminder: You should never need to implement authenticated-encryption schemes yourself. Instead use an off-the-shelf implementation that does the hard work for you. AES-GCM is one popular and widely implemented authenticated encryption scheme

Warning! ***Only*** *use encrypt-then-MAC* There are a number of bad ways to combine encryption and MACs to attempt to build authenticated-encryption schemes. MAC-then-encrypt is one way. Encrypt-and-MAC (i.e., MAC the message instead of the ciphertext) is another. Neither of these constructions is necessarily CCA-secure

when used with a CPA-secure encryption scheme and a secure MAC scheme. So the ***only*** flavor of authenticated encryption you should use is encrypt-then-MAC.

2 *AES-GCM (Galois Counter Mode)*

One of the most widely used authenticated-encryption constructions is AES-GCM. It follows the encrypt-then-MAC paradigm. It uses AES as a pseudorandom function for counter-mode encryption (**??**). It uses a Carter-Wegman-style MAC (**??**) as the MAC scheme.

There are a few optimizations that AES-GCM uses beyond what we have described:

- AES-GCM derives both the encryption and MAC keys from a single short key using a pseudorandom function.
- AES-GCM implements a fast form of the Carter-Wegman MAC that does not need arithmetic modulo a big prime p, as the scheme described in **??** does. Instead, of defining the MAC using $\mathbb{Z}_p$ (integers mod a 128-bit prime p), GCM works with 128-bit strings. The GCM mode of operation replaces addition modulo p with XOR of 128-bit strings and it replaces multiplication modulo p with a somewhat complicated operation on 128-bit strings. (Formally, the scheme works over the field $\mathbb{F}_{2^{128}}$ of order 2^{128}.) This gives a big performance boost with no loss in security.

If you are interested, to implement the multiplication operation: think of both 128-bit strings as polynomials with 128 coefficients in $\mathbb{Z}_2 = \{0, 1\}$. Multiply the polynomials, reducing the coefficients modulo 2. Then reduce the resulting polynomial modulo some fixed polynomial of degree-128. Then interpret the result as a 128-bit string.