Lecture 6
CS 276 Spring 2004

Lecture by Prof. David Wagner
Scribe notes by Arel Cordero and Matt Piotrowski

1 Symmetric-Key Encryption

In symmetric-key encryption (see Figure 1), we have two parties, Alice and Bob, who want to communicate securely, and a third party, Eve, who wants to eavesdrop on the communication. Alice and Bob have pre-shared a secret key that they will use in the communication.

1.1 Definition

A symmetric-key encryption scheme is defined as a pair (E, D), where E is an encryption algorithm and D is a decryption algorithm.

E has the signature:

$$E : keys \text{ x } msgs \rightarrow ciphertexts$$

We require that E be efficient, and we allow it to be randomized and stateful; this implies that the same input can map to more than one ciphertext.

D has the signature:

$$D : keys \text{ x } ciphertexts \rightarrow msgs \cup \{\perp\}$$

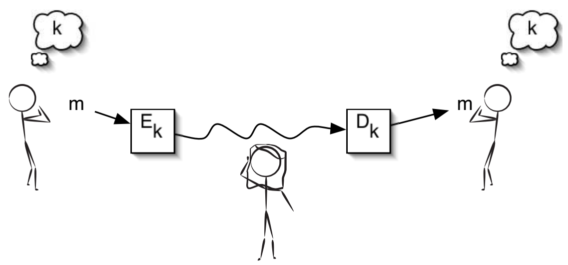

Figure 1: Symmetric-Key Encryption

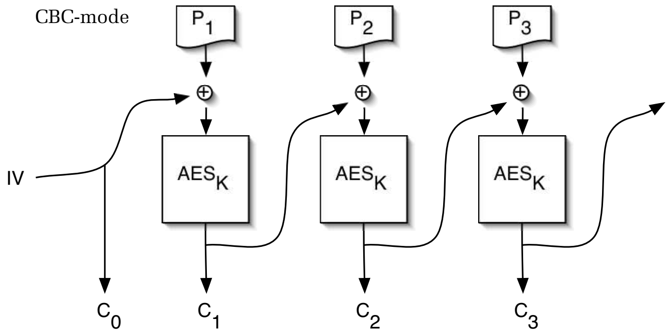

Figure 2: CBC Mode

We require that D be efficient, and we require it to be deterministic and stateless; this implies that the same input cannot map to more than one message (note: contrast this with E). Note that the $\perp$ in the range of D means that some ciphertexts may be invalid.

We also require that the following holds:

$$D_K(E_K(M)) = M$$

1.2 Examples

The following are some examples of symmetric-key encryption schemes.

1.2.1 CBC Mode

Figure 2 shows how CBC Mode works. The Initialization Vector (IV) can come from two sources:

(1) a random source, in which case we refer to the scheme as CBC$. As a notational convention, $ will be used to denote randomness.

(2) 1+ the IV used to encrypt the previous message. This scheme, which we may refer to as CBCC, is necessarily stateful.

1.2.2 CTR Mode

Figure 3 shows how CTR Mode works. The IV can come from two sources:

(1) a random source, in which case we refer to the scheme as CTR$

(2) the final IV of the previous encryption + 1, in which case we refer to the scheme as CTRC.

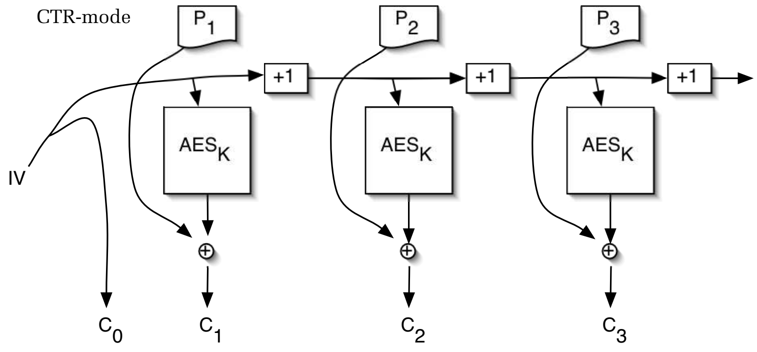

Figure 3: CTR Mode

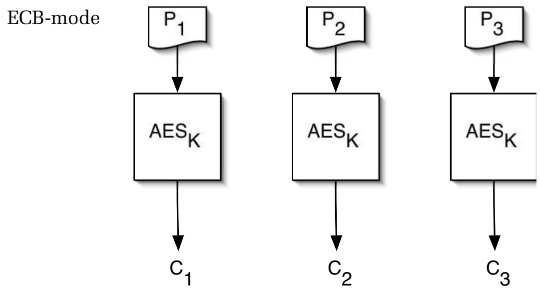

Figure 4: ECB Mode

1.2.3 ECB Mode

Figure 4 shows how ECB Mode works. Later we show its insecurity.

1.3 Definitions of Security

In this section, we define what it means for a symmetric-key encryption algorithm to be secure, but to begin with we take some rough swings at defining security.

