Active Adversary

Lecture 7
CCA Security
MAC
Active Adversary
Active Adversary

An active adversary can inject messages into the channel.
Active Adversary

An active adversary can inject messages into the channel.

Eve can send ciphertexts to Bob and get them decrypted.
Active Adversary

An active adversary can inject messages into the channel.
Eve can send ciphertexts to Bob and get them decrypted.

Chosen Ciphertext Attack (CCA)
Active Adversary

An active adversary can inject messages into the channel.

Eve can send ciphertexts to Bob and get them decrypted.

Chosen Ciphertext Attack (CCA)

If Bob decrypts all ciphertexts for Eve, no security possible.
Active Adversary

An active adversary can inject messages into the channel.

Eve can send ciphertexts to Bob and get them decrypted.

Chosen Ciphertext Attack (CCA)

If Bob decrypts all ciphertexts for Eve, no security possible.

What can Bob do?
Symmetric-Key Encryption

SIM-CCA Security

REAL \approx \text{IDEAL}
Symmetric-Key Encryption

SIM-CCA Security

SIM-CCA secure if:

∀ ∃ s.t. ∀ Key/Enc, ∀ Key/Dec, ∀ Env, Send, Recv, Real, Ideal

Invalid ciphertexts are silently ignored

REAL ≈ IDEAL
Symmetric-Key Encryption

IND-CCA Security

Experiment picks $b \leftarrow \{0,1\}$ and $K \leftarrow \text{KeyGen}$

Adv gets (guarded) access to $\text{Dec}_K$ oracle

For as long as Adversary wants

Adv sends two messages $m_0, m_1$ to the experiment

Expt returns $\text{Enc}(m_b, K)$ to the adversary

Adversary returns a guess $b'$

Experiments outputs 1 iff $b' = b$

IND-CCA secure if for all feasible adversaries $\Pr[b' = b] \approx 1/2$

IND-CCA + ~correctness equivalent to SIM-CCA

Replay Filter: No challenge ciphertext answered
CCA Security
CCA Security

How to obtain CCA security?
CCA Security

How to obtain CCA security?

Use a CPA-secure encryption scheme, but make sure Bob “accepts” and decrypts only ciphertexts produced by Alice.
CCA Security

How to obtain CCA security?

Use a CPA-secure encryption scheme, but make sure Bob “accepts” and decrypts only ciphertexts produced by Alice.

i.e., Eve can’t create new ciphertexts that will be accepted by Bob.
CCA Security

How to obtain CCA security?

Use a CPA-secure encryption scheme, but make sure Bob “accepts” and decrypts only ciphertexts produced by Alice

i.e., Eve can’t create new ciphertexts that will be accepted by Bob

Achieves the stronger guarantee: in IDEAL, Eve can’t send its own messages to Bob
CCA Security

How to obtain CCA security?

Use a CPA-secure encryption scheme, but make sure Bob "accepts" and decrypts only ciphertexts produced by Alice i.e., Eve can’t create new ciphertexts that will be accepted by Bob

Achieves the stronger guarantee: in IDEAL, Eve can’t send its own messages to Bob

CCA secure SKE reduces to the problem of CPA secure SKE and (shared key) message authentication
CCA Security

How to obtain CCA security?

Use a CPA-secure encryption scheme, but make sure Bob “accepts” and decrypts only ciphertexts produced by Alice i.e., Eve can’t create new ciphertexts that will be accepted by Bob

Achieves the stronger guarantee: in IDEAL, Eve can’t send its own messages to Bob

CCA secure SKE reduces to the problem of CPA secure SKE and (shared key) message authentication

MAC: Message Authentication Code
Message Authentication Codes
Message Authentication Codes

A single short key shared by Alice and Bob
Message Authentication Codes

- A single short key shared by Alice and Bob
- Can sign any (polynomial) number of messages
Message Authentication Codes

- A single short key shared by Alice and Bob

- Can sign any (polynomial) number of messages

- A triple (KeyGen, MAC, Verify)
Message Authentication Codes

- A single short key shared by Alice and Bob
  - Can sign any (polynomial) number of messages
- A triple (KeyGen, MAC, Verify)
- Correctness: For all $K$ from KeyGen, and all messages $M$, $\text{Verify}_K(M, \text{MAC}_K(M)) = 1$
Message Authentication Codes

- A single short key shared by Alice and Bob
  - Can sign any (polynomial) number of messages
- A triple (KeyGen, MAC, Verify)
  - Correctness: For all $K$ from KeyGen, and all messages $M$, $\text{Verify}_K(M, \text{MAC}_K(M)) = 1$
  - Security: probability that an adversary can produce $(M, s)$ s.t. $\text{Verify}_K(M, s) = 1$ is negligible unless Alice produced an output $s = \text{MAC}_K(M)$

```latex
\text{Advantage} = \Pr[ \text{Verify}_K(M, s) = 1 \text{ and } (M, s) \not\in \{(M_i, s_i)\} ]
```
CCA Secure SKE
CCA Secure SKE

CCA-Enc_{K_1,K_2}(m) = ( c:= CPA-Enc_{K_1}(m), t:= MAC_{K_2}(c) )
CCA Secure SKE

\[ CCA-Enc_{K_1,K_2}(m) = ( c := CPA-Enc_{K_1}(m), t := MAC_{K_2}(c) ) \]

