Cryptography
CS 555

Topic 9: Message Authentication Codes

Reminder: Homework 1

- Due on **Friday** at the **beginning** of class

- Please typeset your solutions

Recap

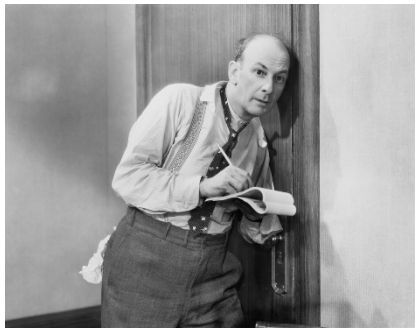
- CPA-Security vs. CCA-Security
- PRFs

Today's Goals:

- Introduce Message Authentication Codes (MACs)
  - Key tool in Construction of CCA-Secure Encryption Schemes
- ~~Build Secure MACs~~

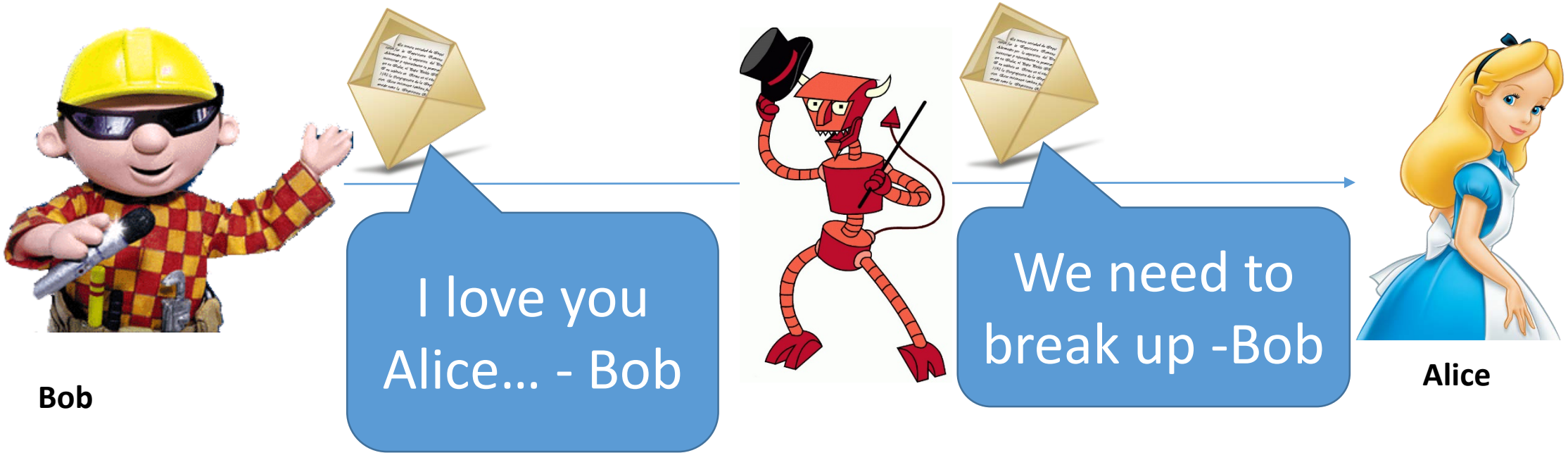
What Does It Mean to “Secure Information”

- Confidentiality (Security/Privacy)
  - Only intended recipient can see the communication



What Does It Mean to “Secure Information”

- Confidentiality (Security/Privacy)
  - Only intended recipient can see the communication
- Integrity (Authenticity)
  - The message was actually sent by the alleged sender

