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, Verify\(_K(M, MAC_K(M)) = 1\)

Security: probability that an adversary can produce (M, s) s.t. Verify\(_K(M, s) = 1\) is negligible unless Alice had computed and output s=MAC\(_K(M)\)

\[ \text{Advantage} = \Pr[ \text{Verify}_K(M, s) = 1 \quad \text{and} \quad (M, s) \notin \{(M_i, s_i)\} ] \]
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) \((r_{i0}, r_{i1})_{i=1..n}\)
To sign a single $n$ bit message

A simple (but inefficient) scheme

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

```
010
```

```
\begin{array}{ccc}
  r^i_0 & r^i_2 & r^i_3 \\
  r^i_1 & r^i_2 & r^i_3 \\
\end{array}
```
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) $(r^i_0, r^i_1)_{i=1..n}$
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 $(r^i_{m_i})_{i=1..n}$
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 \(m'\neq m\)
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 \((r_{mi}^i)_{i=1..n}\)

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

More efficient one-time MACs exist (later)
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 \((r_{imi})_{i=1..n}\)

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

More efficient one-time MACs exist (later)

No computational restriction on adversary
(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!

- $\text{MAC}_K(M) := F_K(M)$ where $F$ is a PRF
(Multi-msg) MAC from PRF

When Each Message is a Single Block

PRF is a MAC!

$$\text{MAC}_K(M) := F_K(M)$$ where F is a PRF
(Multi-msg) MAC from PRF

When Each Message is a Single Block

- PRF is a MAC!
- $\text{MAC}_K(M) := F_K(M)$ 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!
  
  - $\text{MAC}_K(M) := F_K(M)$ 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 $O(\epsilon_{\text{MAC}} - 2^{-m(k)})$ (where $m(k)$ is 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 \( \varepsilon_{\text{MAC}} \), then can break PRF with advantage \( O(\varepsilon_{\text{MAC}}2^{-m(k)}) \) (where \( m(k) \) is the output length of the PRF)

[How?]

If random function \( R \) used as MAC, then probability of forgery, \( \varepsilon_{\text{MAC}}^* = 2^{-m(k)} \)
(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 \( \varepsilon_{\text{MAC}} \), then can break PRF with advantage \( O(\varepsilon_{\text{MAC}}2^{-m(k)}) \) (where \( m(k) \) is the output length of the PRF)

[How?]

- If random function \( R \) used as MAC, then probability of forgery, \( \varepsilon^*_{\text{MAC}} = 2^{-m(k)} \)

Recall: advantage in breaking a PRF \( F \) is the diff in prob a test has of outputting 1, when given \( F \) vs. truly random \( R \)
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
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
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

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

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

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

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

- \( r \) 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):
  - Derive K as $F_{K'}(t)$, where t is the number of blocks
  - 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 block with another key (derived from the original key using the block-cipher). 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 block with another key (derived from the original key using the block-cipher). 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 block with another key (derived from the original key using the block-cipher). Avoids padding when message is integral number of blocks.

Later: Hash-based HMAC used in TLS and IPSec

---

Text uses symbols $\odot$ for bullet points.

**NIST Recommendation. 2005**

**IETF Standard. 1997**
SKE in Practice
Stream Ciphers
Stream Ciphers

- Used for one-time encryption
Stream Ciphers

- Used for one-time encryption
- RC4, eSTREAM portfolio, ...
Stream Ciphers

- Used for one-time encryption
- RC4, eSTREAM portfolio, ...
- In practice, stream ciphers take a key and an "IV" (for initialization vector) as inputs
Stream Ciphers

- Used for one-time encryption
- 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

- Used for one-time encryption
- 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

- Used for one-time encryption
- 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

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

- Used for one-time encryption
- 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
Block Ciphers
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 $f_1, ..., f_t$ 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 $f_1,\ldots,f_t$ 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 \( f_1, \ldots, f_t \) can build a \( t \)-layer Feistel network \( F_{f_1 \ldots f_t} \)
  - Still a permutation from \( \{0,1\}^{2m} \) to \( \{0,1\}^{2m} \)
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 $f_1, ..., f_t$ can build a $t$-layer Feistel network $F_{f_1...f_t}$
- Still a permutation from $\{0,1\}^{2m}$ to $\{0,1\}^{2m}$

- **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]
DES Block Cipher
DES Block Cipher

NIST Standard. 1976

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

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

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

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"

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 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
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 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
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 $10K hardware, running for under a week, in 2006; now, in under a day)
- DES-X: extra keys to pad input and output
- Triple DES: 3 successive applications of DES (or DES$^{-1}$) with 3 keys

NIST Standard. 1976
AES Block Cipher
AES Block Cipher

NIST Standard. 2001

Advanced Encryption Standard (AES)
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
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-Permuate” 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”?
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

$x \cdot a(x) = (a \ll 1) \oplus (a^2 = 1)$

$\log(x \cdot y) = \log(x) + \log(y)$

$use(x+1) = 03$ for log base

S-Box (SRD)

$SRD[a] = f(g(a))$

$g(a) = a^{-1} \mod m(c)$

Think $53 \oplus 63$

$S1S$ and $3P3$ ($0110$ $0011$)

Key Expansion: Round Constants

First Column: $01$, $02$, $04$, $08$

Mix Columns:

$2113$ $2$

$2113$

$3211$

$1321$

$1132$

Inverse Mix

$[EBD9]$

$[EBD9]$ $[a_3]$ $[a_2]$ $[a_1]$ $[a_0]$

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 BC 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 BC 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 BC 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 BC 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