CS 7880 Graduate Cryptography October 15, 2015

Lecture 10: CPA Encryption, MACs, Hash Functions

Lecturer: Daniel Wichs *Scribe: Matthew Dippel*

1 Topic Covered

- Chosen plaintext attack model of security
- MACs in a computational setting
- Definition of collision resistant hash functions

2 Recap of last lecture - PRGs for one time pads

Last lecture, we saw how a PRG could be used to use keys to encrypt messages of a larger size with a one-time-pad. Given a key k and a message m where $|k| < |m|$, we use a PRG G with at least $|m| - |k|$ bit stretch to generate a one time pad $G(k)$. Then, we do bit-wise XORing with m to achieve our ciphertext. Formally, our encryption and decryption functions are:

$$\mathsf{Enc}(k, m) = G(k) \oplus m$$

$$\mathsf{Dec}(k, c) = G(k) \oplus c$$

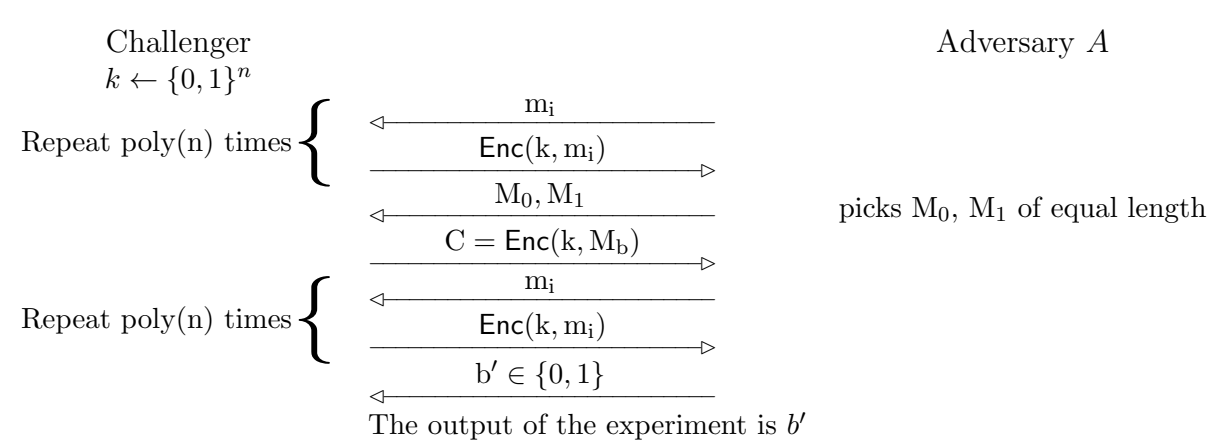
Disregarding other properties, the above still has the problem of not working for more than one message. Our goal for the next section will be to define a model of security that provides multi-message encryption security, and to provide a scheme which achieves it.

3 Chosen Plaintext Attack (CPA)

We will now consider a new model of security in which an adversary has blackbox access to our encryption function with the same key we are using. The adversary can make a polynomial number of queries with any plaintext message they want, and will receive the corresponding ciphertext. The adversary also gets to provide two messages m_0 and m_1. One of them (unknown to the adversary) is chosen, and an encryption of it is returned. The adversary must then determine which message, m_0 or m_1, was encrypted and returned to it.

3.1 CPA Definition

In order to analyze such a scenario, we will consider two experiments, **CPA-Exp-0** and **CPA-Exp-1**. In both experiments the adversary can query a *encryption oracle* by choosing arbitrary messages m_i and getting back ciphertexts $c_i \leftarrow \mathsf{Enc}(k, m_i)$. The adversary can query the oracle as many times as it wants. At some point the adversary chooses two challenge messages M_0, M_1 and gets back an encryption of M_b where $b = 0$ in experiment 0 and $b = 1$ in experiment 1, It will then be the job of the adversary to correctly conclude which experiment he is in. For a bit $b \in 0, 1$, an adversary A and a security parameter n, we define $\textbf{CPA-Exp}^b_A(n)$ as the following procedure:



DEFINITION 1 A symmetric-key encryption scheme $(\mathsf{Enc}, \mathsf{Dec})$ is CPA secure if for any PPT adversary A we have

$$\left| \Pr[\text{CPA-Exp}^b_A(n) = 1] - \Pr[\text{CPA-Exp}^b_A(n) = 1] \right| = \mathsf{negl}(n).$$

♢

Experiment Indistinguishability. We already defined the computational indisitinguishability of two distribution (ensambles) $X \approx Y$. Let's extend this definition to interactive experiments such as the ones defined above. If $Exp_A(n)$ and $Exp'_A(n)$ are two experiments parametrized by an adversary A and a security parameter n, we write $Exp \approx Exp'$ as a shorthand notation to denote that for all PPT adversary A

$$\left| \Pr[Exp_A(n) = 1] - \Pr[Exp'_A(n) = 1] \right| = \mathsf{negl}(n).$$

With this notation, the definition of CPA security for encryption can be written simply as: CPA-Exp$^0 \approx$ CPA-Exp^1.

3.2 A randomized encryption scheme with CPA security

First, it is worth noting that no deterministic encryption protocol can satisfy this definition. This is because the adversary is not limited in what it is allowed to query with its oracle

access to $O(m) = \mathsf{Enc}(k, m)$. Thus, an adversary could first query $O(M_0)$ and $O(M_1)$ before passing these messages to the experiment as its choices of M_0 and M_1. Whichever experiment it is in, it will either be returned $O(M_0)$ or $O(M_1)$ exactly as the oracle originally returned to him. Thus with probability 1 it can distinguish between the experiments it is in.

To get around this, we create a randomized encryption scheme, in which, for fixed k and m, $\mathsf{Enc}(k, m)$ is a random variable, satisfying $\Pr[\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m] = 1$. Let $F_k : \{0,1\}^n \to \{0,1\}^{\ell(n)}$ be a PRF. Then we can define Enc and Dec as follows:

$$\begin{array}{ll} \mathsf{Enc}(k, m): & x \leftarrow \{0,1\}^n \\ & c = (x, F_k(x) \oplus m) \\ \mathsf{Dec}(k, c): & c = (x, y) \\ & m = y \oplus F_k(x) \end{array}$$

Where in the above schemes, Enc samples a uniformly random x and returns c, while Dec unpacks c as described and returns m.

The intuition for the above schemes is that, reusing one time pads is insecure, but access to a PRF gives us an exponential supply of pads we can use. If we wish to decrypt the message, then knowing k will let us use F_k to reconstruct the pad, but to an adversary who can only see x, the pad is indistinguishable from being chosen completely at random, providing encryption similar to a randomly chosen one time pad.

3.3 Proof of Security

To prove the CPA security of our above scheme, we will proceed by a hybrid argument. First, we will define the hybrids. Then, we will prove that each one is indistinguishable from the next. Each hybrid will essentially be a variation of one of the experiments, where we modify how the encryption procedure works.

$$\begin{array}{lll} H_0: & \text{CPA-Exp-0}: & \mathsf{Enc}(k, m) = (x, F_k(x) \oplus m) \\ H_1: & \text{CPA-Exp-0}: & \mathsf{Enc}(k, m) = (x, R(x) \oplus m),\ R(x) \text{ is a true random function} \\ H_2: & \text{CPA-Exp-0}: & \mathsf{Enc}(k, m) = (x, y),\ y \leftarrow \{0,1\}^{\ell(n)} \\ H_2': & \text{CPA-Exp-1}: & \mathsf{Enc}(k, m) = (x, y),\ y \leftarrow \{0,1\}^{\ell(n)} \\ H_3: & \text{CPA-Exp-1}: & \mathsf{Enc}(k, m) = (x, R(x) \oplus m),\ R(x) \text{ is a true random function} \\ H_4: & \text{CPA-Exp-1}: & \mathsf{Enc}(k, m) = (x, F_k(x) \oplus m) \end{array}$$

Note that we labeled H_2, H_2' in this manner because they are trivially identical experiments, so the hybrid step is a one liner. In particular, H_0 is exactly CPA-Exp-0, and H_4 is exactly CPA-Exp-1.

