Computer Network Security

ECE 4112/6612
CS 4262/6262

Prof. Frank Li

```
* Welcome to CityPower Grid Rerouting *
Authorised Users only!
New users MUST notify Sys/Ops.

login:
```

```
EDITV1 sshnuke
rcr ebx, 1
bsr ecx, ecx
shrd ebx, edi, CL
```

```
80/tcp
81/tcp     open             http
```

```
# nmap -v -sS -O 10.2.2.2
Starting nmap V. 2.54BETA25
Insufficient responses for TCP sequencing (3), OS detection may be less
accurate
Interesting ports on 10.2.2.2:
(The 1539 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh

No exact OS matches for host

Nmap run completed -- 1 IP address (1 host up) scanneds
# sshnuke 10.2.2.2 -rootpw="Z10N0101"
Connecting to 10.2.2.2:ssh ... successful.
Attempting to exploit SSHv1 CRC32 ... successful.
Reseting root password to "Z10N0101".
System open: Access Level <9>
# ssh 10.2.2.2 -l root
root@10.2.2.2's password:
```

* Slides adopted from Prof. Manos Antonakakis

Logistics

- HW1 due next Monday midnight

- For those doing the project, start forming teams + brainstorming ideas (feel free to run initial ideas by me in OHs, Piazza, etc.)

Continuing with hashes/message digests

Message Authentication Code (MAC)

- Designed to provide both *authentication* and *integrity*.
- How can we use hash functions to compute a MAC?
  - *H* is a hash function
  - *m* is a message
  - *K* is a secret key.
- Let's talk about different constructions
  - Secret Prefix Construction => H(K ll m)
  - Secret Suffix Construction => H(m ll K)
  - HMAC => H((K⊕opad) ll H((K⊕ipad) ll m))

Secret Prefix Construction

$$MAC = H(K \,||\, m)$$

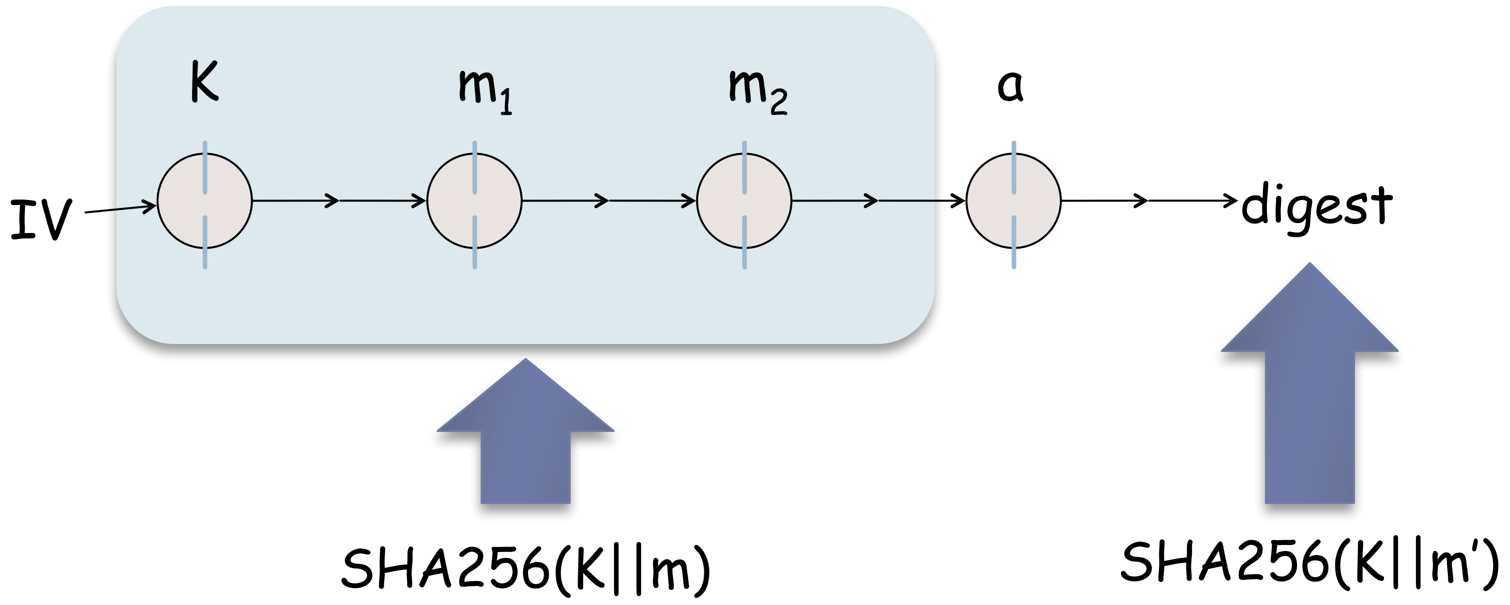
H = cryptographic hash function

K = secret key

m = message to send

Uh oh! Length Extension Attack!

