Integrity and Authentication

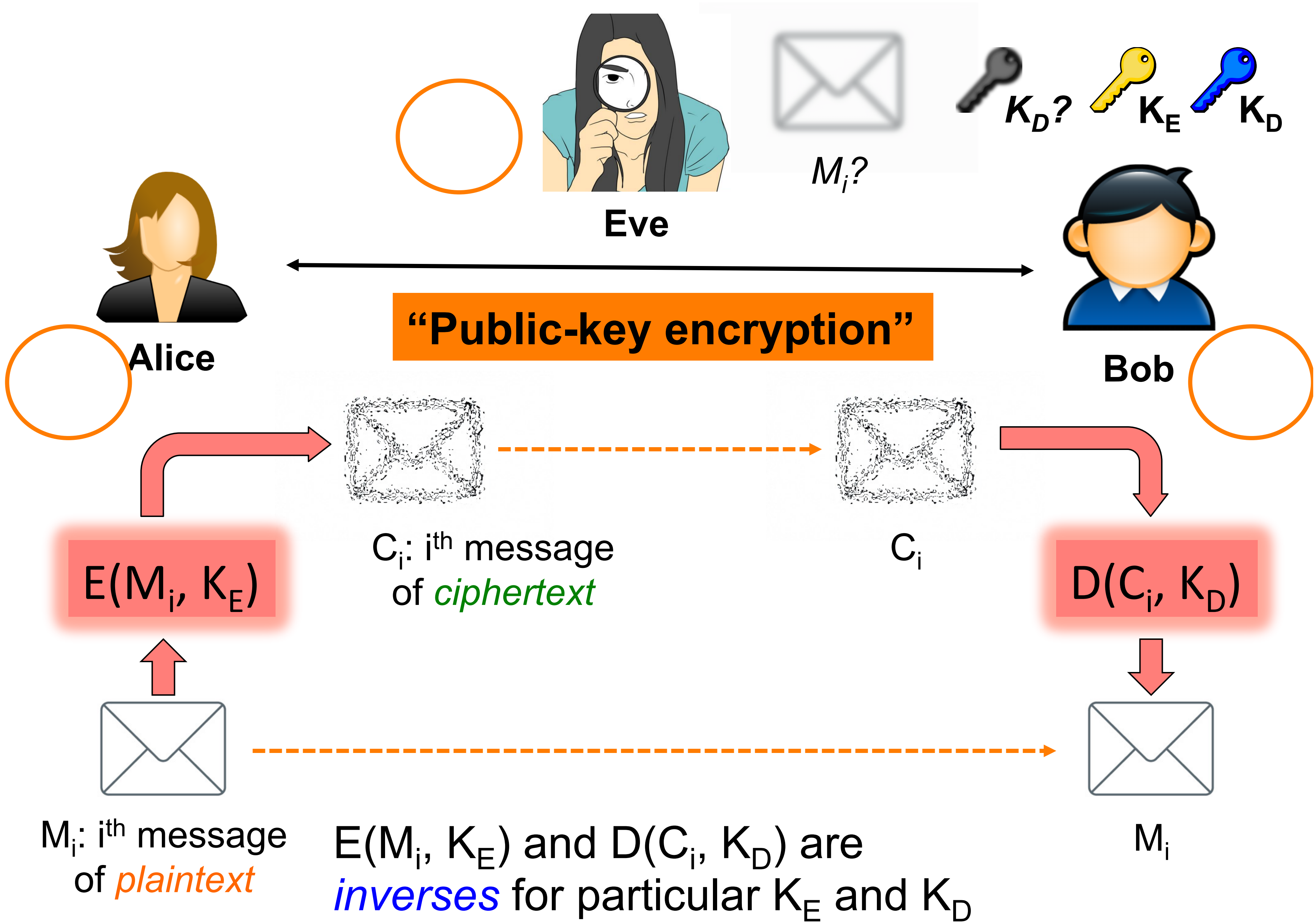
CS 161: Computer Security

Prof. Vern Paxson

TAs: Paul Bramsen, Apoorva Dornadula,
David Fifield, Mia Gil Epner, David Hahn, Warren He,
Grant Ho, Frank Li, Nathan Malkin, Mitar Milutinovic,
Rishabh Poddar, Rebecca Portnoff, Nate Wang

http://inst.eecs.berkeley.edu/~cs161/

February 28, 2017


Eve
M_i?
K_D?
K_E
K_D
Alice
Bob
"Public-key encryption"
$E(M_i, K_E)$
C_i: i^{th} message
of *ciphertext*
C_i
$D(C_i, K_D)$
M_i: i^{th} message
of *plaintext*
$E(M_i, K_E)$ and $D(C_i, K_D)$ are
inverses for particular K_E and K_D
M_i