Lemma 1 $H_0 \approx H_1$

Proof: Suppose A could distinguish between H_0 and H_1. Since they are both running in CPA-Exp-0, this is the same as distinguishing between $(x, F_k(x))$, and $(x, R(x))$. Formally, we could have B_A distinguish between $F_k(X)$ and $R(x)$ by using A as a black box, where whenever A would access the encryption oracle O, B_A will proxy for the result by passing

the input to its unknown function, and giving the output back to A as the result. Since we should have $F_k(x) \approx R(x)$, then $H_0 \approx H_1$. □

Lemma 2 $H_1 \approx H_2$

Proof: First, note that the distribution for $R(x)$ and $R(x)\mathsf{XOR}m$ are the same, since $R(x)$ is truly random. The difference between H_1 and H_2 is the difference between $R(x)\mathsf{XOR}m \approx R(x)$ and y. The only way we could tell the difference between $R(x)$ and y is if we chose the same x at least twice. In this case, $R(x)$ would be consistent, but y will, with probability nearly 1, be a different value. However, x is chosen randomly. If we make q queries to the oracle, then by a union bound, the probability that we choose the same x more than once is $\Pr[\exists i, j : x_i = x_j] \leq q^2/2^n \in negl(n)$. Thus we can only differentiate H_1 and H_2 with negligible probability. □

Lemma 3 $H_2 \approx H_2'$

Proof: The encryption oracle ignores the inputs, so there is no correlation between what the oracle gives as an answer and whether we are in CPA-Exp-0 or CPA-Exp-1. The experiments are identical. □

Lemma 4 $H_2' \approx H_3$

Proof: The same argument that $H_1 \approx H_2$ applies. □

Lemma 5 $H_3 \approx H_4$

Proof: The same argument that $H_0 \approx H_1$ applies. □

Thus by a hybrid argument, we have that $H_0 =$ CPA-Exp-0 $\approx$ CPA-Exp-1 $= H_4$, as desired.

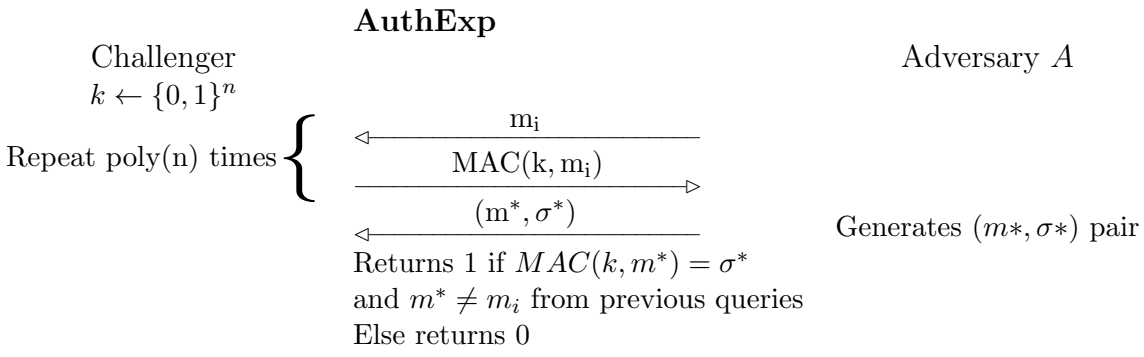
Due to the randomization, it is not possible for a computationally bounded adversary to tell which M_0, M_1 was encrypted and returned. The only real attack is to keep querying on M_0 and M_1 repeatedly, and hope to get the same random parameter x that the experiment used for its encryption of M_b.