CPA secure encryption: Block-cipher/CTR mode construction
CCA Secure SKE

\[ \text{CCA-Enc}_{K_1,K_2}(m) = ( c:= \text{CPA-Enc}_{K_1}(m), t:= \text{MAC}_{K_2}(c) ) \]

- CPA secure encryption: Block-cipher/CTR mode construction
- MAC: from a PRF or Block-Cipher (next time)
CCA Secure SKE

\[ \text{CCA-Enc}_{K_1,K_2}(m) = ( c := \text{CPA-Enc}_{K_1}(m), t := \text{MAC}_{K_2}(c) ) \]

- CPA secure encryption: Block-cipher/CTR mode construction
- MAC: from a PRF or Block-Cipher (next time)

**SKE in practice entirely based on Block-Ciphers** (next time)
CCA Secure SKE

\[ \text{CCA-Enc}_{K_1,K_2}(m) = (c := \text{CPA-Enc}_{K_1}(m), t := \text{MAC}_{K_2}(c)) \]

- CPA secure encryption: Block-cipher/CTR mode construction
- MAC: from a PRF or Block-Cipher (next time)
- **SKE in practice entirely based on Block-Ciphers** (next time)

In principle, PRFs can be constructed (less efficiently) based on any One-Way Permutation or even any One-Way Function.
Making a MAC
One-time MAC
One-time MAC

To sign a single n bit message
One-time MAC

- To sign a single n bit message
- A simple (but inefficient) scheme
One-time MAC

- To sign a single n bit message
- A simple (but inefficient) scheme
- Shared secret key: 2n random strings (each k-bit long) \((r_i^0, r_i^1)_{i=1..n}\)

<table>
<thead>
<tr>
<th>r_0^1</th>
<th>r_0^2</th>
<th>r_0^3</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_1^0</td>
<td>r_1^1</td>
<td>r_1^2</td>
</tr>
<tr>
<td>r_1^1</td>
<td>r_1^2</td>
<td>r_1^3</td>
</tr>
</tbody>
</table>
To sign a single n bit message

A simple (but inefficient) scheme

Shared secret key: 2n random strings (each k-bit long) \((r^i_0, r^i_1)_{i=1..n}\)

One-time MAC
One-time MAC

To sign a single n bit message

A simple (but inefficient) scheme

Shared secret key: 2n random strings (each k-bit long) \((r^i_0, r^i_1)_{i=1..n}\)

Signature for \(m_1...m_n\) be \((r^i_{mi})_{i=1..n}\)
To sign a single $n$ bit message

A simple (but inefficient) scheme

Shared secret key: $2n$ random strings (each $k$-bit long) $(r_{i0}, r_{i1})_{i=1..n}$

Signature for $m_1...m_n$ be $(r_{mi})_{i=1..n}$

Negligible probability that Eve can produce a signature on $m' \neq m$
One-time MAC

To sign a single n bit message

A simple (but inefficient) scheme

Shared secret key: 2n random strings (each k-bit long) \((r^i_0, r^i_1)_{i=1..n}\)

Signature for \(m_1...m_n\) be \((r^i_{mi})_{i=1..n}\)

Negligible probability that Eve can produce a signature on \(m' \neq m\)

Doesn't require any computational restrictions on adversary!
One-time MAC

To sign a single n bit message

A simple (but inefficient) scheme

Shared secret key: 2n random strings (each k-bit long) \((r_{i0}, r_{i1})_{i=1..n}\)

Signature for \(m_1...m_n\) be \((r_{mi})_{i=1..n}\)

Negligible probability that Eve can produce a signature on \(m' \neq m\)

Doesn’t require any computational restrictions on adversary!

More efficient one-time MACs exist (later)
(Multi-msg) MAC from PRF
When Each Message is a Single Block
(Multi-msg) MAC from PRF

When Each Message is a Single Block

PRF is a MAC!
(Multi-msg) MAC from PRF
When Each Message is a Single Block

PRF is a MAC!

MAC_K(M) := F_K(M) where F is a PRF
(Multi-msg) MAC from PRF
When Each Message is a Single Block

PRF is a MAC!

\[ \text{MAC}_K(M) := F_K(M) \text{ where } F \text{ is a PRF} \]
(Multi-msg) MAC from PRF
When Each Message is a Single Block

- PRF is a MAC!
- $\text{MAC}_K(M) := F_K(M)$ where $F$ is a PRF
- $\text{Ver}_K(M,S) := 1$ iff $S = F_K(M)$
(Multi-msg) MAC from PRF

When Each Message is a Single Block

- PRF is a MAC!
  - $\text{MAC}_K(M) := F_K(M)$ where $F$ is a PRF
  - $\text{Ver}_K(M, S) := 1$ iff $S = F_K(M)$
  - Output length of $F_K$ should be big enough
(Multi-msg) MAC from PRF

When Each Message is a Single Block

- **PRF is a MAC!**
  - $MAC_k(M) := F_k(M)$ where $F$ is a PRF
  - $Ver_k(M, S) := 1$ iff $S = F_k(M)$
  - Output length of $F_k$ should be big enough