- Because most hash functions are iterated hash functions
  - Attacker knows the message m and H(K ll m)
  - They could append something to m to get m' = m ll a, and use H(K ll m) to initialize the computation of H(K ll m')



Secret Suffix Construction

$$MAC = H(m \| K)$$

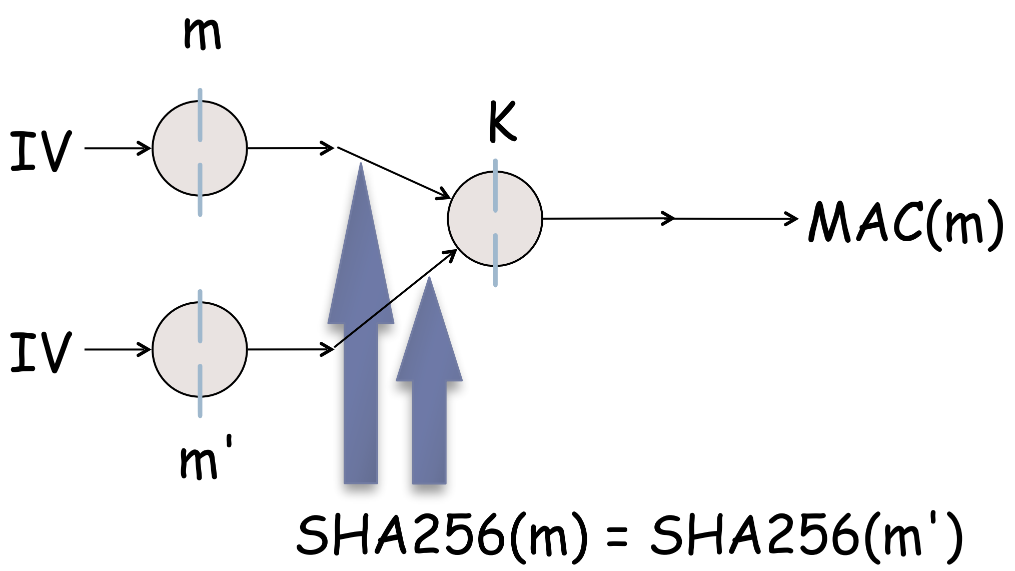
H = cryptographic hash function

K = secret key

m = message to send

Secret Suffix Construction

- Better! But subject to weakness in the hash function.
  - Say the attacker can find a hash collision with the original message m, so an m' where H(m') = H(m).
  - If H is an iterated hash function, MAC(m') = MAC(m)
  - Even w/o the secret key, attacker can find MAC collisions.



HMACs

$$\text{HMAC} = H(K \,||\, H(K \,||\, m))$$

H = cryptographic hash function

K = secret key

m = message to send

HMAC

$$\text{HMAC} = H(K \,||\, H(K \,||\, m))$$

- Inner + outer layers of hashing is much stronger!
- Resistant to length extension attack
- Technically can use same key, but in practice, slightly alters the keys

