

# Applied Cryptography and Network Security

**William Garrison**

bill@cs.pitt.edu

6311 Sennott Square

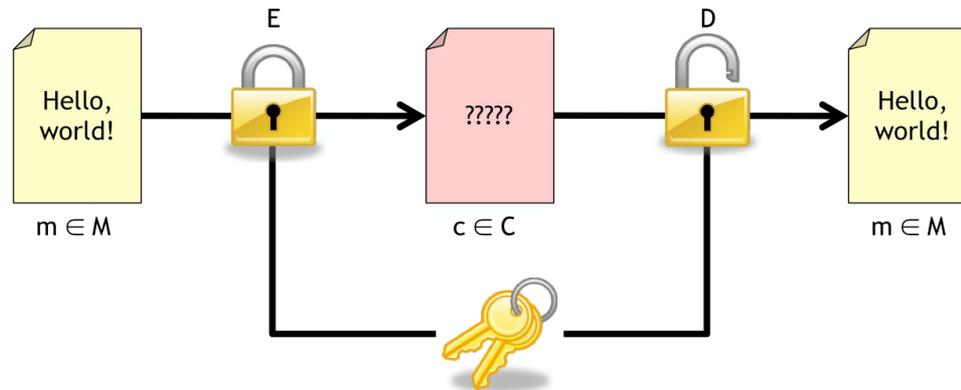
Public Key Cryptography





# Motivation

**Recall:** In a symmetric key cryptosystem, the **same** key is used for both encryption and decryption



**Note:** the sender and recipient need a **shared** secret key

The good news is that symmetric key algorithms

- Have been well-studied by the cryptography community
- Are extremely fast, and thus good for encrypting bulk data
- Provide good security guarantees based on very small secrets

Unfortunately...

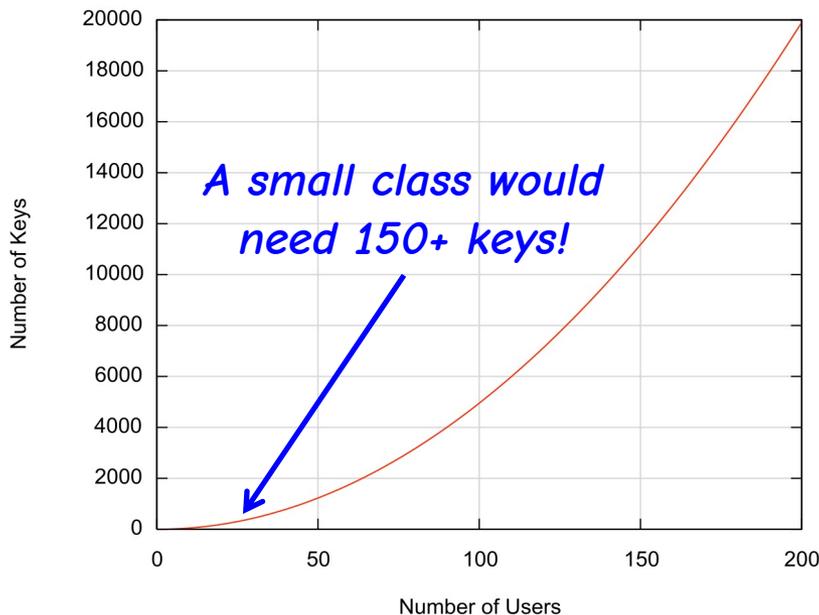


# Symmetric key cryptography is not a panacea

**Question:** What are some ways in which the need for a shared secret key might cause a problem?

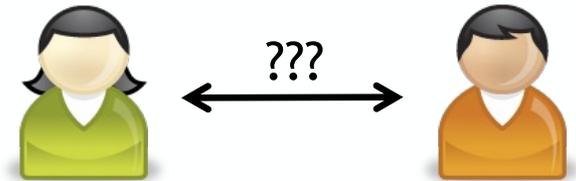
## Problem 1: Key management

- In a network with  $n$  participants,  $\binom{n}{2} = n(n-1)/2$  keys are needed!
- This number grows very rapidly!



## Problem 2: Key distribution

- How do Alice and Bob share keys in the first place?



- What if Alice and Bob have never met in person?
- What happens if they suspect that their shared key  $K_{AB}$  has been compromised?

Wouldn't it be great if we could securely communicate **without** needing pre-shared secrets?



# Thought Experiment

*Forget about bits, bytes, ciphers, keys, and math...*

**The Scenario:** Assume that Alice and Bob have never met in person. Alice has a top-secret widget that she needs to send to Bob using an untrusted courier service. Alice and Bob can talk over the phone if needed but are unable to meet in person. Due to the high-security nature of their work, the phones used by Alice and Bob may be wiretapped by other secret agents.