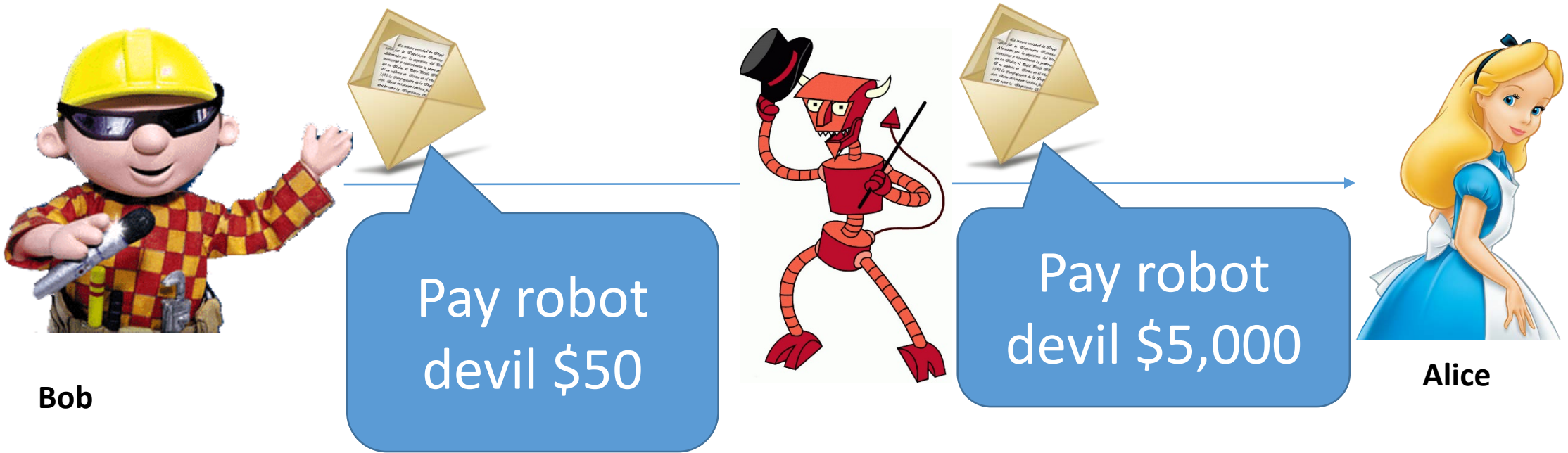


Message Authentication Codes

- CPA-Secure Encryption: Focus on Secrecy
  - But does not promise integrity
- Message Authentication Codes: Focus on Integrity
  - But does not promise secrecy
- CCA-Secure Encryption: Requires Integrity and Secrecy

What Does It Mean to “Secure Information”

- Integrity (Authenticity)
  - The message was actually sent by the alleged sender
  - And the received message matches the original



Error Correcting Codes?

- Tool to detect/correct a *small* number of random errors in transmission
- **Examples:** Parity Check, Reed-Solomon Codes, LDPC, Hamming Codes …
- Provides no protection against a malicious adversary who can introduce an arbitrary number of errors
- Still useful when implementing crypto in the real world (Why?)

Modifying Ciphertexts

$$\text{Enc}_k(\text{m}) = c = \langle r, F_k(r) \oplus m \rangle$$

$$c' = \langle r, F_k(r) \oplus m \oplus y \rangle$$

$$\text{Dec}_k(c') = F_k(r) \oplus F_k(r) \oplus m \oplus y = m \oplus y$$

If attacker knows original message he can forge c' to decrypt to any message he wants.

Even if attacker doesn't know message he may find it advantageous to flip certain bits (e.g., decimal places)

Message Authentication Code Syntax

Definition 4.1: A message authentication code (MAC) consists of three algorithms

- $\mathrm{Gen}(1^n; R)$ (Key-generation algorithm)
  - Input: security parameter 1^n (unary) and random bits R
  - Output: Secret key $\mathrm{k} \in \mathcal{K}$
- $\mathrm{Mac}_{\mathrm{k}}(m; R)$ (Tag Generation algorithm)
  - Input: Secret key $\mathrm{k} \in \mathcal{K}$ and message $\mathrm{m} \in \mathcal{M}$ and random bits R
  - Output: a tag t
- $\mathrm{Vrfy}_{\mathrm{k}}(m, t)$ (Verification algorithm)
  - Input: Secret key $\mathrm{k} \in \mathcal{K}$, a message m and a tag t
  - Output: a bit b (b=1 means “valid” and b=0 means “invalid”)
- Invariant?