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

where *ipad* and *opad* are standardized values

Public Key Cryptography

Public Key Cryptography

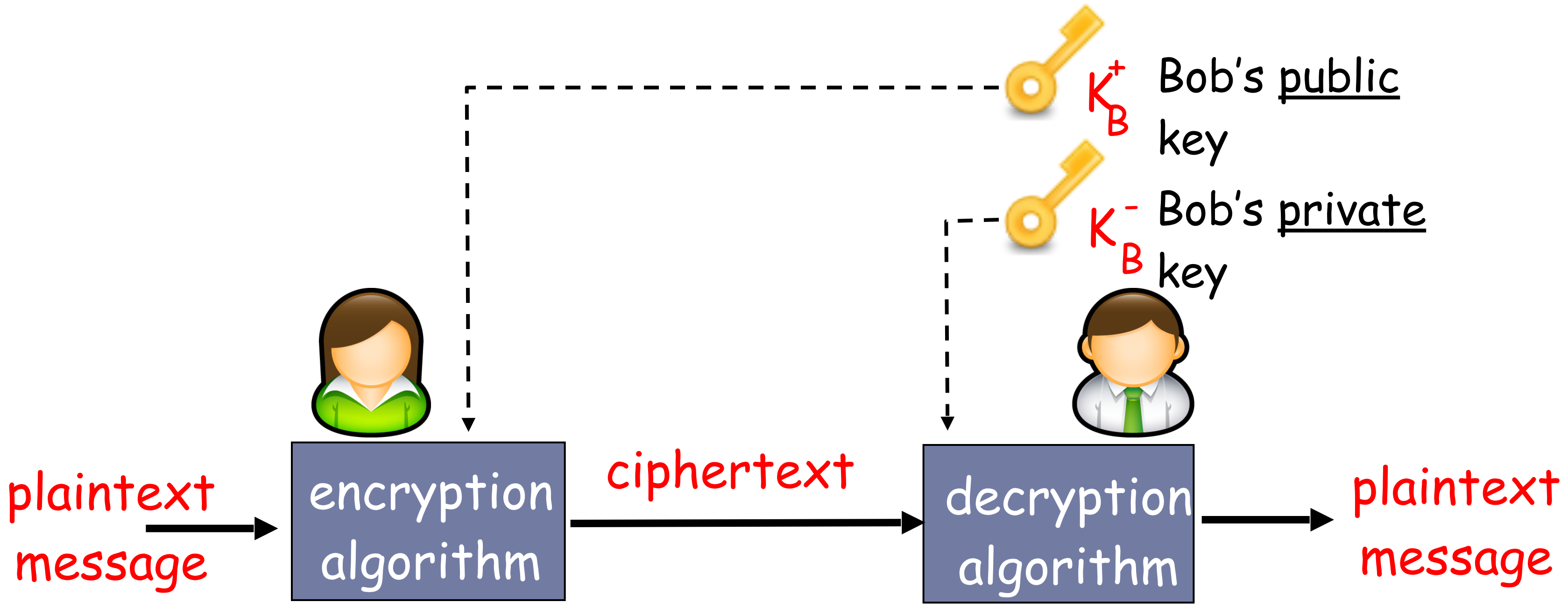
Symmetric Key Crypto

- Requires sender and receiver know shared secret key
- What's the challenge?
    - Key distribution is hard and fraught with peril!

Public Key Cryptography

- Radically different approach [Diffie-Hellman76, RSA78]
- Sender and receiver do not share secret key
    - Receiver has a public encryption key known to all
    - Receiver has a private decryption it keeps secret

Public Key Cryptography



Computational Hardness

- Why do we care about computational hardness?
  - Hard computational problems are cornerstone of modern cryptography.
  - Computational hardness will be the basis of the security for the two public key algorithms we discuss.

The Discrete Logarithm Problem (DLP)

Given x where $x \equiv g^y \bmod p$, find y.

g is a number within group Z_p^*, for prime p.

- What is Z_p^*?
  - Z_p^* is the multiplicative group of integers modulo prime p.
    - It contains the set of all integers from 1 to p-1, so {1, 2,..., p-1}.
    - Only operation permitted is multiplying group members mod p.
  - *Group axioms* (required properties) include closure, associativity, identity existence, and inverse existence
- Is DLP NP-Complete?
  - This problem is NP and seems difficult but probably not NP-complete.

Diffie-Hellman (DH) Key Exchange

- First developed public key cryptosystem
  - Useful to perform key exchange when communication channel is not private
- How does it work?
  1. Two prime numbers **g** and **p** are publicly known (e.g., standard values).
  2. Alice picks a secret number **a,** computes $\mathbf{g^a}$ *mod* **p** = **A,** and sends **A** to Bob.
  3. Bob picks a secret number **b,** computes $\mathbf{g^b}$ *mod* **p** = **B,** and sends **B** to Alice.
  4. Alice takes her secret number **a,** and the **B** received from Bob, and computes the shared secret key as: $\mathbf{B^a}$ *mod* **p = $\mathbf{g^{ab}}$** *mod* **p**.
  5. Bob takes his secret number **b,** and the **A** received from Alice, and computes the shared secret key as: $\mathbf{A^b}$ *mod* **p = $\mathbf{g^{ab}}$** *mod* **p**.

