Asymmetric and Public Key Signatures

Ruta Jawale

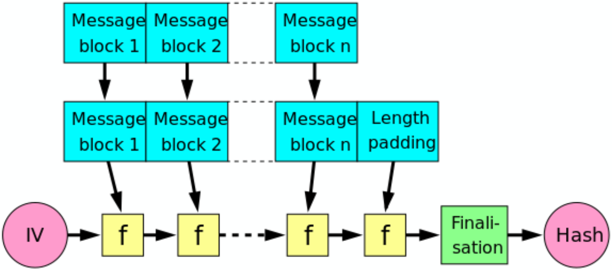
July 9, 2019

Announcements

- Homework 1 will be due **today**! (7/9)
- Project 1 due Thursday! (7/11)
  - Project 1's VM passwords are released
  - If you have a partner, only one submission per group
- Midterm 1 in one week! (7/15)

Hash functions constructions 🏠

Merkle-Damgard construction (used by SHA1, SHA2):

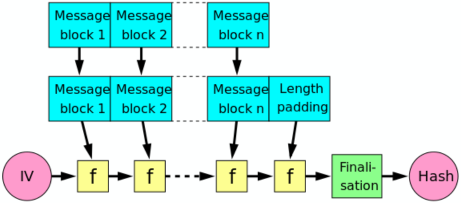


Let N be the message block size in bits. IV is some fixed value, f is some one-way compression function.

Length extension attack

Let H be a hash function depending on Merkle-Damgard. Let PAD be the hash function's internal padding scheme.

An attacker can use the digest $H(m_1)$ for some unknown message m_1 of known length to calculate $H(PAD(m_1) \| m_2)$ for a message m_2 of the attacker's choosing.



SHA3 is not vulnerable to this form of attack.

Symmetric key vs. Public key encryption

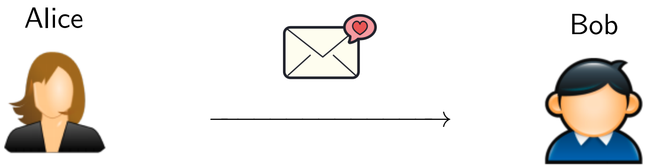
- Symmetric key encryption
  - Inconvenient: need to set up a shared, symmetric key somehow
  - Efficient: bitwise operations are efficient to implement (xor, shift), also can be parallelized
  - Quantum resistant: double the key size!
- Public key encryption
  - Convenient: easy to create public/private key pairs for each person
  - Inefficient: exponentiation of large integers is very slow
  - RSA and El Gamal are broken by quantum computers! Shor's algorithm breaks factorization and discrete log assumptions.

Hybrid encryption

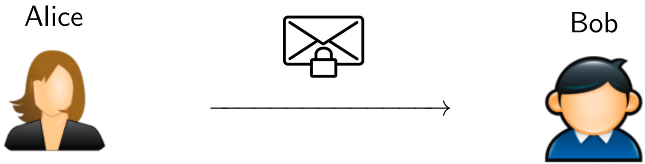
Hybrid encryption is where we use public key encryption to set up a shared secret key, then we use symmetric key encryption to encrypt messages.

The story so far. . .

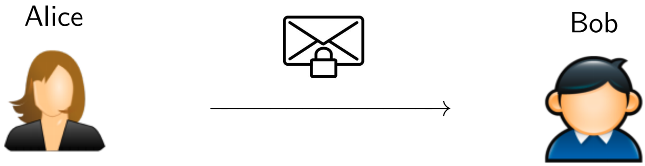
Alice wants to ask Bob on a date. She now knows that if she wants confidentiality. . .



. . . she needs to encrypt her message!



Which encryption?



Let's say Alice prefers symmetric key encryption, so she uses Diffie-Hellman to set up a symmetric key with Bob (if she doesn't have one already), then uses AES-CFB.

