COMP4137 Blockchain Technology and Applications
COMP7200 Blockchain Technology

Lecturer: Dr. Hong-Ning Dai (Henry)

Lecture 2

Cryptography

Outline

- Introduction to Cryptography
- Classical ciphers
- Computer Cryptography

Cryptography ≠ Security

- Cryptography may be a component of a secure system

- Adding cryptography may not make a system secure

Terms

<u>**Plaintext**</u> (cleartext) is denoted by message *M*

<u>**Encryption**</u> is denoted by function E(*M*)

It then produces <u>**ciphertext**</u> denoted by *C*=E(*M*)

<u>**Decryption**</u> the ciphertext and obtain original message *M*=D(*C*)

<u>**Cipher**</u>: Cryptographic algorithm

Terms: types of ciphers

- **Restricted cipher**
- **Symmetric algorithms**
- **Public key algorithms**

Restricted cipher

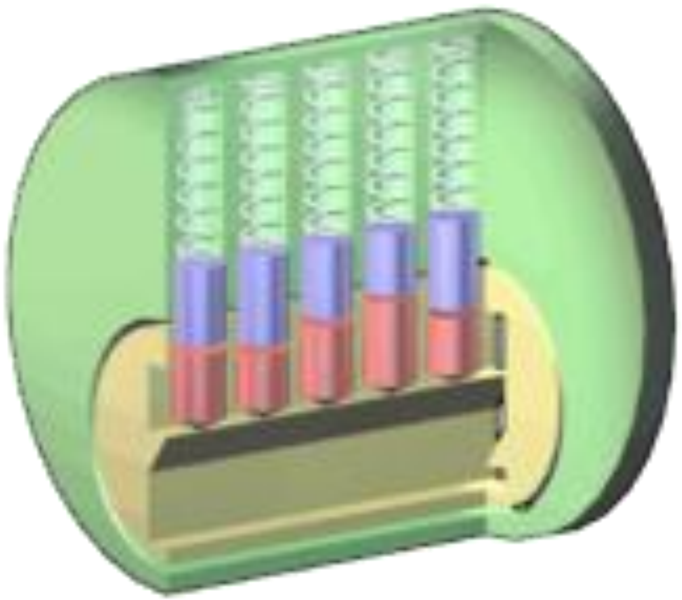
Secret algorithm

- Leaking
- Reverse engineering
  - HD DVD (Dec 2006) and Blu-Ray (Jan 2007)
  - RC4
  - All digital cellular encryption algorithms
  - DVD and DIVX video compression
  - Firewire
  - Enigma cipher machine
  - Every NATO and Warsaw Pact algorithm during Cold War

The key

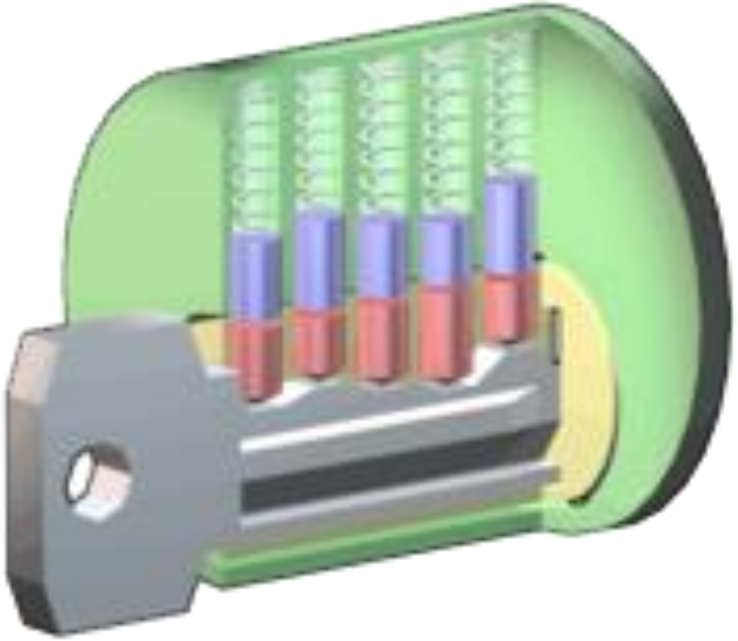
BTW, the above is a *bump key*. See http://en.wikipedia.org/wiki/Lock_bumping .