- If an adversary forges MAC with probability $\varepsilon_{MAC}$, then can break PRF with advantage $O(\varepsilon_{MAC} - 2^{-m(k)})$ ($m(k)$ being the output length of the PRF) [How?]
(Multi-msg) MAC from PRF

When Each Message is a Single Block

- **PRF is a MAC!**
  - $\text{MAC}_K(M) := F_K(M)$ where $F$ is a PRF
  - $\text{Ver}_K(M, S) := 1$ iff $S = F_K(M)$
  - Output length of $F_K$ should be big enough

- If an adversary forges MAC with probability $\epsilon_{\text{MAC}}$, then can break PRF with advantage $O(\epsilon_{\text{MAC}} - 2^{-m(k)})$ ($m(k)$ being the output length of the PRF) [How?]

Recall: Advantage in breaking a PRF $F = \text{diff in prob test has of outputting 1, when given F vs. truly random R}$
(Multi-msg) MAC from PRF

When Each Message is a Single Block

**PRF is a MAC!**

- **MAC**<sub>K</sub>(M) := F<sub>K</sub>(M) where F is a PRF
- **Ver**<sub>K</sub>(M,S) := 1 iff S=F<sub>K</sub>(M)
- Output length of F<sub>K</sub> should be big enough

If an adversary forges MAC with probability ε<sub>MAC</sub>, then can break PRF with advantage O(ε<sub>MAC</sub> − 2<sup>−m(k)</sup>) (m(k) being the output length of the PRF) [How?]

If random function R used as MAC, then probability of forgery, ε<sub>MAC*</sub> = 2<sup>−m(k)</sup>
MAC for Multiple-Block Messages
MAC for Multiple-Block Messages

What if message is longer than one block?
MAC for Multiple-Block Messages

What if message is longer than one block?

MAC’ing each block separately is not secure (unlike in the case of CPA secure encryption)
MAC for Multiple-Block Messages

What if message is longer than one block?

MAC’ing each block separately is not secure (unlike in the case of CPA secure encryption)

Eve can rearrange the blocks/drop some blocks
What if message is longer than one block?

MAC’ing each block separately is not secure (unlike in the case of CPA secure encryption)

Eve can rearrange the blocks/drop some blocks

Could use a PRF that takes longer inputs
MAC for Multiple-Block Messages

What if message is longer than one block?

MAC’ing each block separately is not secure (unlike in the case of CPA secure encryption)

- Eve can rearrange the blocks/drop some blocks

Could use a PRF that takes longer inputs

Can we use a PRF with a fixed block-length (i.e., a block cipher)?
MAC for Multiple-Block Messages
MAC for Multiple-Block Messages

A simple solution: “tie the blocks together”
MAC for Multiple-Block Messages

A simple solution: “tie the blocks together”

Add to each block a random string $r$ (same $r$ for all blocks), total number of blocks, and a sequence number
MAC for Multiple-Block Messages

A simple solution: “tie the blocks together”

Add to each block a random string $r$ (same $r$ for all blocks), total number of blocks, and a sequence number.

$B_i = (r, t, i, M_i)$
MAC for Multiple-Block Messages

A simple solution: “tie the blocks together”

- Add to each block a random string \(r\) (same \(r\) for all blocks), total number of blocks, and a sequence number

\[ B_i = (r, t, i, M_i) \]

\[ \text{MAC}(M) = (r, (\text{MAC}(B_i))_{i=1..t}) \]
MAC for Multiple-Block Messages

A simple solution: “tie the blocks together”

Add to each block a random string $r$ (same $r$ for all blocks), total number of blocks, and a sequence number

$B_i = (r, t, i, M_i)$

$MAC(M) = (r, (MAC(B_i))_{i=1..t})$

$r$ prevents mixing blocks from two messages, $t$ prevents dropping blocks and $i$ prevents rearranging
MAC for Multiple-Block Messages

A simple solution: “tie the blocks together”

Add to each block a random string \( r \) (same \( r \) for all blocks), total number of blocks, and a sequence number \( B_i = (r, t, i, M_i) \)

\[
\text{MAC}(M) = (r, (\text{MAC}(B_i))_{i=1..t})
\]

\( r \) prevents mixing blocks from two messages, \( t \) prevents dropping blocks and \( i \) prevents rearranging

Inefficient! Tag length increases with message length
CBC-MAC
CBC-MAC

PRF domain extension: Chaining the blocks
CBC-MAC

PRF domain extension: Chaining the blocks
CBC-MAC

- PRF domain extension: Chaining the blocks

- cf. CBC mode for encryption (which is not a MAC!)
CBC-MAC

- PRF domain extension: Chaining the blocks
- cf. CBC mode for encryption (which is not a MAC!)
- t-block messages, a single block tag
CBC-MAC

- PRF domain extension: Chaining the blocks
- cf. CBC mode for encryption (which is not a MAC!)
- t-block messages, a single block tag
- Can be shown to be secure
**CBC-MAC**

- **PRF domain extension:** Chaining the blocks

- *cf. CBC mode for encryption* (which is not a MAC!)