**Problem:** How can Alice send her widget to Bob while having very high assurance that Bob is the only person who will be able to access the widget if it is properly delivered?

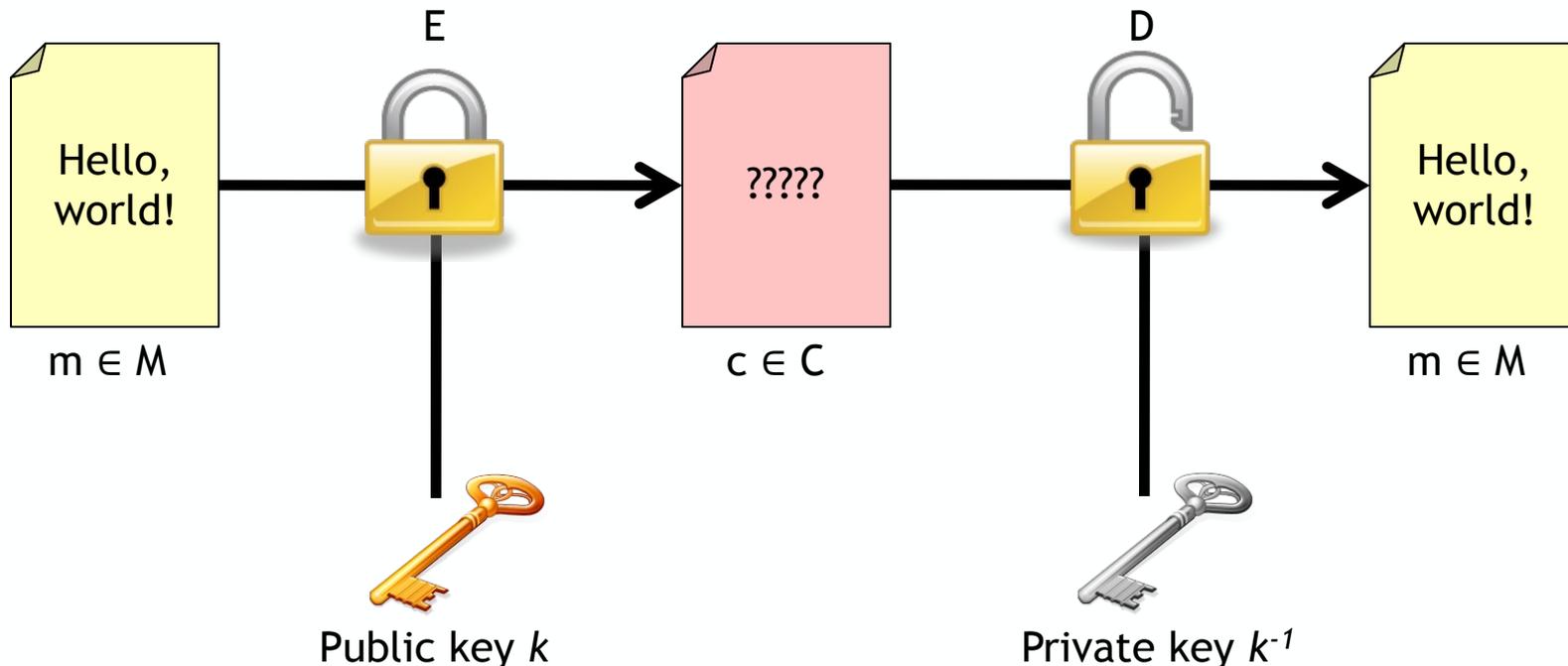
# Public key cryptosystems are a digital counterpart to the strongbox example



Formally, a cryptosystem can be represented as the 5-tuple  $(E, D, M, C, K)$

- $M$  is a message space
- $K$  is a key space
- $E : M \times K \rightarrow C$  is an encryption function
- $C$  is a ciphertext space
- $D : C \times K \rightarrow M$  is a decryption function

