CS 4501-6501 Topics in Cryptography 2 Mar 2018

Lecture 4, CCA Security

Lecturer: M. Mahmoody, *Scribe: Jiefu Liang, Jin Ding, Xinhao Chen*

1 Introduction

Before this class, we have already discussed security against two types of adversaries: a passive adversary that only eavesdrop, and a "partially active" adversary that carries out a chosen-plaintext attack. As we saw for CPA security, the encryption could be "resettable", which means the adversary could modify the ciphertext again and again. The reason this is a drawback of CPA security is that the adversary might get some feedback about the decryption every time he modifies the ciphertext. So we want to have a stronger security definition which makes any modification of an encryption c "useless" (for example, not decrypted). For that purpose, we are going to talk about another (stronger) type of attack, called a chosen-ciphertext attack (CCA) which is even more powerful. In a CCA security game, we provide the adversary not only with the ability to encrypt messages of its choice (as in a chosen-plaintext attack), but also with the ability to decrypt ciphertexts of its choice (except the challenge ciphertext c that is received from the challenger). Formally, we give the adversary access to a decryption oracle in addition to a encryption oracle. Finally, today we also introduce how to combine CPA secure encryption and MACs to achieve CCA security.

2 Review: MAC Authentication

2.1 What's MAC?

MACs (message authentication codes) enable the communicating parties to know whether or not a message was tampered with. The MAC's aim is to prevent an adversary from modifying a message sent by the other party who also holds the secret key (or at least noticing it when it happens). So, a secret key (that is kept secret from the attacker) comes to help.

We use some symbols to represent the message, key and how the MAC is used. Two users who wish to communicate in an authenticated manner begin by generating and sharing a secret key k in advance of their communication. When one party wants to send a message m to the other, she computes a tag t based on the message and the shared key, and sends the message m along with the tag t to the other party. The tag is computed using a tag-generation algorithm that can be denoted by MAC if this satisfied message authentication codes. The sender of a message m computes $t \leftarrow \mathsf{Mac}_k(m)$ and transmits (m, t) to the receiver. Upon receiving (m, t), the second party verifies whether t is a valid tag on the message m (with respect to the shared key) or not. This is done by running a verification

algorithm Vrf that takes as input the shared key as well as a message m and a tag t, and indicates whether the given tag is valid. The mathematical version of what we described is as follow:

Definition 2.1 (Recalling MAC)**.** A message authentication code (MAC) is a tuple of probabilistic polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrf})$ such that:

1. The key-generation algorithm Gen takes as input the security parameter 1^n and outputs a key k with length $|k| \geqslant n$.

2. The tag-generation algorithm of MAC takes as input a key k and a message $m \subseteq \{0,1\}^*$, and outputs a tag t. Since this algorithm may be randomized, we write this as $t \leftarrow \mathsf{Mac}_k(m)$. (In contrast to CPA security, randomness is not necessary here).

3. The verification algorithm Vrf takes as input a key k, a message m, and a tag t .It outputs a bit b, with $b = 1$ meaning valid and $b = 0$ meaning invalid. We assume without loss of generality that Vrf is deterministic, and so write this as $b := \mathsf{Vrf}_k(m, t)$.

It is required that for every n, every key k output by $\mathsf{Gen}(1^n)$ and every $m \leftarrow \{0,1\}^*$ it holds that $\mathsf{Vrf}_k(m, \mathsf{Mac}_k(m)) = 1$. If $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrf})$ is such that for every k output by $\mathsf{Gen}(1^n)$ algorithm Mac_k is only defined for messages $m \leftarrow \{0,1\}^{\ell(n)}$ (and Vrf_k outputs 0 for any $m \not\subseteq \{0,1\}^{\ell(n)}$ then we say that $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrf})$ is a fixed-length MAC for messages of length $\ell(n)$. As with private-key encryption, it is almost always the case that $\mathsf{Gen}(1^n)$ chooses $k \geqslant \{0,1\}^n$ uniformly at random.

2.1.1 The Security of MAC

We give the adversary access to a MAC oracle $\mathsf{Mac}_k(\cdot)$; the adversary can submit any message m to the oracle and is given in return a tag $t \leftarrow \mathsf{Mac}_k(m)$.

This can be considered as a "break" of the scheme if the adversary is able to output any message m along with a tag t such that:

(1) t is a valid tag on the message m (like $\mathsf{Vrf}_k(m, t) = 1$) and (2) the adversary had not previously requested a MAC tag on the message m. The first condition means that the adversarial algorithm could fool the party to think the message originated from the legitimate party. The second condition prevents the situation happened under a replay attack, if we augment the messages with a time stamp. We can also write the message authentication experiment $\mathrm{MACforge}_{\mathcal{A},\pi}(n)$.