- *t*-block messages, a single block tag

- Can be shown to be secure

- If restricted to *t*-block messages (i.e., same length)
**CBC-MAC**

- **PRF domain extension**: Chaining the blocks.
  - cf. CBC mode for encryption (which is not a MAC!)
  - $t$-block messages, a single block tag
  - Can be shown to be secure
    - If restricted to $t$-block messages (i.e., same length)
    - Else attacks possible (by extending a previously signed message)
Patching CBC-MAC
Patching CBC-MAC

Patching CBC MAC to handle message of any (polynomial) length but still producing a single block tag (secure if block-cipher is):
Patching CBC-MAC

Patching CBC MAC to handle message of any (polynomial) length but still producing a single block tag (secure if block-cipher is):

- Derive $K$ as $F_K(t)$, where $t$ is the number of blocks
Patching CBC-MAC

Patching CBC MAC to handle message of any (polynomial) length but still producing a single block tag (secure if block-cipher is):

1. Derive $K$ as $F_K(t)$, where $t$ is the number of blocks.
2. Use first block to specify number of blocks.
Patching CBC-MAC

Patching CBC MAC to handle message of any (polynomial) length but still producing a single block tag (secure if block-cipher is):

- Derive $K$ as $F_K(t)$, where $t$ is the number of blocks
- Use first block to specify number of blocks
- Important that first block is used: if last block, message extension attacks still possible
Patching CBC-MAC

Patching CBC MAC to handle message of any (polynomial) length but still producing a single block tag (secure if block-cipher is):

- Derive $K$ as $F_K(t)$, where $t$ is the number of blocks.
- Use first block to specify number of blocks.
- Important that first block is used: if last block, message extension attacks still possible.
- **EMAC**: Output not the last tag $T$, but $F_{K'}(T)$, where $K'$ is an independent key (after padding the message to an integral number of blocks). No need to know message length a priori.
Patching CBC-MAC

Patching CBC MAC to handle message of any (polynomial) length but still producing a single block tag (secure if block-cipher is):

- Derive \( K \) as \( F_{K'}(t) \), where \( t \) is the number of blocks
- Use first block to specify number of blocks
  - Important that first block is used: if last block, message extension attacks still possible

**EMAC**: Output not the last tag \( T \), but \( F_{K'}(T) \), where \( K' \) is an independent key (after padding the message to an integral number of blocks). No need to know message length a priori.

**CMAC**: XOR last message block with a key (derived from the original key using the block-cipher). Also avoids padding when message is integral number of blocks.
Patching CBC-MAC

Patching CBC MAC to handle message of any (polynomial) length but still producing a single block tag (secure if block-cipher is):

- Derive K as $F_{K'}(t)$, where $t$ is the number of blocks
- Use first block to specify number of blocks
- Important that first block is used: if last block, message extension attacks still possible

**EMAC:** Output not the last tag $T$, but $F_{K'}(T)$, where $K'$ is an independent key (after padding the message to an integral number of blocks). No need to know message length a priori.

**CMAC:** XOR last message block with a key (derived from the original key using the block-cipher). Also avoids padding when message is integral number of blocks.

NIST Recommendation. 2005
Patching CBC-MAC

Patching CBC MAC to handle message of any (polynomial) length but still producing a single block tag (secure if block-cipher is):

- Derive $K$ as $F_{K'}(t)$, where $t$ is the number of blocks
- Use first block to specify number of blocks
- Important that first block is used: if last block, message extension attacks still possible

EMAC: Output not the last tag $T$, but $F_{K'}(T)$, where $K'$ is an independent key (after padding the message to an integral number of blocks). No need to know message length a priori.

CMAC: XOR last message block with a key (derived from the original key using the block-cipher). Also avoids padding when message is integral number of blocks.

Later: Hash-based HMAC used in TLS and IPSec

NIST Recommendation. 2005
IETF Standard. 1997
SKE in Practice
Stream Ciphers
Stream Ciphers

A key should be used for only a single stream
Stream Ciphers

- A key should be used for only a single stream
- RC4, eSTREAM portfolio, ...
Stream Ciphers

- A key should be used for only a single stream
- RC4, eSTREAM portfolio, ...
- In practice, stream ciphers take a key and an “IV” (for initialization vector) as inputs
Stream Ciphers

- A key should be used for only a single stream
- RC4, eSTREAM portfolio, ...
- In practice, stream ciphers take a key and an “IV” (for initialization vector) as inputs

Also used to denote the random nonce chosen for encryption using a block-cipher
Stream Ciphers

- A key should be used for only a single stream
- RC4, eSTREAM portfolio, ...
- In practice, stream ciphers take a key and an “IV” (for initialization vector) as inputs
- Heuristic goal: behave somewhat like a PRF (instead of a PRG) so that it can be used for multi-message encryption

Also used to denote the random nonce chosen for encryption using a block-cipher
Stream Ciphers

- A key should be used for only a single stream
- RC4, eSTREAM portfolio, ...

