MAC.
SKE in Practice.

Lecture 5

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

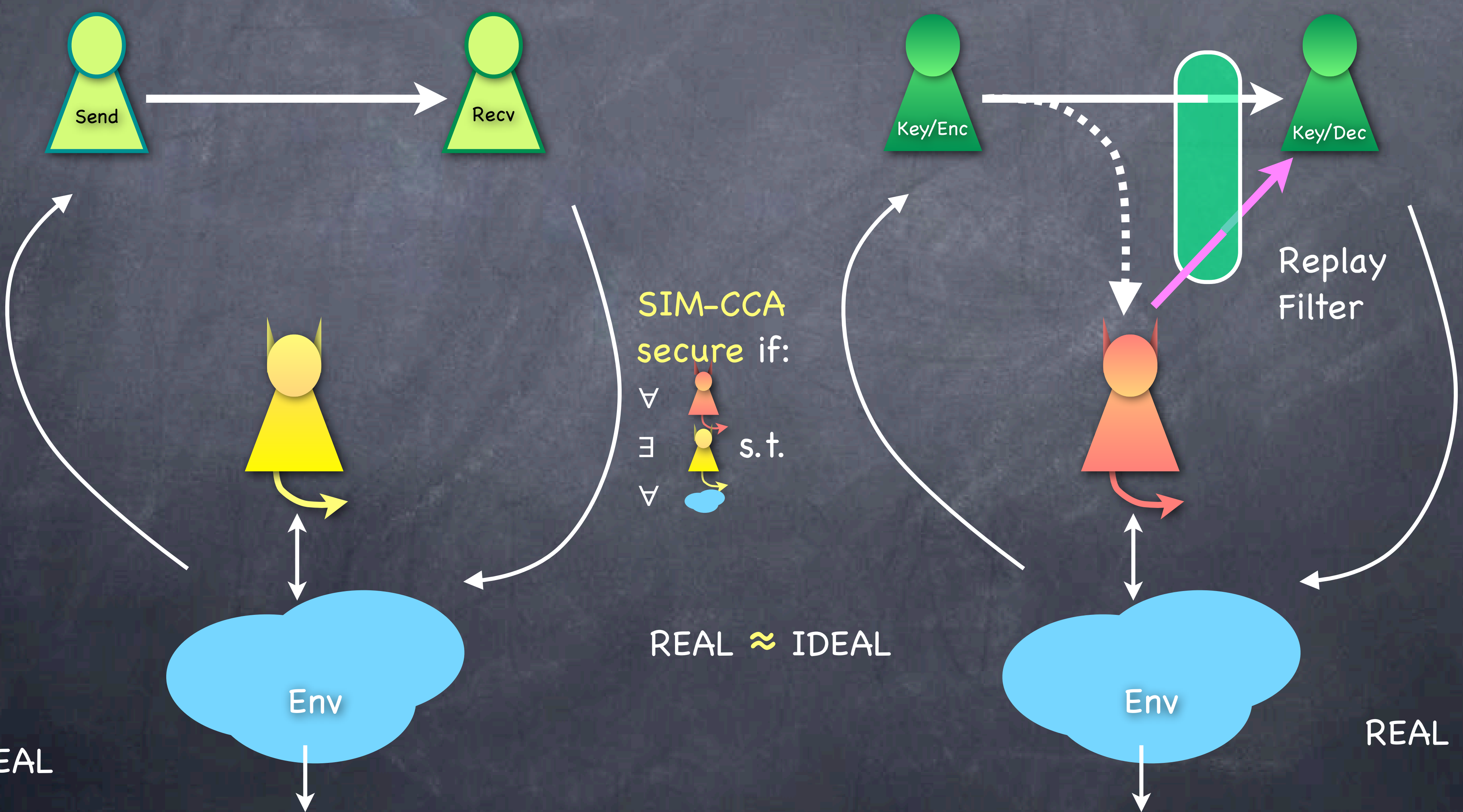
Active Adversary

- 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?