RSA Public-Key Encryption

1. Generate random primes p, q
2. Compute $n = p \cdot q$
3. Compute $\varphi(n) = (p-1)(q-1)$
   Important: if Eve sees n, she can't deduce $\varphi(n)$
   unless she can factor n into p and q
4. Choose $2 < e < \varphi(n)$, where e and $\varphi(n)$ are relatively prime
   Could be something simple like $e=3$, if rel. prime.
5. Public key $K_E = \{ n, e \}$. Both are Well Known.
6. Compute $d = e^{-1} \bmod \varphi(n)$
   d is *multiplicative inverse* of e, modulo $\varphi(n)$
   easy to find if you know $\varphi(n)$

   (believed) HARD to compute if you don't know p, q
7. Private key $K_D = \{ d \}$

RSA Encryption/Decryption

- Let M be a message interpreted as an unsigned integer with $M < n$

  (We'll deal with $M \geq n$ in a minute …)

- $E(M, K_E) = E_{\{n, e\}}(M) = M^e \bmod n$

- $D(C, K_D) = D_{\{d\}}(C) = C^d \bmod n$

  $= (M^e)^d \bmod n$

  $= M^{e \cdot d} \bmod n$

  $= (M^{e \cdot d - 1}) \cdot M \bmod n$

  $= \ldots$

Note: taking modular roots is believed to be **computationally intractable**: otherwise Eve would just extract the e^{th} root of the ciphertext to recover M

RSA Encryption/Decryption, con't

- So we have: $D(C, K_D) = (M^{e \cdot d-1}) \cdot M \bmod n$
- Now recall that d is the multiplicative inverse of e, modulo $\varphi(n)$, and thus:

  $e \cdot d = 1 \bmod \varphi(n)$ (by definition)

  $e \cdot d - 1 = k \cdot \varphi(n)$ for some k
- Therefore $D(C, K_D) = (M^{e \cdot d-1}) \cdot M \bmod n$

  $= (M^{k\varphi(n)}) \cdot M \bmod n$

  $= [(M^{\varphi(n)})^k] \cdot M \bmod n$

  $= (1^k) \cdot M \bmod n$ *by Euler's Theorem*

  $= M \bmod n = M$

(believed) Eve can recover M from C *iff* Eve can factor $n=p \cdot q$

Some Considerations for Public-Key Encryption

- Suppose Eve knows message is one of "Buy!" or "Sell". Problem?
  - Eve can just try encrypting each using {n, e} to see which yields the observed ciphertext
    - $C = (\text{"Buy!"})^e \bmod n$? $C = (\text{"Sell"})^e \bmod n$?
  - Solution: encrypt Encode(M), where Encode adds a random IV (and also adjusts M for some corner-cases that are easy to invert)
    - Encode is well-known, easy to invert

Some Considerations for Public-Key Encryption, con't

- What if $M \geq n$?
  - Decryption $D(C, K_D) = (M^{e \cdot d-1}) \cdot M$ mod n $\Longrightarrow$ can't recover M

- Solution: use Public-Key encryption to encrypt a *random* AES key K*; encrypt M using AES(M, K*)
  - Indeed, this is how public-key encryption is routinely used – because public key operations *so much slower* than block cipher operations