In practice, stream ciphers take a key and an “IV” (for initialization vector) as inputs

- Heuristic goal: behave somewhat like a PRF (instead of a PRG) so that it can be used for multi-message encryption
- But often breaks if used this way
Stream Ciphers

- A key should be used for only a single stream
- RC4, eSTREAM portfolio, ...
- In practice, stream ciphers take a key and an “IV” (for initialization vector) as inputs
  - Heuristic goal: behave somewhat like a PRF (instead of a PRG) so that it can be used for multi-message encryption
  - But often breaks if used this way
- NIST Standard: For multi-message encryption, use a block-cipher in CTR mode

Also used to denote the random nonce chosen for encryption using a block-cipher
Block Ciphers
Block Ciphers

DES, 3DES, Blowfish, AES, ...
Block Ciphers

- DES, 3DES, Blowfish, AES, ...
- Heuristic constructions
Block Ciphers

- DES, 3DES, Blowfish, AES, ...
- Heuristic constructions
- Permutations that can be inverted with the key
Block Ciphers

- DES, 3DES, Blowfish, AES, ...
- Heuristic constructions
- Permutations that can be inverted with the key
- Speed (hardware/software) is of the essence
Block Ciphers

- DES, 3DES, Blowfish, AES, ...
- Heuristic constructions
- Permutations that can be inverted with the key
- Speed (hardware/software) is of the essence
- But should withstand known attacks
Block Ciphers

- DES, 3DES, Blowfish, AES, ...
- Heuristic constructions
- Permutations that can be inverted with the key
- Speed (hardware/software) is of the essence
- But should withstand known attacks
- As a PRP (or at least, against key recovery)
Feistel Network
Feistel Network

Building a permutation from a (block) function
Feistel Network

Building a permutation from a (block) function

Let $f: \{0,1\}^m \rightarrow \{0,1\}^m$ be an arbitrary function
Feistel Network

Building a permutation from a (block) function

Let $f: \{0,1\}^m \rightarrow \{0,1\}^m$ be an arbitrary function

$F_f: \{0,1\}^{2m} \rightarrow \{0,1\}^{2m}$ defined as $F_f(x,y) = (y, x \oplus f(y))$
Feistel Network

Building a permutation from a (block) function

Let $f: \{0,1\}^m \rightarrow \{0,1\}^m$ be an arbitrary function

$F_f: \{0,1\}^{2m} \rightarrow \{0,1\}^{2m}$ defined as $F_f(x,y) = (y, x \oplus f(y))$

$F_f$ is a permutation (Why?)
Feistel Network

Building a permutation from a (block) function

Let $f: \{0,1\}^m \rightarrow \{0,1\}^m$ be an arbitrary function

$F_f: \{0,1\}^{2m} \rightarrow \{0,1\}^{2m}$ defined as $F_f(x,y) = (y, x \oplus f(y))$

$F_f$ is a permutation (Why?)

Can invert (How?)
Feistel Network

Building a permutation from a (block) function

Let $f: \{0,1\}^m \rightarrow \{0,1\}^m$ be an arbitrary function

$F_f: \{0,1\}^{2m} \rightarrow \{0,1\}^{2m}$ defined as $F_f(x,y) = (y, x \oplus f(y))$

$F_f$ is a permutation (Why?)

Can invert (How?)

Given functions $f_1, \ldots, f_t$ can build a $t$-layer Feistel network $F_{f_1 \ldots f_t}$
Feistel Network

Building a permutation from a (block) function

Let $f: \{0,1\}^m \rightarrow \{0,1\}^m$ be an arbitrary function

$F_f: \{0,1\}^{2m} \rightarrow \{0,1\}^{2m}$ defined as $F_f(x,y) = (y, x \oplus f(y))$

$F_f$ is a permutation (Why?)

Can invert (How?)

Given functions $f_1, ..., f_t$ can build a $t$-layer Feistel network $F_{f_1...f_t}$
Feistel Network

Building a permutation from a (block) function

Let \( f: \{0,1\}^m \rightarrow \{0,1\}^m \) be an arbitrary function

\( F_f: \{0,1\}^{2m} \rightarrow \{0,1\}^{2m} \) defined as \( F_f(x,y) = (y, x \oplus f(y)) \)

\( F_f \) is a permutation (Why?)

Can invert (How?)

Given functions \( f_1, ..., f_t \) can build a \( t \)-layer Feistel network \( F_{f_1...f_t} \)

Still a permutation from \( \{0,1\}^{2m} \) to \( \{0,1\}^{2m} \)
Feistel Network

Building a permutation from a (block) function

Let \( f: \{0,1\}^m \rightarrow \{0,1\}^m \) be an arbitrary function

\( F_f: \{0,1\}^{2m} \rightarrow \{0,1\}^{2m} \) defined as \( F_f(x,y) = (y, x \oplus f(y)) \)

\( F_f \) is a permutation (Why?)

Can invert (How?)

Given functions \( f_1, \ldots, f_t \) can build a \( t \)-layer Feistel network \( F_{f_1 \ldots f_t} \)

Still a permutation from \( \{0,1\}^{2m} \) to \( \{0,1\}^{2m} \)

Luby-Rackoff: A 3-layer Feistel network, in which 3 PRFs with independent seeds are the 3 round functions, is a PRP. A 4-layer Feistel gives a strong PRP
Feistel Network

Building a permutation from a (block) function

Let $f: \{0,1\}^m \rightarrow \{0,1\}^m$ be an arbitrary function

$F_f: \{0,1\}^{2m} \rightarrow \{0,1\}^{2m}$ defined as $F_f(x,y) = (y, x \oplus f(y))$

$F_f$ is a permutation (Why?)