The message authentication experiment $\mathrm{MACforge}_{\mathcal{A},\pi}(n)$:

1. A random key k is generated by running $\mathsf{Gen}(1^n)$.

2. The adversary A is given input 1^n and oracle access to $\mathsf{Mac}_k(\cdot)$. The adversary eventually outputs a pair (m, t). Let Q denotes the set of all queries that A asked to its oracle.

3. The output of the experiment is defined to be 1 if and only if (1) $\mathsf{Vrf}_k(m,t) = 1$ and (2) $m \notin Q$.

Therefore, a MAC algorithm is secure if no efficient adversary can succeed in the above experiment with non-negligible probability. Here we can get the mathematical definition of MAC- secure as follow:

Security definition of MAC : A message authentication code $\mathcal{A} = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Vrf})$ is existentially unforgeable under an adaptive chosen-message attack, or just secure, if for all probabilistic polynomial-time adversaries A, there exists a negligible function negl such that:

$$\Pr[\text{MACforge}_{\mathcal{A},\pi}(n)] \leq negl(n).$$

2.2 Constructing Secure MACs using PRFs

The former part is talking about the security of MAC. In this part, we'll focus on how to form a MAC with tool: Pseudorandom functions. The intuitive idea is that if the MAC tag t is obtained by applying a pseudorandom function to the message m, then forging a tag on a previously unauthenticated message requires the adversary to guess the value of pseudorandom function as a new point, which is the message itself. The probability of doing so is $2^{-\ell}$ if the function is truly random and has output length ℓ, and should remain $2^{-\ell} + negl(n)$ if we switch to PRF instead of a truly random function. See last lecture notes for formal description and proof of security, but the construction is simple to describe: the tag is simply $F_k(m)$ using a PRF that can accept arbitrary length messages.

One property of the above mentioned MAC is that it is deterministic. In particular, for every message m there is a unique tag t that can be accepted. This property is useful and we will rely on it in our construction of CCA secure encryption.

Definition 2.2 (Strong MAC)**.** This is defined similar to MAC, but for all (m, k), there is only one unique acceptable tag t that makes the verifier $\mathsf{Vrf}_k(m,t)$ output 1 (i.e., accept).

3 CCA Security

We first recall an example that explains why CPA security is not great for all scenarios.

3.1 Password verification example (Review and Expand)

Scenario:

1. Alice(client) wants to login on Bob's computer(server)
2. Alice's browser has a shared key k with Bob
3. Alice encrypts the password using k and sends $c = \mathsf{Enc}_k(password)$

4. Bob decrypts c and if the password is correct, it allows Alice to log-in

Although this is CPA security, there might have a attack that a resetter algorithm $\mathsf{Res}(c, i, b) \to c'$, c is encrypted by $m = m_1, ..., m_k$, i is the location of m which will be changed, b is the change bit. $\mathsf{Dec}(c') \to m_1, ...m_i, ..., m_k$, where $m_i = b$. The attacker don't need even know the output after producing the c'. But they can know which m be changed from the feedback of this password verification. So if given a 100 bits password, attacker can recover the password for at most 100 tries by using this function. Although it's CPA secure, it's not secure enough. So we want to design some stronger security scheme to cover all the vulnerabilities.

Do it yourself Can you design an actually CPA security which has the above vulnerability? Namely, assume that there is PRF, can you design a CPA secure encryption that has the above vulnerability?

3.2 Stronger security

Definition 3.1 (Tamper resilience – informal definition)**.** If a third party tries to modify ciphertext, the encryption algorithm with stronger security could always ends up with completely useless which means "not decryptable".

Recall the completeness condition: $\mathsf{Enc}(k, m, r) \to c$, $\mathsf{Dec}(k, c) \to m'$. And the correctness condition is that: $\forall k, \forall m, \forall r$: $\mathsf{Enc}(k, m, r) \to c$, $\mathsf{Dec}(k, c) \to m'$, $m = m'$. Note that this definition does not prevent us from rejecting messages that are not properly encrypted.. we will rely on this aspect in our search for a stronger security notion that prevents tampering with ciphertexts. In particular, we can use a specific symbol $\perp$, which is one error symbol. For example, when I want to decrypt something, and I notice it's not a right decryptable message, I might output this specific error symbol instead of any classic output m'.

In order to have a stronger security, we don't even have to change the above definition. Because this definition does not say what happens if we plug in another different ciphertext. If we plug in another ciphertext, this decryption algorithm can potentially output anything.