Integrity & Message Authentication

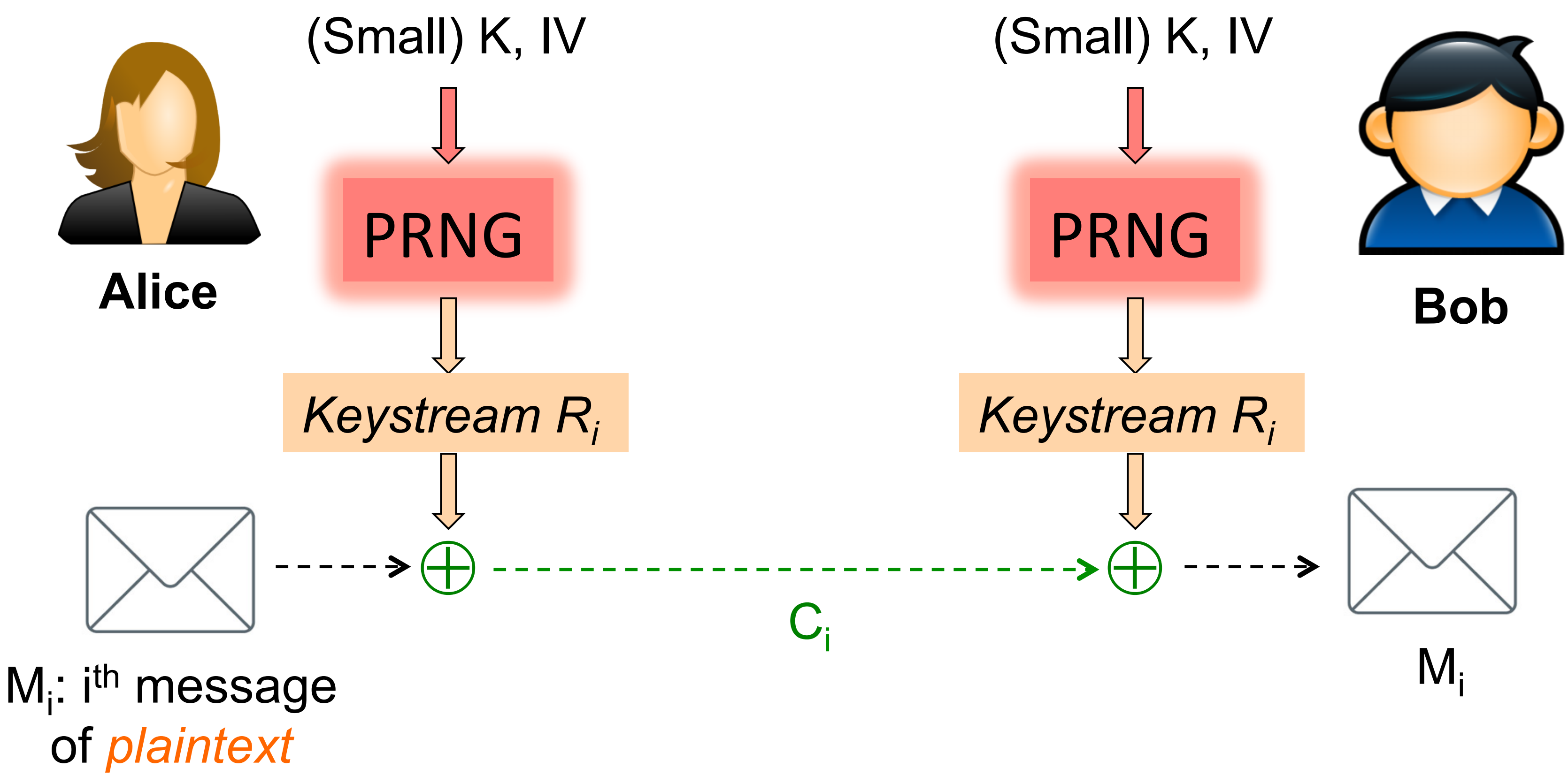
Integrity and Authentication

- Integrity: Bob can confirm that what he's received is exactly the message M that was originally sent
- Authentication: Bob can confirm that what he's received was indeed generated by Alice
- Reminder: for either, confidentiality may-or-may-not matter
  - E.g. conf. not needed when Mozilla distributes a new Firefox binary

Encryption Does Not Provide Integrity

- Simple example: Consider a stream cipher SC_K that uses a cryptographically strong sequence of pseudo-random bytes, R_i.
  - Split message M into plaintext bytes P_i. $C_i = P_i \oplus R_i$

Using a PRNG to Build a Stream Cipher



Encryption Does Not Provide Integrity

- Simple example: Consider a stream cipher SC_K that uses a cryptographically strong sequence of pseudo-random bytes, R_i.
  - Split message M into plaintext bytes P_i. $C_i = P_i \oplus R_i$

- Suppose Mallory knows that Alice sends to Bob "Pay Mal $100". Mallory **intercepts** corresponding C, IV