Message Authentication Code Syntax

Definition 4.1: A message authentication code (MAC) consists of three algorithms

- $\mathrm{Gen}(1^n; R)$ (Key-generation algorithm)
  - Input: security parameter 1^n (unary) and random bits R
  - Output: Secret key $\mathrm{k} \in \mathcal{K}$
- $\mathrm{Mac}_\mathrm{k}(m; R)$ (Tag Generation algorithm)
  - Input: Secret key $\mathrm{k} \in \mathcal{K}$ and message $\mathrm{m} \in \mathcal{M}$ and random bits R
  - Output: a tag t
- $\mathrm{Vrfy}_\mathrm{k}(m, t)$ (Verification algorithm)
  - Input: Secret key $\mathrm{k} \in \mathcal{K}$, a message m and a tag t
  - Output: a bit b (b=1 means “valid” and b=0 means “invalid”)
- Invariant?

Message Authentication Code Syntax

Definition 4.1: A message authentication code (MAC) consists of three algorithms $\Pi = (\mathrm{Gen}, \mathrm{Mac}, \mathrm{Vrfy})$

- $\mathrm{Gen}(1^n; R)$ (Key-generation algorithm)
  - Input: security parameter 1^n (unary) and random bits R
  - Output: Secret key $\mathrm{k} \in \mathcal{K}$
- $\mathrm{Mac}_{\mathrm{k}}(m; R)$ (Tag Generation algorithm)
  - Input: Secret key $\mathrm{k} \in \mathcal{K}$ and message $\mathrm{m} \in \mathcal{M}$ and random bits R
  - Output: a tag t
- $\mathrm{Vrfy}_{\mathrm{k}}(m, t)$ (Verification algorithm)
  - Input: Secret key $\mathrm{k} \in \mathcal{K}$, a message m and a tag t
  - Output: a bit b (b=1 means “valid” and b=0 means “invalid”)

$$\mathrm{Vrfy}_{\mathrm{k}}(m, \mathrm{Mac}_{\mathrm{k}}(m; R)) = 1$$

General vs Fixed Length MAC

$$\mathcal{M} = \{0,1\}^*$$

versus

$$\mathcal{M} = \{0,1\}^{\ell(n)}$$

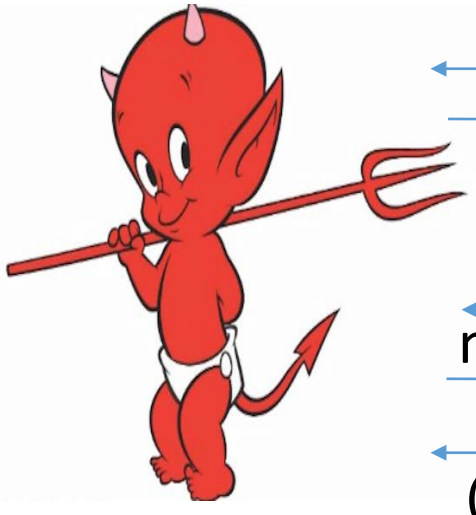
Deterministic MACs

- **Canonical Verification Algorithm**

$$\mathrm{Vrfy}_{\mathrm{k}}(m,t) = \begin{cases} 1 & \text{if } t = \mathrm{Mac}_{\mathrm{k}}(m) \\ 0 & \text{otherwise} \end{cases}$$

- "All real-world MACs use canonical verification" – page 115

MAC Authentication Game $(\text{Macforge}_{A,\Pi}(n))$



m_1

$t_1 = \text{Mac}_K(m_1)$

m_2

$t_2 = \text{Mac}_K(m_2)$

$\vdots$

m_q

$t_q = \text{Mac}_K(m_q)$

(m, t) s.t $m \notin \{m_1, \ldots, m_q\}$

$\text{Macforge}_{A,\Pi}(n) = \text{Vrfy}_k(m, t)$

K = Gen(.)

$\forall PPT\ A\ \exists \mu\ (\text{negligible})\ \text{s.t}$

$\Pr[\text{Macforge}_{A,\Pi}(n) = 1] \leq \mu(n)$

Discussion

- Is the definition too strong?
  - Attacker wins if he can forge any message
  - Does not necessarily attacker can forge a “meaningful message”
  - “Meaningful Message” is context dependent
  - Conservative Approach: Prove Security against more powerful attacker
  - Conservative security definition can be applied broadly
- Replay Attacks?
  - **$t=Mac_K$(“Pay Bob $1,000 from Alice’s bank account”)**
  - Alice cannot modify message to say $10,000, but…
  - She may try to replay it 10 times