So we can use the same definition above, but we then focus on trying to make sure what we achieved would also detect whenever you plug in something which is not correctly generated, and give this specific error symbol $\perp$. In which case, if you try to tamper the ciphertext, you will get error nearly all the time. The reason why it's nearly all the time instead of all the time is that sometimes you might luckily obtain a correct ciphertext which you could only have very very low chance to obtain.

So we come up with a stronger security - CCA security which we will define below.

3.3 Define CCA Security

Definition 3.2 (CCA Security)**.** A private-key encryption scheme π has indistinguishable encryption under a chosen-ciphertext attack (or is CCA-secure), if for all probabilistic

polynomial-time adversaries A, there exists a negligible function negl such that: $\Pr[\text{PrivKey}^{cca}_{A,\pi}(n) = 1] \leq \frac{1}{2} + negl(n)$, where the probability is taken over all random coins used in the experiment.

Consider the following experiment for any private-key encryption scheme $\pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, adversary A and value n for the security parameter.

1. A key k is generated by running $\mathsf{Gen}(1^n)$
2. The adversary A gets the oracle access to $\mathsf{Enc}_k(\cdot)$ and $\mathsf{Dec}_k(\cdot)$. A is given the inputs 1^n, and it outputs two same length message m_0 and m_1.
3. Challenger choose an random bit b from $\{0, 1\}$, and use $\mathsf{Enc}_k(\cdot)$ to compute the challenge cipher-text c and send to Adversary.
4. Now the adversary still have the oracle access to $\mathsf{Enc}_k(\cdot)$ and $\mathsf{Dec}_k(\cdot)$, but with one restriction - it is not allowed to query the challenge cipher-text c.
5. Then the adversary output its guess b'. The adversary wins if $b = b'$, otherwise adversary lose.

This scheme is CCA secure if for every poly-time Adversary A, there is a negligible ε: $\mathbb{N} \mapsto [0, 1]$, and large enough n, $\Pr[A \text{ wins}] \leq \frac{1}{2} + \varepsilon(n)$.

Contrasting CCA with CPA security. The adversaries A for CCA security are given the access to oracle decryption which can decrypt everything but cannot decrypt the cipher-text encrypted by the challenger.

Do it yourself. Check why using CCA secure encryption algorithms in the context of "password encryption" example described above will resolve the issue. Namely, show that if an adversary tries to intercept messages and re-send modified versions of them, he will always get "rejected" and thus would not "learn" anything meaningful from the interaction. More formally, show that the probability of the adversary outputting the password at the end of such interaction is negligible if the passowrd is long enough (i.e., longer than the security parameter) and chosen at random.

4 Achieve CCA Security: MAC+CPA Security

In this section, we try to establish CCA security schema based on our previous achieved security schema. The adversary now has access to the message m_0 m_1, the encryption algorithm $\mathsf{Enc}(m, k)$, and the decryption algorithm $\mathsf{Dec}(c, k)$. In order to achieve CCA security, we have to establish a security schema which can guarantee $\Pr[Adv_{wins}] \leq \frac{1}{2} + \epsilon(n)$. These are the two tools we have for achieving CCA security:

- CPA-Secure encryption: (Enc, Dec) based on key k_1
- Strongly-Secure MAC: (Mac,Vrf) based on key k_2

There are three seemingly "natural" ways to achieve CCA security by combining MAC and CPA security. We will go over them to see which one works better!

0 Do authentication and encryption at the same time. (Wrong: Even not a CPA secure)

1 Do authentication, then encryption.(Wrong: remains CPA secure, but not CCA secure)

2 Do encryption, then authentication(Right: CCA security is achieved).

4.1 0^{th} try (Wrong)

Claim: Doing the authentication and encryption at the same time is not CCA security, even not CPA security.

Proof: Assuming that we do the authentication and encryption together at the same time, encryption algorithm $\mathsf{Enc}_{K_1}(M)$ is a CPA secure algorithm, t_m is a MAC tag unique for a message m. After encrypting the message m, the adversary will be provided with a ciphertext $c = [\mathsf{Enc}_{k_1}(m), t_m]$. The reason why this security schema is not CCA even not a CPA secure schema is that the adversary can easily distinguish which message (m_0 or m_1) the ciphertext c is generated from, by encrypting m_0 and m_1 and comparing the their MAC tags t_{m_0}, t_{m_1} with t_{m_c}. Due to the fact that the MAC tag is unique for every message, the adversary can easily know which message has been encrypted as long as they compare their MAC tags.

4.2 1^{st} try (Still Wrong)

Claim: Doing the authentication then encryption is CPA security but not CCA security, because the encryption scheme π is resettable.