Mallory the Manipulator

- Mallory is an *active attacker*
  - Can introduce new messages (ciphertext)
  - Can “replay” previous ciphertexts
  - Can cause messages to be reordered or discarded
- A “***Man in the Middle***” (MITM) attacker
  - Can be *much more powerful* than just eavesdropping

Encryption Does Not Provide Integrity

- Simple example: Consider a stream cipher SC_K that uses a cryptographically strong sequence of pseudo-random bytes, R_i.
  - Split message M into plaintext bytes P_i. $C_i = P_i \oplus R_i$

- Suppose Mallory knows that Alice sends to Bob "Pay Mal $100". Mallory **intercepts** corresponding C, IV
  - M = "Pay Mal $100". C = "r4ZC#jj8qThM"
  - $M_{10..12}$ = "100". $C_{10..12}$ = "ThM"
  - $R_{10..12}$ = ?

Encryption Does Not Provide Integrity

- $R_{10..12} = ?$
- Mallory computes

$$
\begin{aligned}
\beta &= (\text{“100”} \oplus \text{“999”}) \oplus C_{10..12} \\
&= (\text{“100”} \oplus \text{“999”}) \oplus \text{“ThM”} \\
&= (\text{“100”} \oplus \text{“999”}) \oplus (\text{“100”} \oplus R_{10..12}) \\
&= (\text{“999”} \oplus R_{10..12}) \oplus (\text{“100”} \oplus \text{“100”}) \\
&= \text{“999”} \oplus R_{10..12}
\end{aligned}
$$

- Mallory constructs C' = “r4ZC#jj8q$\beta_1\beta_2\beta_3$”. Sends it and IV to Bob.
- Bob decrypts. SC_K with IV yields same R_i.
  M' = “Pay Mal $999” … *even though Mallory doesn’t know K*
- More general attack: Mallory recovers **all** of $R_i = C_i \oplus M_i$
  - Now can construct valid C' for any desired M' via $C'_i = R_i \oplus M'_i$

Integrity and Authentication

- Integrity: Bob can confirm that what he's received is exactly the message M that was originally sent
- Authentication: Bob can confirm that what he's received was indeed generated by Alice
- Reminder: for either, confidentiality may-or-may-not matter
  - E.g. conf. not needed when Mozilla distributes a new Firefox binary
- Approach using symmetric-key cryptography:
  - *Integrity via MACs (which use a shared secret key K)*
  - *Authentication arises due to confidence that only Alice & Bob have K*
- Approach using public-key cryptography:
  - *"Digital signatures" provide both integrity & authentication together*

- Key building block: *cryptographically strong hash functions*

Hash Functions

- Properties
  - Variable input size
  - Fixed output size (e.g., 512 bits)
  - Efficient to compute
  - Pseudo-random (mixes up input extremely well)

- Provides a "fingerprint" of a document
  - E.g. "shasum -a 256 <exams/mt1-solutions.pdf" prints 0843b3802601c848f73ccb5013afa2d5c4d424a6ef477890ebf8db9bc4f7d13d

Cryptographically Strong Hash Functions

- A *collision* occurs if x≠y but Hash(x) = Hash(y)
  - Since input size > output size, collisions do happen

- A cryptographically strong Hash(x) provides three properties:
  1. One-way: h = Hash(x) easy to compute, but not to invert. (Vivid image: Hash(*cow*) = *hamburger* 😏.)
     - Intractable to find any x' s.t. Hash(x') = h, for a given h
     - Also termed “preimage resistant”

Cryptographically Strong Hash Functions

- The other two properties of a cryptographically strong Hash(x):
  - Second preimage resistant: given x, intractable to find x' s.t. Hash(x) = Hash(x')
  - Collision resistant: intractable to find *any* x, y s.t. Hash(x) = Hash(y)
- Collision resistant $\Longrightarrow$ Second preimage resistant
  - We consider them separately because given Hash might differ in how well it resists each
  - Also, the *Birthday Paradox* means that for n-bit Hash, finding x-y pair takes only $\approx 2^{n/2}$ pairs
    - Vs. potentially 2^n tries for x': Hash(x) = Hash(x') for given x

Cryptographically Strong Hash Functions, con't

- Some contemporary hash functions
  - MD5: 128 bits broken – lack of collision resistance
  - SHA-1: 160 bits broken (as of last week!)
  - SHA-256: 256 bits *at least not currently broken*
- Provide a handy way to unambiguously refer to large documents
  - If hash can be securely communicated, provides integrity
    - E.g. Mozilla securely publishes SHA-256(new FF binary)
    - Anyone who fetches binary can use "`cat binary | shasum -a 256`" to confirm it's the right one, untampered
- Not enough by themselves for integrity, since functions are completely known – Mallory can just compute revised hash value to go with altered message