**Note:** Each “key” in  $K$  is actually a pair of keys,  $(k, k^{-1})$





# What can we do with public key cryptography?

First, we need some way of finding a user's public key



Print it in the newspaper



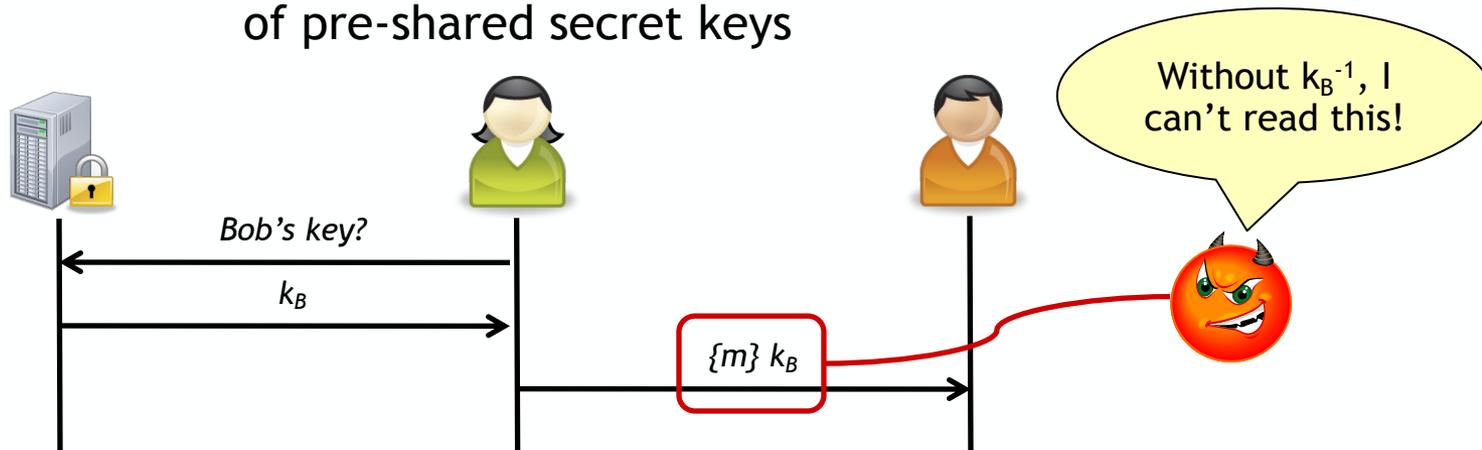
Post it on your webpage



A trusted keyserver (PKI)

**Important:** It is critical to verify the authenticity of any public key! (How?)

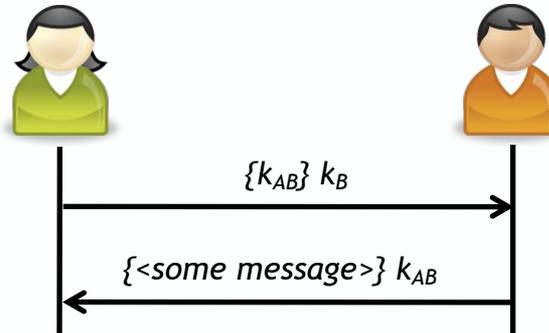
Public key cryptography allows us to send private messages **without** the use of pre-shared secret keys



# Public key cryptography can also help us exchange symmetric keys



1. Generate a key  $k_{AB}$
2. Encrypt  $k_{AB}$  using Bob's public key  $k_B$
3. Transmit



1. Decrypt  $k_{AB}$  using private key  $k_B^{-1}$
2. Encrypt message using  $k_{AB}$
3. Transmit

**Note:** Only Bob can decode  $k_{AB}$ , since only he knows  $k_B^{-1}$

- Unfortunately, Bob doesn't know who this key is from
- Key exchange is not quite this easy in practice, but it isn't *much* harder

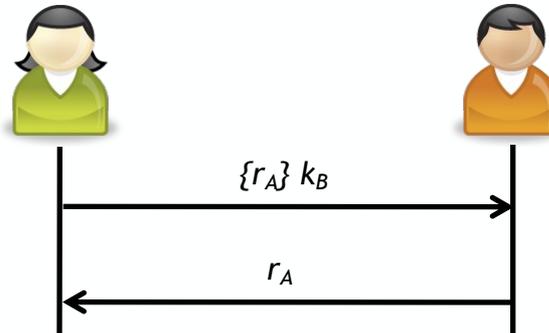
**Question:** Why on earth do we want to exchange symmetric keys?!

