

Symmetric-Key Encryption: constructions

Lecture 5
PRF, Block Cipher

RECALL

PRG from One-Way Permutations

- One-bit stretch PRG, $G_k: \{0,1\}^k \rightarrow \{0,1\}^{k+1}$



- Increasing the stretch

- Can use part of the PRG output as a new seed



- If the intermediate seeds are never output, can keep stretching on demand (for any "polynomial length")

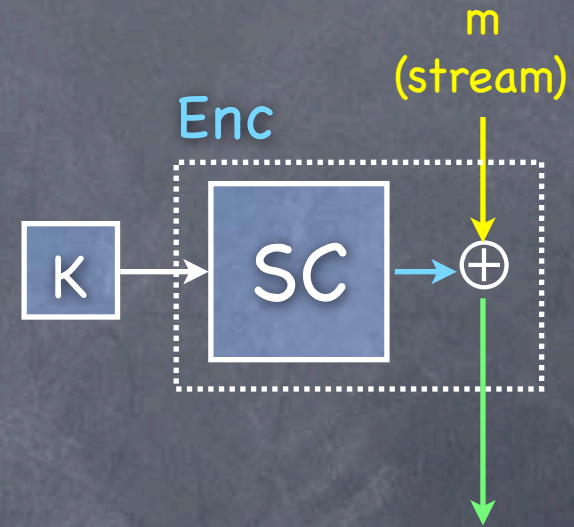
- A stream cipher



RECALL

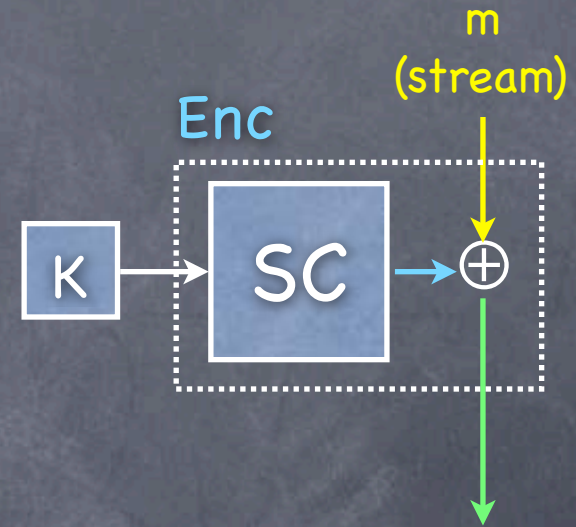
One-time CPA-secure SKE with a Stream-Cipher

- One-time Encryption with a stream-cipher:
 - Generate a one-time pad from a short seed
 - Can share just the seed as the key
 - Mask message with the pseudorandom pad
- Decryption is symmetric: plaintext & ciphertext interchanged
- SC can spit out bits on demand, so the message can arrive bit by bit, and the length of the message doesn't have to be a priori fixed
- Security: indistinguishability from using a truly random pad



RECALL

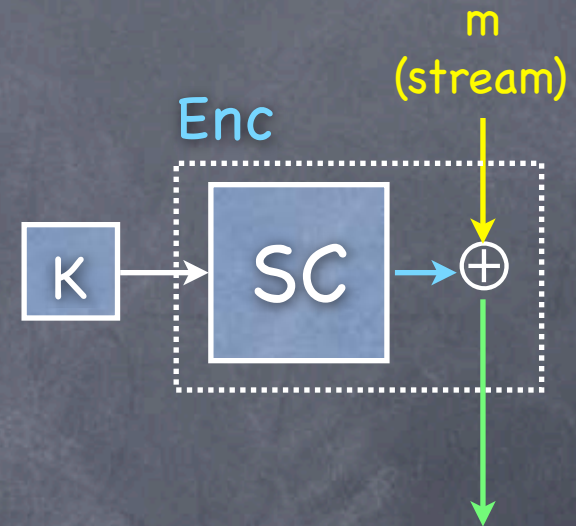
One-time CPA-secure SKE with a Stream-Cipher



RECALL

One-time CPA-secure SKE with a Stream-Cipher

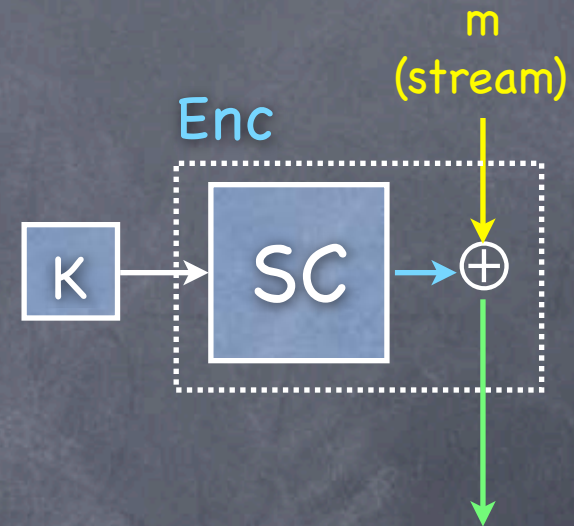
- In IDEAL experiment, consider simulator that uses a truly random string as the ciphertext



RECALL

One-time CPA-secure SKE with a Stream-Cipher

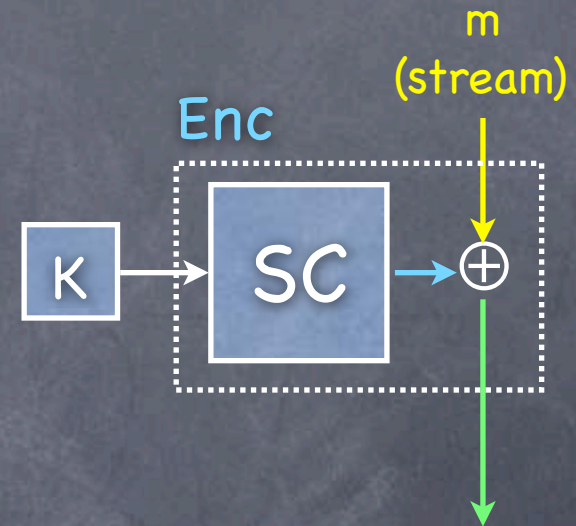
- In IDEAL experiment, consider simulator that uses a truly random string as the ciphertext
- To show $REAL \approx IDEAL$



RECALL

One-time CPA-secure SKE with a Stream-Cipher

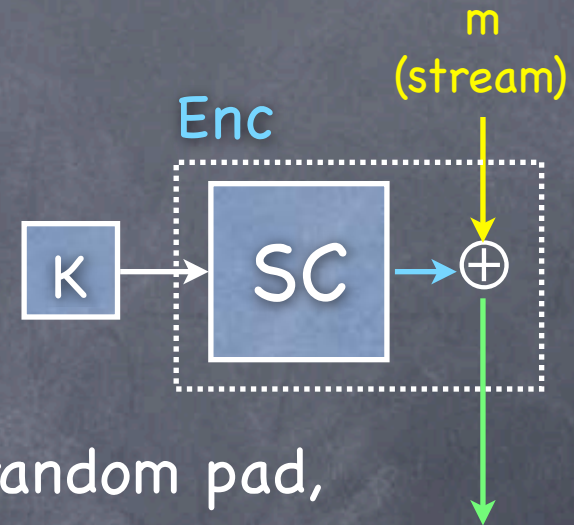
- In IDEAL experiment, consider simulator that uses a truly random string as the ciphertext
- To show $REAL \approx IDEAL$
- Consider an intermediate world, HYBRID:



RECALL

One-time CPA-secure SKE with a Stream-Cipher

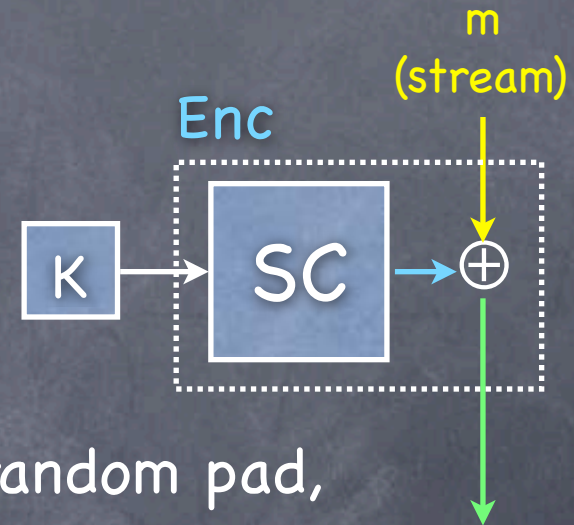
- In IDEAL experiment, consider simulator that uses a truly random string as the ciphertext
- To show $REAL \approx IDEAL$
- Consider an intermediate world, HYBRID:
 - Like REAL, but Enc/Dec use a (long) truly random pad, instead of the output from the stream-cipher



RECALL

One-time CPA-secure SKE with a Stream-Cipher

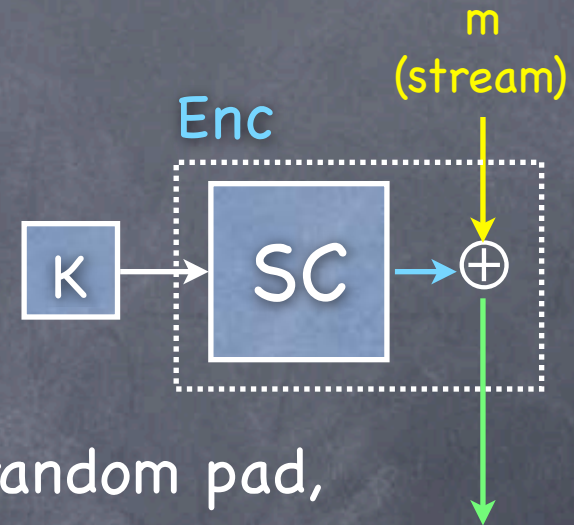
- In IDEAL experiment, consider simulator that uses a truly random string as the ciphertext
- To show $REAL \approx IDEAL$
- Consider an intermediate world, HYBRID:
 - Like REAL, but Enc/Dec use a (long) truly random pad, instead of the output from the stream-cipher
 - $HYBRID = IDEAL$ (recall perfect security of one-time pad)



RECALL

One-time CPA-secure SKE with a Stream-Cipher

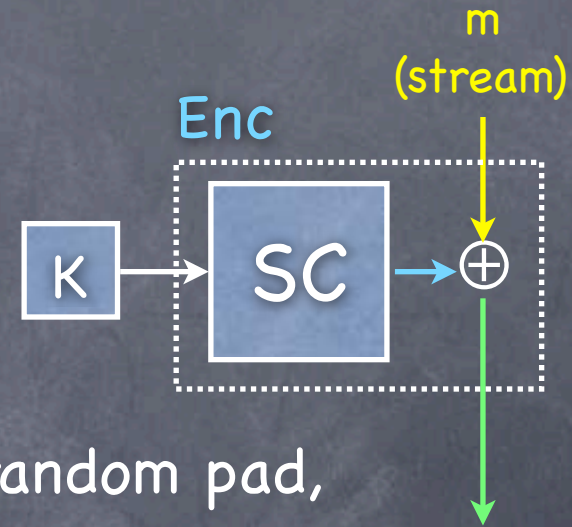
- In IDEAL experiment, consider simulator that uses a truly random string as the ciphertext
- To show $REAL \approx IDEAL$
- Consider an intermediate world, HYBRID:
 - Like REAL, but Enc/Dec use a (long) truly random pad, instead of the output from the stream-cipher
 - $HYBRID = IDEAL$ (recall perfect security of one-time pad)
 - Claim: $REAL \approx HYBRID$



RECALL

One-time CPA-secure SKE with a Stream-Cipher

- In IDEAL experiment, consider simulator that uses a truly random string as the ciphertext
- To show $REAL \approx IDEAL$
- Consider an intermediate world, HYBRID:
 - Like REAL, but Enc/Dec use a (long) truly random pad, instead of the output from the stream-cipher
 - $HYBRID = IDEAL$ (recall perfect security of one-time pad)
 - Claim: $REAL \approx HYBRID$
 - Consider the experiments as a system that accepts the pad from outside ($R' = SC(K)$ for a random K , or truly random R) and outputs the environment's output. This system is PPT, and so can't distinguish pseudorandom from random.