RECALL

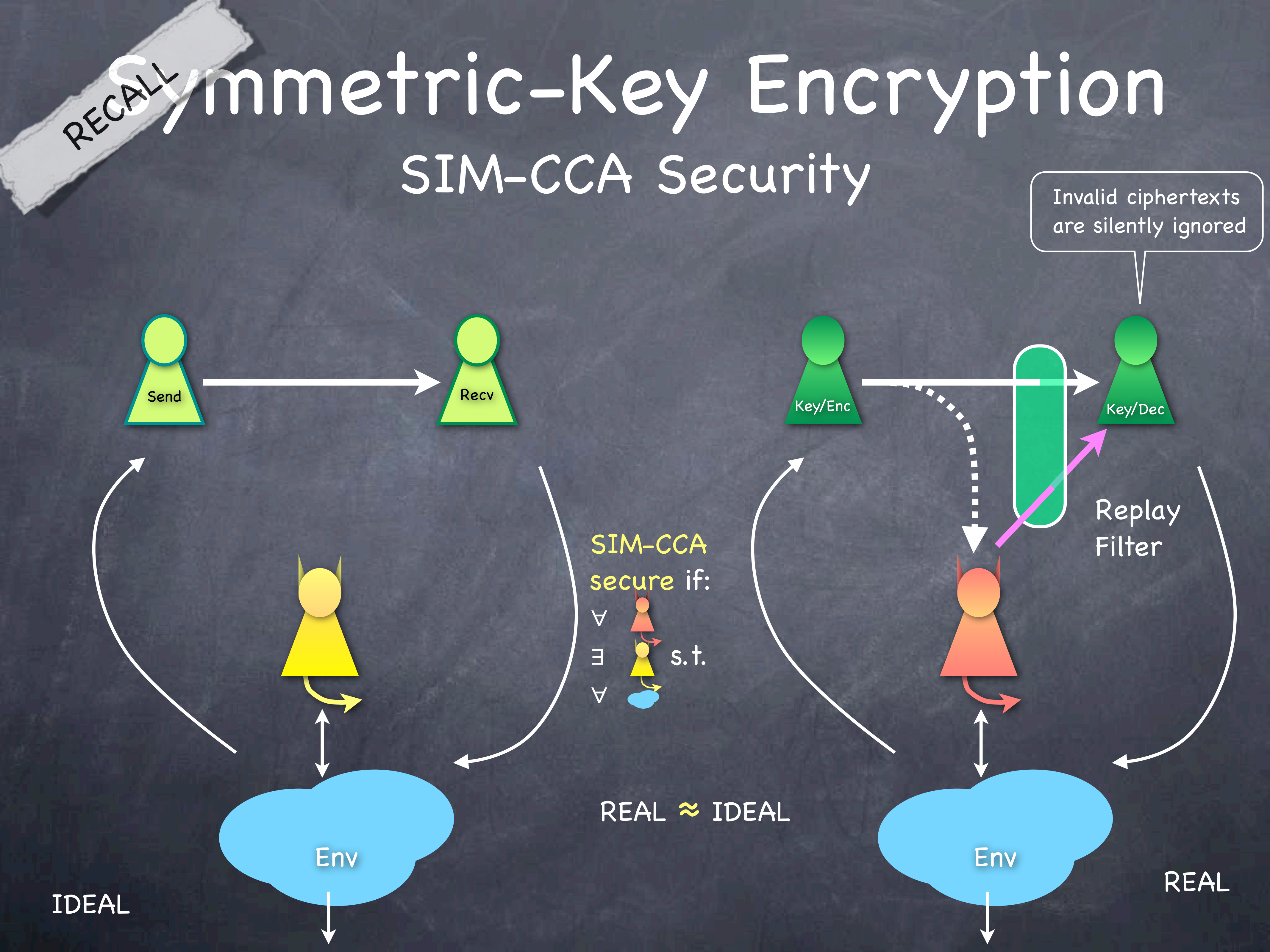
Symmetric-Key Encryption

SIM-CCA Security



Symmetric-Key Encryption

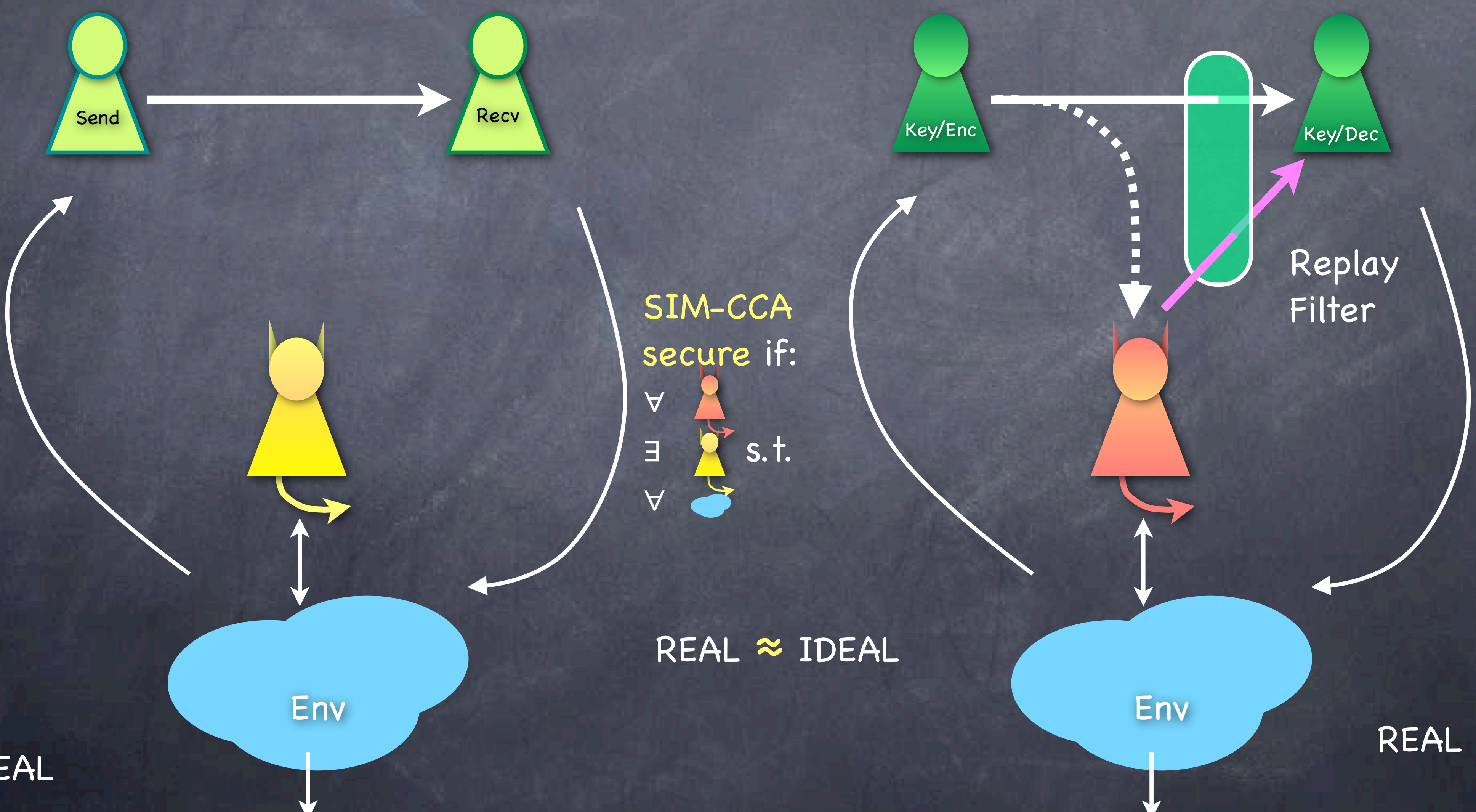
SIM-CCA Security



RECALL

Symmetric-Key Encryption

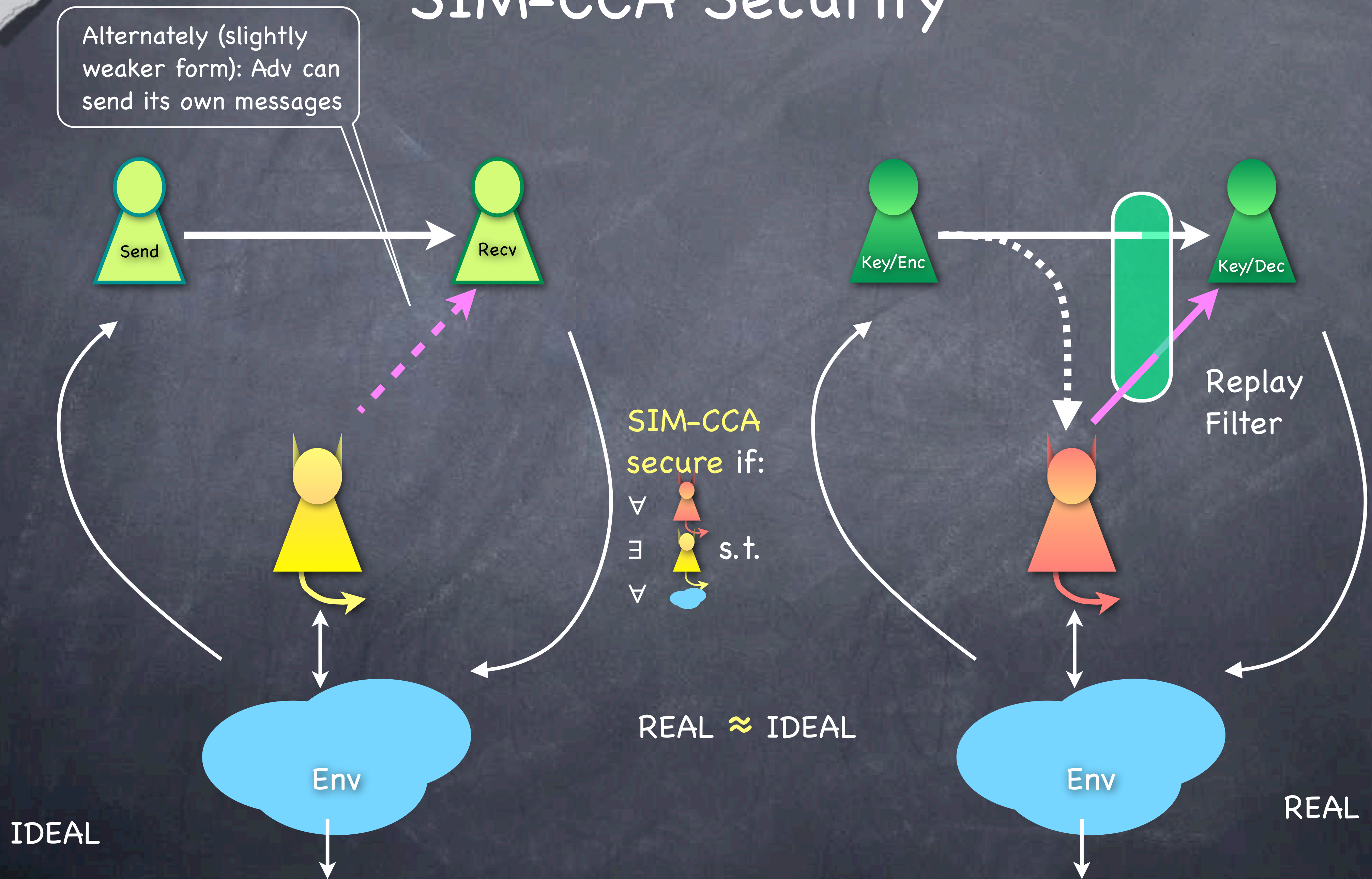
SIM-CCA Security



Symmetric-Key Encryption

RECALL

SIM-CCA Security



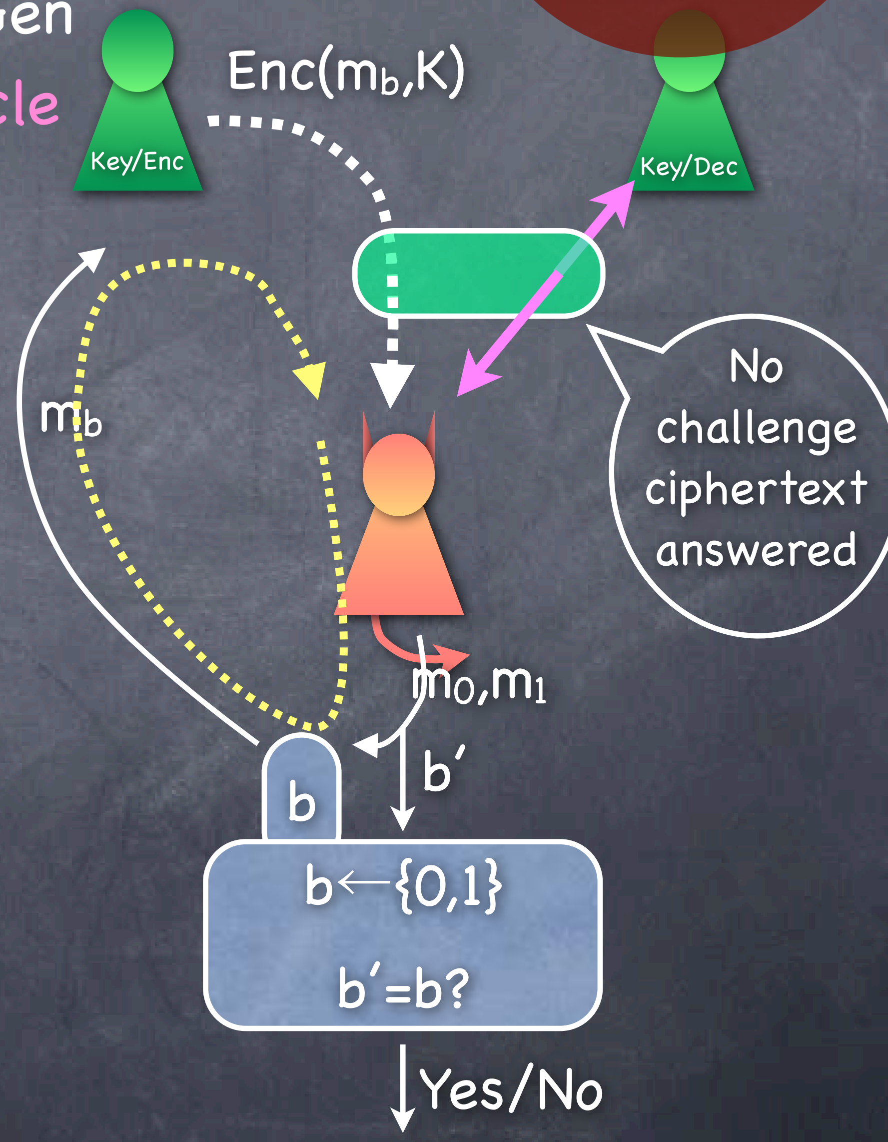
RECALL

Symmetric-Key Encryption

IND-CCA Security

IND-CCA + ~correctness equivalent to SIM-CCA

- Experiment picks b←{0,1} and K←KeyGen
  Adv gets (guarded) access to Dec_K oracle
  - For as long as Adversary wants
    - Adv sends two messages m_0, m_1 to the experiment
    - Expt returns $Enc(m_b,K)$ 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] ≈ 1/2

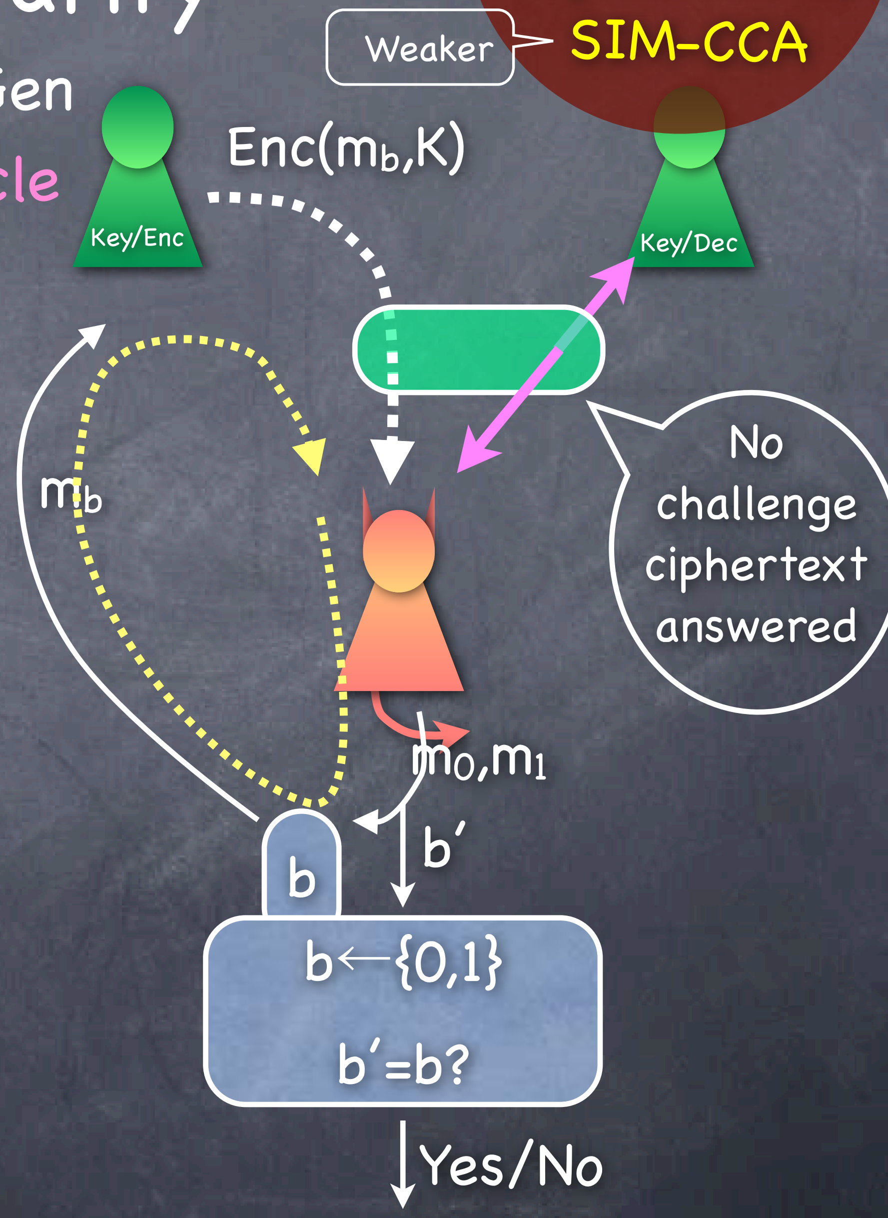


RECALL

Symmetric-Key Encryption

IND-CCA Security

- Experiment picks b←{0,1} and K←KeyGen
- Adv gets (guarded) access to Dec_K oracle
  - For as long as Adversary wants
    - Adv sends two messages m_0, m_1 to the experiment
    - Expt returns $Enc(m_b,K)$ 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] ≈ 1/2