Can invert (How?)

Given functions $f_1, \ldots, f_t$ can build a $t$-layer Feistel network $F_{f_1 \ldots f_t}$

Still a permutation from $\{0,1\}^{2m}$ to $\{0,1\}^{2m}$

Luby-Rackoff: A 3-layer Feistel network, in which 3 PRFs with independent seeds are the 3 round functions, is a PRP. A 4-layer Feistel gives a strong PRP

Fewer layers do not suffice! [Exercise]
Luby-Rackoff
Luby-Rackoff

Using Feistel networks of PRFs to build a PRP
Luby-Rackoff

Using Feistel networks of PRFs to build a PRP

A 3-layer Feistel network, with PRFs with 3 independent seeds as the round functions, is a PRP
Luby-Rackoff

- Using Feistel networks of PRFs to build a PRP

- A 3-layer Feistel network, with PRFs with 3 independent seeds as the round functions, is a PRP

- 1 or 2 layers do not suffice! [Exercise]
Luby-Rackoff

Using Feistel networks of PRFs to build a PRP

A 3-layer Feistel network, with PRFs with 3 independent seeds as the round functions, is a PRP

1 or 2 layers do not suffice! [Exercise]

With 4 layers (and 4 independent seeds), it is a strong PRP
Luby-Rackoff

- Using Feistel networks of PRFs to build a PRP
  - A 3-layer Feistel network, with PRFs with 3 independent seeds as the round functions, is a PRP
    - 1 or 2 layers do not suffice! [Exercise]
  - With 4 layers (and 4 independent seeds), it is a strong PRP
    - 3 layers do not suffice! [Exercise]
Luby-Rackoff

Using Feistel networks of PRFs to build a PRP

A 3-layer Feistel network, with PRFs with 3 independent seeds as the round functions, is a PRP

1 or 2 layers do not suffice! [Exercise]

With 4 layers (and 4 independent seeds), it is a strong PRP

3 layers do not suffice! [Exercise]

OWF/OWP \rightarrow PRG \rightarrow PRF \rightarrow (strong) PRP, i.e., Block Cipher
Luby-Rackoff

Using Feistel networks of PRFs to build a PRP

A 3-layer Feistel network, with PRFs with 3 independent seeds as the round functions, is a PRP

1 or 2 layers do not suffice! [Exercise]

With 4 layers (and 4 independent seeds), it is a strong PRP

3 layers do not suffice! [Exercise]

OWF/OWP $\Rightarrow$ PRG $\Rightarrow$ PRF $\Rightarrow$ (strong) PRP, i.e., Block Cipher

OWF/OWP $\Rightarrow$ PRG $\Rightarrow$ PRF is too slow for standards
DES Block Cipher
DES Block Cipher

Data Encryption Standard (DES), Triple-DES, DES-X

NIST Standard. 1976
DES Block Cipher

NIST Standard. 1976

- Data Encryption Standard (DES), Triple-DES, DES-X
- DES uses a 16-layer Feistel network (and a few other steps)
DES Block Cipher

- NIST Standard. 1976
- Data Encryption Standard (DES), Triple-DES, DES-X
- DES uses a 16-layer Feistel network (and a few other steps)
  - The round functions are not PRFs, but ad hoc
DES Block Cipher

- Data Encryption Standard (DES), Triple-DES, DES-X
- DES uses a 16-layer Feistel network (and a few other steps)
  - The round functions are not PRFs, but ad hoc
  - “Confuse and diffuse”
DES Block Cipher

- Data Encryption Standard (DES), Triple-DES, DES-X
- DES uses a 16-layer Feistel network (and a few other steps)
  - The round functions are not PRFs, but ad hoc
    - “Confuse and diffuse”
- Defined for fixed key/block lengths (56 bits and 64 bits); key is used to generate subkeys for round functions

NIST Standard. 1976
DES Block Cipher

- Data Encryption Standard (DES), Triple-DES, DES-X
- DES uses a 16-layer Feistel network (and a few other steps)
  - The round functions are not PRFs, but ad hoc
    - “Confuse and diffuse”
  - Defined for fixed key/block lengths (56 bits and 64 bits); key is used to generate subkeys for round functions
- DES’s key length too short

NIST Standard. 1976
DES Block Cipher

- Data Encryption Standard (DES), Triple-DES, DES-X
- DES uses a 16-layer Feistel network (and a few other steps)
  - The round functions are not PRFs, but ad hoc
    - “Confuse and diffuse”
  - Defined for fixed key/block lengths (56 bits and 64 bits); key is used to generate subkeys for round functions
- DES’s key length too short
  - Can now mount brute force key-recovery attacks (e.g. using $10K hardware, running for under a week, in 2006; now, in under a day)