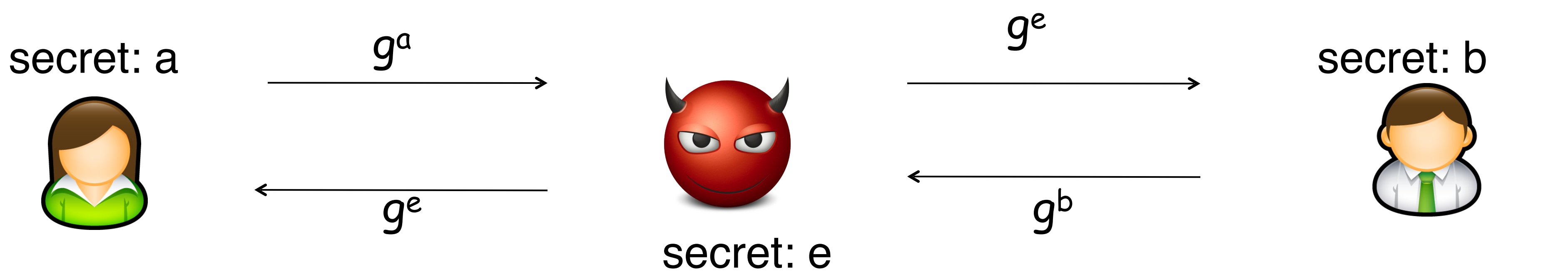
$$(g^a \bmod p)^b \bmod p = g^{ab} \bmod p$$
$$(g^b \bmod p)^a \bmod p = g^{ba} \bmod p$$

Diffie-Hellman (DH) Key Exchange

- First developed public key cryptosystem
  - Useful to perform key exchange when communication channel is not private
- How does it work?
  1. Two prime numbers **g** and **p** are publicly known (e.g., standard values).
  2. Alice picks a secret number **a,** computes $\mathbf{g}^{a}$ *mod* **p** = **A,** and sends **A** to Bob.
  3. Bob picks a secret number **b,** computes $\mathbf{g}^{b}$ *mod* **p** = **B,** and sends **B** to Alice.
  4. Alice takes her secret number **a,** and the **B** received from Bob, and computes the shared secret key as: $\mathbf{B}^{a}$ *mod* **p = $\mathbf{g}^{ab}$** *mod* **p**.
  5. Bob takes his secret number **b,** and the **A** received from Alice, and computes the shared secret key as: $\mathbf{A}^{b}$ *mod* **p = $\mathbf{g}^{ab}$** *mod* **p**.

  - Both parties how have the same secret key, and note that the secret key itself was not directly sent (i.e., $\mathbf{g}^{ab}$ *mod* **p** was never directly sent)
  - Figuring out **a** from **A**=$\mathbf{g}^{a}$ *mod* **p** would require solving DLP (same for **b**).

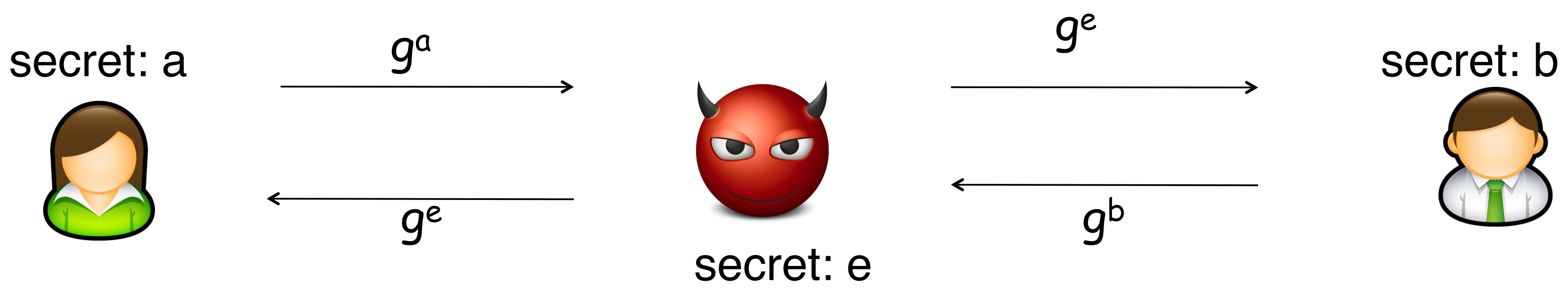
DH: Man-in-the-Middle Attack



Alice-Eve
Shared Key
g^{ae} mod p

Bob-Eve
Shared Key
g^{eb} mod p

DH: Man-in-the-Middle Attack



Alice-Eve
Shared Key
g^{ae} mod p

Bob-Eve
Shared Key
g^{eb} mod p

Need some way to verify that the public key Bob receives is truly from Alice (and vise versa). How to solve? (End of today's lecture)

The Factoring Problem

Given a large number N = p x q, find the prime factors p and q

- Factoring Large Numbers in Practice
  - The fastest factoring algorithm for large numbers is the general number field sieve (GNFS).
  - GNFS complexity is (roughly) = $\exp(1.92 \times (\ln n)^{1/3} (\ln \ln n)^{2/3})$
- Is Factoring NP-Complete?
  - Factoring is NP and "seems hard to solve" (i.e., we don't know how to do so efficiently).
  - However, probably not NP-complete, but no proof for that.