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)

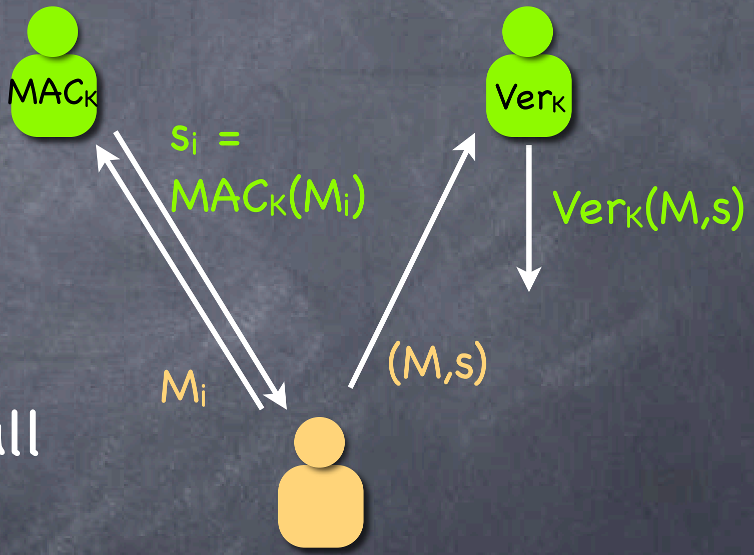
Message Authentication Codes

- 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, $Verify_K(M, MAC_K(M))=1$

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, $Verify_K(M,MAC_K(M))=1$
- Security: probability that an adversary can produce (M,s) s.t. $Verify_K(M,s)=1$ is negligible unless Alice produced an output $s=MAC_K(M)$



Advantage
$= \Pr[\ Ver_K(M,s)=1 \text{ and } (M,s) \notin \{(M_i,s_i)\}\ ]$

CCA Secure SKE

CCA Secure SKE

- $\text{CCA-Enc}_{K1,K2}(m) = (\ c := \text{CPA-Enc}_{K1}(m),\ t := \text{MAC}_{K2}(c)\ )$

CCA Secure SKE

- $\text{CCA-Enc}_{K1,K2}(m) = (\ c := \text{CPA-Enc}_{K1}(m),\ t := \text{MAC}_{K2}(c)\ )$
  - CPA secure encryption: Block-cipher/CTR mode construction

CCA Secure SKE

- $\text{CCA-Enc}_{K1,K2}(m)$ = ($c := \text{CPA-Enc}_{K1}(m)$, $t := \text{MAC}_{K2}(c)$)
  - CPA secure encryption: Block-cipher/CTR mode construction
  - MAC: from a PRF or Block-Cipher (coming up)