1.3.1 Bad Definitions

Attempt 1: Eve shouldn't be able to recover the key.

So an algorithm is secure if given $M, E_K(M)$, an adversary can't find K

Problem: The simple identity algorithm $E_K(M) = M$ fits this description, but intuitively we know it's not secure.

Attempt 2: An algorithm is secure if given $E_K(M)$, an adversary can't find M, or more generally, given pairs of known encryptions $(M_i,\ E_K(M_i))$ and an unknown encryption $E_K(M)$, an adversary can't find M.

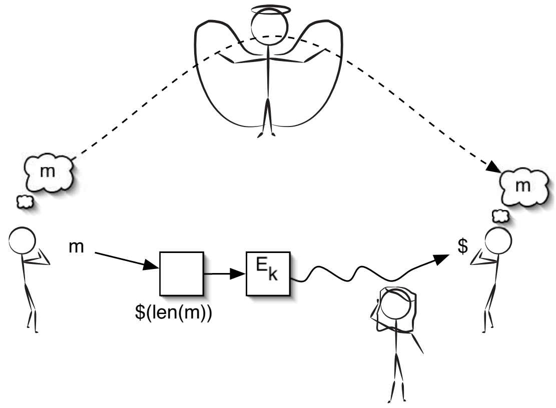

Figure 5: Angel Scenario

Problem: Even if an adversary can't learn M, they may still be able to learn important information about M; for example, if we had an encryption scheme that leaked the first bit of M, this may be very important information – we can imagine a scenario where this bit represents either a buy or sell request in the stock market.

Attempt 3: Given a pairs of known encryptions $(M_i,\ E_K(M_i))$ and an unknown encryption $E_K(M)$, an adversary has no idea what M is.

Problem: This definition is too strong; an adversary may have prior knowledge about M, e.g. it is either yes or no, and we can't require an encryption algorithm to take away this prior knowledge.

1.3.2 Angelic Transport Scenario

To help us in defining what it means for a symmetric-key encryption algorithm to be secure, we imagine an ideal situation. Figure 5 shows the so-called "Angelic" scenario. Ideally, Alice would love if she could just *think* the message M, and have it transported by some devine intervention to Bob, with no chance of it being eavesdropped upon by any adversary.

That sounds a little too good to be true, so lets allow the adversary to gain *some* information, say:

(1) when the message was sent

(2) the length of the message

We implement this by sending a random message of the same length as M (i.e. $\$(len(M))$), encrypted by the algorithm we are testing.

1.3.3 Good Definitions

"Real-or-Random" Security:

Considering the Angelic Scenario, we can formalize the difference between the "real" world and the ideal, or "random", world as follows:

$$Adv^{rr}A = Pr[A^{E_K} = 1] - Pr[A^{E_K \circ \$ \circ len} = 1]$$

In the lefthand probability, the attacker gets an Oracle representing the real world, and in the righthand probability, the attacker gets an Oracle representing the random world.

Just to clarify the notation, len is a function that returns the length of a string, \$ is a function that returns a random string of the specified length, and $\circ$ means the functions are composed.

Since the attacker gets to query the Oracle with any plaintexts of her choosing, we refer to this attack as a chosen plaintext attack.

"Left-or-Right" Security:

$$Adv^{lr}A = Pr[A^{E_K \circ \S_0} = 1] - Pr[A^{E_K \circ \S_1} = 1]$$

$$\text{where } \S_b(x_0,\ x_1) = x_b$$

In this definition of security, the attacker gets to query an Oracle with a pair of messages. In the lefthand probability, the Oracle encrypts the first message; in the righthand probability, the Oracle encrypts the second message. The attacker must distinguish between these two.

Note that for every query $(M_1,\ M_2)$, we require M_1 and M_2 to have the same length.

"Find-then-Guess" Security:

$$\begin{aligned} Adv^{fg}A = Pr[(M_0,\ M_1,\ s) = A^{E_K}(\text{``}find'')\ ;\ A^{E_K}(\text{``}guess'',\ E_K(M_0),\ s) = 1] \\ - Pr[(M_0,\ M_1,\ s) = A^{E_K}(\text{``}find'')\ ;\ A^{E_K}(\text{``}guess'',\ E_K(M_1),\ s) = 1] \end{aligned}$$

In this definition of security, the attacker goes through two phases: in the first phase, the "find" phase, the attacker is allowed to query the Oracle in order to find two messages M_0 and M_1 that it wants to use in the second phase; in the second phase, the "guess" phase, the attacker must decide whether it is given the encryption of M_0 or the encryption of M_1.

Note that s represents some possible state the attacker is allowed to keep across the two stages.

1.3.4 Attacks

As a sanity check on our definitions of security, we should be able to show that ECB Mode is insecure.

In the Real-or-Random definition, we can create an attacker that queries the given Oracle with the message $0\|0$ (0 representing a block of 0's the same size as the blocks that are input in ECB). We see that if we are given an Oracle from the real world, the answer to our query will be a ciphertext whose first block is the same as its second block (see Figure 4 to see why this is). However, if we are given an Oracle from the random world, the answer to our query will be a random ciphertext whose first block is highly unlikely to be the same as its second block. Therefore, our attacker can distinguish between the two Oracles.