Message Authentication Codes (MACs)

- Symmetric-key approach for integrity
  - Uses a shared (secret) key K
- Goal: when Bob receives a message, can confidently determine it hasn't been altered
  - In addition, whomever sent it *must have possessed* K
    (⇒ message authentication)
- Conceptual approach:
  - Alice sends {M, T} to Bob, with tag T = F(K, M)
    - Note, M could instead be C = $E_{K'}$(M), but not required
  - When Bob receives {M', T'}, Bob checks whether T' = F(K, M')
    - If so, Bob concludes message untampered, came from Alice
    - If not, Bob discards message as tampered/corrupted

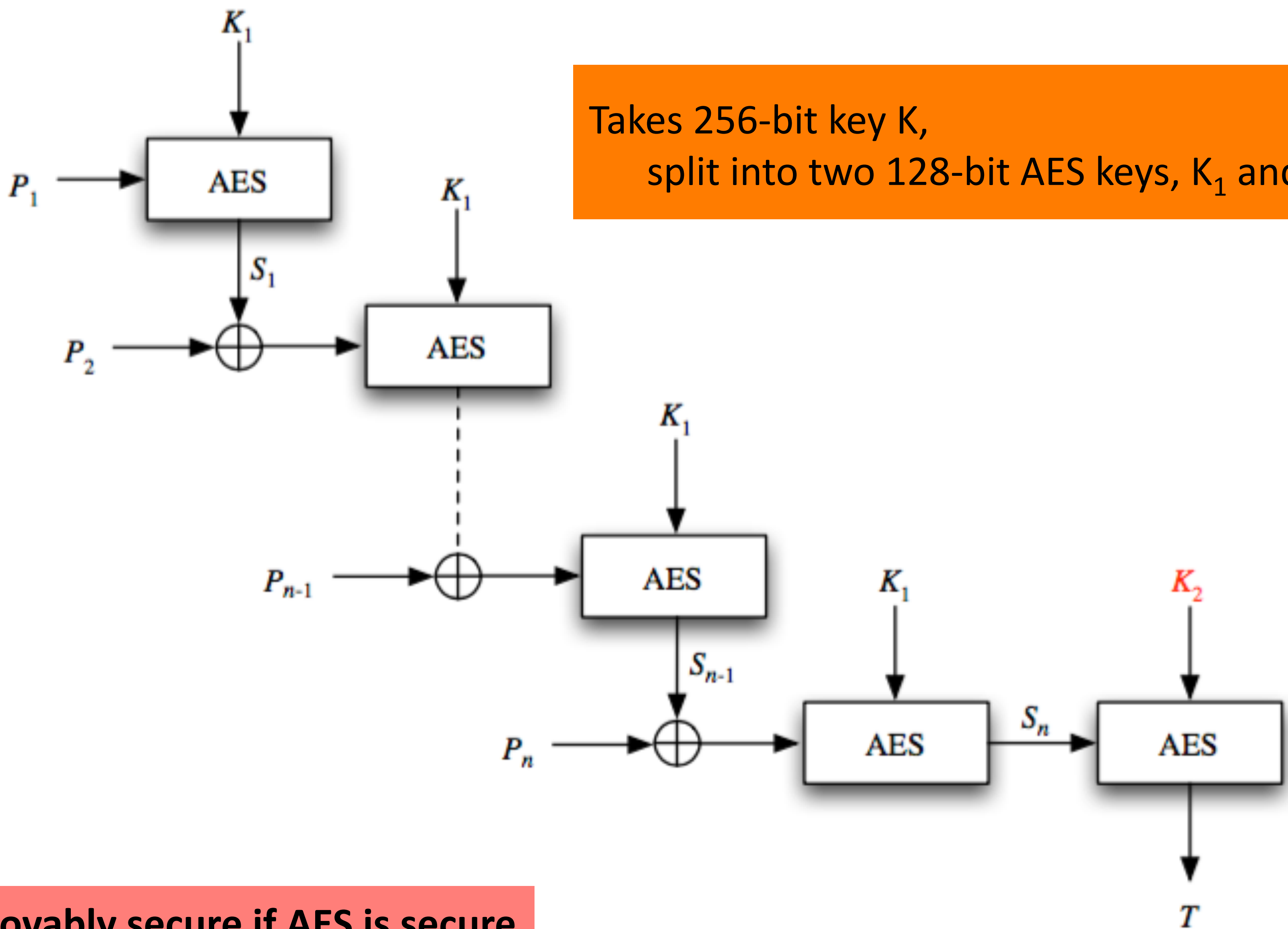
Requirements for Secure MAC Functions

- Suppose MITM attacker *Mallory* intercepts Alice's {M, T} transmission …
  - … and wants to replace M with altered M*
  - … but doesn't know secret key K
- We have secure integrity if MAC function T = F(M, K) has two properties:
  1. Mallory can't compute T* = F(M*, K)
     - Otherwise, could send Bob {M*, T*} and fool him
  2. Mallory can't find M** such that F(M**, K) = T
     - Otherwise, could send Bob {M**, T} and fool him
- *These need to hold even if Mallory can observe many {M_i, T_i} pairs, including for M_i's she chose*

HMAC: Building a MAC Out of a secure hash function