CCA Secure SKE

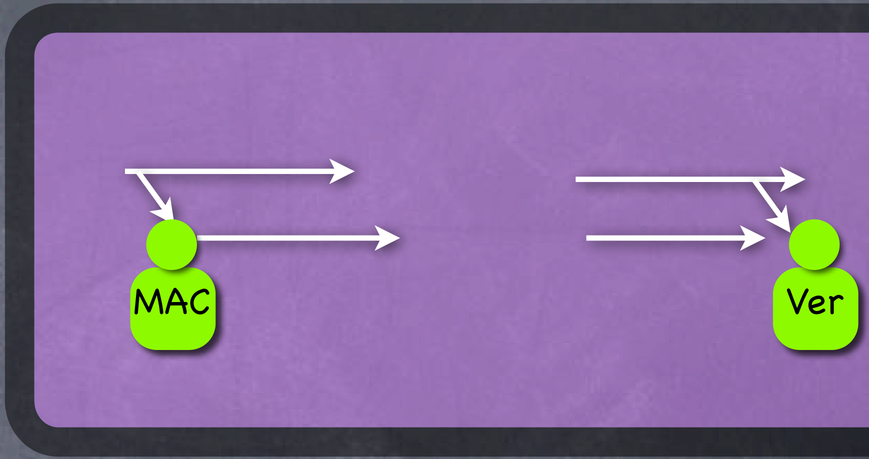
- CCA-Enc$_{K1,K2}$(m) = (c:= CPA-Enc$_{K1}$(m), t:= MAC$_{K2}$(c))
  - CPA secure encryption: Block-cipher/CTR mode construction
  - MAC: from a PRF or Block-Cipher (coming up)
- SKE in practice can just use Block-Ciphers (coming up)

CCA Secure SKE

- CCA-Enc$_{K1,K2}$(m) = (c:= CPA-Enc$_{K1}$(m), t:= MAC$_{K2}$(c))
  - CPA secure encryption: Block-cipher/CTR mode construction
  - MAC: from a PRF or Block-Cipher (coming up)
- SKE in practice can just use Block-Ciphers (coming up)
- In principle, constructions (less efficient) possible based on any One-Way Permutation or even any One-Way Function

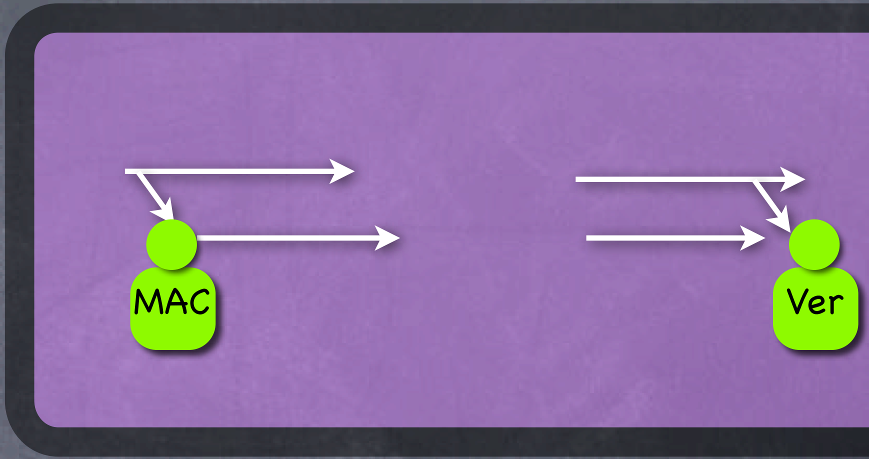
Making a MAC

One-time MAC



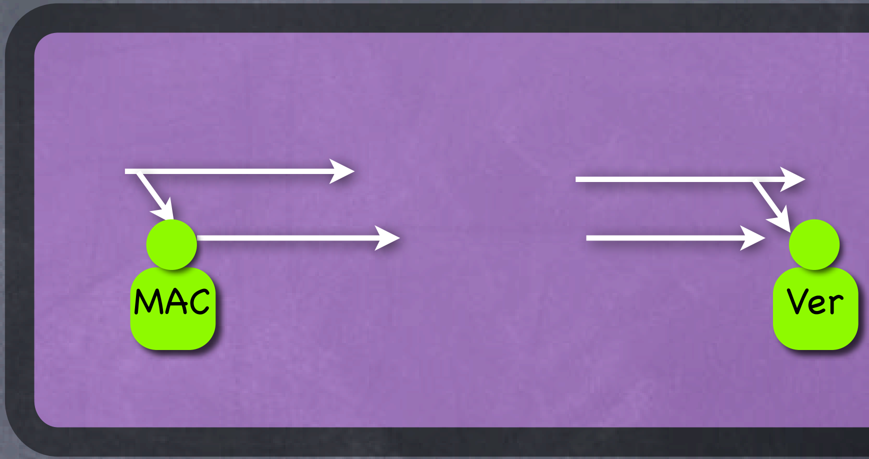
One-time MAC

- To sign a single n bit message



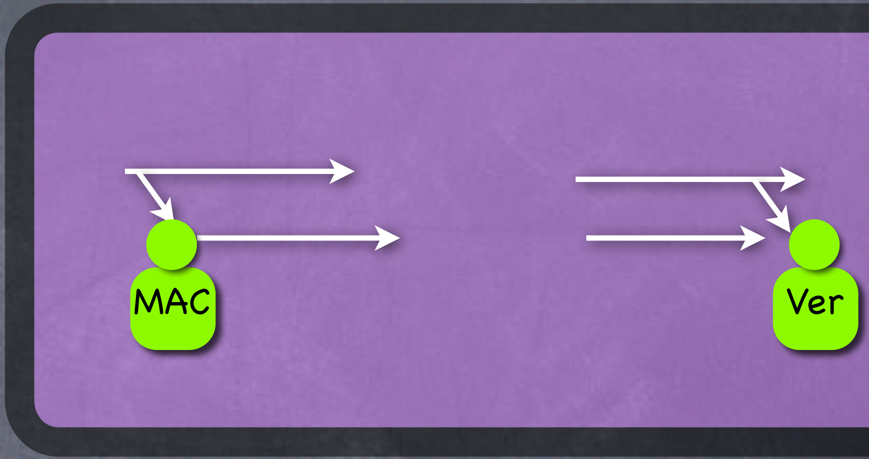
One-time MAC

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



One-time MAC

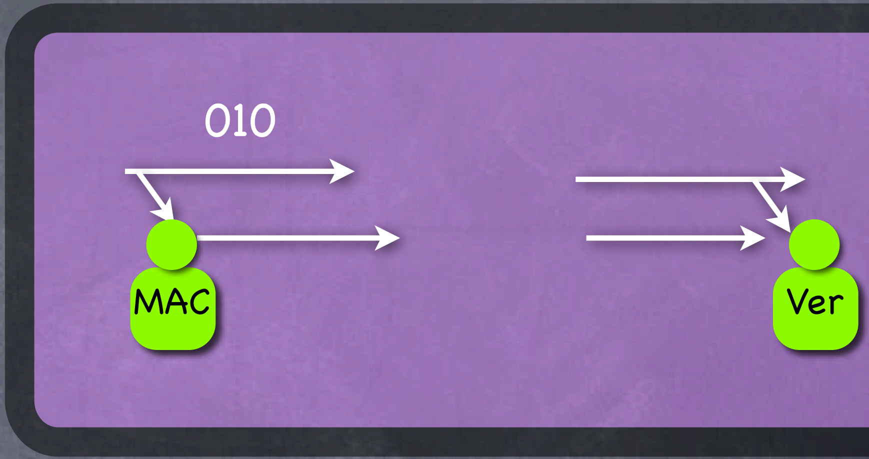
- To sign a single n bit message
- A simple (but inefficient) scheme
  - Shared secret key: 2n random strings (each k-bit long) $(r^i_0, r^i_1)_{i=1..n}$



r	r	r
r	r	r

One-time MAC

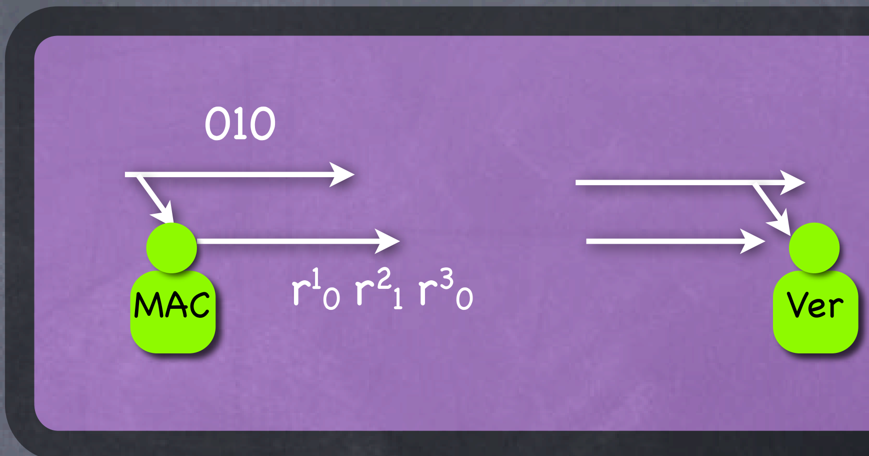
- To sign a single n bit message
- A simple (but inefficient) scheme
  - Shared secret key: 2n random strings (each k-bit long) $(r^i_0, r^i_1)_{i=1..n}$