4 MACs in a computational setting

Previously, we defined 1-time MACs which had information theoretic security, even in the face of unbounded computational power, but could only be used to authenticate a single message. We also saw that we can extend this construction to any a-priori bounded number of messages t, but the size of the key had to grow with t. In this section, we will update our definition by viewing our adversary as being computationally bounded. This new definition will allow us to securely generate tags for arbitrarily many messages with a short key.

4.1 Computational secure MAC definition

As before, the key k is chosen uniformly at random. However, this time, the adversary A has oracle access to $MAC(k, m)$. After making a polynomial number of queries, A can pick any message $m*$, and must attempt to generate a correct tag $\sigma*$. If $MAC(k, m*) = \sigma*$ and $m* \neq m_i$ for any of the previously queried m_i, then the result of the experiment is 1. Else, it is 0. We define this experiment as AuthExp_A. The flow diagram for this is below:



DEFINITION 2 A MAC is computationally secure if for all PPT adversaries A, we have that: $\Pr[\text{AuthExp}_A(n) = 1] \leq \text{negl}(n)$. ♢

4.2 MACs with AuthExp security

Given our above definition, it is true that any PRF $F_k(x)$ will suffice as a MAC. We formalize this and prove it below.

Theorem 1 *Let $F_k(m)$ be any PRF. Define a MAC function as $MAC(k, m) = F_k(m)$. Then $MAC(k, m)$ is computationally secure.*

Proof: First, we will show that AuthExp is indistinguishable from an experiment where in place of F_k we use a truly random function. Then the desired conclusion follows from the security of the modified experiment.

Define the following two hybrids:

H_0 : AuthExp

H_1 : AuthExp, replace F_k with $R(m)$ // R is a truly random function

Lemma 6 *$H_0 \approx H_1$. In other words, for all PPT A, $|\Pr[H_0(A, n) = 1] - \Pr[H_1(A, n) = 1]| \leq negl(n)$.*

Proof: If we could distinguish H_0 and H_1, then we could distinguish any PRF from a random function by plugging them into H_0 and H_1 and distinguishing those experiments. □

Lemma 7 *For all adversaries A:* $\Pr[H_1(A, n) = 1] \leq 2^{-n}$

Proof: Since the MAC function is a truly random function, no matter what queries A makes in experimet H_1, if m^* was not queried then the final tag $MAC_k(m^*) = R(m^*)$ will be random and unknown to the adversary. Thus no matter what tag σ^* the adversary chooses, the probability of it being correct $\sigma^* = R(m^*)$ is at most 2^{-n}. □

Combining the two lemmas we get that for all PPT A:

$$\begin{aligned}
\Pr\left[\text{AuthExp}_A(n) = 1\right] &= \Pr\left[H_0(A, n) = 1\right] \\
&\leq \Pr\left[H_1(A, n) = 1\right] + \mathsf{negl}(n) && \text{By Lemma 6} \\
&\leq 2^{-n} + \mathsf{negl}(n) \leq \mathsf{negl}(n) && \text{By Lemma 7}
\end{aligned}$$

as we wanted to show. □

5 Combining encryption and authenticity

In previous lectures, we had viewed encryption and authenticity as separate problems. In reality, these two processes are combined to create encryption schemes where messages can also be verified. Not only that, but they are more related than they appear.

Consider the following simple protocol between parties A and B: A has a message (b, m) to encrypt for B. If m is a top secret message, she will set $b = 1$, indicating to B that it is not to be shared. If m can be freely shared after B is done with it, she will set $b = 0$. A uses proper encryption to send $\mathsf{Enc}(k, (b, m))$ to B, who then uses his decryption function to obtain the original (b, m) and act according to the value of b.