The key



Source: en.wikipedia.org/wiki/Pin_tumbler_lock

The key

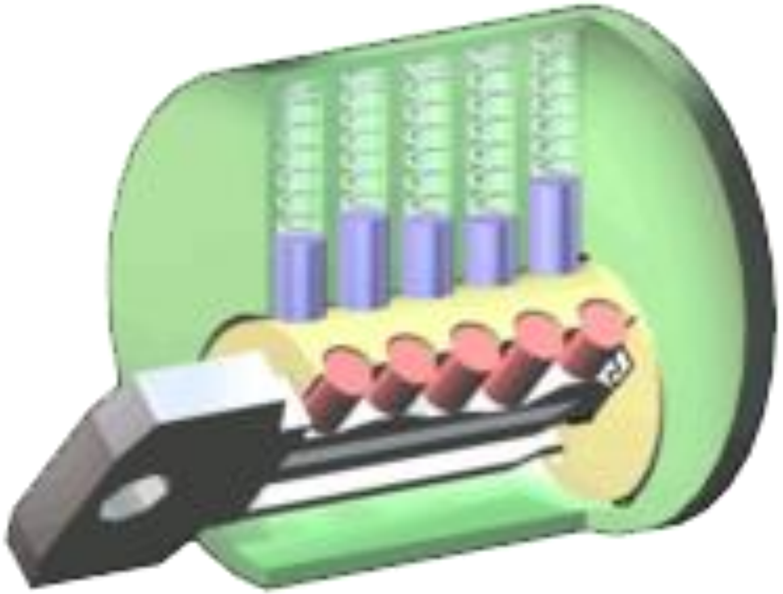


Source: en.wikipedia.org/wiki/Pin_tumbler_lock

The key

- We understand how it works:
  - Strengths
  - Weaknesses

- Based on this understanding, we can assess how much to trust the key & lock.



Source: en.wikipedia.org/wiki/Pin_tumbler_lock

Symmetric algorithm

<u>Secret key</u>

$$C = \mathrm{E_K}(M)$$

$$M = \mathrm{D_K}(C)$$

Public key algorithm

Public key and private keys

$C_1 = \mathrm{E}_{\mathrm{public}}(M)$
$M = \mathrm{D}_{\mathrm{private}}(C_1)$

also:

$C_2 = \mathrm{E}_{\mathrm{private}}(M)$
$M = \mathrm{D}_{\mathrm{public}}(C_2)$

McCarthy’s puzzle (1958)

- Two countries are at war
- One country sends spies to the other country
- To return safely, spies must give the border guards a password

- Spies can be trusted
- Guards chat – information given to them may leak

Challenge!

How can a guard authenticate a person without knowing the password?

Enemies cannot use the guard’s knowledge to introduce their own spies

Solution to McCarthy's puzzle

Michael Rabin, 1958

Use **one-way function**, $B=f(A)$

- Guards get *B* …
  - Enemy cannot compute *A*
- Spies give *A*, guards compute $f(A)$
  - If the result is *B,* the password is correct.

Example function:

Middle squares

- Take a 100-digit number (A), and square it
- Let B = middle 100 digits of 200-digit result

McCarthy’s puzzle example

Example with an 18 digit number

A = 289407349786637777

A^2 = 837566141105253089484453382203501729

Middle square, B = 110525308948445338

Given A, it is easy to compute B

Given B, it is extremely hard to compute A

One-way functions

- Easy to compute in one direction
- Difficult to compute in the other

Examples:

Factoring:

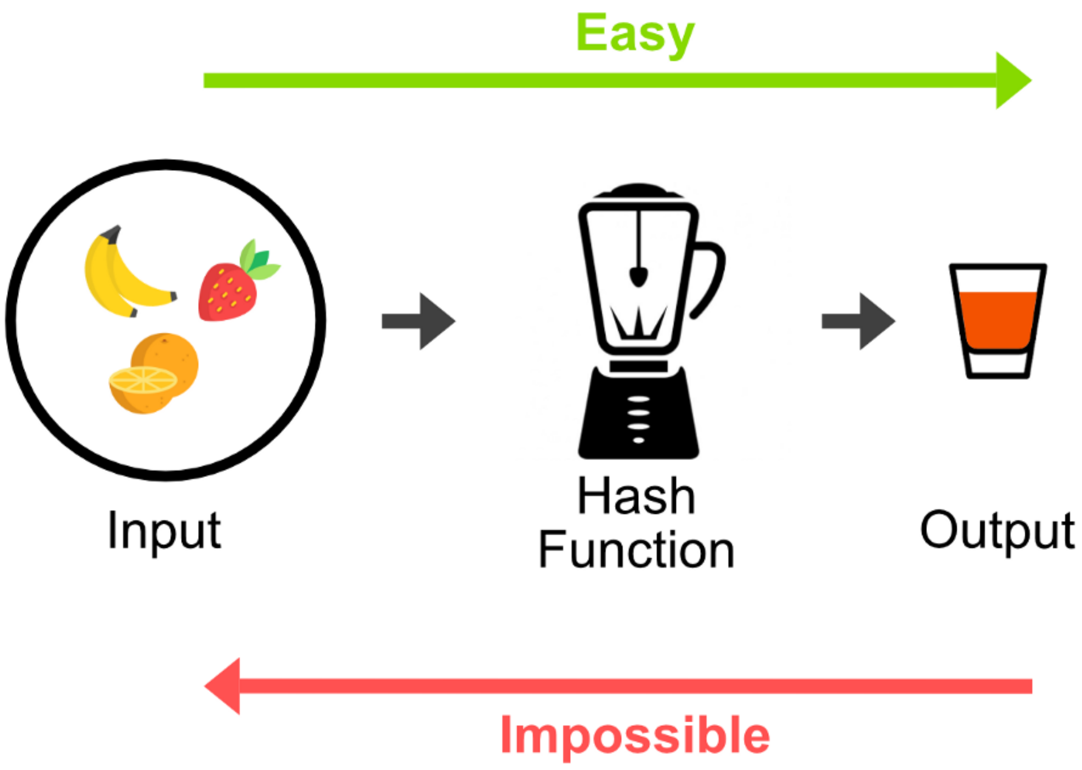
$pq = N$ EASY

find p,q given N DIFFICULT

Discrete Log:

$a^b \bmod c = N$ EASY

find b given a, c, N DIFFICULT



More terms

- **one-way function**
  - Rabin, 1958: McCarthy's problem
  - middle squares, exponentiation, …
- [one-way] **hash function**
  - message digest, fingerprint, cryptographic checksum, integrity check
- **encrypted hash**
  - message authentication code
  - only possessor of key can validate message

More terms

- **Stream cipher**
  - Encrypt a message a character at a time
- **Block cipher**
  - Encrypt a message a chunk at a time
- **Digital Signature**
  - Authenticate, not encrypt message
  - Use pair of keys (private, public)
  - Owner encrypts message with private key
  - Sender validates by decrypting with public key
  - Generally use *hash*(message).

Outline

- Introduction to Cryptography
- Classical ciphers
- Computer Cryptography

Cryptography: what is it good for?

- **Authentication**
  - determine origin of message
- **Integrity**
  - verify that message has not been modified
- **Nonrepudiation**
  - sender should not be able to falsely deny that a message was sent
- **Confidentiality**
  - others cannot read contents of the message

Cæsar cipher

Earliest documented military use of cryptography

- Julius Caesar c. 60 BC
- shift cipher: simple variant of a substitution cipher
- each letter replaced by one *n* positions away
  modulo alphabet size

  n = shift value = key

Similar scheme used in India

- early Indians also used substitutions based on phonetics

  similar to pig latin

Last seen as ROT13 on usenet to keep the reader from seeing offensive messages unwillingly