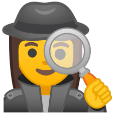
Reminder: Alice's security specifications

- Confidentiality
  - only Alice and Bob should know the message
- Integrity
  - Bob should be able to verify Alice's message was not modified or tampered with
    - If it was modified, Bob should realize it!
- Authentication
  - Bob should be able to verify Alice sent the message

All about Eve

Excepting Mallory's brief cameo (see MITM attacks), so far it's been all about Eve.

Eve the Eavesdropper




Likes: Reading messages
Dislikes: Confidentiality

Mallory the Manipulator

Likes: Altering messages
Dislikes: Integrity/Authenticity

Today let's talk about Mallory!

Achieve integrity/authenticity to upset Mallory

Mallory likes to manipulate messages. How can we ensure that Mallory can't tamper with Alice's correspondence? *(Another rhetorical question)*

Let's send a "tag" with Alice's message!

Types of “tags”

Signature key

Verification key

- Symmetric key “tag”  $=$ 
  - same private key for signing and verifying
- Asymmetric key “tag”  $\neq$ 
  - separate public verification key and private signing key

Both types of “tags” achieve integrity/authenticity, necessary to prevent Mallory’s plans. We’ll see both in today’s lecture.

Learning Objectives

- Learn a symmetric key integrity/authenticity
  - MAC (ex: HMAC)
- Learn asymmetric key integrity/authenticity
  - Digital signatures (ex: RSA signature)

Message Authentication Codes (MACs)

Gen$(1^n) \to k$:

Input: 1^n where n is the security parameter
Output: secret key k

Sign$(k, m) \to \sigma$:

Input: secret key k and message m
Output: signature σ

Verify$(k, m, \sigma) \to \{0, 1\}$:

Input: secret key k, message m, and signature σ
Output: 1 on success, 0 otherwise

Important: We will write $MAC(k, m)$ or $MAC_k(m)$ or MIC when we mean **Sign**(k, m) to avoid confusion with digital signatures!

MAC correctness

$$\forall m \qquad \textbf{Verify}(k, m, \textbf{Sign}(k, m)) = 1$$

MAC security: unforgeability 

Phases	Challenger		Adversary $\mathcal{A}$
setup	$k \leftarrow \textbf{Gen}(1^n)$		
signature query or		$\xleftarrow{m_i \text{ or } (m_j, \sigma_j)}$	for $i, j \in \text{poly}(n)$
verification query	$\sigma_i \leftarrow \textbf{Sign}(k, m_i)$ or		where $(m_j, \sigma_j) \neq (m_i, \sigma_i)$
	$b \leftarrow \textbf{Verify}(k, m_j, \sigma_j)$		
determine win	$b = 1$, $\mathcal{A}$ wins	$\xrightarrow{\sigma_i \text{ or } b = 0}$	

MAC security: unforgeability 

Phases	Challenger		Adversary $\mathcal{A}$
setup	$k \leftarrow \textbf{Gen}(1^n)$		
signature query or		$\xleftarrow{m_i \text{ or } (m_j, \sigma_j)}$	for $i, j \in \text{poly}(n)$
verification query	$\sigma_i \leftarrow \textbf{Sign}(k, m_i)$ or		where $(m_j, \sigma_j) \neq (m_i, \sigma_i)$
	$b \leftarrow \textbf{Verify}(k, m_j, \sigma_j)$		
determine win	$b = 1$, $\mathcal{A}$ wins	$\xrightarrow{\sigma_i \text{ or } b = 0}$	

What does $(m_j, \sigma_j) \neq (m_i, \sigma_i)$ mean? Cannot submit verification queries on signature queries. In other words, cannot claim something signed by the challenger is a forgery of the challenger's signature.