Do it by yourself: Figure out how to prove the encryption scheme π is CPA security.

Proof: Next we will prove the encryption scheme π is not CCA secure *in general*. In other words, we will show that some CPA secure encryption and secure MAC, when used this way, will be not CCA secure. In particular, we will use a CPA secure encryption scheme that is possible to be resettable. In the encryption scheme, the message m firstly is authenticated by MAC with k_1, generating a MAC tag t_m. Then a combined message $[m + t]$ is encrypted by a CPA encryption scheme, generating a ciphertext $c = \mathsf{Enc}([m + t_m])$. In order to prove the encryption scheme is not CCA security, we can provide an attack to break the encryption within polynomial time.

Here we want to prove the encryption scheme is resettable, meaning the adversary can break the encryption scheme by modifying ciphertext and decrypting the modified ciphertext. Suppose the adversary can modify the first bit of the message m in the ciphertext $\mathsf{Enc}([m, t_m])$ and get a new ciphertext $\mathsf{Enc}([m', t_m])$. Then the adversary can try to decrypt the new ciphertext. Based on the decryption results (rejection or not), the adversary can know which

message was encrypted then break the encryption scheme. Suppose the adversary is provided with message $m = \left\{ \begin{array}{ll} m_0 = & 000 \\ m_1 = & 111 \end{array} \right\}$. Of course the adversary does not know which message is encrypted. For the ciphertext $\mathsf{Enc}([m, t_m])$, suppose the adversary resets the first bit of the message m to 0 then gets new ciphertext $\mathsf{Enc}([m', t_m])$, of which $m' = \left\{ \begin{array}{ll} m'_0 = & 000 \\ m'_1 = & 011 \end{array} \right\}$. Note here we do not modify the MAC tag t_m, the t_m is still generated from one of previous message m_0 and m_1. Therefore, the adversary gets the new ciphertext $c' = \mathsf{Enc}([m', t_m])$, m' is one of the two 000 and 011. Then the adversary can use the decryption scheme to decrypt c' and get the decryption results. If the adversary gets a rejected result, it means 111 was encrypted in this challenge, otherwise 000 was encrypted.

4.3 2^{nd} try (Right)

Claim: Doing the encryption then authentication is CCA security.

Big picture of proof: In order to prove a private-key encryption scheme is CCA security, the intuition is to prove the decryption $\mathsf{Dec}()$ provided for the adversary is useless, then back to CPA security. Assuming that we construct a private-key encryption scheme π which is a CCA secure. The aim is to prove the encryption scheme is CCA security. Here we have two assumptions:

1. a private-key $(\mathsf{Enc}, \mathsf{Dec})$ scheme as π_1 which is CPA security.
2. a MAC authentication algorithm $(\mathsf{Mac}, \mathsf{Vrf})$ as π_2 which has *stronger* security, meaning that for every message the MAC tag is unique.

The big picture of the proof as usual is to show that if an adversary A can break the CCA security of the new encryption scheme with polynomial time, then using A as a subroutine, one can break either the above two assumptions. In other words, if the two assumptions are correct, the private-key encryption scheme is CCA secure.

Suppose A distinguishes the encryption of m_0 from m_0 with advantage ε. We will break the situation into two cases: Either, the following happens with probability $\geq \varepsilon/2$ or at mot $\varepsilon/2$: Event E: during its attack A asks its decryption oracle a query c' where c' is not the challenge ciphertext and is not obtained by A from its encryption oracle and that c' is decrypted properly without returning the error $\perp$.

- **Breaking MAC:** If $\Pr[E] \geq \varepsilon/2$, then note that $\varepsilon/2$ is also non-negligible (just like ε) and so we can get an attacker B_1 who can break the security of MAC scheme π_2 with $\approx \varepsilon/2$ probability. The way we can do this is as follows. B_1 will simulate a copy of the CPA secure encryption π_1 in its head. Then it runs A and tries to provide what A needs. It will use π_1 and its Mac oracle to implement the (supposedly) CCA secure encryption scheme. The first moment that A makes the event E happen, it means that it has found a ciphertext $c' = [C, t]$ where t is a correct tag for C and that A has not obtained this c' from B_1.

- Breaking CPA encryption: If $\Pr[E] \leq \varepsilon$, it means that with probability at least ε, A succeeds without asking any query like c' described above. It means that with probability $\varepsilon/2$ all the queries that A asks from the decryption oracle, is something that it knows the answer to them already! So if we remove the decryption oracle from A, it can still win in the CPA security game with probability at least $\varepsilon/2$.

Do it by yourself: Foralize the above sketch of the proof.