- Public key cryptography is usually pretty slow...
  - Based on “fancy” math, not bit shifting
  - Symmetric key algorithms are orders of magnitude faster
- It's always a good idea to change keys periodically

# Public key cryptography can also be used to authenticate users



1. Pick  $r_A$  at random
2. Encrypt  $r_A$  using Bob's public key  $k_B$
3. Transmit



1. Decrypt  $r_A$  using private key  $k_B^{-1}$
2. Transmit

**Note:** As in the previous key exchange, only Bob can decrypt  $r_A$ , since only Bob knows  $k_B^{-1}$ .

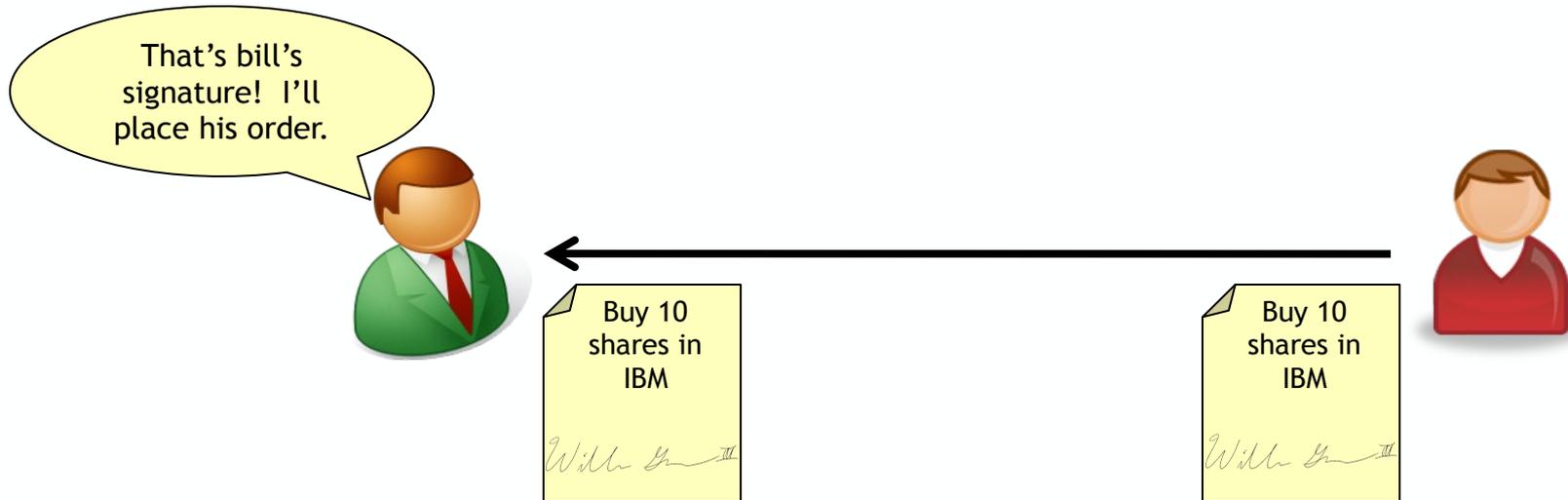
It is of absolute importance that the random numbers used during this type of protocol are **not predictable** and are **never reused** (**Why?**)

- **Unpredictable:**
  - The security of this protocol is a proof of possession of  $k_B^{-1}$
  - If predictable, an adversary can guess the “challenge” without decrypting!
  - (This is bad news)
- **Reusing** challenges may\* lead to replay attacks (**When?**)

# In addition to encryption, public key systems also let us create digital signatures



**Goal:** If Bob is given a message  $m$  and a signature  $S(m)$  supposedly computed by bill, he can determine whether or not bill actually wrote the message  $m$



In order for this to occur, we require that

- The signature  $S(m)$  must be **unforgeable**
- The signature  $S(m)$  must be **verifiable**