In the Left-or-Right definition, we can create an attacker that queries the given Oracle with the pair of messages $(0\|0,\ 0\|1)$. We see that if we are given an Oracle that encrypts the left input, the answer to our query will be a ciphertext whose first block is the same as its second block. However, if we are given an Oracle that encrypts the right input, the answer to our query will be the encryption of 0 concatenated with the encryption of 1, and the two encryptions are highly unlikely to be the same. Therefore, our attacker can distinguish between the two Oracles.

In the Find-then-Guess definition, we can create an attacker who doesn't even need the "find" stage. It inputs to the "guess" stage the pair of messages $(0\|0, 0\|1)$. We see that if we are given the encryption of the first message, it will be a ciphertext whose first block is the same as its second block. However, if we are given the encryption of the second message, it will be the encryption of 0 concatenated with the encryption of 1, and the two encryptions are highly unlikely to be the same. Therefore, our attacker can distinguish between the encryption of the two messages.

We now show a somewhat more surprising attack: an attack on any deterministic, stateless E.

In the Real-or-Random definition, we can create an attacker that queries the Oracle with a message M and then queries the Oracle again with the same message M. If the Oracle is from the real world, it will return the same ciphertext to both queries because E is deterministic and stateless; however, if the Oracle is from the random world, it is highly unlikely that it will return the same ciphertext to both queries because \$ is not deterministic. Therefore, we can distinguish between the two Oracles.

In the Left-or-Right definition, we can create an attacker that queries the Oracle with the pair of messages $(M_0,\ M_0)$ and then queries the Oracle with the pair of messages $(M_0,\ M_1)$. If the Oracle is encrypting the left input, then the answer to both queries will be the same. However, if the Oracle is encrypting the right input, then the answer to both queries is highly unlikely to be the same. Therefore, we can distinguish between the two Oracles.

In the Find-then-Guess Definition, we can create an attacker whose "find" phase consists of querying the Oracle with message M_0 and remembering the answer, and who supplies the pair (M_0, M_1) to the "guess" phase. If the encrypted message that the attacker sees in the "guess" phase is the same as in the "find" phase, then with very high probability, the attacker is seeing the encryption of the first message. Similarly, if the encrypted message that the attacker sees in the "guess" phase is different from the one remembered in the "find" phase, then with very high probability, the attacker is seeing the encryption of the second message. Therefore, the attacker can distinguish between the encryption of the two messages.

As a final example of an attack, we show that CBCC Mode is not secure according to the Real-or-Random definition of security. We do this by creating an attacker that queries the Oracle with a message M and then queries the Oracle with the message $M \oplus 1$. Assuming the IV is 0 in the initial state, and each message encryption increments the IV by one, then if we are given the Oracle representing the *real* world, we will see the same answer to both queries.

To see why this is the case, see Figure 2. The first answer from the Oracle will be $E_K(M \oplus 0) = E_K(M)$ and the second answer from the Oracle will be $E_K((M \oplus 1) \oplus 1) = E_K(M)$. On the other hand, if we are given a random Oracle, the chance of seeing the same answer from both queries is very small. Note, if the starting IV value were n, we would want to query the Oracle first with message M, followed by message $M \oplus n\prime$ such that $n\prime \oplus (n+1) = n$. Therefore, we can distinguish between the two Oracles.

1.3.5 Equivalence of Good Definitions

You may have started to notice that there are similarities in how we performed the attacks in the previous section, regardless of the security definition being used. It turns out that all 3 definitions of security, Real-or-Random, Left-or-Right, and Find-then-Guess, are equivalent. We say that an encryption algorithm that is secure according to these definitions is IND-CPA (INDistinguishable under Chosen Plaintext Attacks). In this section, we begin to prove the equivalence of the IND-CPA definitions.

Claim: Left-or-Right Security $\Leftrightarrow$ Real-or-Random Security

Proof: In two parts, $\Rightarrow$ and then $\Leftarrow$

First Part: Left-or-Right Security $\Rightarrow$ Real-or-Random Security

Suppose A rr-breaks E

let B^g:

1. simulate running A^f where $f(M) = g(M, M')$ and where $M' \leftarrow \$(len(M))$
2. output whatever A does

Note that

$B^{E_K \circ \S_0} = A^{E_K}$

and

$B^{E_K \circ \S_1} = A^{E_K \circ \$ \circ len}$

therefore,

$AdvB = AdvA$

Second Part: Real-or-Random Security $\Rightarrow$ Left-or-Right Security

We prove this using a hybrid argument.

Proof sketch: Show that $A^{E_K \circ \S_0}$ is close to $A^{E_K \circ \$ \circ len \circ \S_0}$, which is close to $A^{E_K \circ \S_1}$

Next lecture: Show Find-then-Guess is equivalent to the other two.