- For a given secret key K & message M, let:
  - H be a cryptographically strong hash function
  - Pad_i, Pad_o = well-known strings
  - K* = a lightly adjusted version of K (padded if K too short)
- HMAC(M, K) = H[(K* $\oplus$ Pad_o) || H((K* $\oplus$ Pad_i) || M)]
- Most widely used MAC on the Internet
- Currently believed to be safe even if underlying hash function is somewhat flawed (e.g., SHA-1)
  - though of course not prudent to bet on that continuing …

AES-EMAC: Building a MAC out of a secure block cipher



Takes 256-bit key K,
split into two 128-bit AES keys, K_1 and K_2

Provably secure if AES is secure

Considerations when using MACs

- Along with messages, can use for data at rest
  - E.g. laptop left in hotel, providing you don't store the key on the laptop
  - Can build an efficient data structure for this that doesn't require re-MAC'ing over entire disk image when just a few files change
- MACs in general provide *no promise not to leak* info about message
  - Though the ones we've seen don't
  - Compute MAC on ciphertext if this matters

Considerations when using MACs, con’t

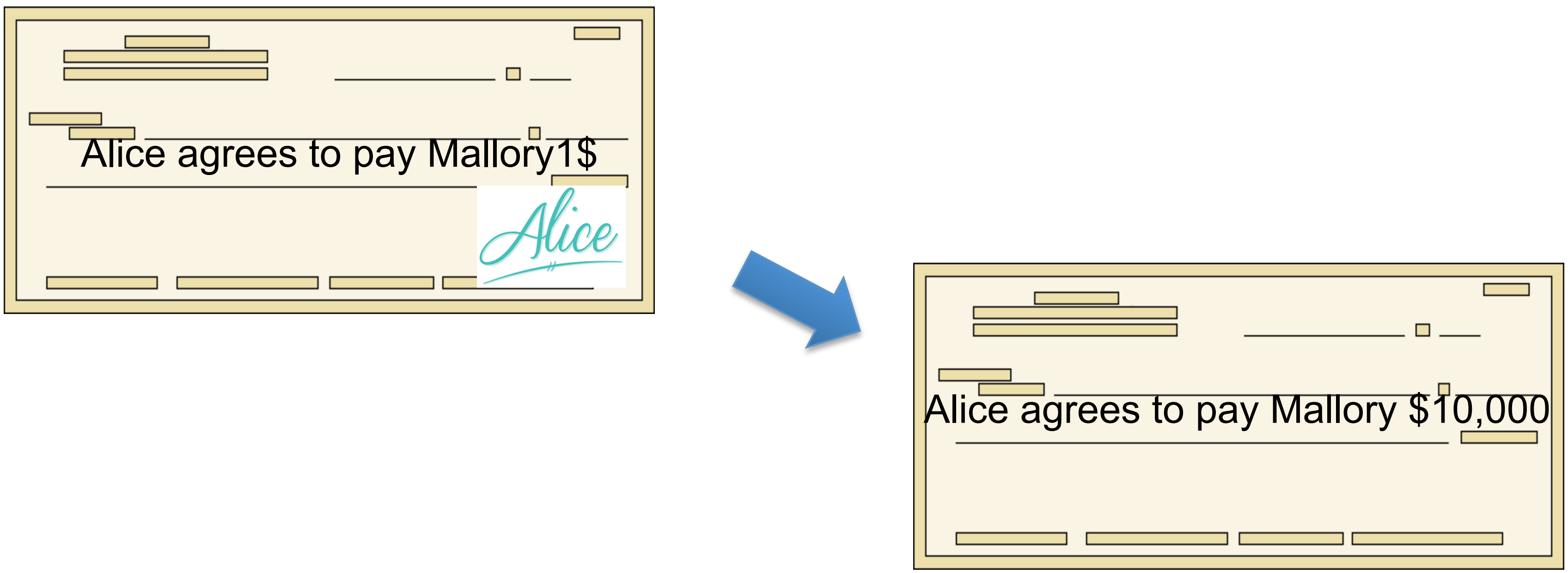
- If also encrypting, do not use the same key to encrypt and for the MAC
  - some MACs can then leak info about crypto stages

