Active Adversary

Lecture 7

CCA Security

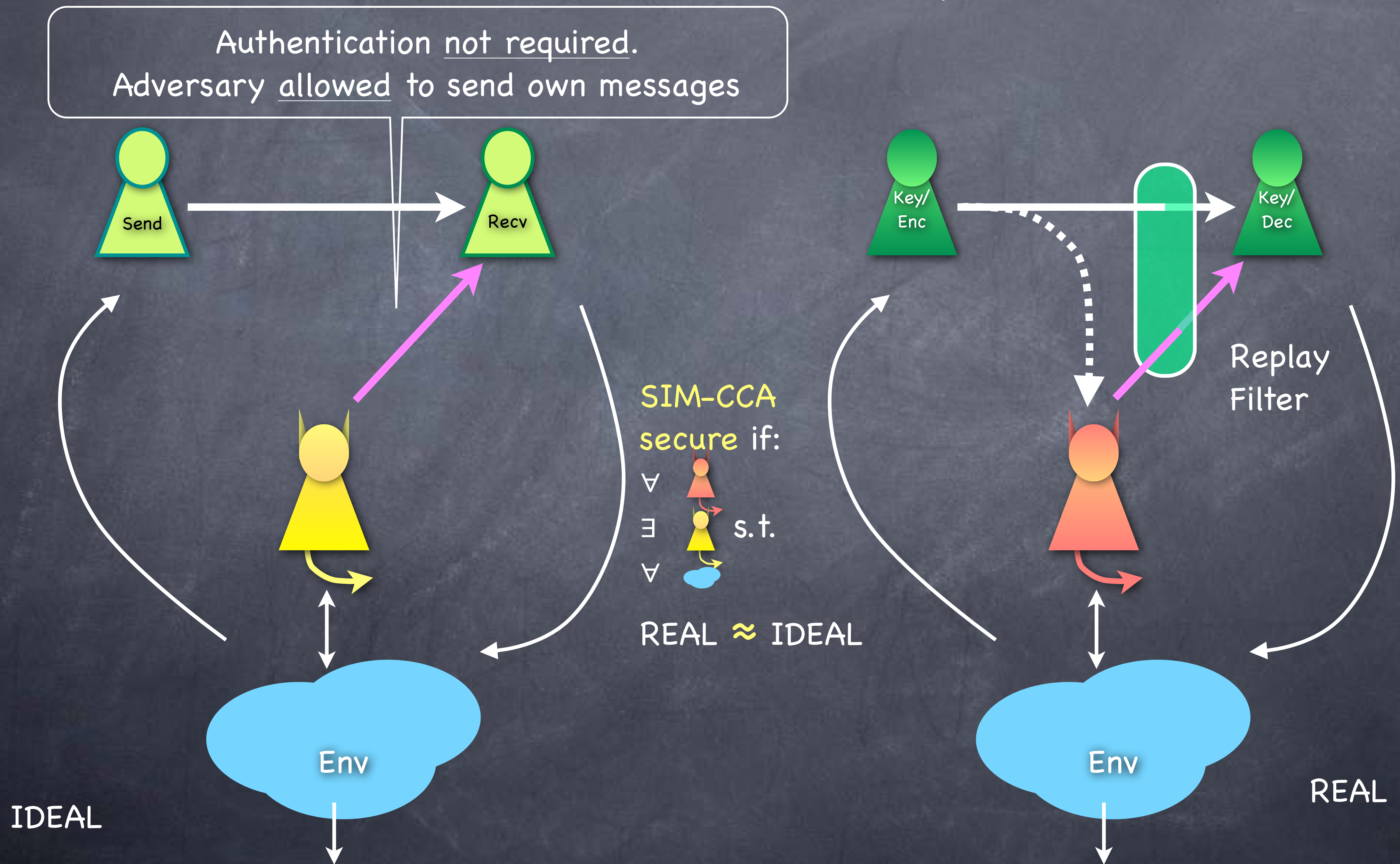
MAC

Active Adversary

- An active adversary can inject messages into the channel
  - Eve can send ciphertexts to Bob and get them decrypted
    - Chosen Ciphertext Attack (CCA)
  - If Bob decrypts all ciphertexts for Eve, no security possible
    - What can Bob do?