Cæsar cipher

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Cæsar cipher

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

⟶ *shift alphabet by n (6)*

Cæsar cipher

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

Cæsar cipher

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

G

Cæsar cipher

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

GS

Cæsar cipher

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

GSW

Cæsar cipher

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

GSWU

Cæsar cipher

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

GSWUN

Cæsar cipher

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

GSWUNB

Cæsar cipher

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

GSWUNBU

Cæsar cipher

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

GSWUNBUM

Cæsar cipher

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

GSWUNBUMZ

Cæsar cipher

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

GSWUNBUMZF

Cæsar cipher

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

GSWUNBUMZFY

Cæsar cipher

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

GSWUNBUMZFYU

Cæsar cipher

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

GSWUNBMUFZYUM

Cæsar cipher

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

GSWUNBMUFZYUM

- Convey one piece of information for decryption: *shift value*
- trivially easy to crack (26 possibilities for a 26 character alphabet)

Ancient Hebrew variant (ATBASH)

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

NBXZGSZHUOVZH

- c. 600 BC
- No information (key) needs to be conveyed!

Substitution cipher

MY CAT HAS FLEAS

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
M	P	S	R	L	Q	E	A	J	T	N	C	I	F	Z	W	O	Y	B	X	G	K	U	D	V	H

IVSMXAMBQCLMB

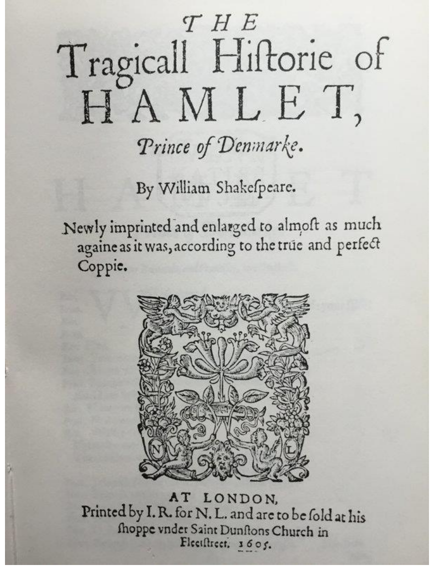
- General case: arbitrary mapping
- both sides must have substitution alphabet

Substitution cipher

Easy to decode:

- vulnerable to frequency analysis

Moby Dick (1.2M chars)		Shakespeare (55.8M chars)	
e	12.300%	e	11.797%
o	7.282%	o	8.299%
d	4.015%	d	3.943%
b	1.773%	b	1.634%
x	0.108%	x	0.140%



Statistical Analysis

Letter frequencies

- E: 12%
- A, H, I, N, O, R, S, T: 6 – 9%
- D, L: 4%
- B, C, F, G, M, P, U, W, Y: 1.5 – 2.8%
- J, K, Q, V, X, Z: < 1%

Common digrams:

- TH, HE, IN, ER, AN, RE, …

Common trigrams

- THE, ING, AND, HER, ERE, …

Strong password:

- At least 12 characters long but 14 or more is better.
- A combination of uppercase letters, lowercase letters, numbers, and symbols.