MAC security: unforgeability 

Phases	Challenger		Adversary $\mathcal{A}$
setup	$k \leftarrow \mathbf{Gen}(1^n)$		
signature query or		$\xleftarrow{m_i \text{ or } (m_j, \sigma_j)}$	for $i, j \in \text{poly}(n)$
verification query	$\sigma_i \leftarrow \mathbf{Sign}(k, m_i)$ or		where $(m_j, \sigma_j) \neq (m_i, \sigma_i)$
	$b \leftarrow \mathbf{Verify}(k, m_j, \sigma_j)$		
determine win	$b = 1$, $\mathcal{A}$ wins	$\xrightarrow{\sigma_i \text{ or } b = 0}$	

$$\Pr[\mathcal{A} \text{ wins game}] = \textit{negligible}.$$

MAC has integrity/authenticity

- Integrity
  - No one can forge a valid "tag" without knowing the key
  - So if Mallory changes Alice's message, Mallory can't forge a matching "tag"
  - When Bob goes to verify the "tag", he can determine if Mallory changed Alice's message
- Authenticity
  - No one can forge a valid "tag" without knowing the key
  - So only Alice and Bob can create a correct "tag"
  - Knowing the key, authenticates Alice and Bob

Does MAC provide confidentiality?

Given just the output of a MAC, is the input to the MAC confidential?

No, not in general. We can construct a MAC that leaks the entire message and is still unforgeable.

MAC has plausible deniability

In our scenario, Alice could deny that she sent the message and claim Bob sent the message to her. A third person cannot determine which one of them, Alice or Bob, sent the "tag", since both Alice and Bob know the same key.

HMAC

Let $H : \{0,1\}^* \rightarrow \{0,1\}^n$ be some hash function of our choice.

We want to use this hash function to construct a MAC scheme!

Idea # 1

Let $H : \{0,1\}^* \rightarrow \{0,1\}^n$ be some hash function of our choice.

Gen$(1^n) \rightarrow k$:

- $k \xleftarrow{\$} \{0,1\}^n$

Sign$(k, m) \rightarrow \sigma$:

- output $H(k \| m)$

Verify$(k, m, \sigma) \rightarrow \{0,1\}$:

- check $H(k \| m) \stackrel{?}{=} \sigma$
- if equal, output 1. else, output 0.

What if the hash function we use is SHA-256 or SHA-512?

Given a valid signature

$$\sigma = H(k \| m_1)$$

adversary could forge the signature

$$\sigma^* = H(k \| PAD(m_1) \| m_2)$$

using length extension attack!

Idea # 1: Can easily break unforgeability

Phases	Challenger		Adversary $\mathcal{A}$
setup	$k \leftarrow \textbf{Gen}(1^n)$		
signature query or verification query		$\xleftarrow{m_1}$	
	$H(k\|m_1) \leftarrow \textbf{Sign}(k, m_1)$ or	$\xrightarrow{H(k\|m_1)}$	
		$\xleftarrow{(m^*, \sigma^*)}$	$m^* = PAD(m_1)\|m_2$
			compute $\sigma^* = H(k\|m^*)$ using length extension
	$b \leftarrow \textbf{Verify}(k, m^*, \sigma^*)$		
determine win	$b = 1$, $\mathcal{A}$ wins		

$$\Pr[\mathcal{A} \text{ wins game}] = 1.$$

Idea # 2

Let $H : \{0,1\}^* \to \{0,1\}^n$ be some hash function of our choice.

Gen$(1^n) \to k$:

- $k \xleftarrow{\$} \{0,1\}^n$

Sign$(k, m) \to \sigma$:

- output $H(k \| H(k \| m))$

Verify$(k, m, \sigma) \to \{0,1\}$:

- check $H(k \| H(k \| m)) \stackrel{?}{=} \sigma$
- if equal, output 1.
  else, output 0.

Is it unforgeable?

No known length extension attacks! The outer hash function appears to hide the inner hash functions's internal state.

However, we shouldn't use the same key twice.

