In the last lecture, we discussed properties of pseudo-random permutations (PRPs) and how to construct a PRP from a pseudo-random function (PRF). In this lecture, we continue exploring the properties of PRPs. We also introduce the notion of Block Ciphers. In practice, PRPs are known as Block Ciphers. We discuss how to get a PRBG from a block cipher. We then discuss some different “modes of operation”. We present a sketch of a security proof for CBC encryption mode and show how to use block ciphers for message authentication.

1 PRPs and block ciphers

PRPs in practice are known as block ciphers. Two examples of block ciphers are DES and AES. Block ciphers are parameterized by:

1. \( n \) - number of bits per block
2. \( l \) - bit length of the key

For example, DES uses 64 bits per block and 56 bit keys. AES typically uses 128 bits per block and 128 or 256 bit keys. We would like to know what can be done with a block cipher. Using the standard encryption notation: \( E_k, D_k = E_k^{-1} \), and assume \( E_k \) is a PRP.

(Note that it is possible to design a distinguisher that tells whether \( E_k \) is a PRP vs a PRF – its just a collision detector. However, for many bits, the PRP “looks like” a PRF, since by the “birthday paradox”, this distinguisher should take an exponential number of queries to have a realistic chance of succeeding.

We’ve already seen that given a PRBG we can get a PRF, and given a PRF we can get a PRP. In fact, we can also obtain a PRBG from a block cipher:

**Example 1.** To obtain a PRBG from a block cipher, use \( k \) as a seed, and generate
\[ E_k(<0>)E_k(<1>)... \]
This is called “counter mode”. □

Traditionally, block ciphers are used for encryption. However, using a block cipher, one cannot encrypt more than a small block of data. Thus we need to find ways to encrypt more than a single block using a block cipher.

2 Modes of Operation

This leads us to our next topic, *modes of operation* for block ciphers. To encrypt longer messages, several *modes of operations* may be used. These modes preserve the confidentiality of a block cipher, over longer inputs. In fact, some of them may also be used to preserve data integrity over longer messages.
2.1 Cipher Block Chaining (CBC) Mode

CBC is essentially a probabilistic encryption scheme. In the following example, the plaintext is the message $x_1x_2x_3$ and the ciphertext is $c_0c_1c_2c_3$.

In CBC mode, we always put the tap right before the encryption box. CBC is secure for variable length messages, and achieves security for multiple messages. One of the properties desired from a “mode of operation” is that it should be possible to decrypt the ciphertext on the fly. In the case of CBC, $x_i = D_k(c_i) \oplus c_{i-1}$, And, thus you can decrypt the $i^{th}$ ciphertext block, as soon as you get it.

2.2 Cipher Feedback (CFB) mode

The CFB mode is shown below,

One nice property of the CFB mode is that the decryption is just: $x_i = E_k(c_{i-1}) \oplus c_i$. Thus, you can implement this using just a PRF, since you don’t even need the invertability property from $E_k$.

2.3 Output Feedback (OFB) mode

OFB mode is also called a stream cipher. It is really just a PRBG used as a one time pad.
2.4 Counter mode

Counter mode is has more recently been accepted. It is also a stream cipher. N.B. You need some deterministic function to standardize the position of the counter, which leads to the issue of synchronization between the sender and receiver.

2.5 Note on Stream Ciphers

With stream ciphers such as the two examples above, if you flip a bit of ciphertext and decrypt, you’ll get the plaintext with one bit flipped. Thus, these modes do not preserve data integrity, for which you will need to add a MAC on top.

3 Security of CBC encryption

What follows is an analysis of the security of CBC encryption with respect to multi-message attack. Suppose there are a number of plaintext queries, so that plaintext\(i\) = \(x_1^{(i)} x_2^{(i)} \ldots\). Also assume the key is fixed for the encryption.
if there are \( q \) queries, \( i \) runs from 1...\( q \) and query \( i \) has length \( \leq T \) This is a variable length scheme. The adversary chooses messages adaptively. As usual, our proof will proceed as a sequence of indistinguishable games.

- **Game 0**: Original game
  Let, \( S_0 = \text{[adversary outputs 1 in game 0]} \)

- **Game 1**: Replace block cipher by a random function
  Note that if all \( y \) values are distinct, the \( c \) values will be random. Also if the \( c \) values are random, then there will be no collisions in the \( y \) values. Thus there is an apparent circularity in this argument which we need to break.

Thus it is clear that, \( Pr[S_0] = Pr[S_1] \).