Beyond One-Time?

Beyond One-Time?

- Need to make sure same part of the one-time pad is never reused

Beyond One-Time?

- Need to make sure same part of the one-time pad is never reused
 - Sender and receiver will need to maintain state and stay in sync (indicating how much of the pad has already been used)

Beyond One-Time?

- Need to make sure same part of the one-time pad is never reused
 - Sender and receiver will need to maintain state and stay in sync (indicating how much of the pad has already been used)
 - Or only sender maintains the index, but sends it to the receiver. Then receiver will need to run the stream-cipher to get to that index.

Beyond One-Time?

- Need to make sure same part of the one-time pad is never reused
 - Sender and receiver will need to maintain state and stay in sync (indicating how much of the pad has already been used)
 - Or only sender maintains the index, but sends it to the receiver. Then receiver will need to run the stream-cipher to get to that index.
 - A PRG with direct access to any part of the output stream?

Beyond One-Time?

- Need to make sure same part of the one-time pad is never reused
 - Sender and receiver will need to maintain state and stay in sync (indicating how much of the pad has already been used)
 - Or only sender maintains the index, but sends it to the receiver. Then receiver will need to run the stream-cipher to get to that index.
 - A PRG with direct access to any part of the output stream?
- Pseudo Random Function (PRF)

Pseudorandom Function (PRF)

Pseudorandom Function (PRF)

- A compact representation of an exponentially long (pseudorandom) string

Pseudorandom Function (PRF)

- A compact representation of an exponentially long (pseudorandom) string
 - Allows “random-access” (instead of just sequential access)

Pseudorandom Function (PRF)

- A compact representation of an exponentially long (pseudorandom) string
 - Allows “random-access” (instead of just sequential access)
 - A function $F(s;i)$ outputs the i^{th} block of the pseudorandom string corresponding to seed s

Pseudorandom Function (PRF)

- A compact representation of an exponentially long (pseudorandom) string
 - Allows “random-access” (instead of just sequential access)
 - A function $F(s;i)$ outputs the i^{th} block of the pseudorandom string corresponding to seed s
 - Exponentially many blocks (i.e., large domain for i)

Pseudorandom Function (PRF)

- A compact representation of an exponentially long (pseudorandom) string
 - Allows “random-access” (instead of just sequential access)
 - A function $F(s;i)$ outputs the i^{th} block of the pseudorandom string corresponding to seed s
 - Exponentially many blocks (i.e., large domain for i)
- Pseudorandom Function

Pseudorandom Function (PRF)

- A compact representation of an exponentially long (pseudorandom) string
 - Allows “random-access” (instead of just sequential access)
 - A function $F(s;i)$ outputs the i^{th} block of the pseudorandom string corresponding to seed s
 - Exponentially many blocks (i.e., large domain for i)
- Pseudorandom Function
 - Need to define pseudorandomness for a function (not a string)

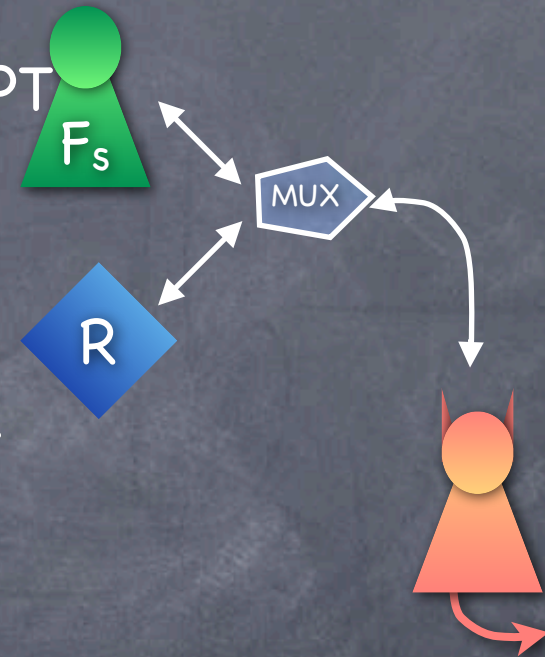
Pseudorandom Function (PRF)

Pseudorandom Function (PRF)

- $F: \{0,1\}^k \times \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}$ is a PRF if all PPT adversaries have negligible advantage in the PRF experiment

Pseudorandom Function (PRF)

- $F: \{0,1\}^k \times \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}$ is a PRF if all PPT adversaries have negligible advantage in the PRF experiment

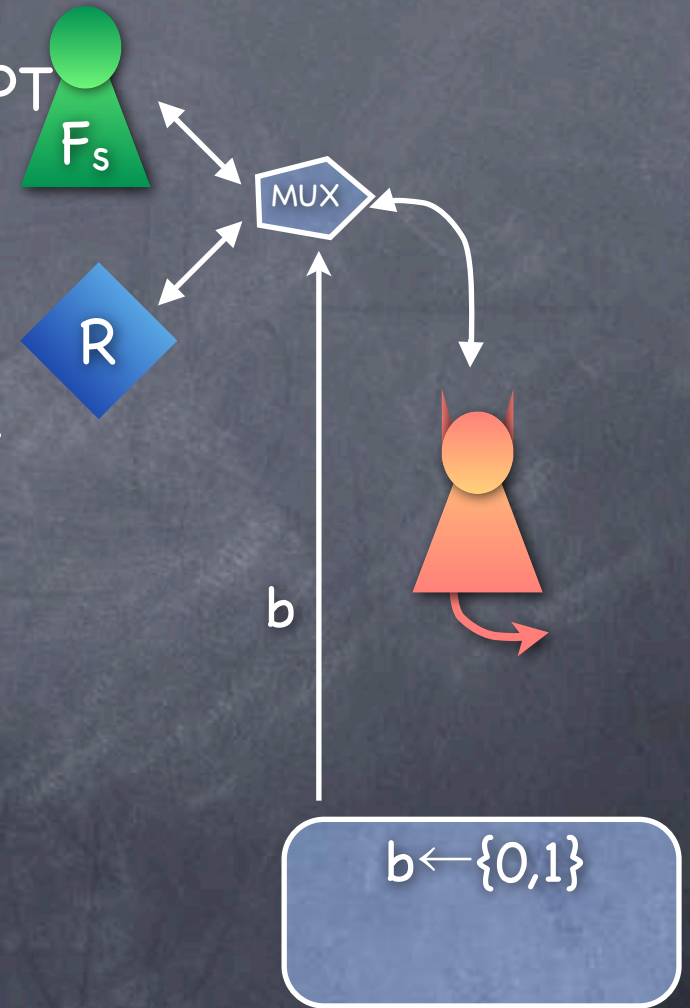


- Adversary given oracle access to either F with a random seed, or a random function $R: \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}$. Needs to guess which.



Pseudorandom Function (PRF)

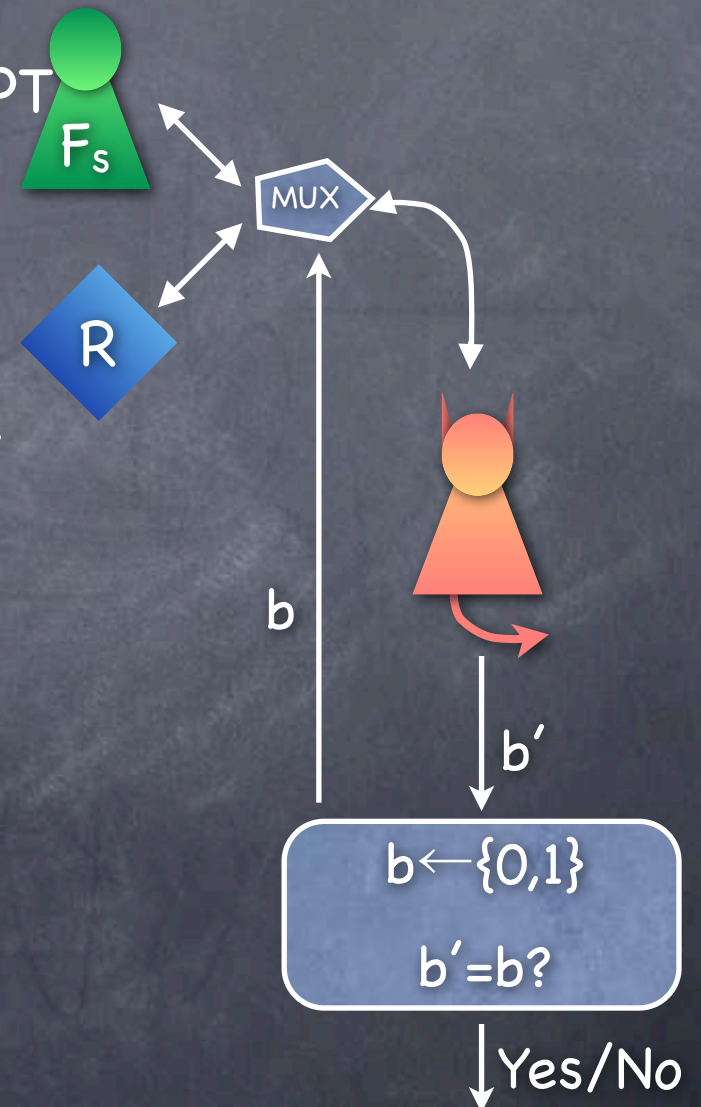
- $F: \{0,1\}^k \times \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}$ is a PRF if all PPT adversaries have negligible advantage in the PRF experiment



- Adversary given oracle access to either F with a random seed, or a random function $R: \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}$. Needs to guess which.