**Question:** *How can we do this?*

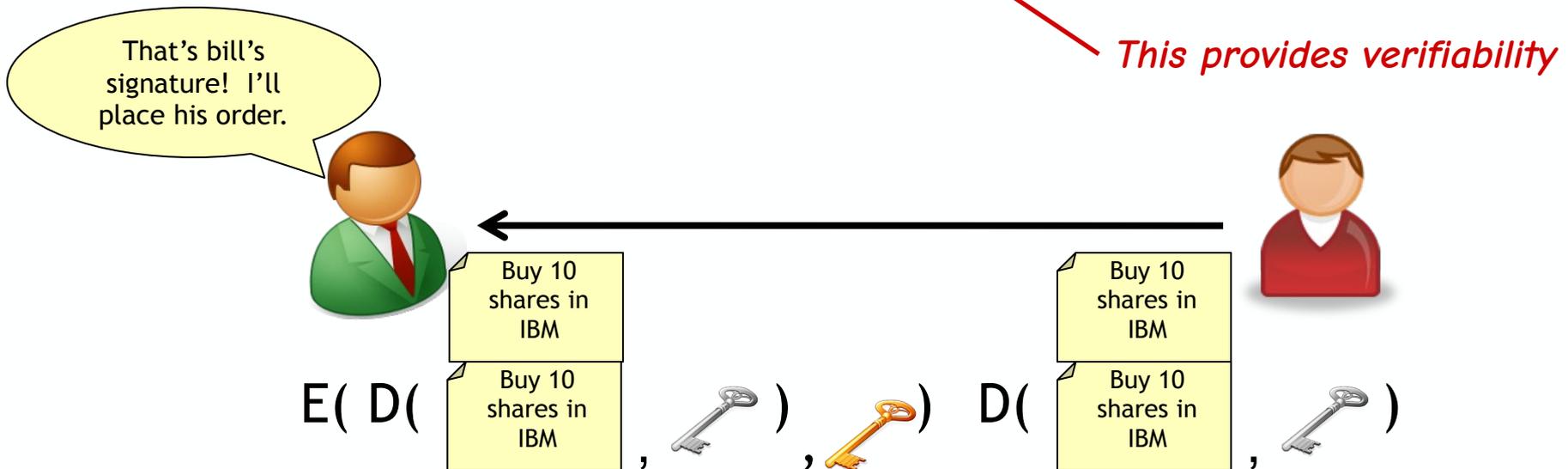
# In many public key cryptosystems, the encryption and decryption operations are **commutative**



That is,  $D(E(m, k), k^{-1}) = E(D(m, k^{-1}), k) = m$

In such a system, we can use digital signatures as follows:

- To sign a message, compute  $D(m, k^{-1})$  ← *This is unforgeable*
- Transmit  $m$  and  $D(m, k^{-1})$  to the recipient
- The recipient uses the sender's public key to verify that  $E(D(m, k^{-1}), k) = m$



**Question:** Does encryption with a shared key have the same properties?



# Features and Requirements

These features all require that for a given key pair  $(k, k^{-1})$ ,  $k$  can be made public, but  $k^{-1}$  must remain secret

So, in a public key cryptosystem it must be

1. Computationally **easy** to encipher or decipher a message
2. Computationally **infeasible** to derive the private key from the public key
3. Computationally **infeasible** to determine the private key using a chosen plaintext attack

Informally, **easy** means “polynomial complexity”, while **infeasible** means “no easier than a brute force search”

---

*How do public key cryptosystems work?*

# Diffie and Hellman proposed\* the notion of public key cryptography

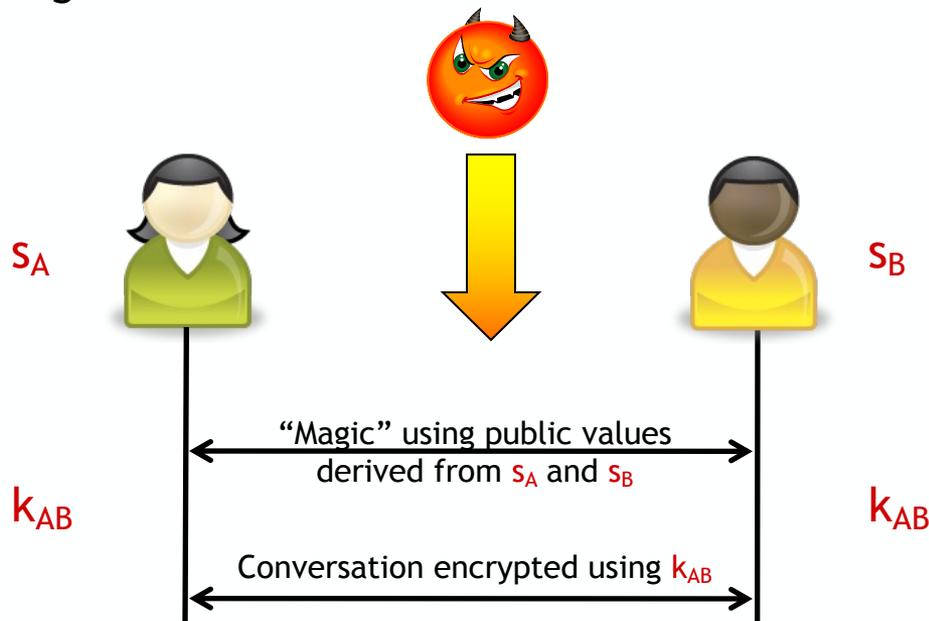


Diffie and Hellman **did not** succeed in developing a full-fledged public key cryptosystem

- i.e., their system cannot be used to encrypt/decrypt document directly
- Rather, it allows two parties to agree on a shared secret using an entirely public channel

**Question:** Why is this an interesting problem to solve?

- Key exchange!





