CS 4501-6501 Topics in Cryptography 23 Feb 2018

Lecture 6, Authentication

Lecturer: Mahmoody, Scribe: Paul Sanders, Jack Herd, Mainuddin Jonas, Jack Schumann

1 Reviewing aspects of CPA security

Extending CPA secure encryption to long messages. Last time, we covered Pseudo Random Functions (PRFs) and how we can use them easily to obtain CPA secure encryption. It is important to note (as we show in problem set 2) that CPA secure encryption automatically extends to longer messages. For example, if we wanted to encrypt a long message $m = m_1, m_2, \dots m_\ell$ we could do so as follows:

$$\text{Enc}(key, m; r) = [\text{Enc}(key, m_1, r_1) || \dots || \text{Enc}(key, m_\ell, r_\ell)]$$

Note that in this scheme, r represent a stream of randomness that can be broken into $r = r_1, r_2, \dots r_\ell$; however, as we saw, if the length of the randomness $|r_i|$ is short, the encryption scheme even for small blocks becomes not secure (as defined by CPA security) simply because an adversary can cycle through all possible randomness combinations (so any CPA secure encryption needs to use sufficiently long randomness, in particular of length $\log(n)^{\omega(1)}$ for n being the security parameter).

Contrasting with weaker security notion of single message security. Single-message security does *not* necessarily scale if we use weaker notions of security than CPA security, meaning that trying to extend a key to work on a long message $m = m_1, m_2, \dots$ as follows

$$\text{Enc}(k, [m_1, m_2, \dots]) = (\text{Enc}(k, m_1), \text{Enc}(k, m_2), \dots)$$

will not be single-message (semantic or indistinguishable) -secure even if we assume that the scheme *is* secure for blocks of length ℓ (i.e., $m_i \in \{0,1\}^\ell$). We saw how to break such extended schemes if no randomness is used. (Note that randomness is *not* necessary for single message security, but here we see a different aspect of what is wrong with weak security notions: an example that direct extension to long messages fails).

Revealing or not revealing randomness of encryption Encryption's own randomness is usually **not** revealed despite our revealing of randomness in the encryption scheme of Lecture 05 where the decryption algorithm was passed randomness as part of the plaintext. That scheme functioned as follows:

If we have a PRF $F_k(x) \to y$ $|y| = \ell$, $x \in \{0,1\}^*$, and k is a secret key for the PRF:
To encrypt m of length $|m| = \ell$ bits pick r of length n output $c = [r, m \oplus F_k(r)]$.
To decrypt $\text{Dec}(k, c)$ where $c = [r, y]$ output $m = F_k(r) \oplus y$

What CPA security does or does not guarantee

- It guarantees multi-message security against a **passive attacker** who only observes messages that are encrypted.
- It even guarantees security against a semi-active attacker who as access to an oracle that holds the same key as the encryption access.
- But, it does **not** guarantee security against truly **active attackers**.

As a result, CPA security is vulnerable to a variety of attacks, including:

1. The re-sending of saved messages (these are called replay attacks and can *possibly* be fixed with a time-stamp, if done properly).
2. Forwarding a modified ciphertext, and pretending to be sent from the honest party with the hope of getting some "feedback" from the decryption process.
3. Changing the ciphertext in order to have it decrypted to something else: e.g., set the last bit of the plaintext to 0 or 1 with certainty (this is a big problem for server/password verification)

An interesting observation is that in (3) the adversary is not making any progress in terms of understanding the message, but they are able to cause unintended results. Thus, very informally speaking, (3) is an attack on completeness rather than security. Yet, the question stands: how do we define a security definition that holds against an active attacker? Also note that (2) and (3) (and even (3)) are all related as they all involve adversary sending something on behalf of Alice while it did not really come from Alice.