Symmetric-Key Encryption

SIM-CCA Security

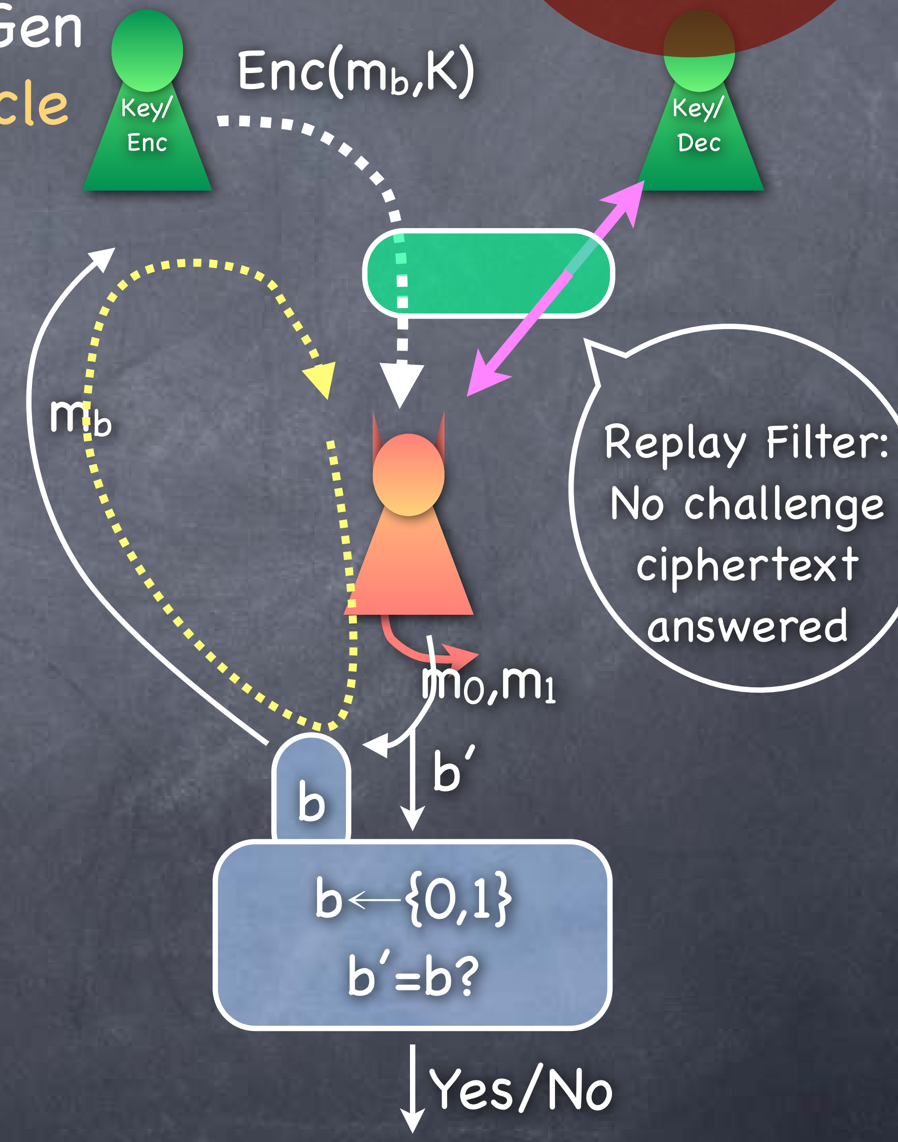


Symmetric-Key Encryption

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 $Pr[b'=b] \approx 1/2$

IND-CCA + ~correctness equivalent to SIM-CCA

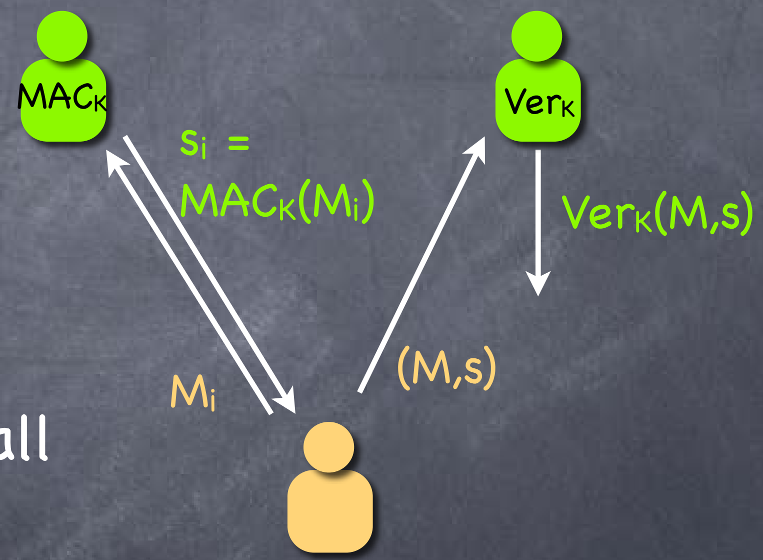


CCA Security

- 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 (symmetric key) message authentication
  - Symmetric-key solution for message authentication: Message Authentication Code (MAC)

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



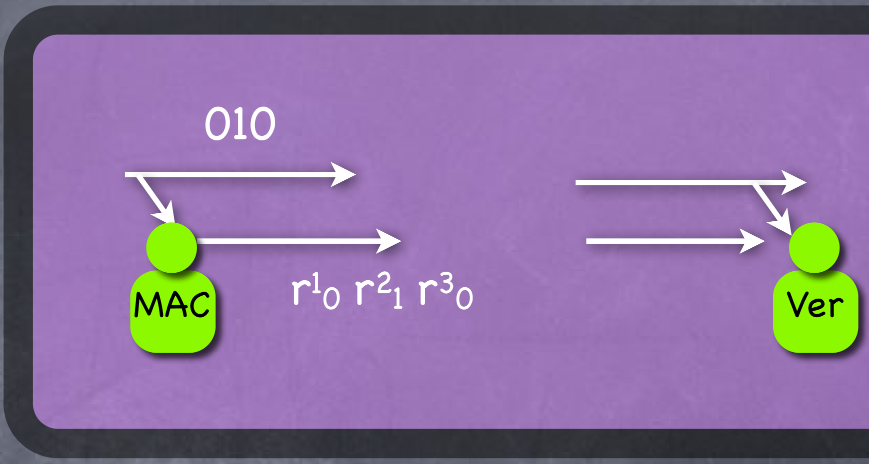
Advantage
$= \Pr[\ \mathrm{Ver}_K(M,s)=1$ and
$(M,s) \notin \{(M_i,s_i)\}\ ]$