- **Game 2**: Use **gnomes**. The gnome is supposed to maintain consistency, i.e. if the gnome saw a \( y \) value before, he’ll output the same result. Define random variables \( R_j^{(i)} \) for \( i = 1 \ldots q, j = 0 \ldots q \). The gnome’s algorithm to determine \( C_j^{(i)} \) is below:
if \( j = 0 \) then output \( C_j \leftarrow R_j \)
else if \( y_j = y_j' \) for some previous \( i' \), \( j' \)
then we output \( C_j \leftarrow C_j' \)
(i.e. if the gnome remembers input then use previous value)
else \( C_j \leftarrow R_j \)

Define, \( S_2 = \text{adversary outputs 1 in game 2} \)
Claim: \( Pr[S_2] = Pr[S_1] \)

– Game 3: Now we’ll remove the consistency check from the gnome’s algorithm. So kick the gnome in the head so it forgets all the queries it previously responded to.
As usual \( S_3 = \text{adversary outputs 1 in game 3} \)
Now, define the event \( F := y_j = y_j' \) for some \( (i', j') \).

Then, \(|Pr[S_3] - Pr[S_2]| \leq Pr[F]\)

by the difference lemma. In game 3, the boxes are now just random responders, so

\[ F : R_j \oplus x_{j+1} = R_j' \oplus x_{j+1}' \text{ for some pairs (i,j), (i',j')} \]

There are \( q^2 T^2 \) pairs of index-pairs, and each event occurs with probability \( \frac{1}{2^n} \).
Therefore, \( Pr[F] = O(\frac{q^2 T^2}{2^n}) \)

People have analyzed other “modes of operation as well, and get a quadratic degradation as in the case of CBC.

4 Using block ciphers for Message Authentication

Use CBC-MAC. Employ a constant IV, and output only the last block.

\[ y = \text{CBC-MAC}(x_1, \ldots, x_T) \]

There are some issues with the CBC map. Facts about CBC-MACs: (not proved in class)

– for fixed length inputs, it is a PRF.
Here’s a break with variable length messages:

\[ y = MAC_k(x_1) \]
\[ z = MAC_k(y \bigoplus x_2) \]

Then, \[ z = MAC_k(x_1 \circ x_2) \]

– appending the length (a) to the beginning and (b) to the end do not work (or people don’t like them) They can be fixed though.

5 Using Symmetric key cryptography in a network

The last part of symmetric key cryptography is using it in a network setting. Suppose there are users \( U_1, \ldots, U_n \), and each pair of users \((U_i, U_j)\) share some keys \( k_{ij} \). So there’s a quadratic number of keys. In encryption, we have an eavesdropping adversary. There are homework exercises for multi-message security and for multi-key security.

5.1 Message Authentication

In this situation, a more malicious adversary attempts to modify messages in transit.

1. We already have multi-message security, so let’s not worry about secrecy. The adversary won’t be able to predict another tag, so he can’t create new message/tag combos.

2. multi-user security/multi-key security

Note that (2) follows from (1)

Another formulation:

– keys \( k_1, \ldots, k_s \)

– The adversary can make queries to an oracle: message \( x \), index \( i = 1, \ldots, s \) and obtain \( F_{k_i}(x) \)

– The adversary wins by predicting something he didn’t ask.

The adversary outputs \( x, t, i \) and wins if \( F_{k_i}(x) = t \) and \((x,i)\) are never queried.
Everywhere there is a check denotes that the adversary queried \((x_i, k_j)\) Everywhere else he shouldn’t know about. We can prove security using the “plug and pray” argument:

- Guess where break will occur
- **plug** challenge instance at the guessed breakpoint

Simulated multi-key challenger

- The adversary’s view is identical to that in the actual game.
- His view is independent of \(i'\).
With this argument, once we have an unpredictable function in the single key sense, we’ll have one for multi-key.

5.2 Encrypting and Authenticating

Suppose we have an encryption \( E_{ek}, D_{ek} \) for an encryption key \( ek \), a MAC key \( mk \) and function \( F_{mk} \).

\[
\begin{align*}
\text{Encrypt then MAC:} & \quad E_{ek}(x') , F_{mk}(c) \\
\text{Encrypt/MAC:} & \quad E_{ek}(x) , F_{mk}(c)
\end{align*}
\]

\[
\begin{align*}
\text{Sender} & \quad \text{Receiver} \\
\text{c} & \quad \text{c} \\
\text{t} & \quad \text{t} \\
\text{\textbf{check that } t = F_{mk}(c)} & \quad \text{\textbf{compute } x = D_{ek}(c)} \\
\text{\textbf{check that } t = F_{mk}(x)} & \quad \text{\textbf{compute } x = D_{ek}(c)}
\end{align*}
\]

(comment: if we only assume \( \{F_{mk}\} \) is unpredictable then it may be insecure. \( F_{mk} \) could leak \( x \) entirely and still be unpredictable. For example, start with any unpredictable function \( F \). Define \( F'(x) := F(x)||x \).)

6 next week

In the next lecture we’ll begin studying Public Key cryptography.