NIST Standard. 1976
**DES Block Cipher**

- Data Encryption Standard (DES), Triple-DES, DES-X
- DES uses a 16-layer Feistel network (and a few other steps)
  - The round functions are not PRFs, but ad hoc
    - “Confuse and diffuse”
  - Defined for fixed key/block lengths (56 bits and 64 bits); key is used to generate subkeys for round functions
- DES’s key length too short
  - Can now mount brute force key-recovery attacks (e.g. using $10K hardware, running for under a week, in 2006; now, in under a day)
- DES-X: extra keys to pad input and output

NIST Standard. 1976
Data Encryption Standard (DES), Triple-DES, DES-X

DES uses a 16-layer Feistel network (and a few other steps)

- The round functions are not PRFs, but ad hoc
  - “Confuse and diffuse”
- Defined for fixed key/block lengths (56 bits and 64 bits); key is used to generate subkeys for round functions
- DES’s key length too short
  - Can now mount brute force key-recovery attacks (e.g. using $10K hardware, running for under a week, in 2006; now, in under a day)

DES-X: extra keys to pad input and output

Triple DES: 3 successive applications of DES (or DES⁻¹) with 3 keys
AES Block Cipher
AES Block Cipher

Advanced Encryption Standard (AES)

NIST Standard. 2001
AES Block Cipher

- Advanced Encryption Standard (AES)
- AES-128, AES-192, AES-256 (3 key sizes; block size = 128 bits)

NIST Standard. 2001
AES Block Cipher

- Advanced Encryption Standard (AES)
  - AES-128, AES-192, AES-256 (3 key sizes; block size = 128 bits)
  - Very efficient in software implementations (unlike DES)

NIST Standard. 2001
AES Block Cipher

- Advanced Encryption Standard (AES)
  - AES-128, AES-192, AES-256 (3 key sizes; block size = 128 bits)
  - Very efficient in software implementations (unlike DES)
  - Uses “Substitute-and-Permute” instead of Feistel networks

NIST Standard. 2001
AES Block Cipher

- Advanced Encryption Standard (AES)
  - AES-128, AES-192, AES-256 (3 key sizes; block size = 128 bits)
- Very efficient in software implementations (unlike DES)
- Uses “Substitute-and-Permute” instead of Feistel networks
- Has some algebraic structure

NIST Standard. 2001
AES Block Cipher

- Advanced Encryption Standard (AES)
- AES-128, AES-192, AES-256 (3 key sizes; block size = 128 bits)
- Very efficient in software implementations (unlike DES)
- Uses “Substitute-and-Permute” instead of Feistel networks
- Has some algebraic structure
  - Operations in a vector space over the field $GF(2^8)$

NIST Standard. 2001
AES Block Cipher

Advanced Encryption Standard (AES)

- AES-128, AES-192, AES-256 (3 key sizes; block size = 128 bits)

- Very efficient in software implementations (unlike DES)

- Uses “Substitute-and-Permute” instead of Feistel networks

- Has some algebraic structure
  - Operations in a vector space over the field GF(2^8)
  - The algebraic structure may lead to “attacks”?

NIST Standard. 2001
AES Block Cipher

- Advanced Encryption Standard (AES)
  - AES-128, AES-192, AES-256 (3 key sizes; block size = 128 bits)
- Very efficient in software implementations (unlike DES)
- Uses “Substitute-and-Permute” instead of Feistel networks
- Has some algebraic structure
  - Operations in a vector space over the field GF(2^8)
  - The algebraic structure may lead to “attacks”?
- Some implementations may lead to side-channel attacks (e.g. cache-timing attacks)

NIST Standard. 2001
AES Block Cipher

- Advanced Encryption Standard (AES)
- AES-128, AES-192, AES-256 (3 key sizes; block size = 128 bits)
- Very efficient in software implementations (unlike DES)
- Uses “Substitute-and-Permute” instead of Feistel networks
- Has some algebraic structure
  - Operations in a vector space over the field GF(2^8)
  - The algebraic structure may lead to “attacks”?
- Some implementations may lead to side-channel attacks (e.g. cache-timing attacks)
- No “simple” hardness assumption known to imply any sort of security for AES

NIST Standard. 2001
AES Crib Sheet
(Handy for memorizing)

General Math
11B = AES Polynomial = 19(x)

\[ x^4 + x + 1 \]

Fast Multiply
\[ x \cdot a(x) = (a \oplus 1) \oplus (a_7 = 1) \]

1B:00

\[ \log(x \cdot y) = \log(x) + \log(y) \]

Use \( x^1 = 03 \) for log base

5-Box (SRD)
\[ SRD[a] = f(g(a)) \]

Think \( 5^3 \oplus 6^3 \)

5 is and 3's [0110 0011]

Key Expansion: Round Constants

First Column: 01 02 04 08 1C 32 64 128

Mix Columns:

\[ \begin{bmatrix} 2113 2 \\ 3211 1321 1132 \end{bmatrix} \]

Inverse Mix

\[ \begin{bmatrix} EBDQ \\ EBDQ \end{bmatrix} \]

Cryptanalysis
Cryptanalysis

Attacking stream ciphers and block ciphers
Cryptanalysis

- Attacking stream ciphers and block ciphers
- Typically for key recovery
Cryptanalysis

- Attacking stream ciphers and block ciphers
- Typically for key recovery
- Brute force cryptanalysis, using specialized hardware
Cryptanalysis