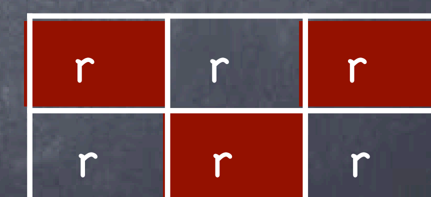


r	r	r
r	r	r

One-time MAC

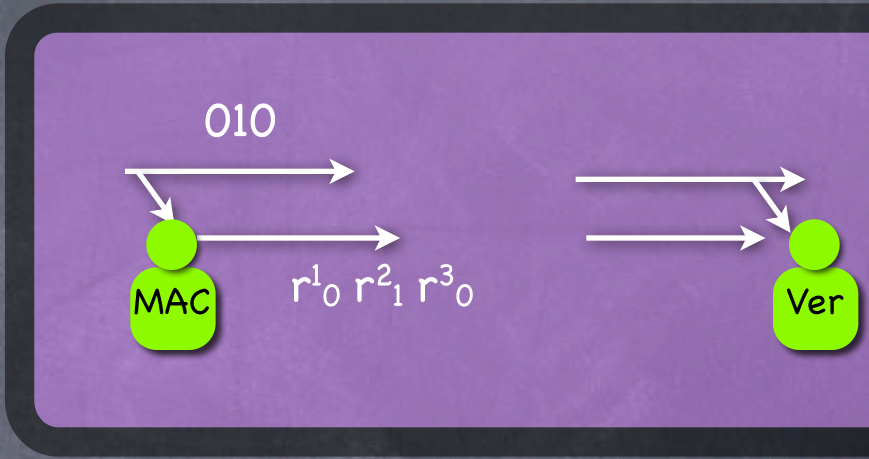
- To sign a single n bit message
- A simple (but inefficient) scheme
  - Shared secret key: 2n random strings (each k-bit long) $(r^i_0, r^i_1)_{i=1..n}$
  - Signature for $m_1 \dots m_n$ be $(r^i_{m_i})_{i=1..n}$

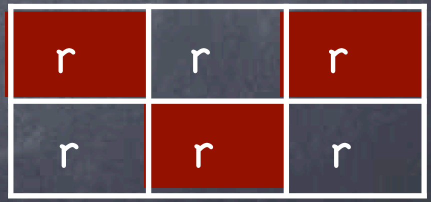




One-time MAC

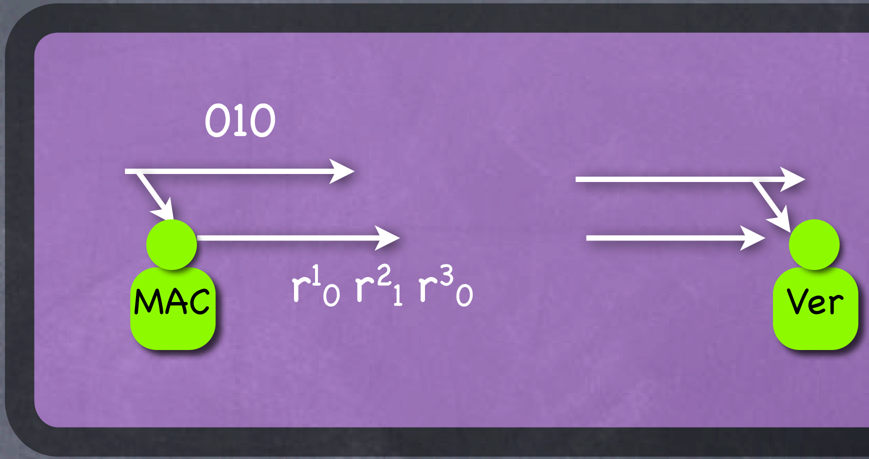
- To sign a single n bit message
- A simple (but inefficient) scheme
  - Shared secret key: 2n random strings (each k-bit long) $(r^i_0, r^i_1)_{i=1..n}$
  - Signature for $m_1 \ldots m_n$ be $(r^i_{m_i})_{i=1..n}$
  - Negligible probability that Eve can produce a signature on $m' \neq m$

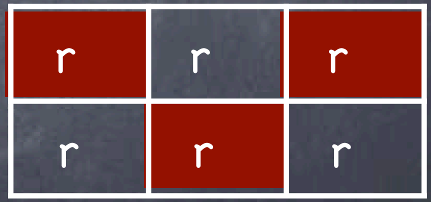




One-time MAC

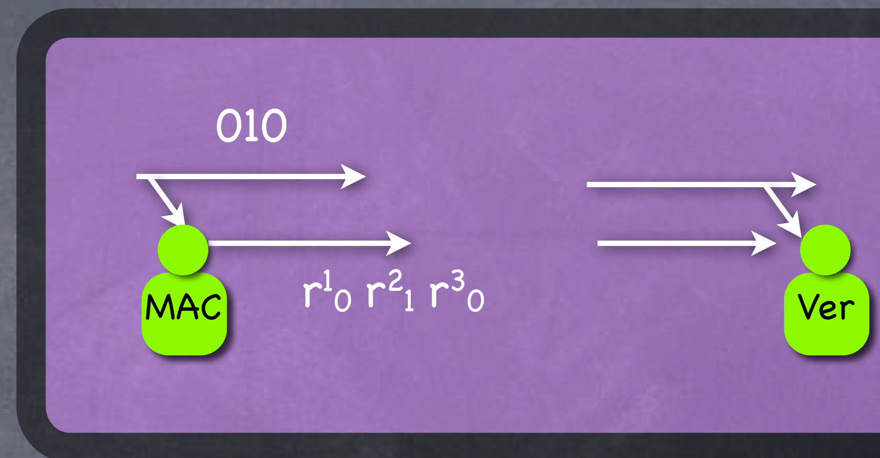
- To sign a single n bit message
- A simple (but inefficient) scheme
  - Shared secret key: 2n random strings (each k-bit long) $(r^i_0, r^i_1)_{i=1..n}$
  - Signature for $m_1 \ldots m_n$ be $(r^i_{m_i})_{i=1..n}$
  - Negligible probability that Eve can produce a signature on $m' \neq m$
- Doesn't require any computational restrictions on adversary!

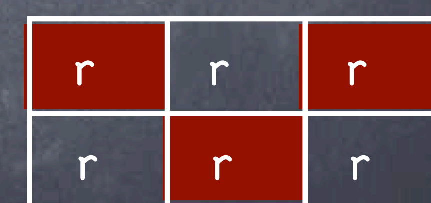




One-time MAC

- To sign a single n bit message
- A simple (but inefficient) scheme
  - Shared secret key: 2n random strings (each k-bit long) $(r^i_0, r^i_1)_{i=1..n}$
  - Signature for $m_1 \ldots m_n$ be $(r^i_{m_i})_{i=1..n}$
  - 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

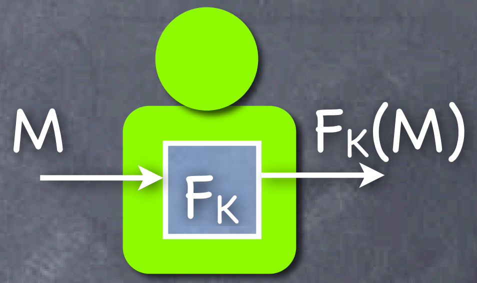
When Each Message is a Single Block

- 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

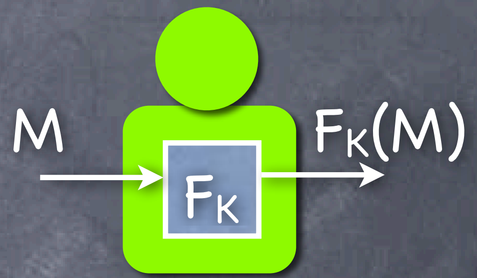
- PRF is a MAC!
  - $MAC_K(M) := F_K(M)$ where F is a PRF



(Multi-msg) MAC from PRF

When Each Message is a Single Block

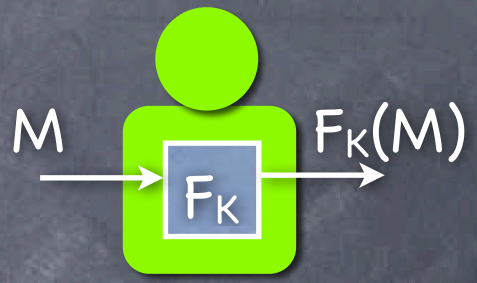
- PRF is a MAC!
    - $MAC_K(M) := F_K(M)$ where F is a PRF
    - $Ver_K(M,S) := 1$ iff $S=F_K(M)$