- If confidentiality doesn’t matter, fine to send the computed MAC in the clear

Digital Signatures

The Problem with *Digitized* Signatures

Goal: demonstrate that author produced/endorsed document

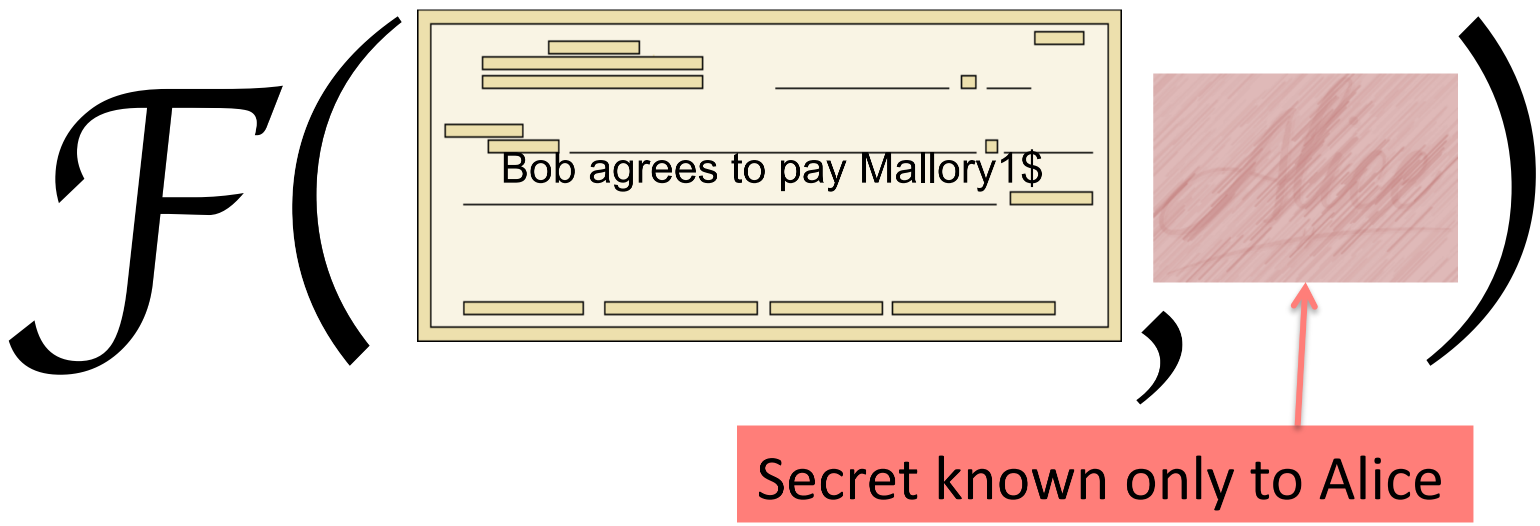


Problem: attacker can copy
Alice's sig from one doc to another

Digital Signatures

Solution: make signature depend on document

Signature S =



Given signature S and document, need to be able to confirm that only Alice could have produced S using some verification function V(S, Alice). Discard as forgery/corrupted if not.

Digital Signatures, con't

- Idea: as with public-key encryption, leverage a function that's easy to compute but intractable to invert … *unless* one possesses some private information
  - But instead, do this for a function that's hard to compute without private info, but easy to invert
- One way to produce such a function: use the inverse of a public-key encryption function
- *For example*, consider RSA ...