- Attacking stream ciphers and block ciphers
  - Typically for key recovery
  - Brute force cryptanalysis, using specialized hardware
    - e.g. Attack on DES in 1998
Cryptanalysis

- Attacking stream ciphers and block ciphers
- Typically for key recovery
- Brute force cryptanalysis, using specialized hardware
  - e.g. Attack on DES in 1998
- Several other analytical techniques to speed up attacks
Cryptanalysis

Attacking stream ciphers and block ciphers

Typically for key recovery

Brute force cryptanalysis, using specialized hardware

e.g. Attack on DES in 1998

Several other analytical techniques to speed up attacks

Sometimes “theoretical”: on weakened (“reduced round”) constructions, showing improvement over brute-force attack
Cryptanalysis

Attacking stream ciphers and block ciphers

Typically for key recovery

Brute force cryptanalysis, using specialized hardware

e.g. Attack on DES in 1998

Several other analytical techniques to speed up attacks

Sometimes “theoretical”: on weakened (“reduced round”) constructions, showing improvement over brute-force attack

Meet-in-the-middle, linear cryptanalysis, differential cryptanalysis, impossible differential cryptanalysis, boomerang attack, integral cryptanalysis, cube attack, ...
Authenticated Encryption
Authenticated Encryption

Doing encryption + authentication better
Authenticated Encryption

- Doing encryption + authentication better

- Generic composition: encrypt, then MAC
Authenticated Encryption

- Doing encryption + authentication better
- Generic composition: encrypt, then MAC
- Needs two keys and two passes
Authenticated Encryption

Doing encryption + authentication better

- Generic composition: encrypt, then MAC
- Needs two keys and two passes
- AE aims to do this more efficiently
Authenticated Encryption

Doing encryption + authentication better

Generic composition: encrypt, then MAC

Needs two keys and two passes

AE aims to do this more efficiently

Several constructions based on block-ciphers (modes of operation) provably secure modeling block-cipher as PRP
Authenticated Encryption

Doing encryption + authentication better

- Generic composition: encrypt, then MAC
- Needs two keys and two passes
- AE aims to do this more efficiently
- Several constructions based on block-ciphers (modes of operation) provably secure modeling block-cipher as PRP
  - One pass: IAPM, OCB, ... [patented]
Authenticated Encryption

Doing encryption + authentication better

- Generic composition: encrypt, then MAC
- Needs two keys and two passes
- AE aims to do this more efficiently
  - Several constructions based on block-ciphers (modes of operation) provably secure modeling block-cipher as PRP
    - One pass: IAPM, OCB, ... [patented]
    - Two pass: CCM, GCM, SIV, ... [included in NIST standards]
Authenticated Encryption

Doing encryption + authentication better

Generic composition: encrypt, then MAC

Needs two keys and two passes

AE aims to do this more efficiently

Several constructions based on block-ciphers (modes of operation) provably secure modeling block-cipher as PRP

One pass: IAPM, OCB, ... [patented]

Two pass: CCM, GCM, SIV, ... [included in NIST standards]

AE with Associated Data: Allows unencrypted (but authenticated) parts of the plaintext, for headers etc.
SKE today
SKE today

- SKE in IPsec, TLS etc. mainly based on AES block-ciphers
SKE today

- SKE in IPsec, TLS etc. mainly based on AES block-ciphers
- AES-128, AES-192, AES-256
SKE today

- SKE in IPsec, TLS etc. mainly based on AES block-ciphers
  - AES-128, AES-192, AES-256
- Recommended: AES Counter-mode + CMAC (or HMAC)
SKE today

SKE in IPsec, TLS etc. mainly based on AES block-ciphers

AES-128, AES-192, AES-256

Recommended: AES Counter-mode + CMAC (or HMAC)

Gives CCA security, and provides authentication
SKE today

- SKE in IPsec, TLS etc. mainly based on AES block-ciphers
  - AES-128, AES-192, AES-256
- Recommended: AES Counter-mode + CMAC (or HMAC)
  - Gives CCA security, and provides authentication
- Older components/modes still in use
SKE today

- SKE in IPsec, TLS etc. mainly based on AES block-ciphers
  - AES-128, AES-192, AES-256
- Recommended: AES Counter-mode + CMAC (or HMAC)
- Gives CCA security, and provides authentication
- Older components/modes still in use
- Supported by many standards for legacy purposes
SKE today

- SKE in IPsec, TLS etc. mainly based on AES block-ciphers
  - AES-128, AES-192, AES-256
- Recommended: AES Counter-mode + CMAC (or HMAC)
  - Gives CCA security, and provides authentication
- Older components/modes still in use
  - Supported by many standards for legacy purposes
  - In many applications (sometimes with modifications)
SKE today

- SKE in IPsec, TLS etc. mainly based on AES block-ciphers
  - AES-128, AES-192, AES-256
  - Recommended: AES Counter-mode + CMAC (or HMAC)
  - Gives CCA security, and provides authentication

- Older components/modes still in use
  - Supported by many standards for legacy purposes
  - In many applications (sometimes with modifications)
    - e.g. RC4 in BitTorrent, Skype, PDF