(Multi-msg) MAC from PRF

When Each Message is a Single Block

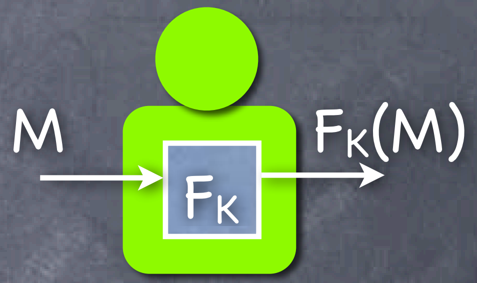
- PRF is a MAC!
  - $MAC_K(M) := F_K(M)$ where F is a PRF
  - $Ver_K(M,S) := 1$ iff $S=F_K(M)$
  - Output length of F_K should be big enough



(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

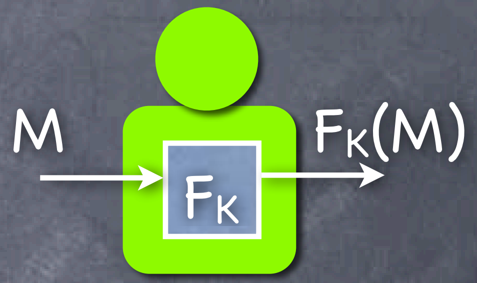


- If an adversary forges MAC with probability ε_{MAC}, then can break PRF with advantage $O(\varepsilon_{MAC} - 2^{-m(k)})$ (m(k) being the output length of the PRF) [How?]

(Multi-msg) MAC from PRF

When Each Message is a Single Block

- PRF is a MAC!
  - $MAC_K(M) := F_K(M)$ where F is a PRF
  - $Ver_K(M,S) := 1$ iff $S=F_K(M)$
  - Output length of F_K should be big enough



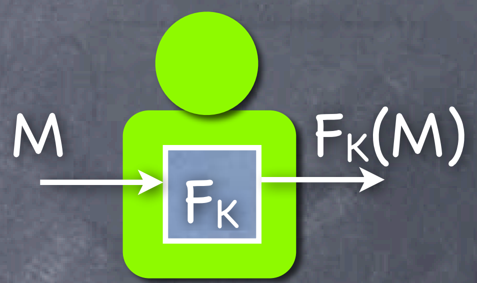
- If an adversary forges MAC with probability ε_{MAC}, then can break PRF with advantage $O(\varepsilon_{MAC} - 2^{-m(k)})$ (m(k) being the output length of the PRF) [How?]

Advantage in breaking a PRF F: diff in prob a 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!
  - $MAC_K(M) := F_K(M)$ 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 ε_{MAC}, then can break PRF with advantage $O(\varepsilon_{MAC} - 2^{-m(k)})$ (m(k) being the output length of the PRF) [How?]
  - If random function R used as MAC, then probability of forgery, $\varepsilon_{MAC^*} = 2^{-m(k)}$

Advantage in breaking a PRF F: 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

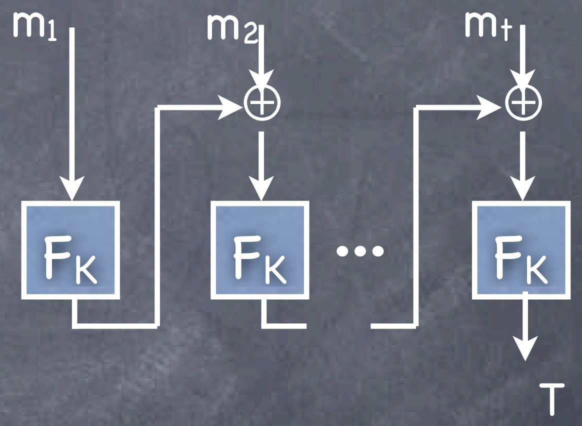
CBC-MAC

CBC-MAC

- PRF domain extension: Chaining the blocks

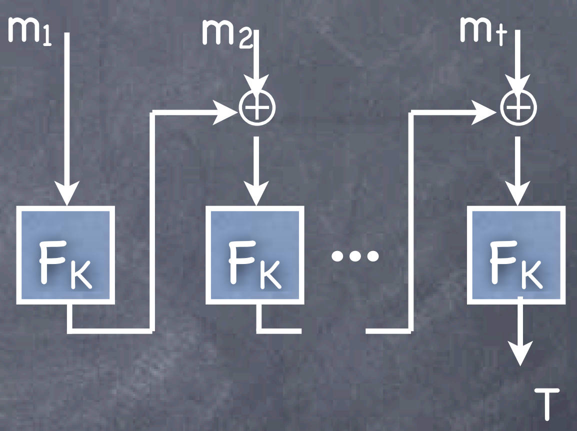
CBC-MAC

- PRF domain extension: Chaining the blocks



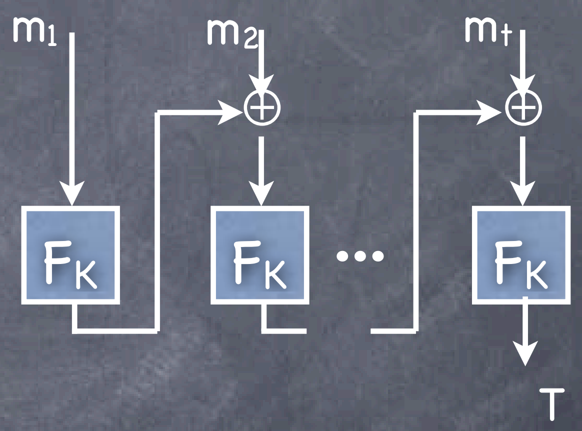
CBC-MAC

- PRF domain extension: Chaining the blocks
  - cf. CBC mode for encryption (which is not a MAC!)



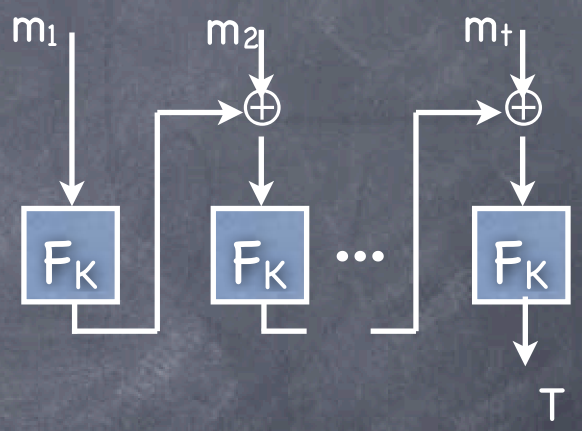
CBC-MAC

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



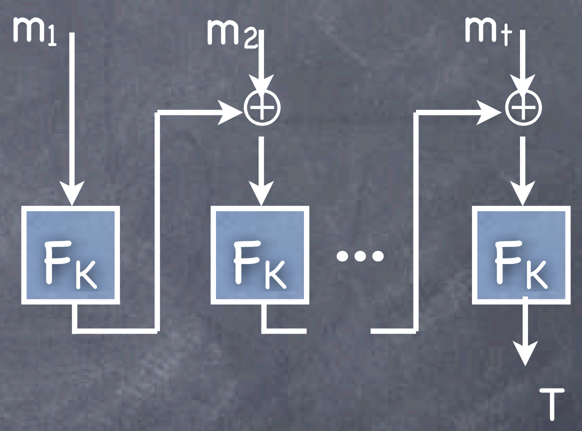
CBC-MAC

- 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



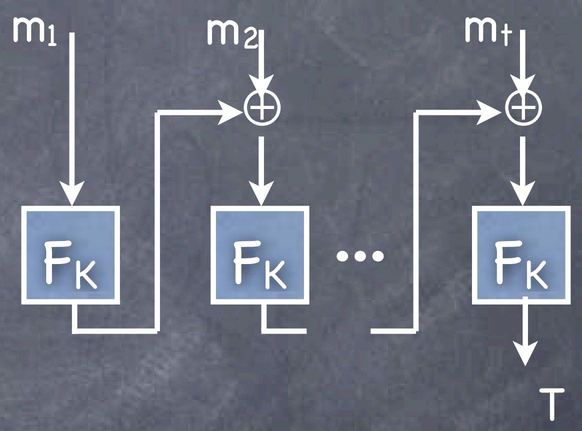
CBC-MAC

- 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
  - <u>EMAC:</u> 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.
  - <u>CMAC:</u> 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. ← NIST Recommendation. 2005
- Later: Hash-based HMAC used in TLS and IPSec ← 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 <u>block-cipher</u>