RSA Overview

- A message is a bit pattern.
- A bit pattern can be uniquely represented by an integer number.
- Thus encrypting a message is equivalent to encrypting a number.

Example

- m = 10010001 = 145
  - This message is uniquely represented by the decimal number 145.
  - To encrypt m, we encrypt the corresponding number, which gives a new number (the ciphertext).

RSA: Creating Public/Private Key Pair

1. Choose two large prime numbers p, q. (e.g., 1024 bits each)
2. Compute n = pq, z = (p-1)(q-1)
3. Choose e (with e<n) that has no common factors with z. (e, z are "relatively prime").
4. Choose d such that ed-1 is exactly divisible by z. (in other words: ed mod z = 1).
5. Public key is (n,e). Private key is (n,d).

RSA: Encryption and Decryption

0. Given $\overset{\text{public}}{(n,e)}$ and $\overset{\text{private}}{(n,d)}$ as computed above

1. To encrypt message m (<n), compute
   $c = m^e \bmod n$

2. To decrypt received bit pattern, c, compute
   $m = c^d \bmod n$

Magic happens!

$$m = (\underbrace{m^e \bmod n}_{c})^d \bmod n$$

RSA: Example

Bob chooses $p=5$, $q=7$. Then $n=35$, $z=24$.

$e=5$ (so e, z relatively prime).

$d=29$ (so $ed-1$ exactly divisible by z)

$ed = 145 \equiv 1 \bmod z$ (b/c $144/24=6$)

Encrypting 8-bit messages.

	bit pattern	m	m^e	$c = m^e \bmod n$
encrypt:	0000l000	12	24832	17

	c	c^d	$m = c^d \bmod n$
decrypt:	17	481968572106750915091411825223071697	12

RSA

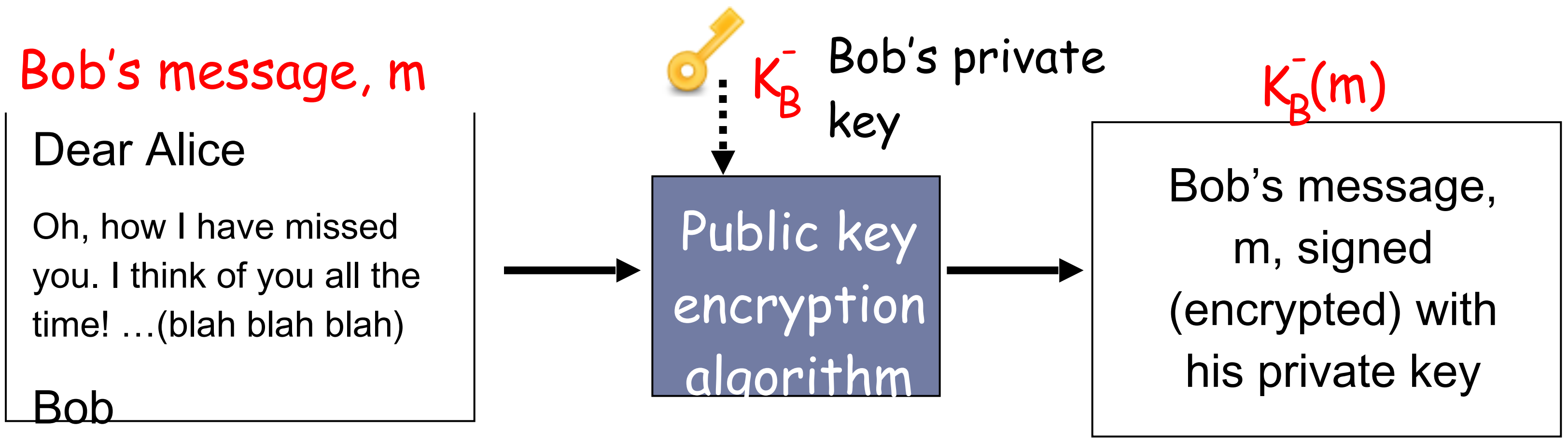
- Why is this secure?
  - Suppose you know Bob’s public key (n,e). It’s really hard to determine *d* without knowing the factors of *n* (factoring is hard!)