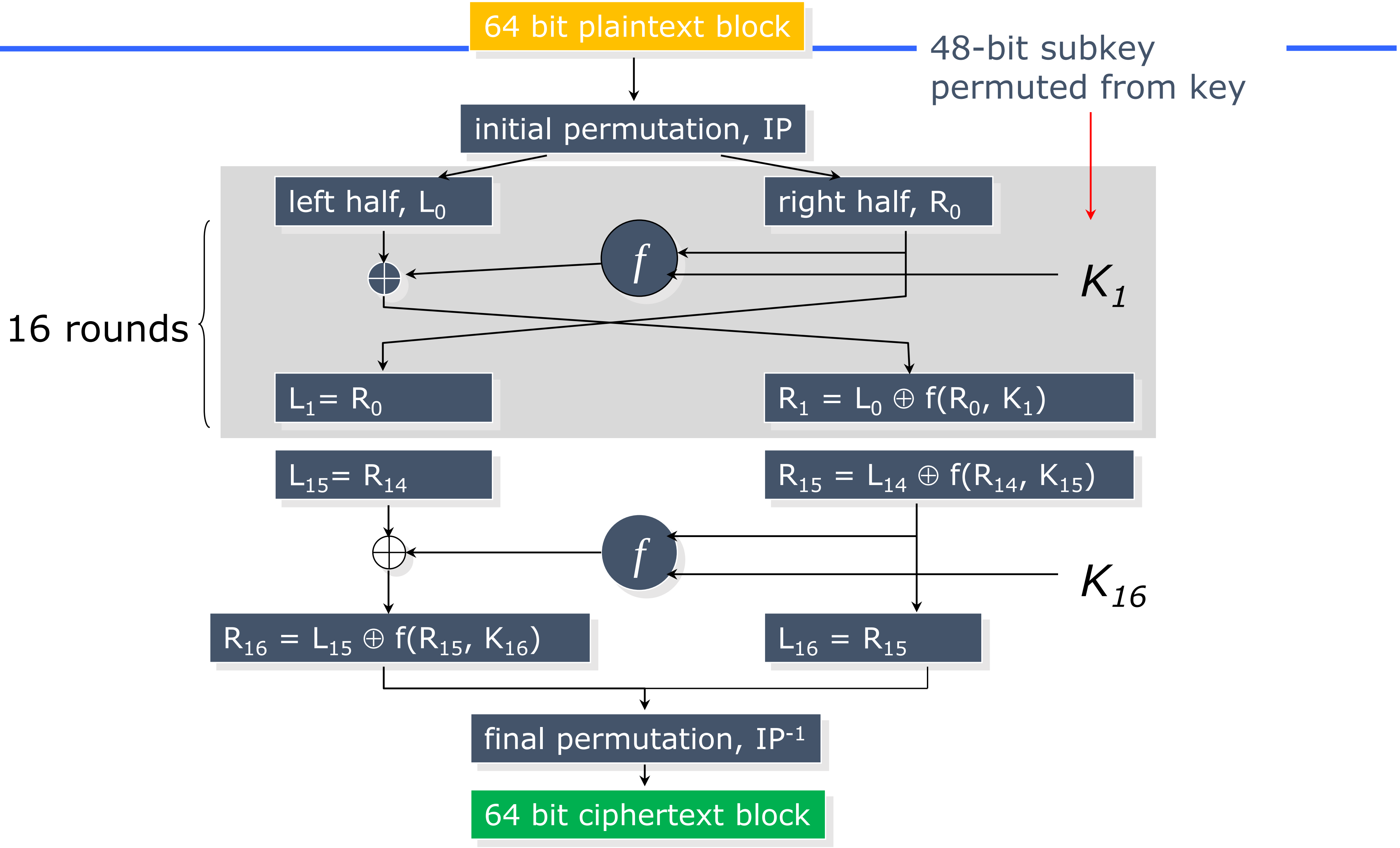
Outline

- Introduction to Cryptography
- Classical ciphers
- Computer Cryptography

DES

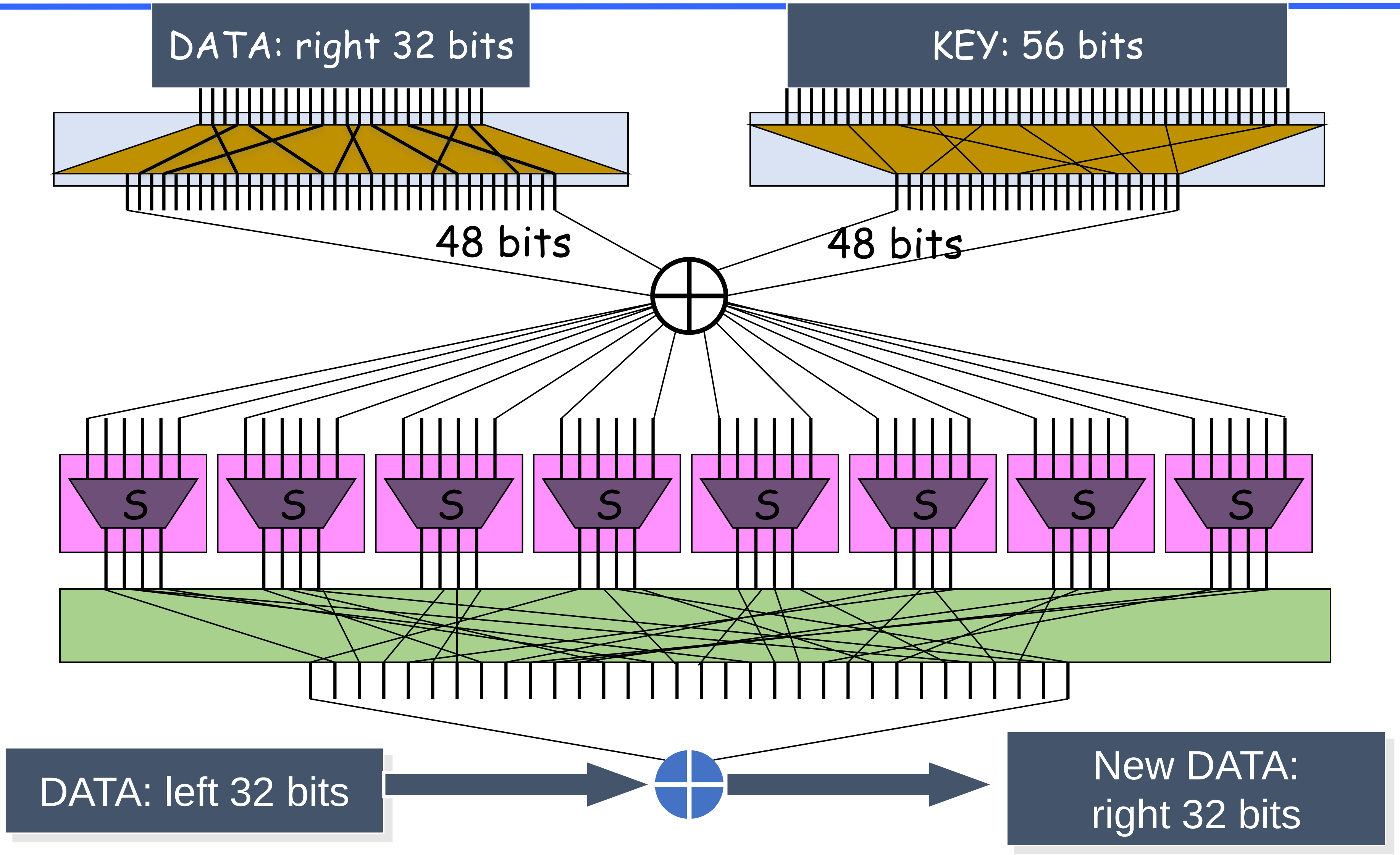
- Data Encryption Standard
  - adopted as a federal standard in 1976
- block cipher, 64 bit blocks
- 56 bit key
  - all security rests with the key
- substitution followed by a permutation (transposition)
  - same combination of techniques is applied on the plaintext block 16 times

DES



64 bit plaintext block
48-bit subkey
permuted from key
initial permutation, IP
left half, L_0
right half, R_0
f
K_1
16 rounds
$L_1 = R_0$
$R_1 = L_0 \oplus f(R_0, K_1)$
$L_{15} = R_{14}$
$R_{15} = L_{14} \oplus f(R_{14}, K_{15})$
f
K_{16}
$R_{16} = L_{15} \oplus f(R_{15}, K_{16})$
$L_{16} = R_{15}$
final permutation, IP^{-1}
64 bit ciphertext block

DES: f



DATA: right 32 bits
KEY: 56 bits
48 bits
48 bits
S
S
S
S
S
S
S
S
DATA: left 32 bits
New DATA: right 32 bits

DES: S-boxes

- After compressed key is XORed with expanded block
  - 48-bit result moves to substitution operation via 8 **substitution boxes** (s-boxes)
- Each S-box has
  - 6-bit input
  - 4-bit output
- 48 bits divided into eight 6-bit sub-blocks
- Each block is operated by a separate S-box
- key components of DES’s security
- net result: 48-bit input generates 32-bit output

Is DES secure?

56-bit key makes DES relatively weak

- 7.2×10^{16} keys
- Brute-force attack

Late 1990's:

- DES cracker machines built to crack DES keys in a few hours
- DES Deep Crack: 90 billion keys/second
- Distributed.net: test 250 billion keys/second

The power of 2

Adding an extra bit to a key doubles the search space.

Suppose it takes 1 second to attack a 20-bit key:

- 21-bit key: 2 seconds
- 32-bit key: 1 hour
- 40-bit key: 12 days
- 56-bit key: 2,178 years
- 64-bit key: >557,000 years!