Stream Ciphers

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

- 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

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

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

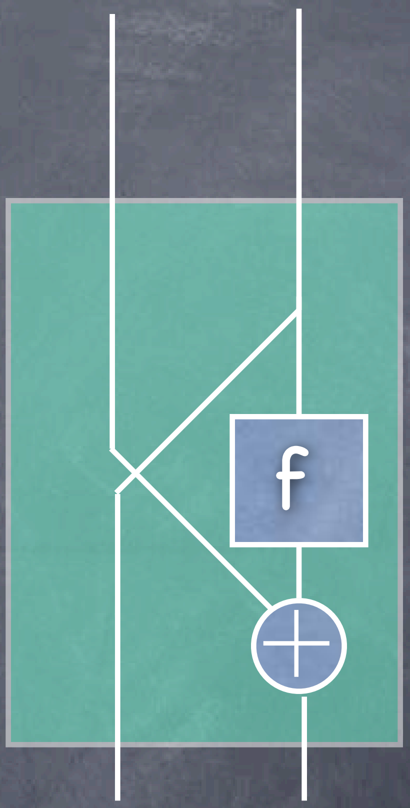
- Building a permutation from a (block) function

Feistel Network

- Building a permutation from a (block) function
  - Let f: $\{0,1\}^m \rightarrow \{0,1\}^m$ be an arbitrary function

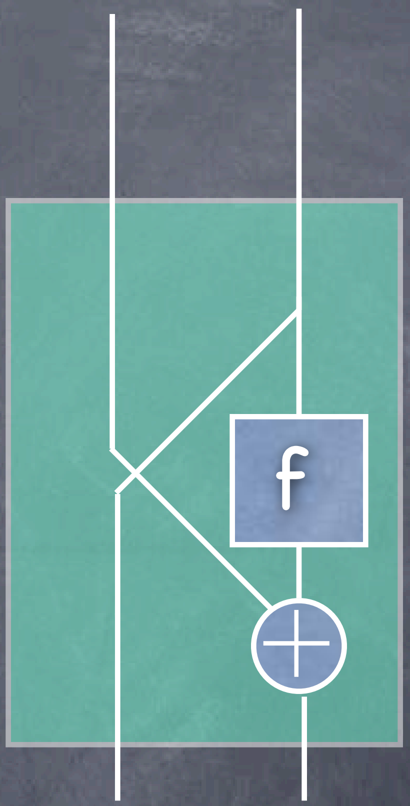
Feistel Network

- 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 $F_f(x,y) = (\ y,\ x \oplus f(y)\ )$



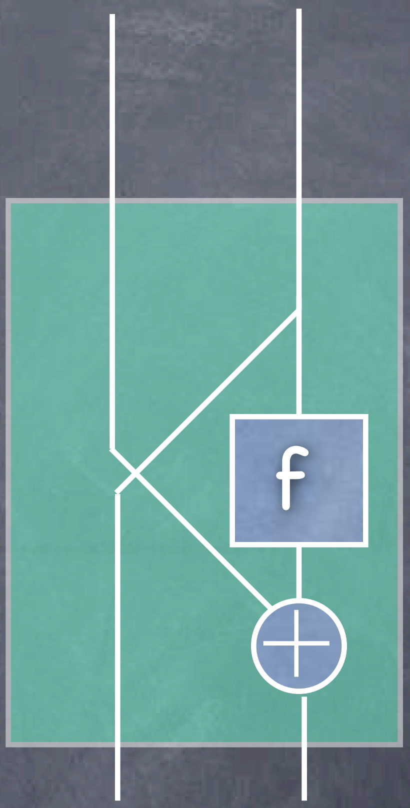
Feistel Network

- Building a permutation from a (block) function
  - Let f: $\{0,1\}^m \to \{0,1\}^m$ be an arbitrary function
  - F_f: $\{0,1\}^{2m} \to \{0,1\}^{2m}$ defined as $F_f(x,y) = (\ y,\ x \oplus f(y)\ )$
    - F_f is a permutation (Why?)



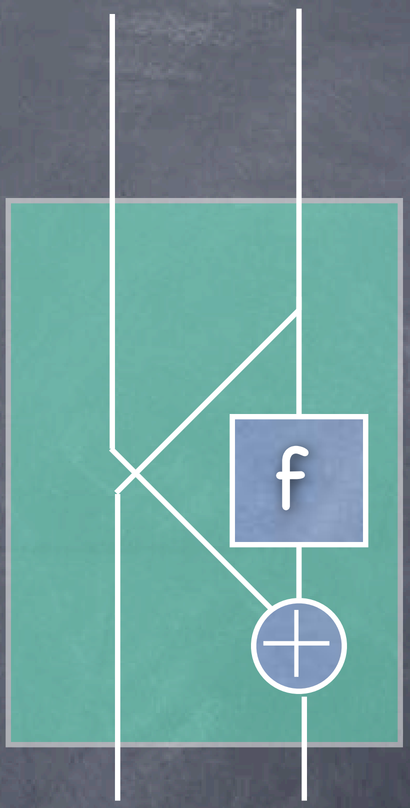
Feistel Network

- 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?)