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 can be entirely based on Block-Ciphers**
  - A tool that can make things faster: Hash functions (later)
  - Or, in principle, from any One-Way Function

Making a MAC

One-time MAC



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

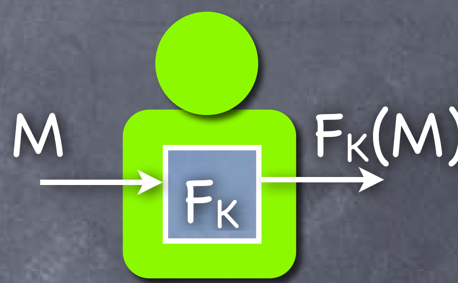
r^1_0	r^2_0	r^3_0
r^1_1	r^2_1	r^3_1

- Doesn't require any computational restrictions on adversary!
  - Has a statistical security parameter k (unlike one-time pad which has perfect security)
- More efficient one-time MACs exist (later)

(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, $\varepsilon_{MAC^*} = 2^{-m(k)}$

Adversary for PRF using forger: Given access to truly random R or PRF, use it to get MAC tags. Output 1 if forger succeeds.

MAC for Multiple-Block Messages

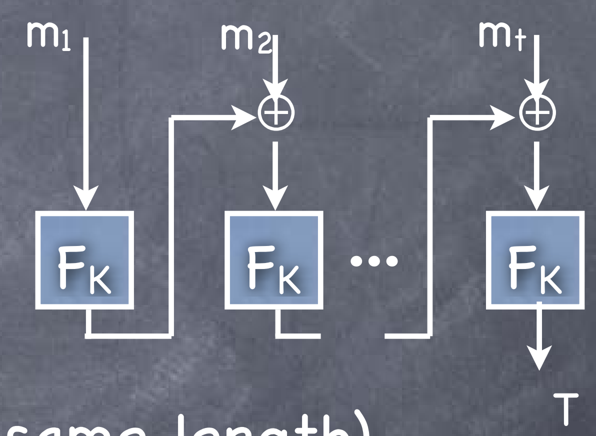
- What if message is longer than one block?
- MAC'ing each block separately is not secure (unlike in the case of CPA secure encryption)
  - Eve can rearrange the blocks/drop some blocks
- Coming up: two solutions
  1. A simple but inefficient scheme from MAC for single-block messages
  2. From a PRF (block cipher), build a PRF that takes <u>longer inputs</u>

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

- 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)
    - Security crucially relies on not revealing intermediate output blocks

Patching CBC-MAC

- Patching CBC MAC to handle message of any (polynomial) length but still producing a single block tag (secure if block-cipher is):
  - Derive K as $F_{K'}(t)$, where t is the number of blocks
  - Use first block to specify number of blocks
    - Important that first block is used: if last block, message extension attacks still possible
  - EMAC: Output not the last tag T, but $F_{K'}(T)$, where K' is an independent key (after padding the message to an integral number of blocks). No need to know message length a priori.
  - CMAC: XOR last message block with a key (derived from the original key using the block-cipher). Also avoids padding when message is integral number of blocks. NIST Recommendation. 2005
- Later: Hash-based HMAC used in TLS and IPSec IETF Standard. 1997

Authenticated Encryption

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

MAC-then-encrypt is not necessarily CCA-secure

- AE aims to do this more efficiently
  - Several constructions based on block-ciphers (modes of operation) provably secure modelling block-cipher as PRP
    - One pass: IAPM, OCB, ... [patented]
    - Two pass: CCM, GCM, GCM-SIV, ... [included in NIST standards]
  - AE with Associated Data: Allows unencrypted (but authenticated) parts of the plaintext, for headers etc.

SKE today

- SKE in IPsec, TLS etc. mainly based on AES block-ciphers
  - AES-128, AES-192, AES-256
- A recommended choice: AES Counter-mode + CMAC (or HMAC), encrypt-then-MAC. Gives CCA security, and provides authentication.
  - Alternately, more optimised AES modes for Authenticated Encryption with Associated Data (e.g., AES-GCM)
- Older components/modes still in use
  - Supported by many standards for legacy purposes
  - In many applications (sometimes with modifications)
    - e.g. RC4 still used in BitTorrent
- API of libraries tend to be "too low-level," e.g., Block-cipher, instead of Enc, Dec, KeyGen, leading to errors (e.g., use of ECB mode, nonce reuse, key cycles)