2 Authentication

How does Bob know Alice, and not Eve, sent the message? We note that authentication could be meaningful even without asking any notion of privacy; so, for now we can just think about authenticating some message (whether it is plaintext or ciphertext) and it would be a worthwhile goal. Also note that, when it comes to only guaranteeing authentication, without loss of generality we can assume that the message m that we want to authenticated is part of the communicated message. The extra information attached to m for the purpose of authentication is usually called **Message Authentication Code (MAC)**. Additionally, we will later discuss a "public-key" version of authentication known as "Digital Signatures". If we could combine an authentication protocol and **properly** combine this with a CPA-secure protocol, we would have a more secure encryption that handles "active" attackers.

3 Message Authentication Code

Let us first informally introduce the concept of MAC before defining it formally. Informally, MAC can be described as follows:

- Alice and Bob share a key k.
- Alice generates $\mathsf{Mac}_k(m) \to tag_m$ and sends: $[m, tag_m]$.
- Bob receives $[m, tag_m]$ and runs $\mathsf{Vrf}_k(m, tag_m)$. Vrf_k outputs 1 if the message ought to be accepted, and it outputs 0 if the message ought to be rejected.

Completeness condition: The MAC is *complete* if for all m, k it holds that

$$\mathsf{Vrf}_k(m, \mathsf{Mac}_k(m)] = 1$$

Security condition (informal): The MAC is secure if it is computationally unfeasible for an adversary to generate a valid $[m, tag_m]$ pair, even if the adversary gets to see $poly(n)$ (n being the security parameter) many valid $[m, tag_m]$ pairs before forging a tag for a **new** m. Before we can define MAC security more formally, we first formally define the experiment.

Message authentication security game (experiment): The message authentication experiment $\mathrm{MacForge}_{\mathcal{A},\Pi}(n)$ dealing with adversary $\mathcal{A}$ is formally defined as follows:

1. A key k is generated by running $\mathrm{Gen}(1^n)$.
2. The adversary $\mathcal{A}$ is given input 1^n and oracle access to $\mathsf{Mac}_k(\cdot)$. The adversary outputs (m, t). Let $\mathcal{Q}$ denote the set of all queries that $\mathcal{A}$ asked to its oracle.
3. $\mathcal{A}$ *succeeds* if and only if (1) $\mathsf{Vrf}_k(m, t) = 1$ and (2) $m \notin \mathcal{Q}$. In that case the output of the experiment is defined to be 1.

Definition 3.1 (Formal definition of MAC security)**.** A *message authentication code* $\Pi = (\mathrm{Gen}, \mathsf{Mac}, \mathsf{Vrf})$ is existentially unforgeable under an adaptive chosen-message attack, or just simply secure, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there is a negligible function *negl* such that:

$$\Pr[\mathrm{MacForge}_{\mathcal{A},\Pi}(n) = 1] \leq negl(n).$$

Constructing MACs using Pseudorandom functions (PRFs): One obvious choice for generating MAC tags is to use PRFs. Let $F_k(\cdot)$ be a PRF with key, input and output lengths $n, *, l$. Now we can define a MAC scheme as follows:

$$\mathsf{Mac}_k(m) = F_k(m)$$
$$\mathsf{Vrf}_k(m, tag_m) = 1 \text{ if } F_k(m) = tag_m,\ 0 \text{ otherwise.}$$

Next, we will prove that, if the length of the output of the PRF, $\ell = n$, where n is the security parameter, then, the MAC scheme above is secure according to the previous definition of security.

3.1 Proof of Security

Let us define a scheme $(\mathsf{Mac}, \mathsf{Vrf})$ as follows:

$$\mathsf{Mac}_k(m) = F_k(m)$$
$$\mathsf{Vrf}_k(m, tag_m) = 1 \text{ if } F_k(m) = tag_m, \text{ 0 otherwise.}$$

Claim 3.2. *Given that F_k is a keyed pseudo-random function of key length n and output length $\ell = n$, the above scheme is secure when n is the security parameter.*

Proof Outline: We will prove this using a proof for the contrapositive statement; that is, we will show that if our scheme is not secure, than the PRF is not secure. We will achieve this by constructing an attacker on the PRF using the attacker on the MAC scheme.