- How to handle large messages?
  - Combine RSA with symmetric crypto.
  - Use RSA to securely transmit a symmetric key, then use symmetric cipher to encrypt message.
  - RSA is slow anyways, AES is fast

Digital Signature

Simple digital signature for message m:

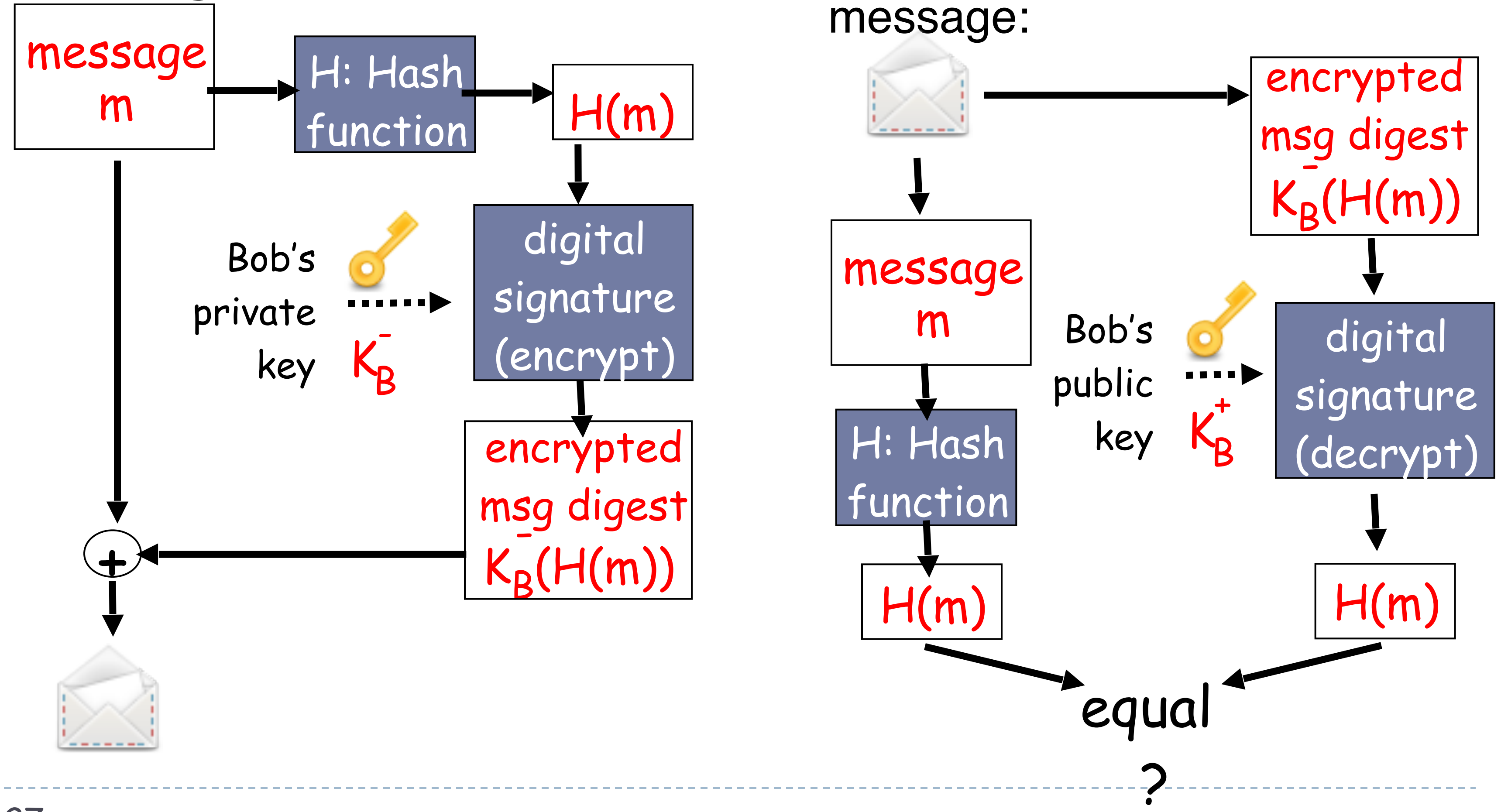
- Bob signs m by encrypting with his private key K_B^-, creating "signed" message, $K_B^-(m)$. (Could use RSA)


Bob's message, m
Dear Alice
Oh, how I have missed you. I think of you all the time! …(blah blah blah)
Bob
K_B^-
Bob's private key
Public key encryption algorithm
$K_B^-(m)$
Bob's message, m, signed (encrypted) with his private key

Digital Signature with Hash

Bob sends digitally signed message:

Alice verifies signature and integrity of digitally signed message:



Digital Signatures Summary

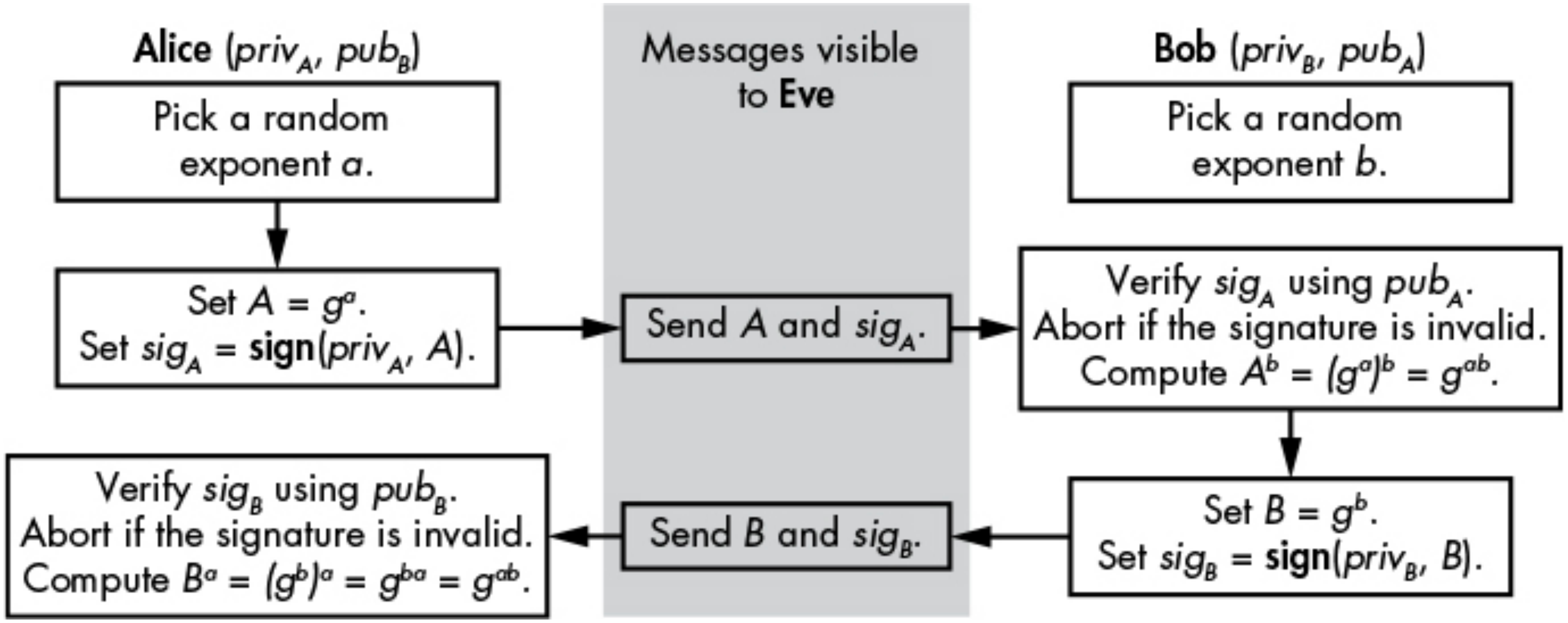
<u>What are the general steps?</u>

1. Sender signs a message with private key.
2. Receiver verifies message with public key.

<u>What does this get us?</u>

1. Verifiability
   - Receiver can verify sender signed message (m).
   - No one else could have signed message (m).
2. Non-repudiation:
   - Sender cannot claim to have signed m’ and not m.