HMAC

Let $H : \{0,1\}^* \rightarrow \{0,1\}^n$ be some hash function of our choice.

$$\text{HMAC}(K, m) = \text{H}(\quad (K' \oplus opad) \quad \| \quad H(\ (K' \oplus ipad) \| \ m\ ) \quad)$$

$$\begin{aligned} \text{where} \quad opad &= n \text{ bit block of repeating ``0x5c''} \\ ipad &= n \text{ bit block of repeating ``0x36''} \end{aligned}$$

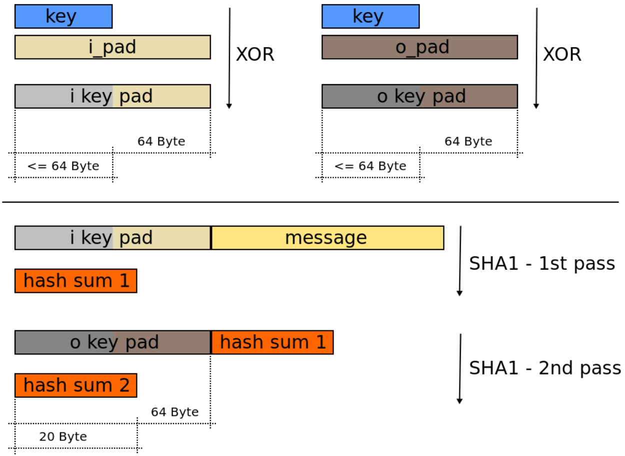
$$K' = \begin{cases} K \| \text{``0x00''} & K \text{ is shorter than block size } n \\ \text{H}(K) & K \text{ is larger than block size } n \\ K & \text{otherwise} \end{cases}$$

HMAC

```
function hmac (key, message) {
    if (length(key) > blocksize) {
        key = hash(key)
    }
    while (length(key) < blocksize) {
       key = key || 0x00
    }
    o_key_pad = 0x5c5c... ⊕ key
    i_key_pad = 0x3636... ⊕ key
    return hash(o_key_pad ||
                hash(i_key_pad || message))
}
```

HMAC constructions 🏠

- HMAC-SHA1



Announcements
Review
Objectives
MAC
HMAC
Digital Signatures
RSA Signatures
Conclusion
Summary

HMAC constructions

- HMAC-SHA1
  - SHA1 is insecure!
- HMAC-SHA256
  - block size: 512 bits or 64 bytes
- HMAC-SHA512
  - block size: 1024 bits or 128 bytes
- HMAC-SHA3
  - No length extension attack!
  - Could actually use Idea # 1: $H(k \| m)$

Does HMAC provide confidentiality?

Yes. If the underlying hash function has pre-image resistance, then HMAC should not leak much information about its input.

Where's the reduction proof? *Left as exercise to the reader.*

Break time∼

Stand up, stretch, ask a neighbor how they're planning to study for the midterm.

Coming up next: public key signatures

Digital Signatures

$\textbf{Gen}(1^n) \to (vk, sk)$:

Input: 1^n where n is the security parameter
Output: secret signing key sk and public verification key vk

$\textbf{Sign}(sk, m) \to \sigma$:

Input: secret key sk, and message m
Output: signature σ

$\textbf{Verify}(vk, m, \sigma) \to \{0, 1\}$:

Input: verification key vk, message m, and signature σ
Output: 1 on success, otherwise 0.