Pseudorandom Function (PRF)

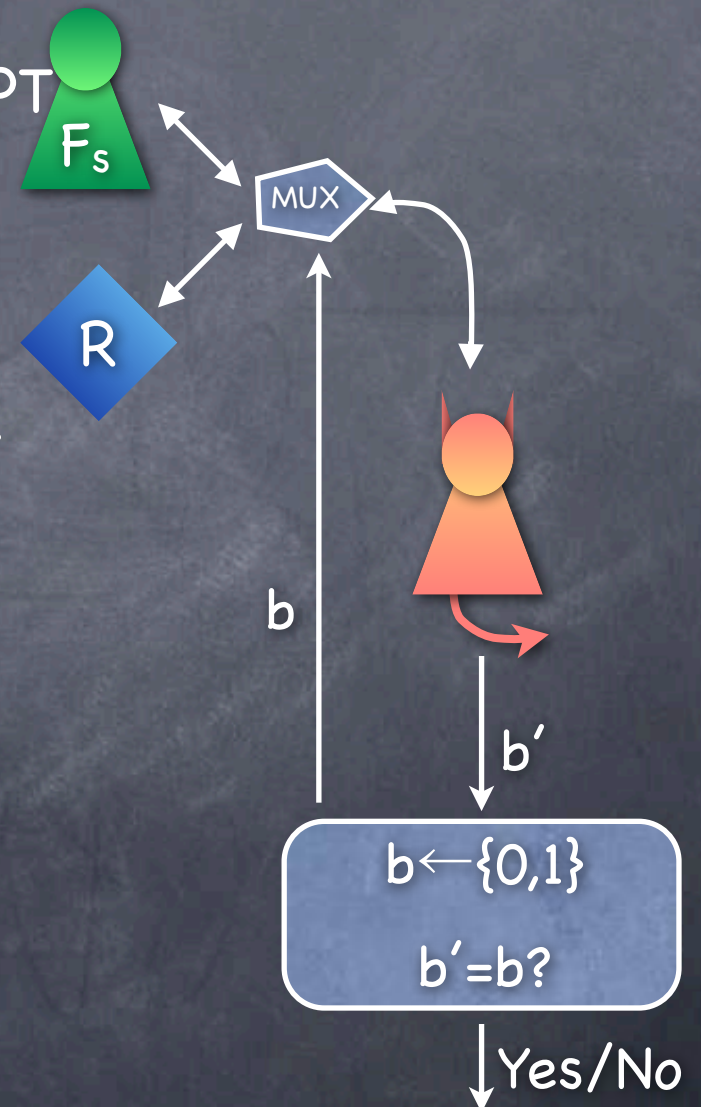
- $F: \{0,1\}^k \times \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}$ is a PRF if all PPT adversaries have negligible advantage in the PRF experiment



- Adversary given oracle access to either F with a random seed, or a random function $R: \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}$. Needs to guess which.

Pseudorandom Function (PRF)

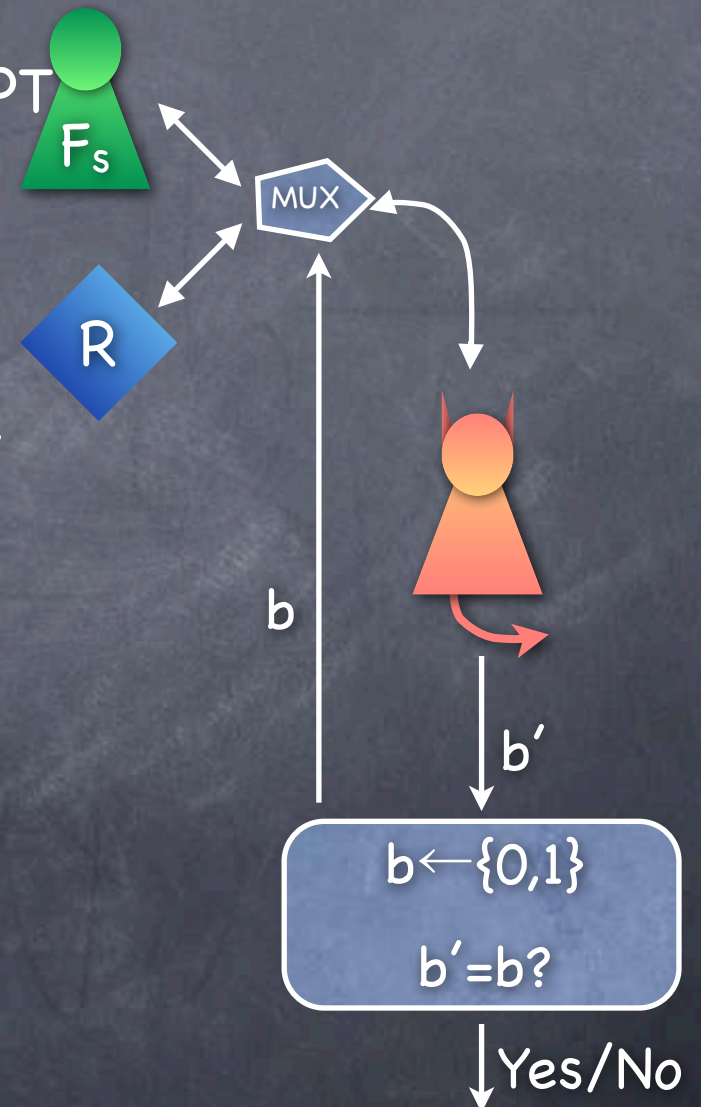
- $F: \{0,1\}^k \times \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}$ is a PRF if all PPT adversaries have negligible advantage in the PRF experiment



- Adversary given oracle access to either F with a random seed, or a random function $R: \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}$. Needs to guess which.
- Note: Only 2^k seeds for F

Pseudorandom Function (PRF)

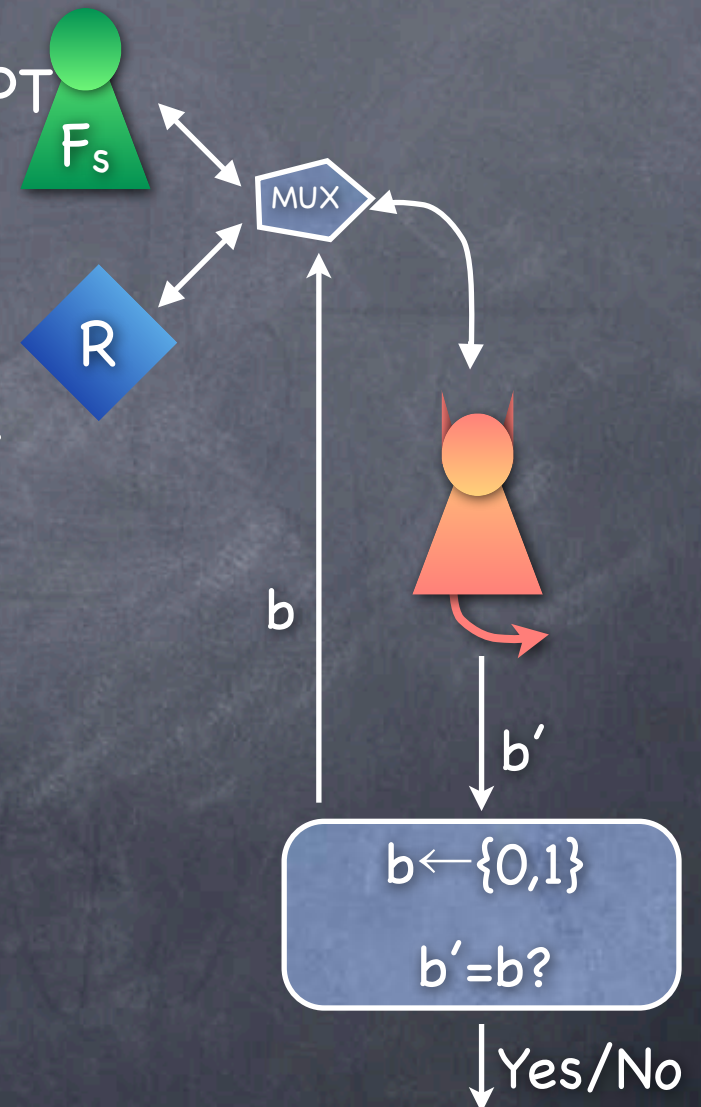
- $F: \{0,1\}^k \times \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}$ is a PRF if all PPT adversaries have negligible advantage in the PRF experiment



- Adversary given oracle access to either F with a random seed, or a random function $R: \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}$. Needs to guess which.
- Note: Only 2^k seeds for F
 - But $2^{(n2^m)}$ functions R

Pseudorandom Function (PRF)

- $F: \{0,1\}^k \times \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}$ is a PRF if all PPT adversaries have negligible advantage in the PRF experiment



- Adversary given oracle access to either F with a random seed, or a random function $R: \{0,1\}^{m(k)} \rightarrow \{0,1\}^{n(k)}$. Needs to guess which.
- Note: Only 2^k seeds for F
 - But $2^{(n2^m)}$ functions R
- PRF stretches k bits to $n2^m$ bits

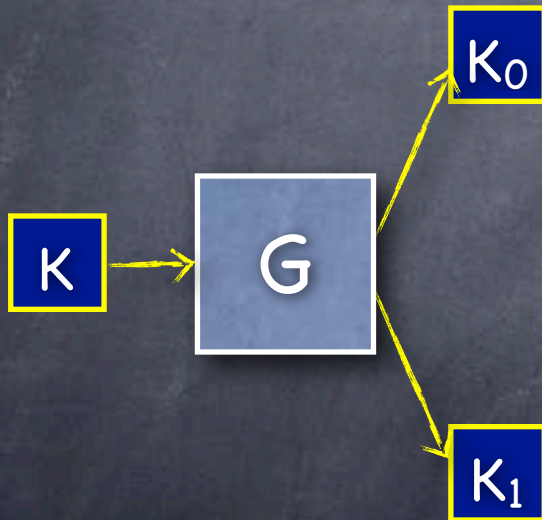
Pseudorandom Function (PRF)

Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG

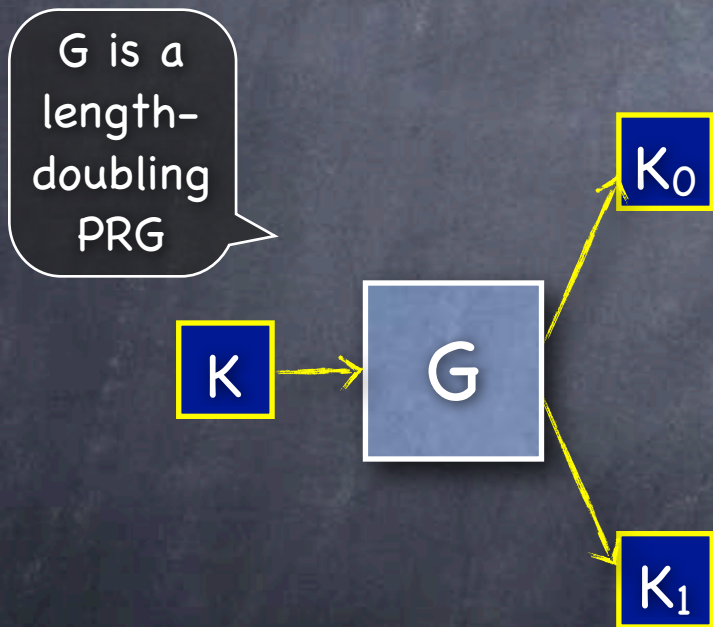
Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG



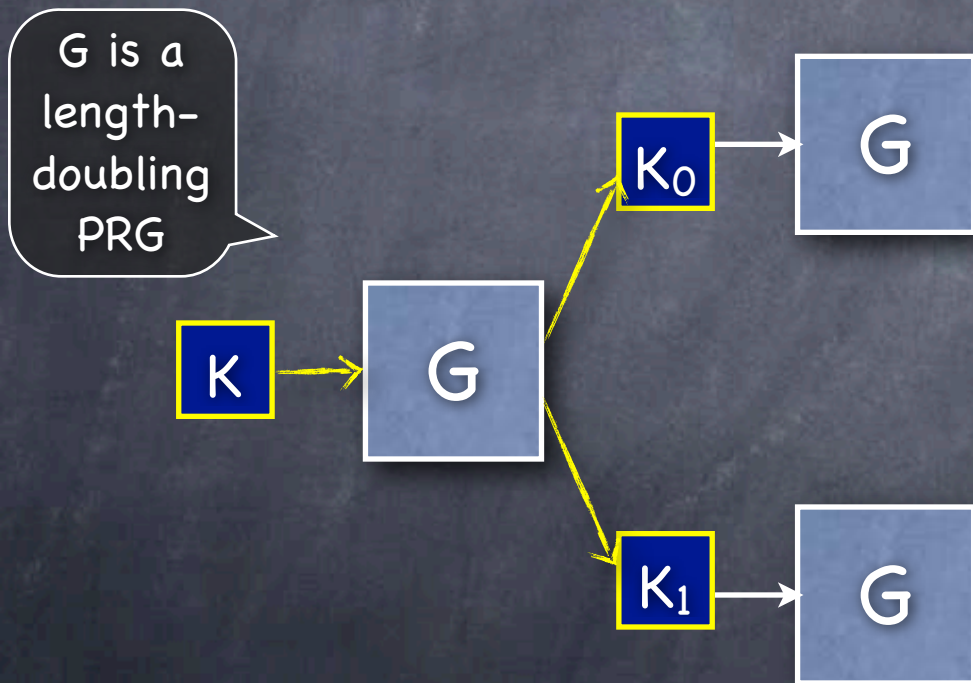
Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG



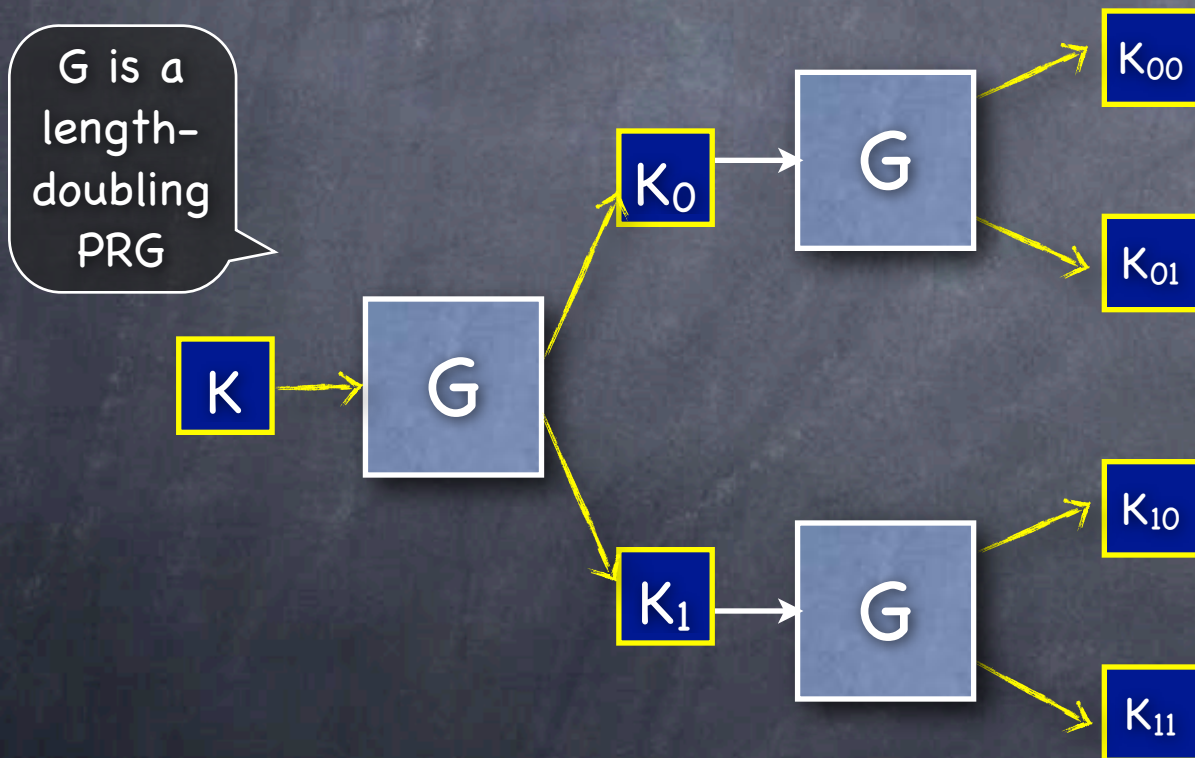
Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG



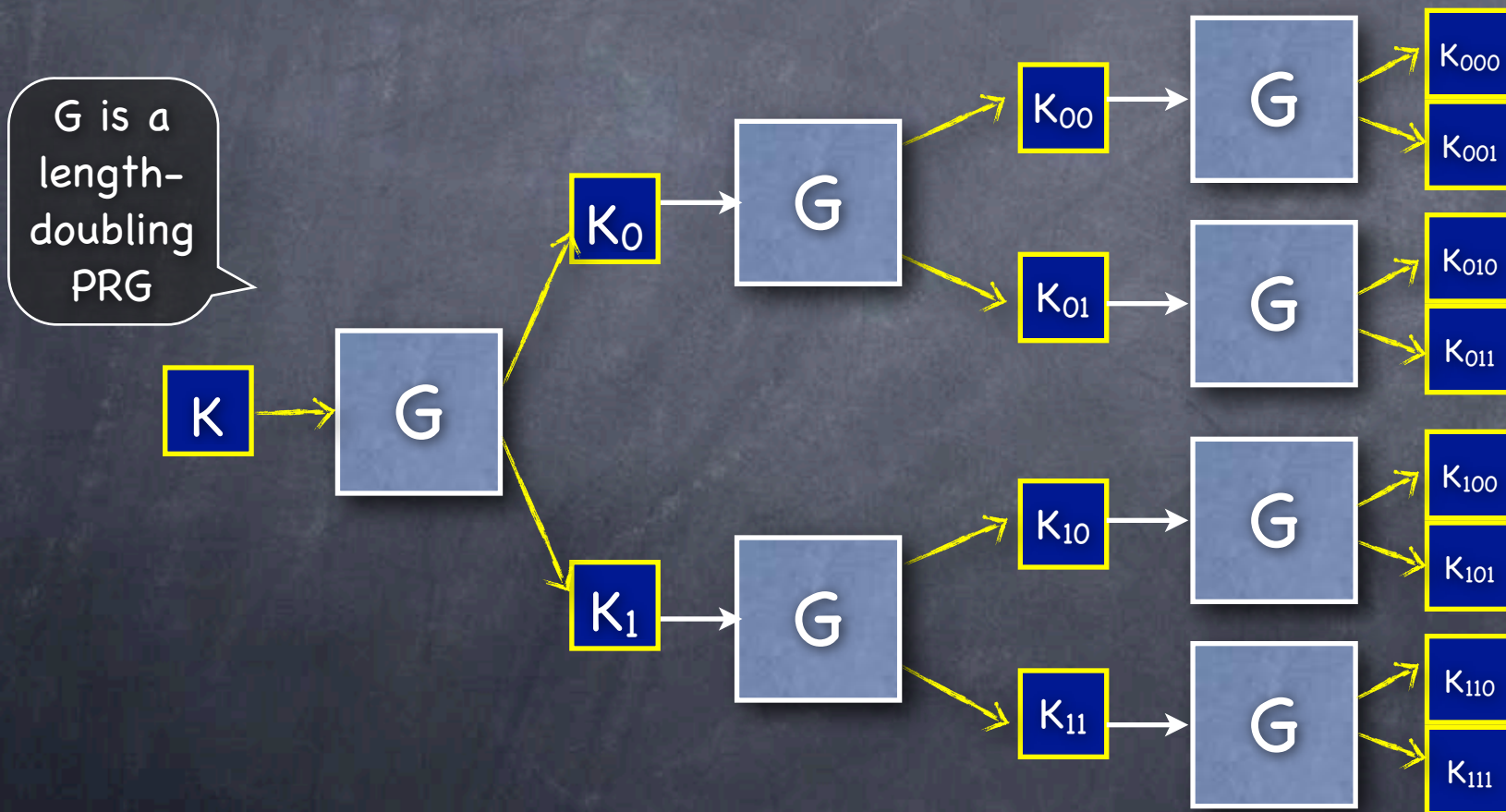
Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG



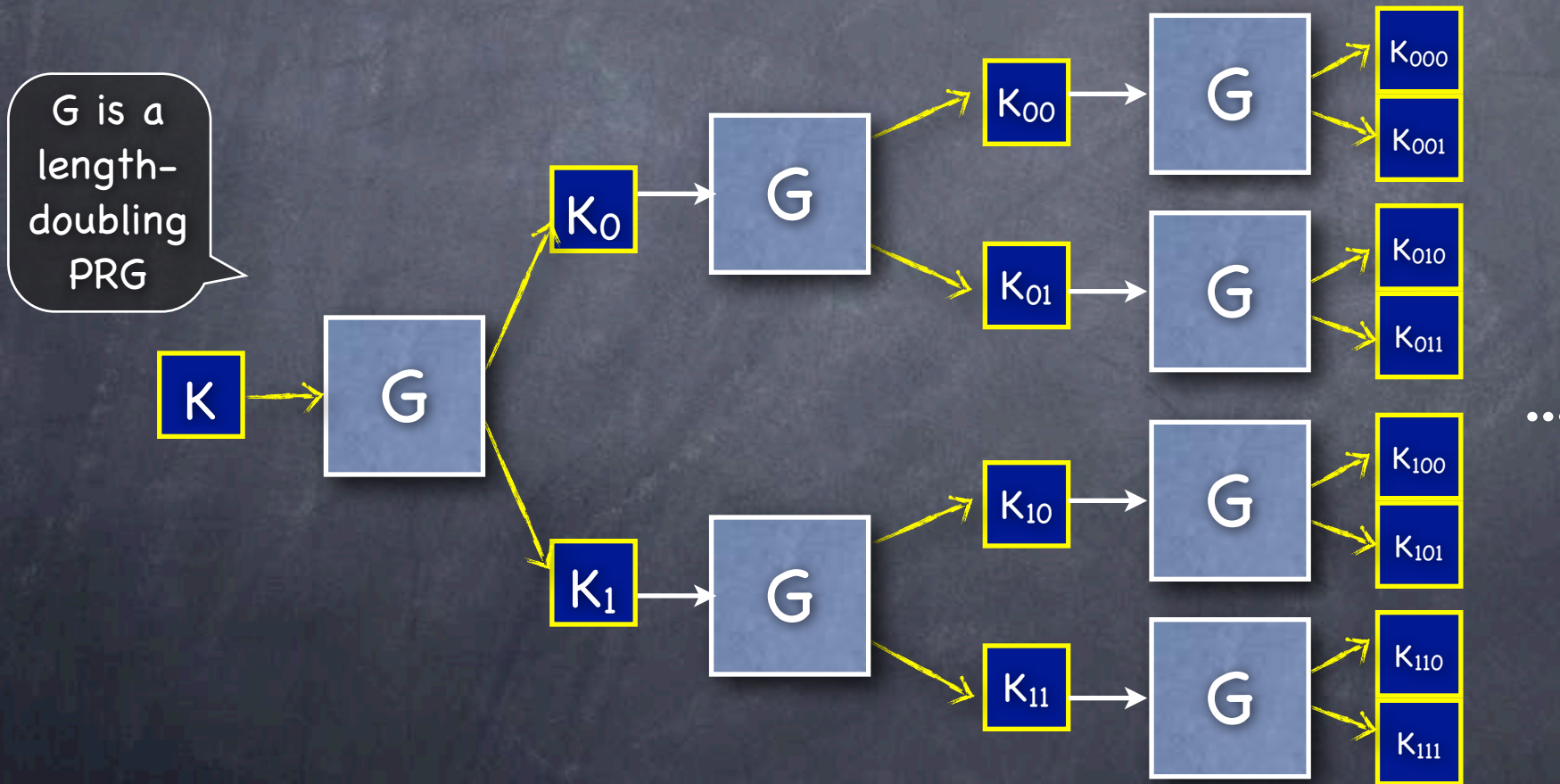
Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG



Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG



Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG



Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG



Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG



Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG



Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG

Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG
 - Not blazing fast

Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG
 - Not blazing fast
 - Faster constructions based on specific number-theoretic computational complexity assumptions

Pseudorandom Function (PRF)

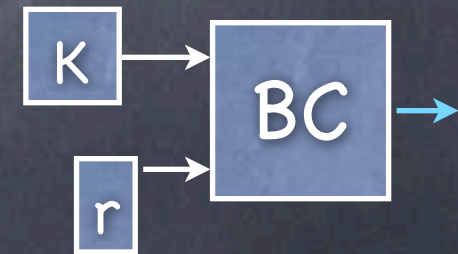
- A PRF can be constructed from any PRG
 - Not blazing fast
 - Faster constructions based on specific number-theoretic computational complexity assumptions
 - Fast heuristic constructions

Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG
 - Not blazing fast
 - Faster constructions based on specific number-theoretic computational complexity assumptions
 - Fast heuristic constructions
- PRF in practice: Block Cipher

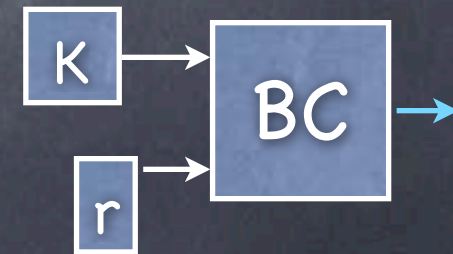
Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG
 - Not blazing fast
 - Faster constructions based on specific number-theoretic computational complexity assumptions
 - Fast heuristic constructions
- PRF in practice: Block Cipher



Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG
 - Not blazing fast
 - Faster constructions based on specific number-theoretic computational complexity assumptions
 - Fast heuristic constructions
- PRF in practice: Block Cipher
 - Extra features/requirements:



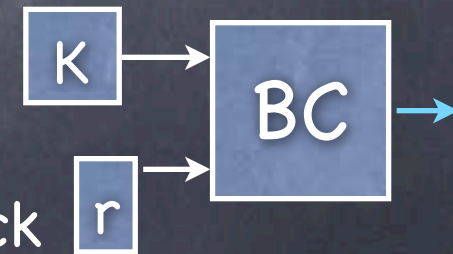
Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG
 - Not blazing fast
 - Faster constructions based on specific number-theoretic computational complexity assumptions
 - Fast heuristic constructions

• PRF in practice: Block Cipher

• Extra features/requirements:

• Permutation: input block (r) to output block



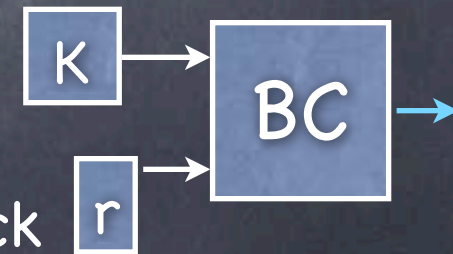
Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG
 - Not blazing fast
 - Faster constructions based on specific number-theoretic computational complexity assumptions
 - Fast heuristic constructions

- PRF in practice: Block Cipher

- Extra features/requirements:

- Permutation: input block (r) to output block
 - Key can be used as an inversion trapdoor



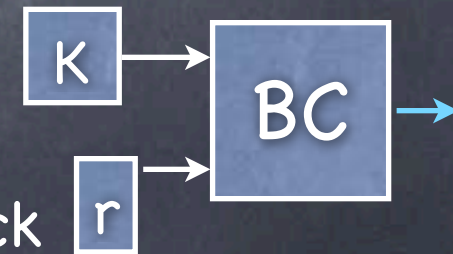
Pseudorandom Function (PRF)

- A PRF can be constructed from any PRG
 - Not blazing fast
 - Faster constructions based on specific number-theoretic computational complexity assumptions
 - Fast heuristic constructions

- PRF in practice: Block Cipher

- Extra features/requirements:

- Permutation: input block (r) to output block
 - Key can be used as an inversion trapdoor
 - Pseudorandomness even with access to inversion



CPA-secure SKE with a Block Cipher

CPA-secure SKE with a Block Cipher

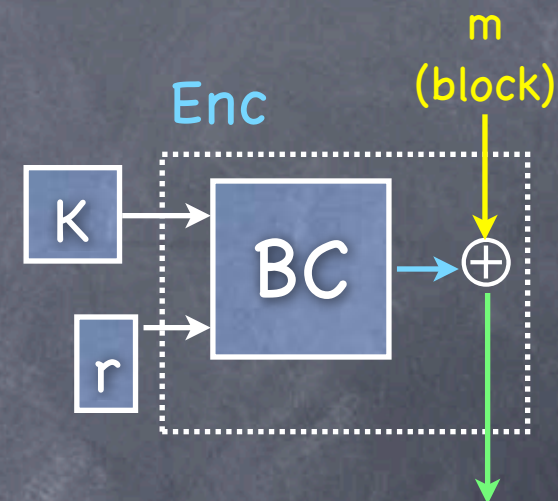
- Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) BC

CPA-secure SKE with a Block Cipher

- Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) BC
- For each encryption, Alice will pick a fresh pseudorandom pad, by picking a fresh value r and setting $\text{pad} = \text{BC}_k(r)$

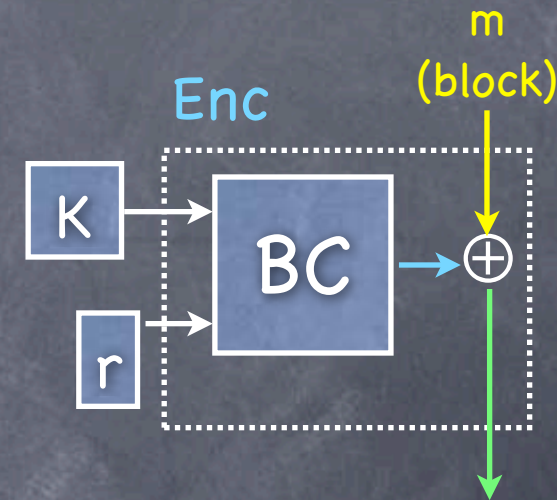
CPA-secure SKE with a Block Cipher

- Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) BC
- For each encryption, Alice will pick a fresh pseudorandom pad, by picking a fresh value r and setting $pad = BC_K(r)$



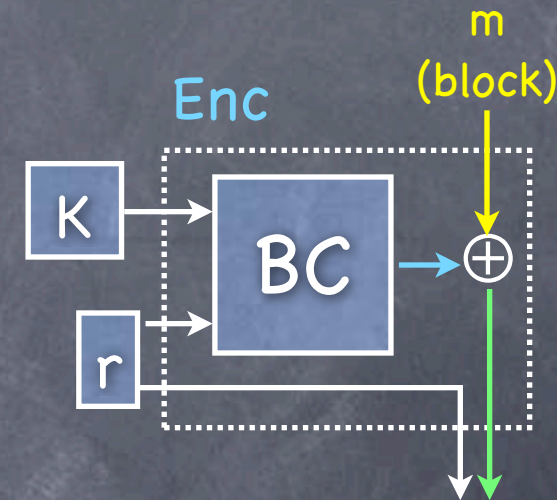
CPA-secure SKE with a Block Cipher

- Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) BC
- For each encryption, Alice will pick a fresh pseudorandom pad, by picking a fresh value r and setting $pad = BC_K(r)$
- Bob needs to be able to generate the same pad, so Alice sends r (in the clear, as part of the ciphertext) to Bob



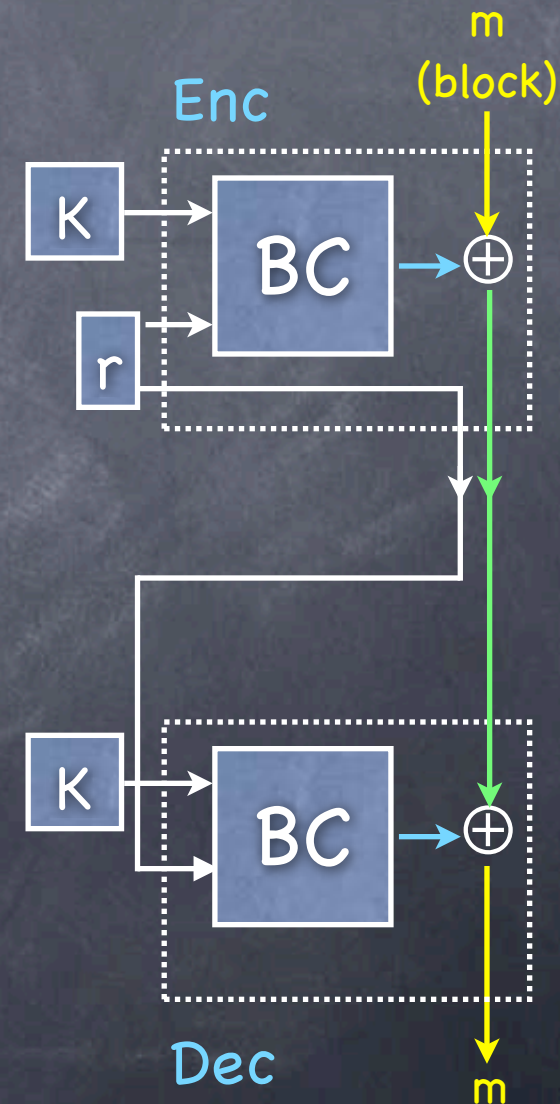
CPA-secure SKE with a Block Cipher

- Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) BC
- For each encryption, Alice will pick a fresh pseudorandom pad, by picking a fresh value r and setting $pad = BC_K(r)$
- Bob needs to be able to generate the same pad, so Alice sends r (in the clear, as part of the ciphertext) to Bob



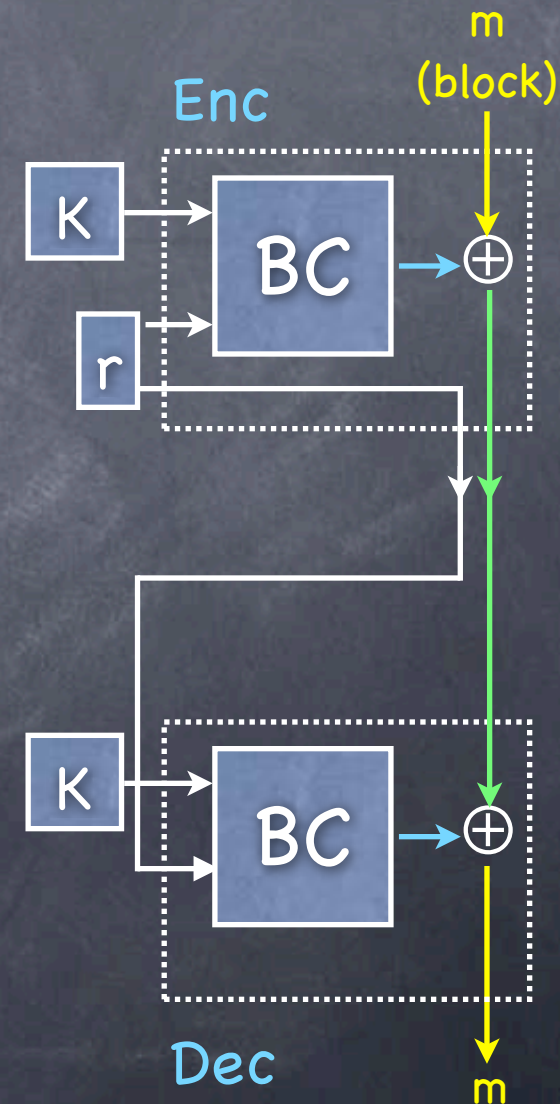
CPA-secure SKE with a Block Cipher

- Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) BC
- For each encryption, Alice will pick a fresh pseudorandom pad, by picking a fresh value r and setting $pad = BC_K(r)$
- Bob needs to be able to generate the same pad, so Alice sends r (in the clear, as part of the ciphertext) to Bob