Proof. Let $(\mathsf{Mac}, \mathsf{Vrf})$ be the scheme defined above, with a key $k \leftarrow \{0,1\}^n$.

First we must prove completeness. This is easy to show. For any key k and message m, $\mathsf{Mac}_k(m) = F_k(m)$. So, $\mathsf{Vrf}_k(m, \mathsf{Mac}_k(m)) = \mathsf{Vrf}(m, F_k(m))$. Trivially, $F_k(m) = F_k(m)$, so Bob will always correctly verify messages sent from Alice.

Then, we must prove the security of this system. Consider two worlds, Real and Ideal. In both worlds, the adversary A outputs (m, t) and wins if $\mathsf{Mac}_k(m) = t$.

Real World	Ideal World
$\mathsf{Mac}_k(m) = F_k(m)$	$\mathsf{Mac}_k(m) = R(m)$
$P_R = \Pr[A(\mathsf{Mac}_k(\cdot)) = 1] = \Pr[t = \mathsf{Mac}_k(m)]$	$P_I = \Pr[A(R(\cdot)) = 1] = \Pr[t = R(m)]$

In the Ideal world, the function $R(\cdot)$ is a truly random function with ℓ bits of output. Note that it is still a *function*, so asking the same query twice will lead to the same output. In other words, for each new input, it chooses a random output, and each time that input is given, it gives the same output.

Step 1: Showing that P_I is small. As for all messages m, $R(m)$ gives a uniformly random output of length n, we see that the probability of $R(m)$ being any given tag is $\frac{1}{2^n}$. So, no matter what algorithm A uses, $P_I = \Pr[t = R(m)] = \frac{1}{2^n}$. The crucial point here is that m should *not* be equal to any of the previously asked queries, as otherwise the adversary will lose anyway.

Step 2: Showing that $|P_R - P_I|$ must be large. Here, we use our earlier assumption that our scheme was not secure. So, we have that P_R is non-negligible, or $P_R \geq \frac{1}{\text{poly}(n)}$. Then note that P_I *is* negligible, so the difference between the two must be non-negligible.

Step 3: Showing that F_k is not a secure PRF. We will use that $|P_R - P_I| \geq 1/\operatorname{poly}(n)$ to build a poly-time attacker on F_k that we call B. Recall that in the PRF game, B also has two worlds with two different oracles available – one where the oracle is a PRF and one where it is a truly random function. First, B will run A internally and try to answer its oracle queries, and at the end A will output a pair (m, t). B can simply forward all of A's queries to its oracle and get the answer and then forward the answer to A. Note that A is good at outputting a pair m such that $O(m) = t$ is more likely to happen if O is the random function compared to the PRF case (because $|P_R - P_I| \geq 1/\operatorname{poly}(n)$). So B will try to rely on this point to win the security game of the PRF¿ Then, B will query the oracle that it has O itself with input m. If $O(m) = t$, then B will output that it is in the PRF world (e.g., by outputting 0). Otherwise, it will output that it is in the random world (e.g., by outputting 1). The probability that it outputs that it is in the PRF world, given it is in the PRF world, is equal to $Pr[t = F_k(m)] = P_R$. The probability that it outputs that it is in the PRF world, given it is in the random world, is $Pr[t = R(m)] = P_I$. But, we know that $|P_R - P_I|$ non-negligible. So, B is indeed breaking the security game of PRF with non-negligible probability, and so F_k is not a PRF!

So, the contraceptive of what we proved above is that, if F_k is a PRF, then the scheme described above must be secure as well!