# Diffie and Hellman proposed their system in 1976

**Seminal paper:** Whitfield Diffie and Martin E. Hellman, “New Directions in Cryptography,” IEEE Transactions on Information Theory (22)6 : 644 - 654, Nov. 1976

---

**Problem:** The widening use of telecommunications coupled with the key distribution problems inherent with secret key cryptography point to the fact that current solutions are **not** scalable!

---

This paper accomplishes many things:

- Clearly articulates why the key distribution problem must be solved
- Motivates the need for digital signatures
- Presents the first public key cryptographic algorithm
- Opened the “challenge” of designing a general-purpose public key cryptosystem

*Variants of the Diffie-Hellman key exchange algorithm are still used today!*

# How does the Diffie-Hellman protocol work?



**Step 0:** Alice and Bob agree on a finite cyclic group  $G$  of (large) prime order  $q$ , and a generator  $g$  for this group. This information is all **public**.

$a$  is Alice's private key

$g^a \pmod{q}$  is Alice's public key

## Step 1:

- Randomly choose  $a \in \{1, 2, \dots, q-1\}$
- Compute  $g^a \pmod{q}$
- Send  $g^a \pmod{q}$



$g^a \pmod{q}$

$g^b \pmod{q}$

## Step 2:

- Randomly choose  $b \in \{1, 2, \dots, q-1\}$
- Compute  $g^b \pmod{q}$
- Send  $g^b \pmod{q}$

## Step 3:

- Compute  $(g^b \pmod{q})^a \pmod{q} = g^{ba} \pmod{q} = K_{ab}$

## Step 3':

- Compute  $(g^a \pmod{q})^b \pmod{q} = g^{ab} \pmod{q} = K_{ab}$

# Why is the Diffie-Hellman key exchange protocol safe?



**Recall:** We need to show that it is hard for a “bad guy” to learn any of the secret information generated by this protocol, assuming that they know all public information

**Public information:**  $G, g, q, g^a \pmod{q}, g^b \pmod{q}$

**Private information:**  $a, b, K_{ab} = g^{ab} \pmod{q}$

---

**Tactic 1:** Can we get  $g^{ab} \pmod{q}$  from  $g^a \pmod{q}$  and  $g^b \pmod{q}$ ?

- We can get  $g^{am+bn} \pmod{q}$  for arbitrary  $m$  and  $n$ , but this is no help...

**Tactic 2:** Can we get  $a$  from  $g^a \pmod{q}$ ?

- This called taking the discrete logarithm of  $g^a \pmod{q}$
- The discrete logarithm problem is widely believed to be very hard to solve in certain types of cyclic groups

**Conclusion:** If solving the discrete logarithm problem is hard, then the Diffie-Hellman key exchange is secure!



# Hm, interesting...

Recall from last week that:

- Block ciphers secure data through confusion and diffusion
- Designing block cipher mechanisms is equal parts art and science
- The security of a block cipher is typically accepted over time (**Assurance!**)
  - Recall the initial skepticism over DES
  - The NIST competitions promote this as well

---

In public key cryptography, the relationship between  $k$  and  $k^{-1}$  is intrinsically mathematical

**Result:** The security of these systems is also rooted in mathematical relationships, and proofs of security involve reductions to mathematically “hard” problems

- e.g., Diffie-Hellman safe if the discrete logarithm is hard

# The RSA cryptosystem picks up where Diffie and Hellman left off



RSA was proposed by Ron Rivest, Adi Shamir, and Leonard Adelman in 1978. It can be used to encrypt/decrypt and digitally sign arbitrary data!

---

## Key generation:

- Choose two large prime numbers  $p$  and  $q$ , compute  $n = pq$
- Compute  $\varphi(n) = (p-1)(q-1)$
- Choose an integer  $e$  such that  $\gcd(e, \varphi(n)) = 1$
- Calculate  $d$  such that  $ed \equiv 1 \pmod{\varphi(n)}$
- **Public key:**  $n, e$
- **Private key:**  $p, q, d$

We'll discuss how to do these steps, and why they work, next time

## Usage:

- Encryption:  $M^e \pmod{n}$
- Decryption:  $C^d \pmod{n} = M^{ed} \pmod{n} = M^{k\varphi(n) + 1} \pmod{n} = M^1 \pmod{n} = M$



# An RSA Example

$p = 7$        $e = 17$   
 $q = 11$       $d = 53$   
 $n = 77$   
 $\phi(n) = 60$

Alice: (77, 17)  
Key Server



HELLO WORLD



07 04 11 11 14 26 22 14 17 11 03



$28^{53} \text{ mod } 77 = 07$

$16^{53} \text{ mod } 77 = 04$

...