Digital Signatures correctness

$$\forall m \qquad \textbf{Verify}(vk, m, \textbf{Sign}(sk, m)) = 1$$

Reminder: verification key vk is public, signing key sk is private

Digital Signatures security: unforgeability

Phases	Challenger		Adversary $\mathcal{A}$
setup	$vk, sk \leftarrow \textbf{Gen}(1^n)$	$\xrightarrow{vk}$	
signature query or		$\xleftarrow{m_i \text{ or } (m_j, \sigma_j)}$	for $i, j \in \text{poly}(n)$
verification query	$\sigma_i \leftarrow \textbf{Sign}(sk, m_i)$ or		where $(m_j, \sigma_j) \neq (m_i, \sigma_i)$
	$b \leftarrow \textbf{Verify}(vk, m_j, \sigma_j)$		
determine win	$b = 1$, $\mathcal{A}$ wins	$\xrightarrow{\sigma_i \text{ or } b = 0}$	

Digital Signatures security: unforgeability

Phases	Challenger		Adversary $\mathcal{A}$
setup	$vk, sk \leftarrow \textbf{Gen}(1^n)$	$\xrightarrow{vk}$	
signature query or		$\xleftarrow{m_i \text{ or } (m_j, \sigma_j)}$	for $i, j \in \text{poly}(n)$
verification query	$\sigma_i \leftarrow \textbf{Sign}(sk, m_i)$ or		where $(m_j, \sigma_j) \neq (m_i, \sigma_i)$
	$b \leftarrow \textbf{Verify}(vk, m_j, \sigma_j)$		
determine win	$b = 1$, $\mathcal{A}$ wins	$\xrightarrow{\sigma_i \text{ or } b = 0}$	

Adversary $\mathcal{A}$ has the verification key *vk*. They don't need to ask the challenger for verification queries. They only need to submit the forged signature.

Digital Signatures security: unforgeability

Phases	Challenger		Adversary $\mathcal{A}$
setup	$vk, sk \leftarrow \textbf{Gen}(1^n)$	$\xrightarrow{vk}$	
signature query		$\xleftarrow{m_i}$	for $i \in \text{poly}(n)$
	$\sigma_i \leftarrow \textbf{Sign}(sk, m_i)$	$\xrightarrow{\sigma_i}$	
forgery		$\xleftarrow{m^*, \sigma^*}$	where $(m^*, \sigma^*) \neq (m_i, \sigma_i)$
	$b \leftarrow \textbf{Verify}(vk, m^*, \sigma^*)$		
determine win	If $b = 1$, $\mathcal{A}$ wins		

$$\Pr[\mathcal{A} \text{ wins game}] = \textit{negligible}.$$

Do Digital Signatures have non-repudiation?

Non-repudiation is the assurance that someone cannot deny the validity of something. The opposite of deniability.

Can we determine whether Alice sent the message to Bob or Bob sent the message to Alice?

Yes, depending on who's public key / private key pairing was used, so Digital Signatures have non-repudiation.

RSA signatures: key generation

Reminder: verification key vk is public, signing key sk is private

$\mathbf{Gen}(1^n) \to (vk, sk)$:

- choose primes p and q
- define $N = p \cdot q$
- choose small prime $e \in \{1, \ldots, N-1\}$
- compute d to satisfy $e \cdot d = 1 \pmod{(p-1)(q-1)}$
- define $vk = (N, e)$ and $sk = d$

RSA signatures: signature 🏠

Let H be a cryptographic hash function.

$\mathbf{Sign}(sk, m) \rightarrow \sigma$:

- compute $\sigma = H(m)^d \pmod{N}$

RSA signatures: verification 🏠

Let H be our cryptographic hash function.

Verify$(vk, m, \sigma) \rightarrow \{0, 1\}$:

- check $H(m) \stackrel{?}{=} \sigma^e \pmod{N}$
- if equal, output 1. else, output 0

Is RSA signatures correct?

Gen$(1^n) \to (vk, sk)$:

- choose primes p and q
- define $N = p \cdot q$
- $e \in \{1, \ldots, N-1\}$
- compute d s.t. $e \cdot d = 1 \pmod{(p-1)(q-1)}$
- $vk = (N, e)$, $sk = d$

Sign$(sk, m) \to \sigma$:

- $\sigma = H(m)^d \pmod{N}$

Verify$(vk, m, \sigma) \to \{0, 1\}$:

- $H(m) \stackrel{?}{=} \sigma^e \pmod{N}$
- if equal, output 1

Does **Verify**(vk, m, σ) return 1?

$$\begin{aligned}\sigma^e \pmod{N} &= (H(m)^d \pmod{N})^e \pmod{N} \\ &= H(m)^{e \cdot d} \pmod{N} = H(m) \pmod{N}\end{aligned}$$

by application of the Chinese Remainder Theorem.