$\square$

4 Next Week: CCA Security

Next week, we will continue our discussion of MACs by demonstrating how to achieve security against **C**hosen **C**iphertext **A**ttacks. Our goal is to come up with an encryption scheme that is both private and preserves integrity. In order to achieve this stricter security standard, we need construct a method to *safely* combine CPA security with MAC to handle both passive and active attacks.

4.1 Password Verification Example

Consider the following example and corresponding figure that highlights our need for provably safe CCA security. Namely, we describe a setting where CPA security guarantees nothing meaningful, if the adversary can get come feedback about decryption of certain tampered ciphertexts.

Set-Up: Suppose we have two parties, Alice and Bob, who both share the same secret key k. In our scenario we will think of Bob as a server that Alice only interacts with through her browser, so really Alice's browser and the Server share k.

Scenario: Alice begins by typing her password into the browser and submitting. The browser then sends $c = \text{Enc}_k(pass)$ to the server Bob, who uses his shared knowledge of k to decrypt the password and verifies a successful login attempt by responding "ok". Let us

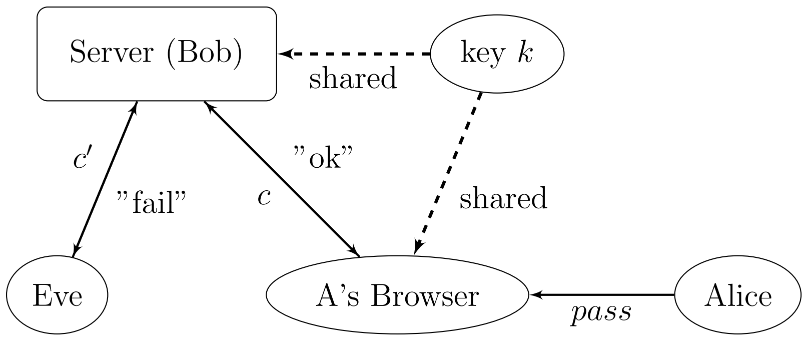

Figure 1: Password Verification Ex

assume the encryption is CPA secure, so that the encryption of any password is indistinguishable from any other password.

Question: What would happen if an active adversary Eve were to tamper with the ciphertext c and instead send some c' where $c' \neq c$?

Answer: One of the following events would occur:

- Eve succeeds in gaining access to the server (unlikely)
- Eve fails in her attempt (most likely)

Now, suppose the server responds to Eve with some message like "login failed". Then, under certain circumstances if Eve is allowed to continue to send fake ciphertexts many times it may be possible for her to recover the password entirely!

The Attack: To see how this attack could potentially occur, first lets just suppose we are dealing with a CPA encryption scheme that has a "last-bit vulnerability". All this means is that on certain occasions I can tamper with the last bit of the ciphertext c such that I can be ensured that the last bit of c is now set to zero. Call this last bit zeroed ciphertext c'. Note that the way we have described this process, we are not violating the fact that the encryption scheme is CPA secure because we remain oblivious as to whether the last bit of c was zero or one before we chose to set it to zero. We have also not stipulated that this should always be possible with every ciphertext, as we require this to happen only sometimes. But the bottom line is that there *are* CPA secure encryption algorithms that *do* have this "feature" (here of the form of a huge vulnerability). Now, if we send this c' to the server, one of two things will occur based on whether or not the bit I reset was originally zero to begin with:

- If bit was 0, then the server will respond "ok" and login is successful
- if bit was 1, then the server will respond "fail" but now we have the knowledge that the bit was 1.

In either case, we can use the feedback that the server provides to deduce what the bit was originally. Now if we are allowed to repeat this process many times, there is nothing stopping us for repeating this process on each bit, allowing us to recover the password entirely.

The Bigger Picture: The reason why this attack happens is because we currently do not have a way to maintain the integrity of a message, which is ultimately an authentication problem. Ideally, we want all the modified variants $c' \neq c$ be be "undecryptable" and return no meaningful information. This example provides a strong motivation for finding a way to fuse CPA security with message integrity.