$75^{53} \text{ mod } 77 = 03$

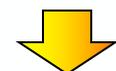


28 16 44 44 42 38 22 42 19 44 75

HELLO WORLD



07 04 11 11 14 26 22 14 17 11 03



$07^{17} \text{ mod } 77 = 28$

$04^{17} \text{ mod } 77 = 16$

...

$03^{17} \text{ mod } 77 = 75$



28 16 44 44 42 38 22 42 19 44 75





# What is involved in breaking RSA?

To break RSA, an attacker would need to derive the decryption exponent  $d$  from the public key  $(n, e)$

*Mathematicians think that this is a hard problem*

This is **conjectured** to be as hard as factoring  $n$  into  $p$  and  $q$ . Why?

- Given  $p$ ,  $q$ , and  $e$ , we can compute  $\phi(n)$
- This allows us to compute  $d$  easily!

But what if there is some entirely unrelated way to derive  $d$  from the public key  $(n, e)$ ?

**Question:** Should this make you uneasy? Why or why not?

**My Answer:** Probably not, since this bizarre new attack would also have applications to factoring large numbers.



# As always, nothing is really that easy..

**The bad news:** Naive implementations of RSA are vulnerable to chosen ciphertext attacks

**The good news:** These attacks can be prevented by using a padding scheme like OAEP prior to encryption

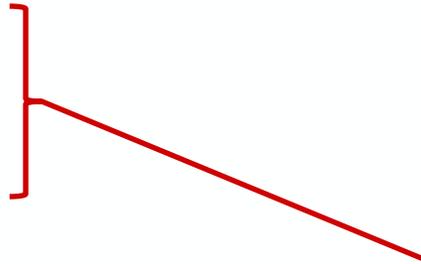
---

*Don't implement cryptography yourself! Use a standardized implementation and verify that it is standards compliant.*

---

Lastly, don't forget that implementations can be subjected to attacks

- Timing attacks
- Power consumption attacks
- ...



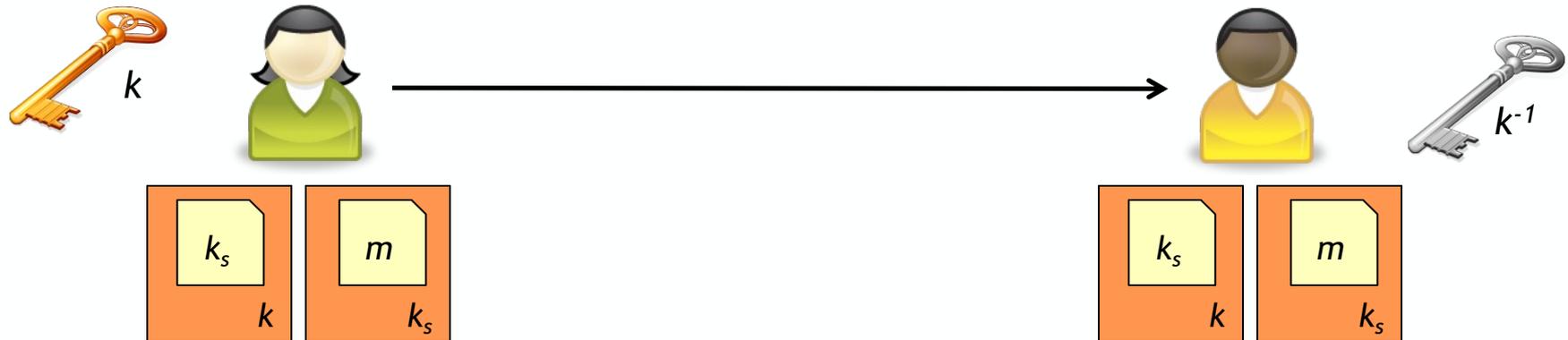
*More on this later...  
and much more in  
CS 1657*



# Unfortunately, RSA is slow when compared to symmetric key algorithms like AES or HMAC-\*

Using RSA as part of a **hybrid cryptosystem** can speed up **encryption**

- Generate a symmetric key  $k_s$
- Encrypt  $m$  with  $k_s$
- Use RSA to encrypt  $k_s$  using public key  $k$
- Transmit  $E_{k_s}(m)$ ,  $E_k(k_s)$



Using hash functions can help speed up **signing** operations

- **Intuition:**  $H(m) \ll m$ , so signing  $H(m)$  takes far less time than signing  $m$
- Why is this safe?  $H$ 's **preimage resistance** property!