Replay Attacks

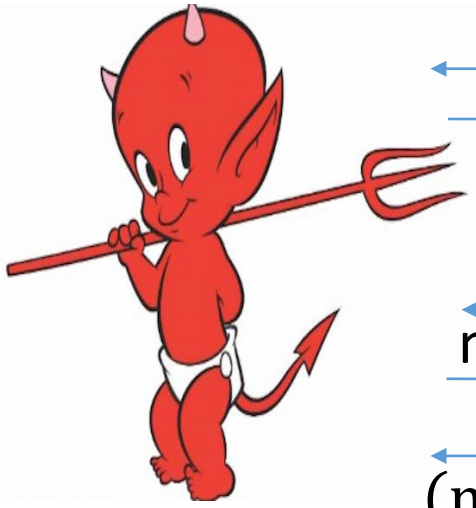
- MACs alone do not protect against replay attacks (they are stateless)

- Common Defenses:
  - Include Sequence Numbers in Messages (requires synchronized state)
    - Can be tricky over a lossy channel
  - Timestamp Messages
    - Double check timestamp before taking action

Strong MACs

- Previous game ensures attacker cannot generate a valid tag for a new message.
- However, attacker may be able to generate a second valid tag t’ for a message m after observing (m,t)
- Strong MAC: attacker cannot generate second valid tag, even for a known message

Strong MAC Authentication ($\mathrm{Macsforge}_{A,\Pi}(n)$)



m_1

$t_1 = \mathrm{Mac}_K(m_1)$

m_2

$t_2 = \mathrm{Mac}_K(m_2)$

$\vdots$

m_q

$t_q = \mathrm{Mac}_K(m_q)$

(m, t) s.t $(m, t) \notin \{(m_1, t_1), \ldots, (m_q, t_q)\}$

$\mathrm{Macsforge}_{A,\Pi}(n) = \mathrm{Vrfy}_k(m, t)$

K = Gen(.)

$\forall PPT\ A\ \exists \mu$ (negligible) s. t

$\Pr[\mathrm{Macsforge}_{A,\Pi}(n) = 1] \leq \mu(n)$

Strong MAC vs Regular MAC

Proposition 4.4: Let $\Pi = (\mathrm{Gen}, \mathrm{Mac}, \mathrm{Vrfy})$ be a secure MAC that uses canonical verification. Then Π is a strong MAC.

"All real-world MACs use canonical verification" – page 115

Should attacker have access to $\mathbf{Vrfy_K(.)}$ oracle in games?

(e.g., CPA vs CCA security for encryption)

Irrelevant if the MAC uses canonical verification!

Timing Attacks (Side Channel)

Naïve Canonical Verification Algorithm

Input: m,t’

```
t=Mac_K(m)
for i=1 to tag-length
   if t[i] != t’[i] then
      return 0
return 1
```

Example

t= 1 0 1 0 1 1 1 0

t’= 1 0 1 0 1 0 1 1

Returns 0 after 8 steps

Timing Attacks (Side Channel)

Naïve Canonical Verification Algorithm

Input: m,t’

t=Mac_K(m)
for i=1 to tag-length
  if t[i] != t’[i] **then**
    return 0
return 1

Example

t= 1 0 1 0 1 1 1 0
t’= 0 0 1 0 1 0 1 0

Returns 0 after 1 step

Timing Attack

- MACs used to verify code updates for Xbox 360
- Implementation allowed different rejection times (side-channel)
- Attacks exploited vulnerability to load pirated games onto hardware
- **Moral**: Ensure verification is time-independent

Improved Canonical Verification Algorithm

Input: m,t’

B=1
t=Mac_K(m)
for i=1 to tag-length
  if t[i] != t’[i] **then**
    B=0
  else (dummy op)
return B

Example

t= 1 0 1 0 1 1 1 0
t’= 0 0 1 0 1 0 1 0

Returns 0 after 8 steps

Side-Channel Attacks

- Cryptographic Definition
  - Attacker only observes outputs of oracles (Enc, Dec, Mac) and nothing else
- When attacker gains additional information like timing (not captured by model) we call it a side channel attack.

Other Examples

- Differential Power Analysis
- Cache Timing Attack
- Power Monitoring
- Acoustic Cryptanalysis
- …many others

Next Class

- Read Katz and Lindell 4.3
- Message Authentication Codes (MACs) Part 2
  - Constructing Secure MACs