Increasing The Key

Can double encryption work for DES?

- Useless if we could find a key K such that:

$$E_K(P) = E_{K2}(E_{K1}(P))$$

- This does not hold for DES

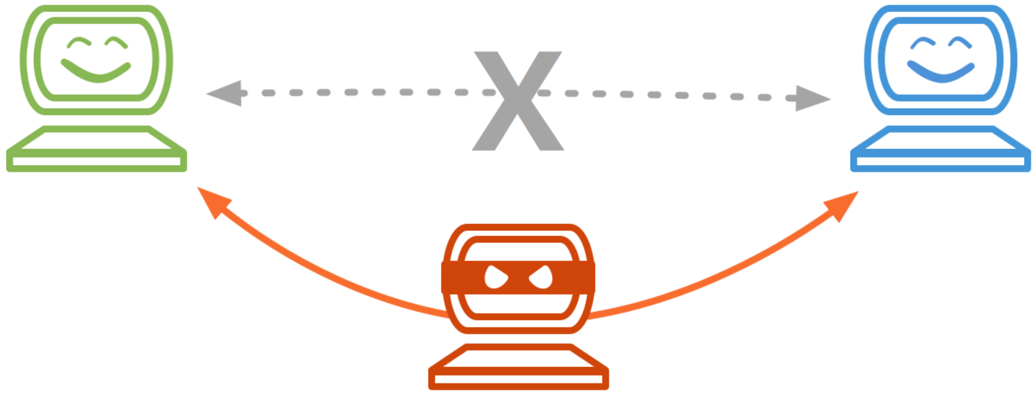
Double DES

Vulnerable to meet-in-the-middle attack

If we know some pair (P, C), then:

[1] Encrypt P for all 2^{56} values of K_1

[2] Decrypt C for all 2^{56} values of K_2



For each match where [1] = [2]

- test the two keys against another P, C pair
- if match, you are assured that you have the key

Triple DES

Triple DES with two 56-bit keys:

$$C = E_{K1}(D_{K2}(E_{K1}(P)))$$

Triple DES with three 56-bit keys:

$$C = E_{K3}(D_{K2}(E_{K1}(P)))$$

Decryption used in middle step for compatibility with DES ($K_1=K_2=K_3=k$)

$$C = E_K(D_K(E_K(P))) \equiv C = E_{K1}(P)$$

Triple DES

Prevent meet-in-the-middle attack with

- three stages
- and two keys

Triple DES:

$$C = E_{K1}(D_{K2}(E_{K1}(P)))$$

Decryption used in middle step for compatibility with DES

$$C = E_K(D_K(E_K(P))) \equiv C = E_{K1}(P)$$

Popular symmetric algorithms

IDEA - International Data Encryption Algorithm

- 1992
- 128-bit keys, operates on 8-byte blocks (like DES)
- algorithm is more secure than DES

RC4, by Ron Rivest

- 1995
- key size up to 2048 bits
- not secure against multiple messages encrypted with the same key

AES - Advanced Encryption Standard

- NIST proposed successor to DES, chosen in October 2000
- based on Rigndael cipher
- 128, 192, and 256 bit keys