# Discussion



**Question 1:** Why might malleability be an **desirable** property for a cryptosystem to have?

- Tallying electronic votes
- Aggregating private values
- A primitive for privacy-preserving computation
- ...

**Question 2:** Why might this be **bad**?

- Modifications by an active attacker!
- *Example:* Modifying an encrypted payment

---

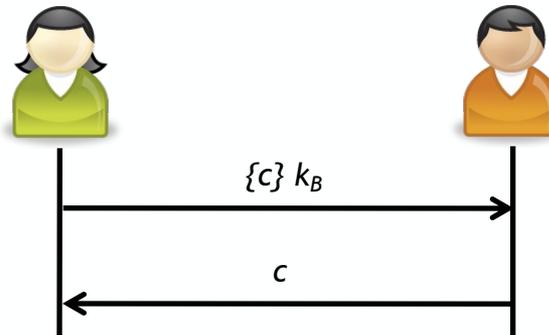
*In short, these types of cryptosystems have interesting properties, but require care to use properly.*

Note that public key cryptography allows us to prove knowledge of a secret **without** revealing that secret



**Example:** Decrypting a challenge

1. Pick challenge  $c$  at random
2. Encrypt  $c$  using Bob's public key  $k_B$
3. Transmit



1. Decrypt  $c$  using private key  $k_B^{-1}$
2. Transmit

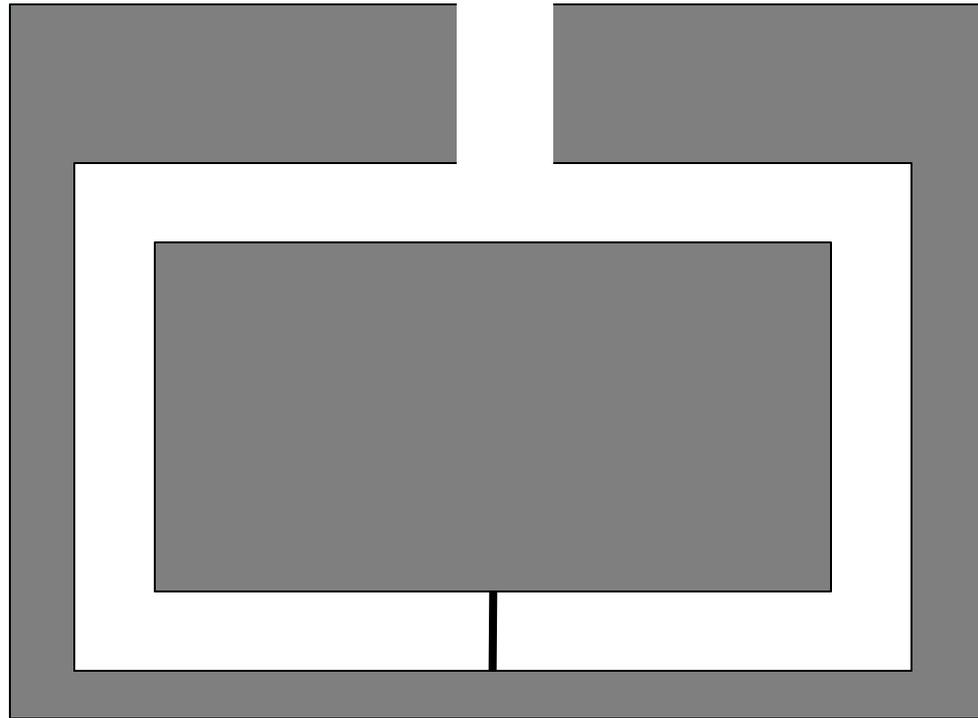
**Note:** Revealing the challenge,  $c$ , does not leak information about the private key  $k_B^{-1}$ , yet Alice is (correctly) convinced that Bob knows  $k_B^{-1}$

This type of protocol is called a **zero-knowledge** protocol

# Zero-knowledge proofs are easy in the abstract: Ali Baba's "open sesame"



*Example:* The secret cave



**Note:** To ensure correctness, this "protocol" needs to be run multiple times (**Why?**)

# Conclusions



Secret key cryptography has a key distribution problem

**Public key** cryptography overcomes this problem!

- Public encryption key
- Private decryption key

Digital signatures provide both **integrity** protection and **non-repudiation**

Malleable cryptosystems are useful, but their usage entails certain risks

Zero knowledge proof systems have many interesting applications

**Next time:** *Really* understanding RSA (and DSA)