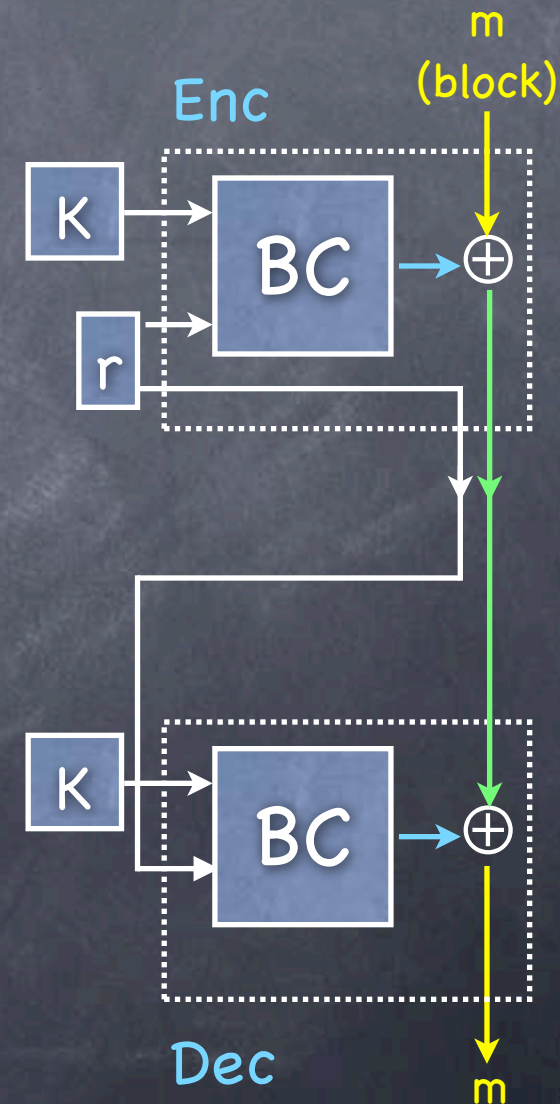
CPA-secure SKE with a Block Cipher

- Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) BC
- For each encryption, Alice will pick a fresh pseudorandom pad, by picking a fresh value r and setting $pad = BC_K(r)$
- Bob needs to be able to generate the same pad, so Alice sends r (in the clear, as part of the ciphertext) to Bob
- Even if Eve sees r , PRF security guarantees that $BC_K(r)$ is pseudorandom. (In fact, Eve could have picked r , as long as we ensure no r is reused.)



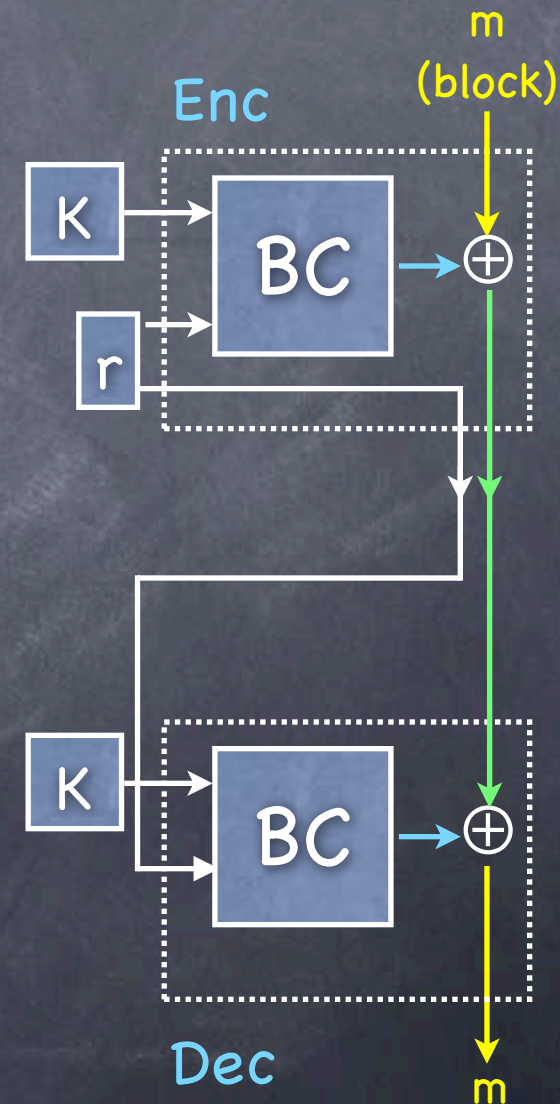
CPA-secure SKE with a Block Cipher

- Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) BC
- For each encryption, Alice will pick a fresh pseudorandom pad, by picking a fresh value r and setting $pad = BC_K(r)$
- Bob needs to be able to generate the same pad, so Alice sends r (in the clear, as part of the ciphertext) to Bob
- Even if Eve sees r , PRF security guarantees that $BC_K(r)$ is pseudorandom. (In fact, Eve could have picked r , as long as we ensure no r is reused.)
- How to pick a fresh r ?



CPA-secure SKE with a Block Cipher

- Suppose Alice and Bob have shared a key (seed) for a block-cipher (PRF) BC
- For each encryption, Alice will pick a fresh pseudorandom pad, by picking a fresh value r and setting $pad = BC_K(r)$
- Bob needs to be able to generate the same pad, so Alice sends r (in the clear, as part of the ciphertext) to Bob
- Even if Eve sees r , PRF security guarantees that $BC_K(r)$ is pseudorandom. (In fact, Eve could have picked r , as long as we ensure no r is reused.)
- How to pick a fresh r ?
 - Pick at random!



CPA-secure SKE with a Block Cipher

CPA-secure SKE with a Block Cipher

- How to encrypt a long message (multiple blocks)?

CPA-secure SKE with a Block Cipher

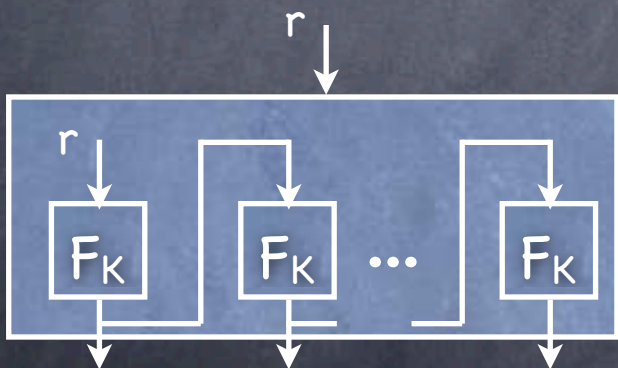
- How to encrypt a long message (multiple blocks)?
 - Can chop the message into blocks and independently encrypt each block as before. Works, but ciphertext size is double that of the plaintext (if $|r|$ is one-block long)

CPA-secure SKE with a Block Cipher

- How to encrypt a long message (multiple blocks)?
 - Can chop the message into blocks and independently encrypt each block as before. Works, but ciphertext size is double that of the plaintext (if $|r|$ is one-block long)
- Extend output length of PRF (w/o increasing input length)

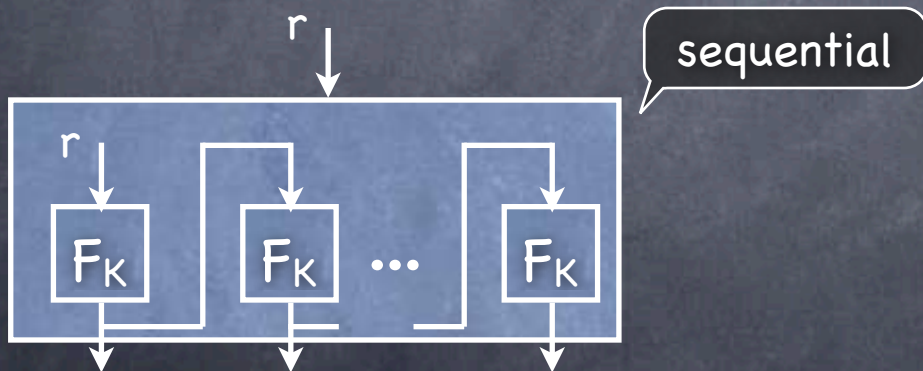
CPA-secure SKE with a Block Cipher

- How to encrypt a long message (multiple blocks)?
 - Can chop the message into blocks and independently encrypt each block as before. Works, but ciphertext size is double that of the plaintext (if $|r|$ is one-block long)
- Extend output length of PRF (w/o increasing input length)



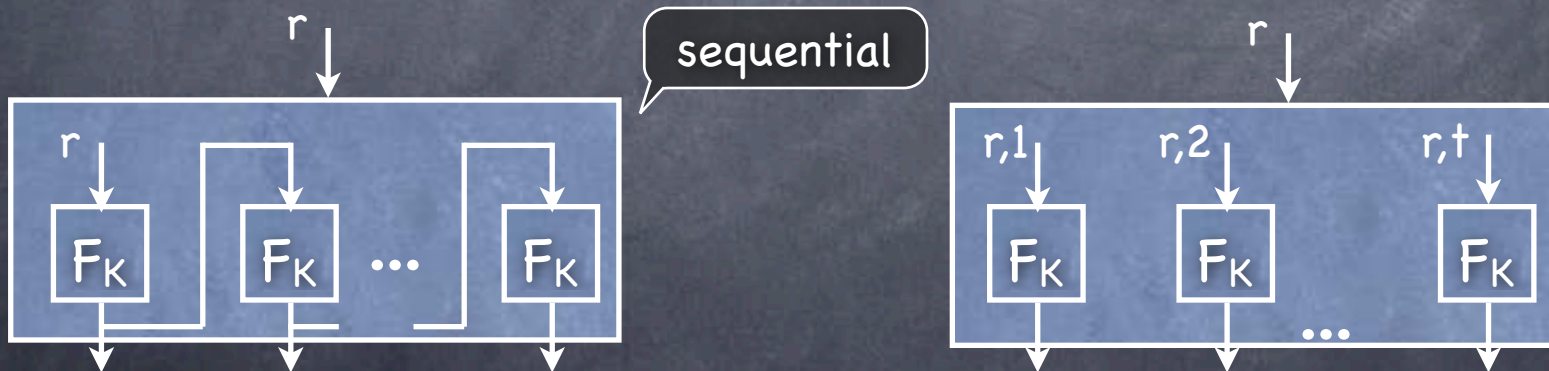
CPA-secure SKE with a Block Cipher

- How to encrypt a long message (multiple blocks)?
 - Can chop the message into blocks and independently encrypt each block as before. Works, but ciphertext size is double that of the plaintext (if $|r|$ is one-block long)
- Extend output length of PRF (w/o increasing input length)