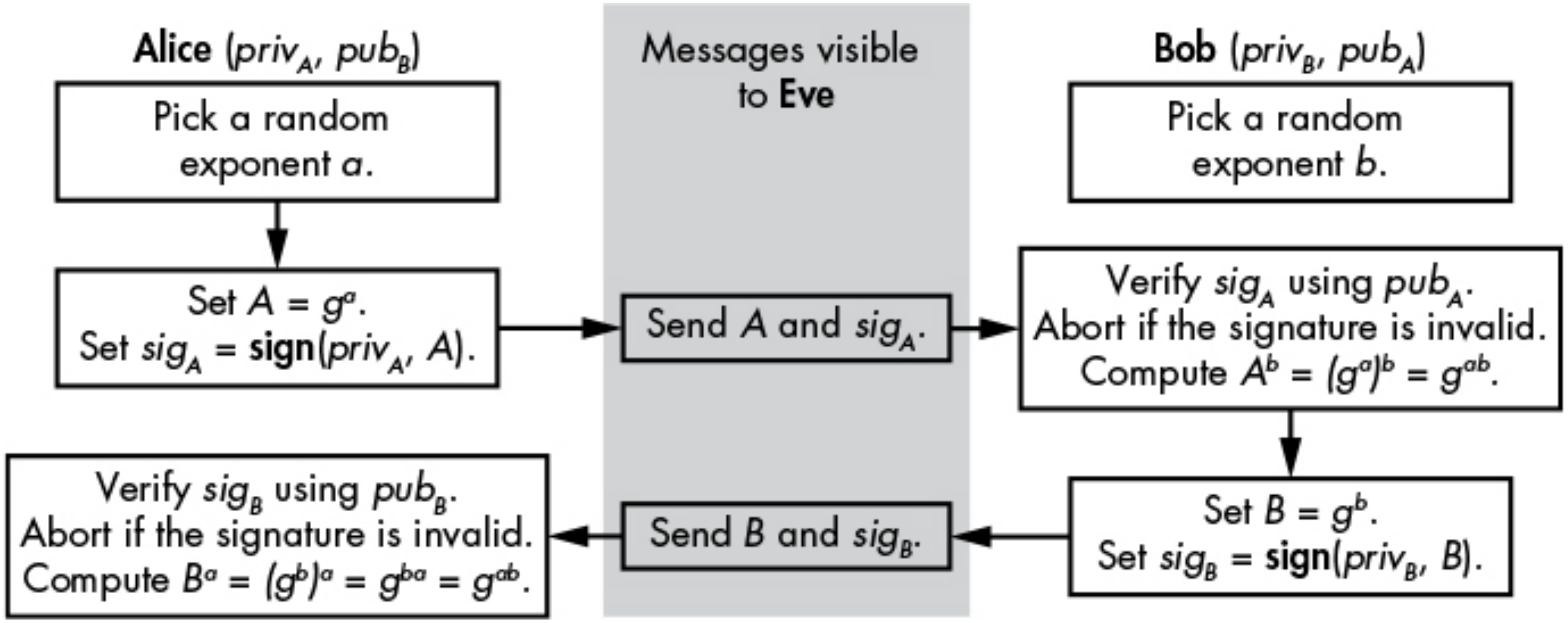
Back to Diffie-Hellman: Man-in-the-Middle Defense



Alice $(priv_A, pub_B)$
Pick a random exponent a.
Set $A = g^a$. Set sig_A = **sign**$(priv_A, A)$.
Verify sig_B using pub_B. Abort if the signature is invalid. Compute $B^a = (g^b)^a = g^{ba} = g^{ab}$.
Messages visible to **Eve**
Send A and sig_A.
Send B and sig_B.
Bob $(priv_B, pub_A)$
Pick a random exponent b.
Verify sig_A using pub_A. Abort if the signature is invalid. Compute $A^b = (g^a)^b = g^{ab}$.
Set $B = g^b$. Set sig_B = **sign**$(priv_B, B)$.

The authenticated Diffie-Hellman protocol

Back to Diffie-Hellman: Man-in-the-Middle Defense



Alice ($priv_A$, pub_B)
Pick a random exponent a.
Set $A = g^a$.
Set sig_A = **sign**($priv_A$, A).
Verify sig_B using pub_B.
Abort if the signature is invalid.
Compute $B^a = (g^b)^a = g^{ba} = g^{ab}$.
Messages visible to **Eve**
Send A and sig_A.
Send B and sig_B.
Bob ($priv_B$, pub_A)
Pick a random exponent b.
Verify sig_A using pub_A.
Abort if the signature is invalid.
Compute $A^b = (g^a)^b = g^{ab}$.
Set $B = g^b$.
Set sig_B = **sign**($priv_B$, B).

The authenticated Diffie-Hellman protocol

Let's say Eve recorded all traffic b/w Alice and Bob. After Alice and Bob finish communicating, they "forget" *a* and *b* (they are ephermeral/short-lived keys). **Perfect Forward Secrecy:** Even if Alice or Bob's private key is leaked, Eve still can't decrypt the recorded traffic (w/o solving the DLP).

Putting it all together

Use public-key crypto to establish symmetric keys:

- RSA <u>encryption</u> to secretly agree on symmetric key

- Diffie-Hellman key-exchange with RSA <u>signatures</u> to prevent MITM attack, provides perfect forward secrecy

With the symmetric key:

- Encrypt/decrypt messages using AES in some secure block cipher mode (e.g., CBC, CTR)

- Use HMAC (which relies on a secret key) to achieve integrity + authentication

General recommendation: use different keys for different algorithms

- different RSA key pair for signing and for encrypting

- different secret keys for AES and HMACs