Why hash the message?

Gen$(1^n) \to (vk, sk)$:

- choose primes p and q
- define $N = p \cdot q$
- $e \in \{1, \ldots, N-1\}$
- compute d s.t. $e \cdot d = 1 \pmod{(p-1)(q-1)}$
- $vk = (N, e)$, $sk = d$

Sign$(sk, m) \to \sigma$:

- $\sigma = H(m)^d \pmod{N}$

Verify$(vk, m, \sigma) \to \{0, 1\}$:

- $H(m) \stackrel{?}{=} \sigma^e \pmod{N}$
- if equal, output 1

You'll see why during Wednesday's discussion section.

Reminder: Attend discussion sections!

Is RSA signature unforgeable?

Assuming the hash function is secure, the RSA signature will be unforgeable.

Can Alice just use HMAC since it has CIA?

No, remember how a “tag” is used in our scenario when Alice wants to send a message. The message also needs to be sent for Bob to verify!

Lucky 7 step plan

Alice still prefers symmetric keys, so she will use Diffie-Hellman to set up **two** symmetric keys with Bob (if she doesn't have two already). One for encryption and one for "tag" generation.

Then she follows "encrypt then MAC" strategy: she first encrypts her message using AES-CFB and then appends a tag of the ciphertext using HMAC-SHA256.

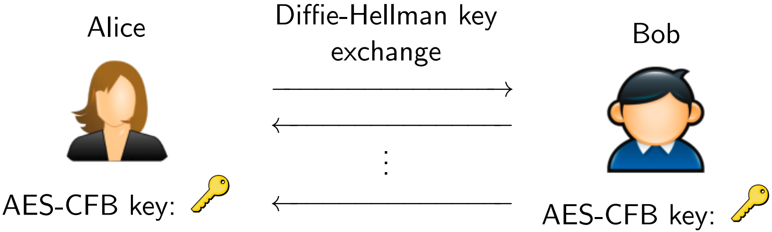
Announcements
Review
Objectives
MAC
HMAC
Digital Signatures
RSA Signatures
Conclusion
Summary

Let's put it together

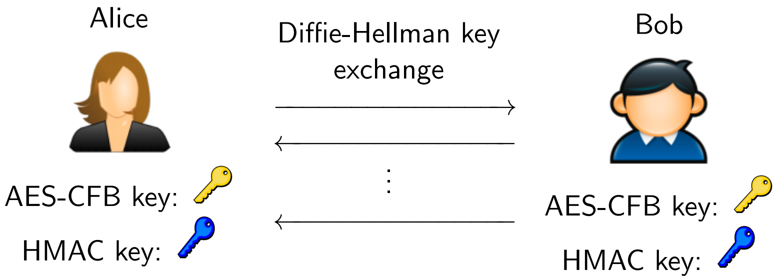
Alice

Bob

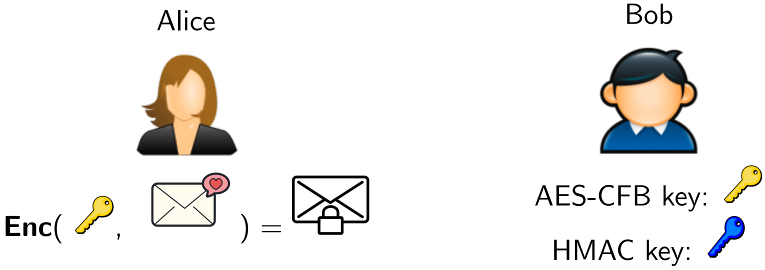
Step # 1: Diffie-Hellman to share encryption key