CPA-secure SKE with a Block Cipher

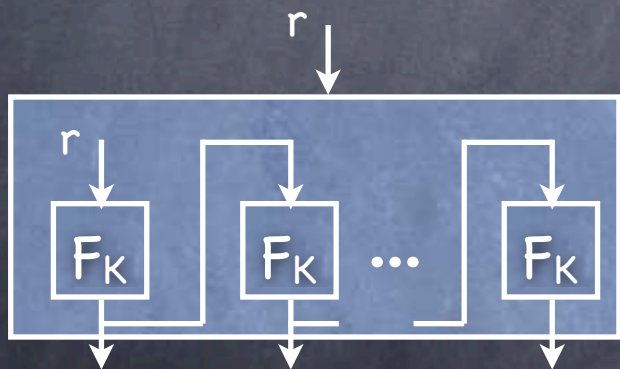
- How to encrypt a long message (multiple blocks)?
 - Can chop the message into blocks and independently encrypt each block as before. Works, but ciphertext size is double that of the plaintext (if $|r|$ is one-block long)
- Extend output length of PRF (w/o increasing input length)



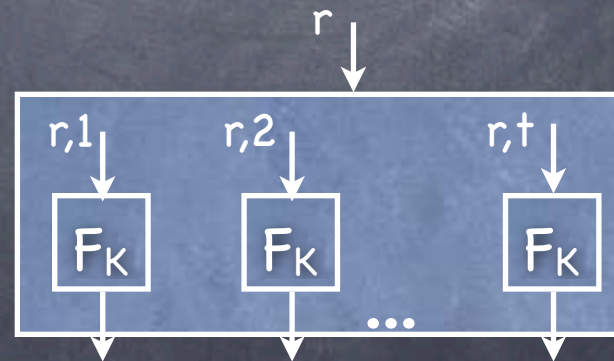
CPA-secure SKE with a Block Cipher

- How to encrypt a long message (multiple blocks)?
 - Can chop the message into blocks and independently encrypt each block as before. Works, but ciphertext size is double that of the plaintext (if $|r|$ is one-block long)

- Extend output length of PRF (w/o increasing input length)



sequential

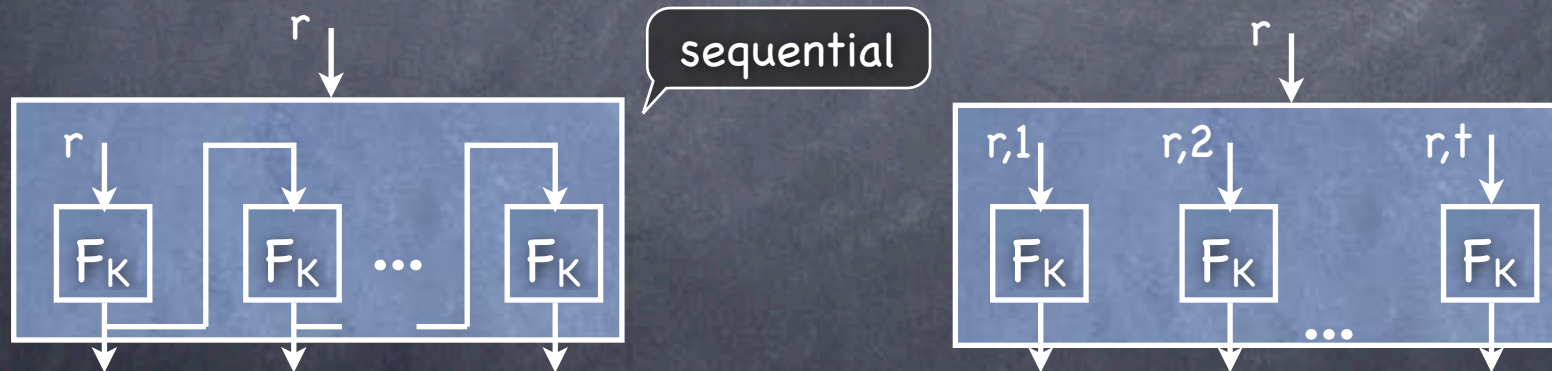


input length slightly decreased, based on an a priori limit on t

CPA-secure SKE with a Block Cipher

- How to encrypt a long message (multiple blocks)?
 - Can chop the message into blocks and independently encrypt each block as before. Works, but ciphertext size is double that of the plaintext (if $|r|$ is one-block long)

- Extend output length of PRF (w/o increasing input length)



input length slightly decreased, based on an a priori limit on t

- Output is indistinguishable from t random blocks (even if input to F_K known/chosen)

CPA-secure SKE with a Block Cipher

CPA-secure SKE with a Block Cipher

- Various “modes” of operation of a Block-cipher (i.e., encryption schemes using a block-cipher). All with one block overhead.

CPA-secure SKE with a Block Cipher

- Various “modes” of operation of a Block-cipher (i.e., encryption schemes using a block-cipher). All with one block overhead.
- **Output Feedback (OFB) mode:** Extend the pseudorandom output using the first construction in the previous slide

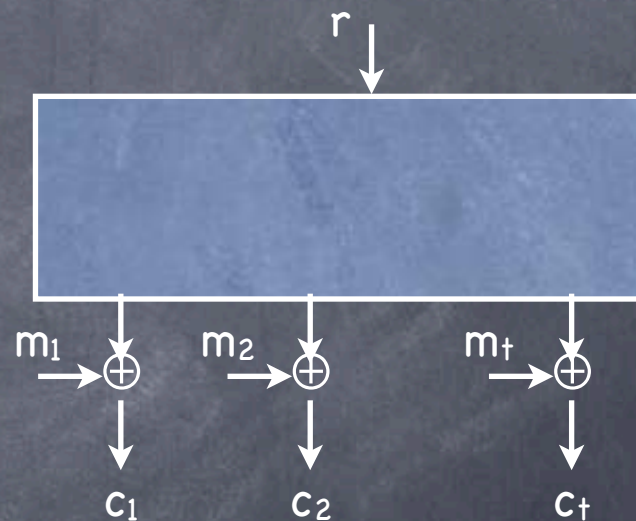
CPA-secure SKE with a Block Cipher

- Various "modes" of operation of a Block-cipher (i.e., encryption schemes using a block-cipher). All with one block overhead.
- **Output Feedback (OFB) mode:** Extend the pseudorandom output using the first construction in the previous slide



CPA-secure SKE with a Block Cipher

- Various "modes" of operation of a Block-cipher (i.e., encryption schemes using a block-cipher). All with one block overhead.
- **Output Feedback (OFB) mode:** Extend the pseudorandom output using the first construction in the previous slide

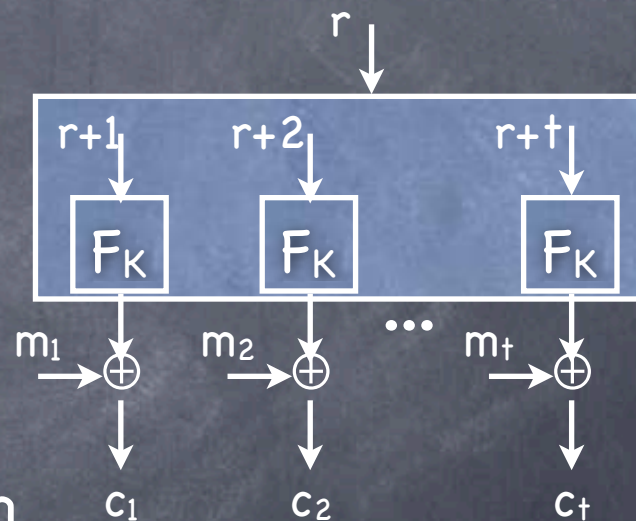


CPA-secure SKE with a Block Cipher

- Various “modes” of operation of a Block-cipher (i.e., encryption schemes using a block-cipher). All with one block overhead.

- **Output Feedback (OFB) mode:** Extend the pseudorandom output using the first construction in the previous slide

- **Counter (CTR) Mode:** Similar idea as in the second construction. No a priori limit on number of blocks in a message. Security from low likelihood of $(r+1, \dots, r+t)$ running into $(r'+1, \dots, r'+t')$

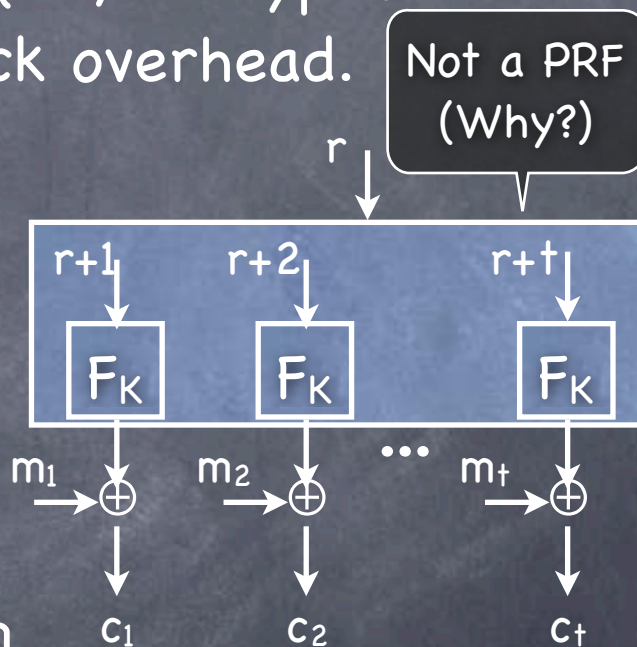


CPA-secure SKE with a Block Cipher

- Various “modes” of operation of a Block-cipher (i.e., encryption schemes using a block-cipher). All with one block overhead.

- **Output Feedback (OFB) mode:** Extend the pseudorandom output using the first construction in the previous slide

- **Counter (CTR) Mode:** Similar idea as in the second construction. No a priori limit on number of blocks in a message. Security from low likelihood of $(r+1, \dots, r+t)$ running into $(r'+1, \dots, r'+t')$



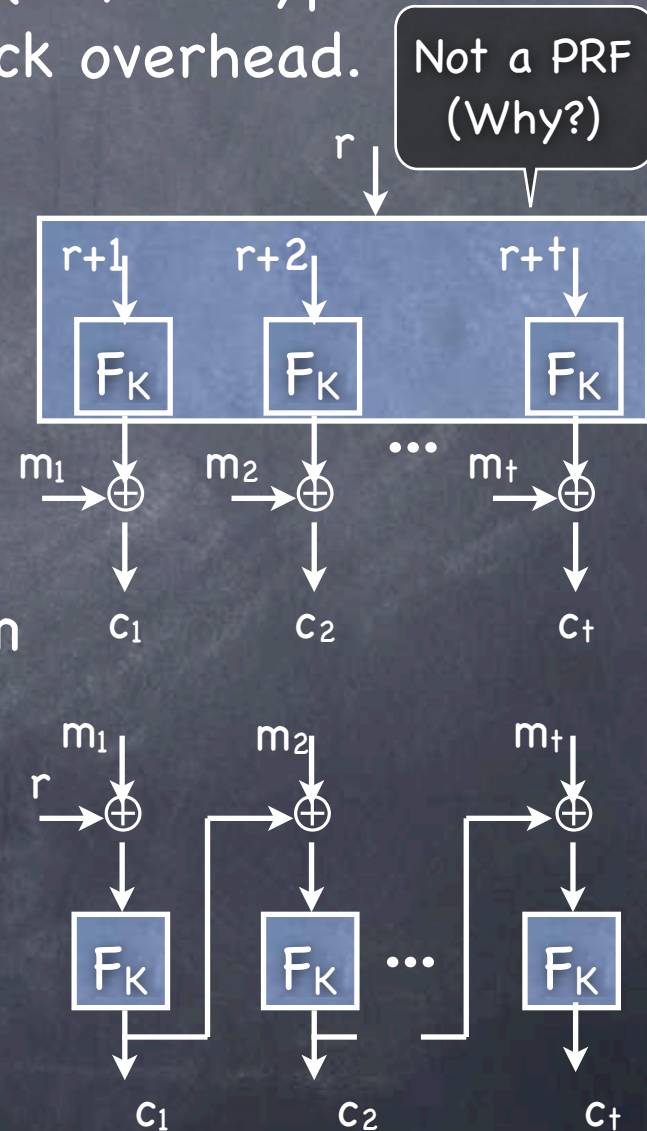
CPA-secure SKE with a Block Cipher

Various "modes" of operation of a Block-cipher (i.e., encryption schemes using a block-cipher). All with one block overhead.