Suppose an actively malicious party E could intercept ciphertexts, and although cannot read them, could flip any bit of the plaintext message by appropriately tampering with the ciphertext. Then E could flip the first bit of the plaintext, changing the behavior of B. If this was a top secret message, B would mistakingly publish it for all to see. Although this example may seem forced, in practice there have been attacks on cryptography implementations which work in a way very similar to the above.

The key problem in the above example is that B will act differently based on what he reads from the plaintext message. B could also be viewed as a system, which decrypts incoming ciphertexts and acts according to the plaintext. For example, imagine a bank server which receives encrypted transactions, and must make transfers according to the plaintext descriptions. In order for correctness of this system to be guaranteed, the cryptography schemes must provide their own guarantees as to what can and cannot be decrypted into a plaintext message. This is the role of the MACs in the cryptography schemes.

The question to answer is, in what way should encryption and authenticate be composed, so that neither is compromised? To motivate this question, consider the following *incorrect* combination:

- $c = \mathsf{Enc}(k, m)$
- $\sigma \leftarrow \text{MAC}((, k), m)$
- Send (σ, c)

In this case, we are signing the plaintext, not the ciphertext. B can only authenticate after decrypting c. There are two potential problems with this. First, we have no guarantees that σ doesn't reveal any information about the message, which we want to hide. Second, it is still possible that just by decrypting c, information about the message or the secret key can be revealed. This is mostly due to $\mathsf{Dec}(k, c)$ having undefined behavior if c is not in its domain, which can occur if it was tampered with by an active eavesdropper.

What we really want is to make sure that bad cipher texts do not get to be passed to the decryption function. Thus, we do the following:

- $c = \mathsf{Enc}(k, m)$
- $\sigma \leftarrow \text{MAC}((, k), c)$
- Send (σ, c)

When B receives (σ, c), he first checks that $MAC(k, c) = \sigma$, and only attempts decryption if this is the case. This prevents both the MAC from leaking information about m, and accidental decryption of faulty ciphertexts. To reiterate, the main point is to not decrypt bad cipher texts.

6 Collision Resistant Hash Functions

A collision resistant hash function is used to compress a long message x to a short digest $y = H(x)$. Since H is compressing, there are neceserally collisions $x \neq x'$ such that $H(x) = H(x')$. However, we require that such collisions are hard to find.

More formally, we consider a family of functions $H_s : \{0,1\}^{\ell(n)} \to \{0,1\}^n$, where $\ell(n) > n$ and $s \in \{0,1\}^n$. We think of s as a public seed of the hash function which is chosen randomly but known to everyone. Then we say H_s is a collision resistant family of hash functions if:

1. $H_s(x)$ can be computed in polynomial (in $|s|, |x|$) time.
2. For all PPT A, the following experiment has negligible probability of returning 1:
   - $s \leftarrow \{0,1\}^n$
   - A gets s.
   - A outputs $x, x' \in \{0,1\}^{\ell(n)}$.
   - The experiment returns 1 iff $x \neq x'$ and $H_s(x) = H_s(x')$, 0 otherwise.

   More compactly, we can write that for all PPT A we have:

$$\Pr[(H_s(x) = H_s(x')) \wedge (x \neq x') \ : \ s \leftarrow \{0,1\}^n, (x, x') \leftarrow A(s)] = \mathsf{negl}(n).$$

Discussion on seeds. One might ask whether the seed s is needed at all - couldn't we have just a single hash function H for which it's hard to find collisions? Intuitively this seems possible and is actually done in practice. However, for a technical reason, we need the formal definition to have a random seed. The reason is that, if we had a single fixed hash function H then there would always be a very simple (non-uniform) adversary A that has a hard-coded collision $x \neq x'$ such that $H(x) = H(x')$ and simply outputs x, x' without doing any computation. By choosing a random seed s, we prevent this since the adversary would need to know a different collision for each s (or at least many seeds s) which would require an exponential amount of data. In practice, it seems unnecessary to have a seed since, even though there exists some A with the hard-coded collision x, x', we as humans do not know the code of this A. However, we do not have very good ways of formalizing this mathematically.