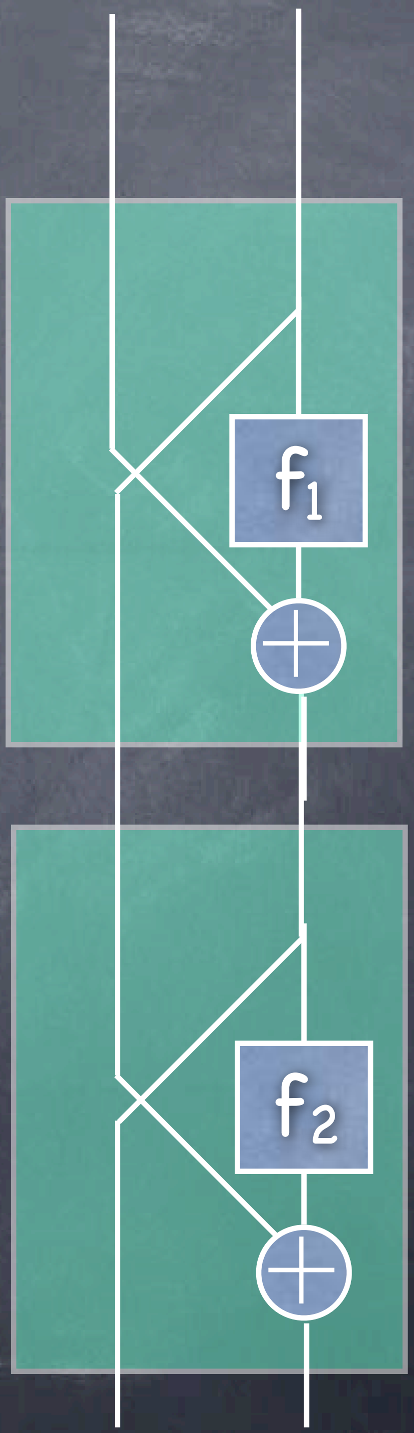
Feistel Network

- 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 $F_{f1\ldots ft}$



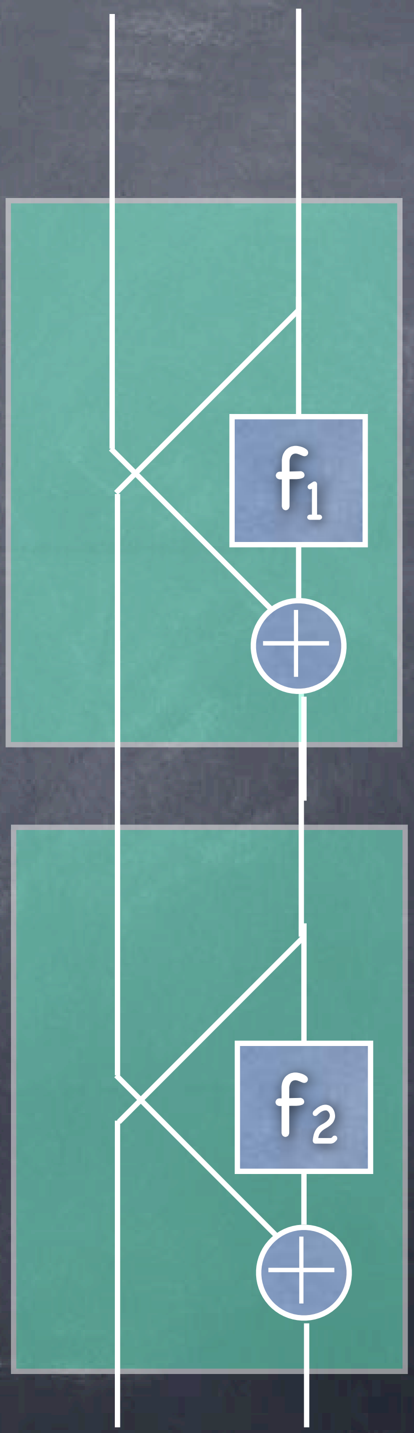
Feistel Network

- 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 $F_{f1 \ldots ft}$



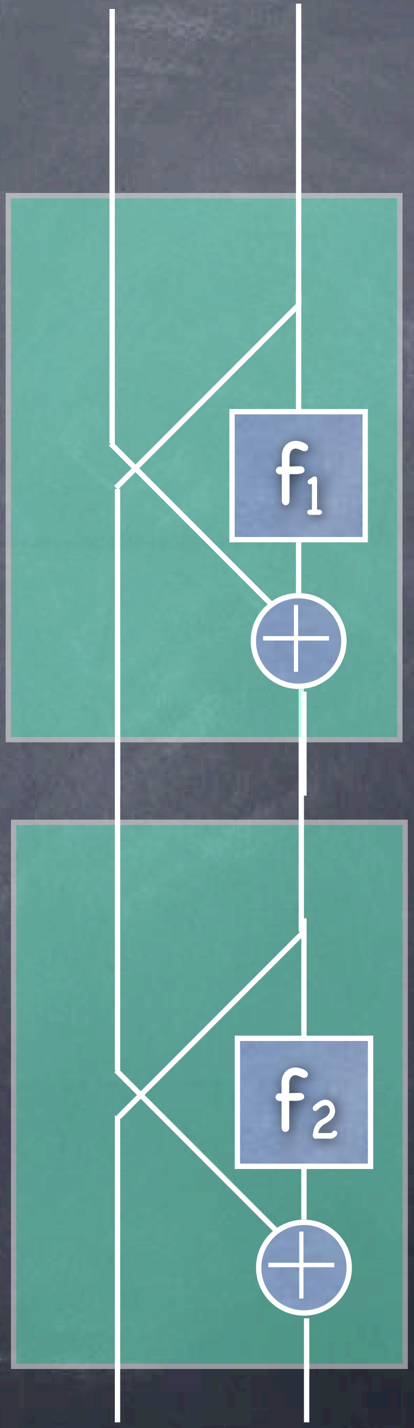
Feistel Network

- 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 $F_{f1\ldots ft}$
    - Still a permutation from $\{0,1\}^{2m}$ to $\{0,1\}^{2m}$



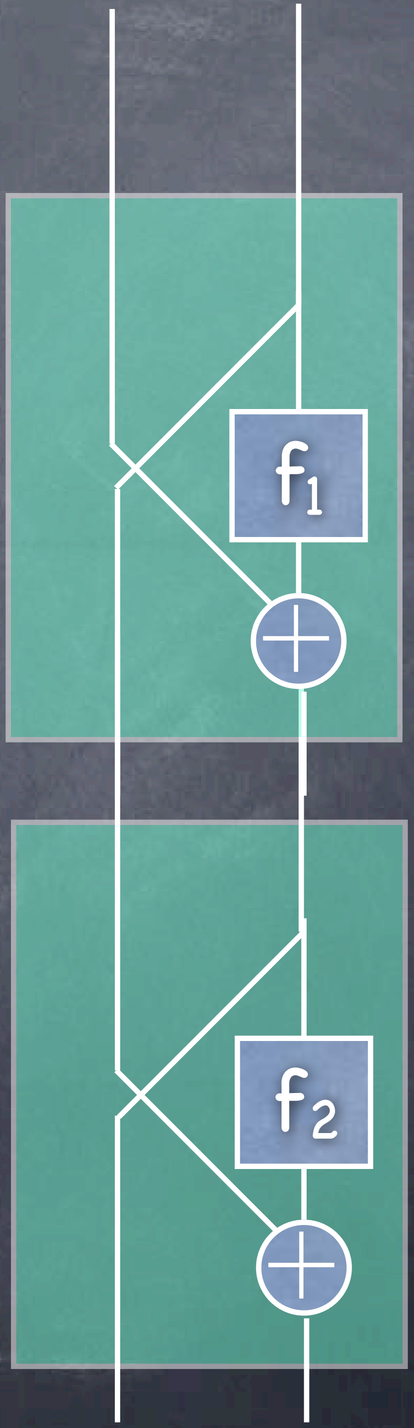
Feistel Network

- 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 $F_{f1\ldots ft}$
    - 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 $F_{f1\ldots ft}$
    - 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

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

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"

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
    - "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

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

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
    - "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

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

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
    - "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

NIST Standard. 2001

- Advanced Encryption Standard (AES)

AES Block Cipher

NIST Standard. 2001

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

AES Block Cipher

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)

AES Block Cipher

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-Permute" instead of Feistel networks

AES Block Cipher

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-Permute" instead of Feistel networks
  - Has some algebraic structure

AES Block Cipher

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-Permute" instead of Feistel networks
  - Has some algebraic structure
    - Operations in a vector space over the field $GF(2^8)$

AES Block Cipher

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-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"?

AES Block Cipher

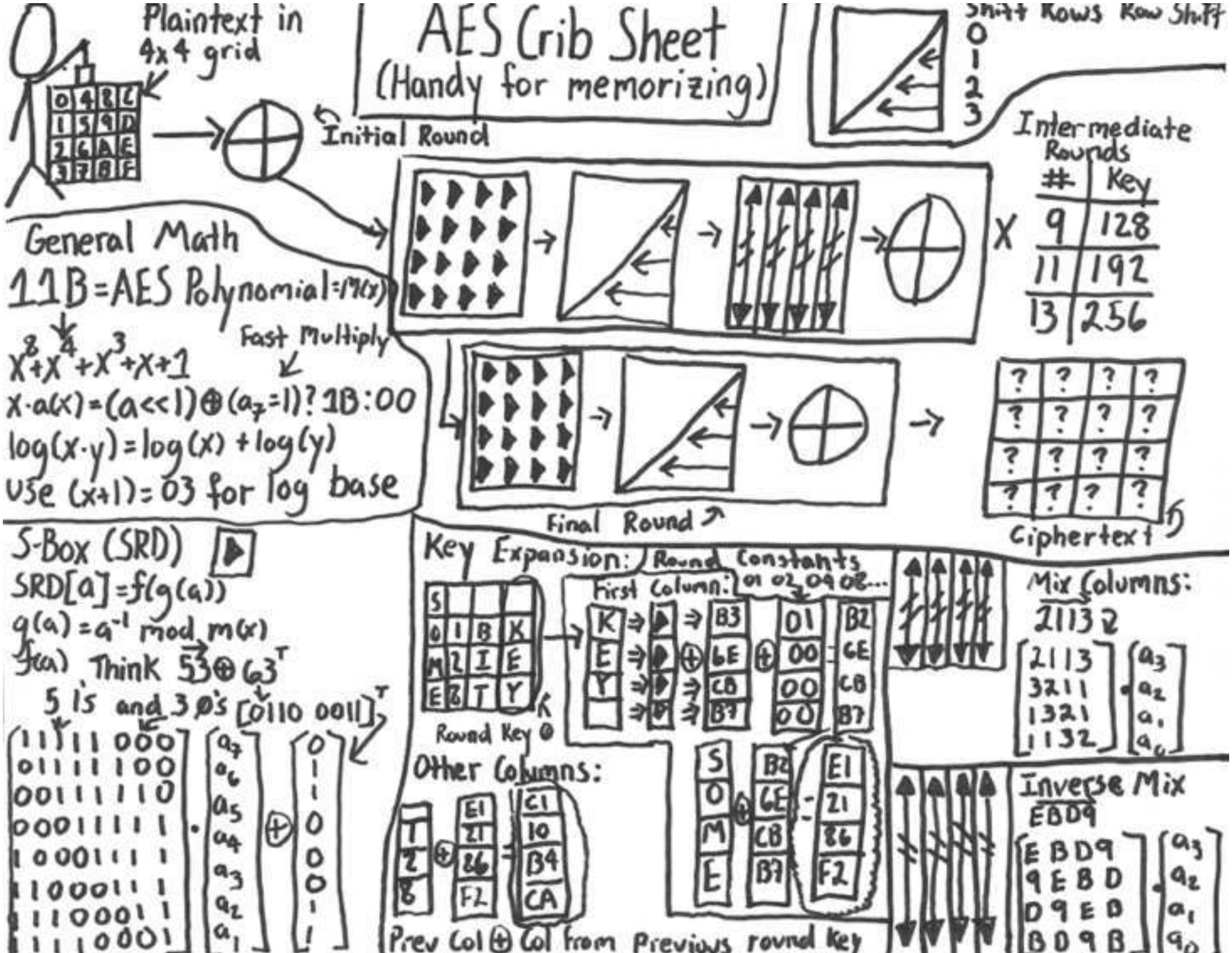
NIST Standard. 2001

- 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-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)

AES Block Cipher

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-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



By Jeff Moser (http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html)

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