Output Feedback (OFB) mode: Extend the pseudorandom output using the first construction in the previous slide

Counter (CTR) Mode: Similar idea as in the second construction. No a priori limit on number of blocks in a message. Security from low likelihood of $(r+1, \dots, r+t)$ running into $(r'+1, \dots, r'+t')$

Cipher Block Chaining (CBC) mode: Sequential encryption. Decryption uses F_K^{-1} . Ciphertext an integral number of blocks.



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

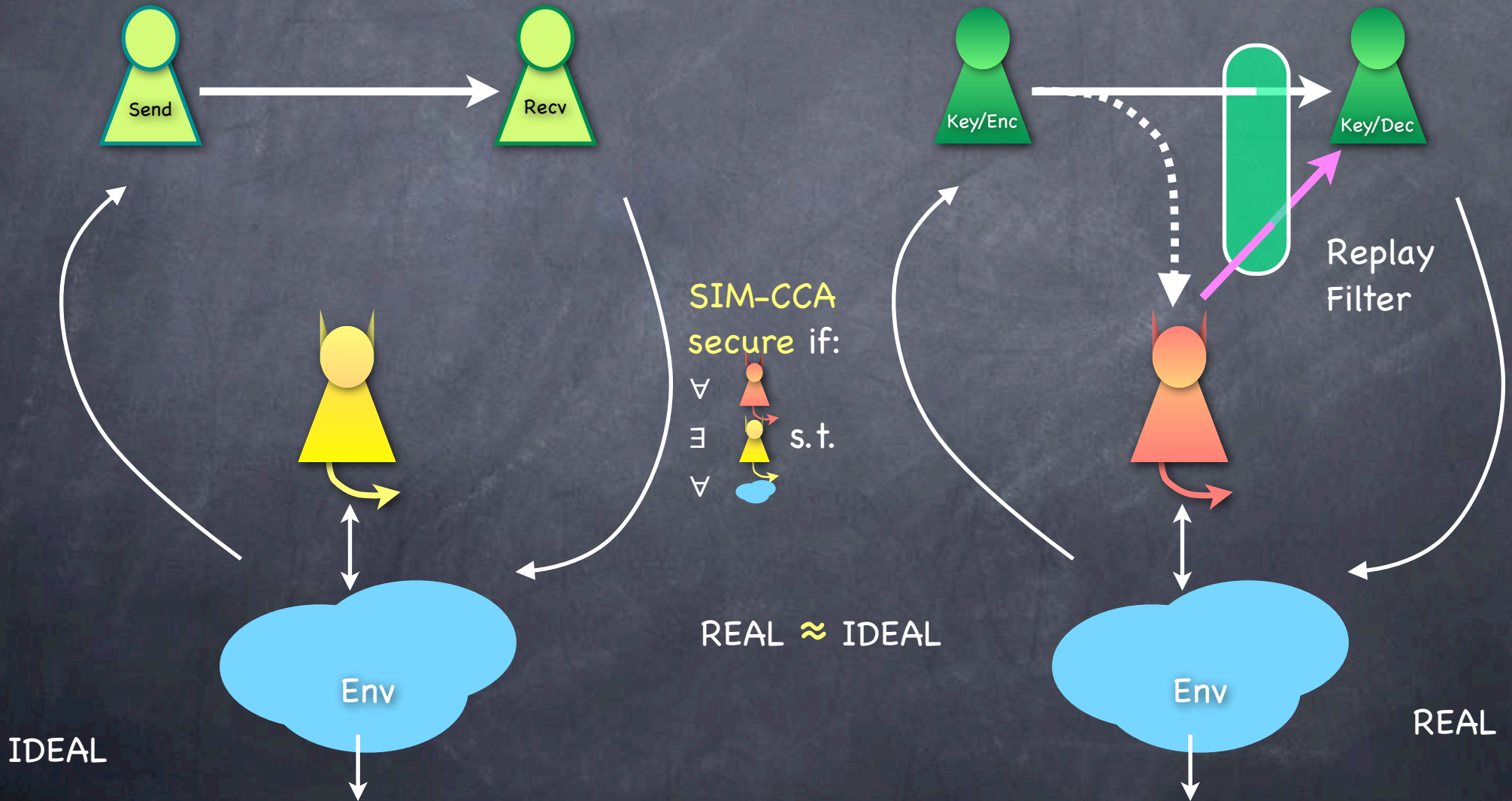
Active Adversary

- 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



RECALL

Symmetric-Key Encryption

IND-CCA +
~ correctness
equivalent to
SIM-CCA

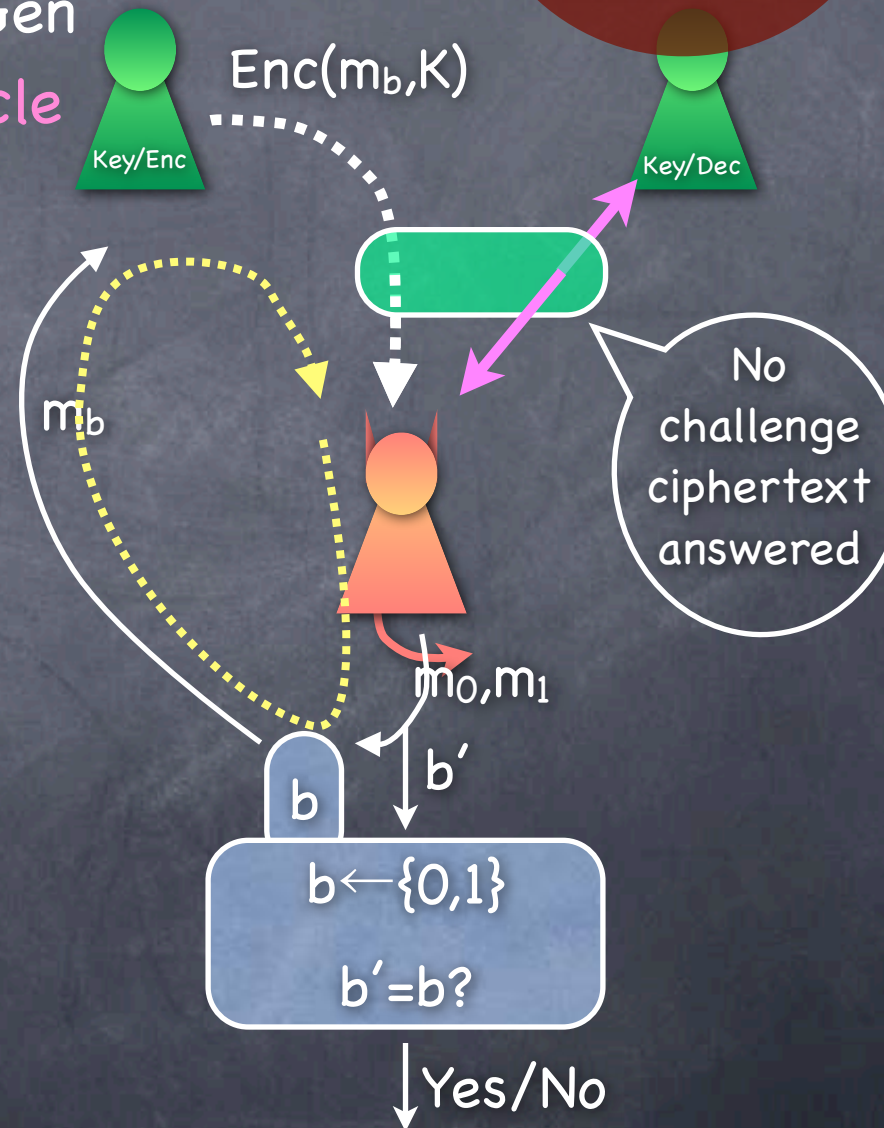
IND-CCA Security

- Experiment picks $b \leftarrow \{0,1\}$ and $K \leftarrow \text{KeyGen}$
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 $\text{Enc}(m_b, K)$ to the adversary

- Adversary returns a guess b'
- Experiment outputs 1 iff $b' = b$
- IND-CCA secure if for all feasible adversaries $\Pr[b' = b] \approx 1/2$



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
- 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
- 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

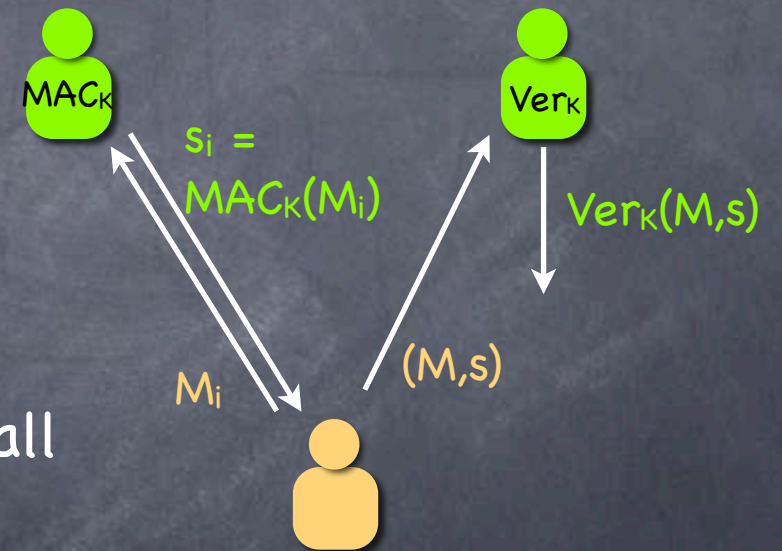
- Can sign any (polynomial) number of messages



- 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 , $\text{Verify}_K(M, \text{MAC}_K(M))=1$
- Security: probability that an adversary can produce (M,s) s.t. $\text{Verify}_K(M,s)=1$ is negligible unless Alice produced an output $s=\text{MAC}_K(M)$



Advantage

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

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

- $\text{CCA-Enc}_{K_1, K_2}(m) = (c := \text{CPA-Enc}_{K_1}(m), t := \text{MAC}_{K_2}(c))$
- CPA secure encryption: Block-cipher/CTR mode construction

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
 - 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 uses Block-Cipher standards (next time)

CCA Secure SKE

- $CCA\text{-}Enc_{K_1, K_2}(m) = (c := CPA\text{-}Enc_{K_1}(m), t := MAC_{K_2}(c))$
 - CPA secure encryption: Block-cipher/CTR mode construction
 - MAC: from a PRF or Block-Cipher (next time)
- SKE in practice uses Block-Cipher standards (next time)
- In principle, constructions (less efficient) possible based on any One-Way Permutation or even any One-Way Function