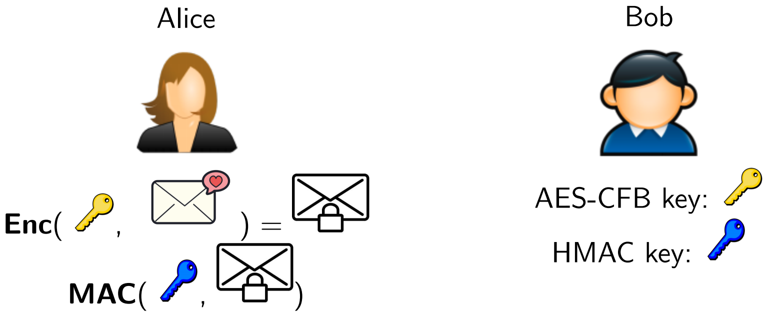
Step # 2: Diffie-Hellman to share MAC key



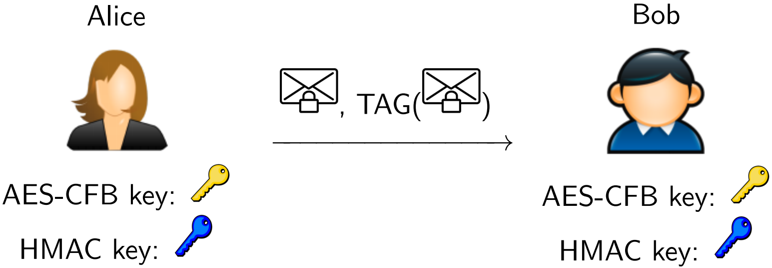
Step # 3: Encrypt the message using AES-CFB



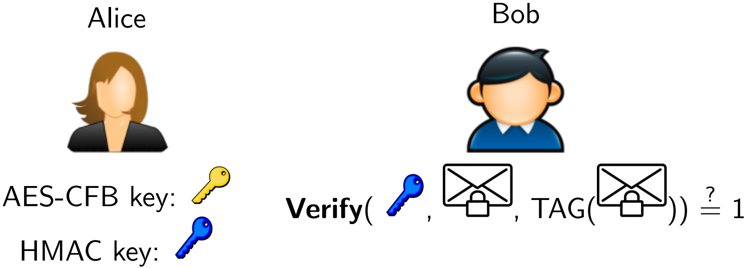
Step # 4: Compute the “tag” using HMAC



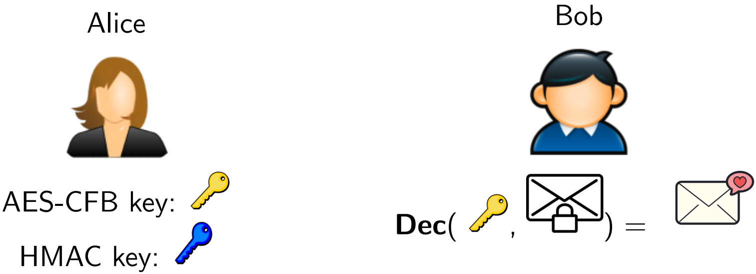
Step # 5: Send message via “Encrypt then MAC”



Step # 6: Bob will verify the "tag"



Step # 7: Bob will decrypt the message



Fin∼

Alice

Bob

Epilogue 

Announcements
Review
Objectives
MAC
HMAC
Digital Signatures
RSA Signatures
Conclusion
Summary

Since our symmetric key systems, such as MAC, have deniability. . .

Many years later, Bob still jokingly insists that he was the one who asked Alice out first!

Alice learned today that . . .

- If she wants integrity and authentication,
  - she can use HMAC or RSA Signatures
- How to finally ask Bob on a date!
  - "Encrypt then MAC"