AES

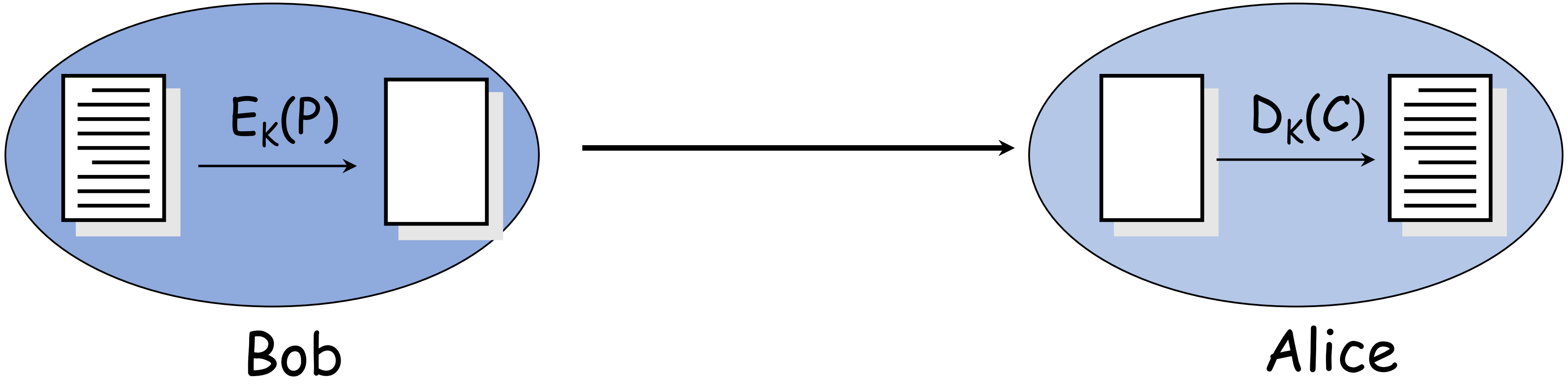
From NIST:

> Assuming that one could build a machine that could recover a DES key in a second (i.e., try 2^{56} keys per second), then it would take that machine approximately 149 trillion years to crack a 128-bit AES key. To put that into perspective, the universe is believed to be less than 20 billion years old.

http://csrc.nist.gov/encryption/aes/

Symmetric cryptography

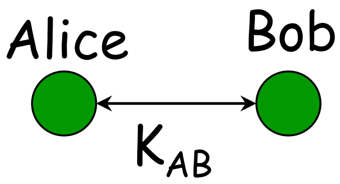
- Both parties must agree on a secret key, *K*
- message is encrypted, sent, decrypted at other side



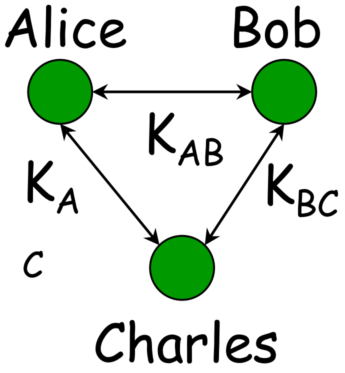
- Key distribution must be secret
  - otherwise messages can be decrypted
  - users can be impersonated

Key explosion

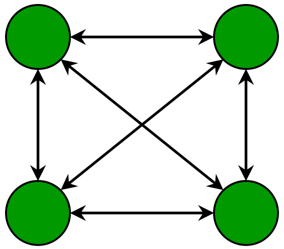
Each pair of users needs a separate key for secure communication



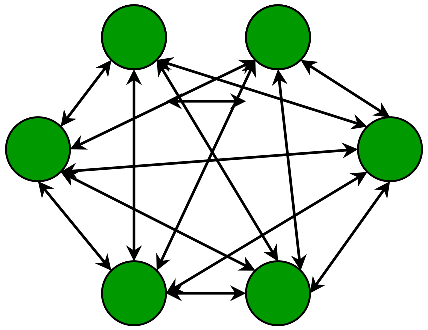
2 users: 1 key



3 users: 3 keys



4 users: 6 keys



6 users: 15 keys

100 users: 4950 keys

1000 users: 399500 keys

n users: $\frac{n(n-1)}{2}$ keys

Key distribution

Secure key distribution is the biggest problem with symmetric cryptography

How can you communicate securely with someone you've never met?

Whit Diffie: idea for a *public key* algorithm

Challenge: can this be done securely?

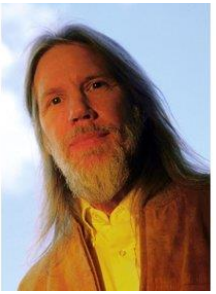
Knowledge of public key should not allow derivation of private key

Diffie-Hellman exponential key exchange

Key distribution algorithm

- first algorithm to use public/private keys
- not public key encryption
- based on difficulty of computing discrete logarithms in a finite field compared with ease of calculating exponentiation

Allows us to negotiate a secret session key without fear of eavesdroppers



Diffie-Hellman exponential key exchange

- All arithmetic performed in
  field of integers modulo some large number
- Both parties agree on
  - a **large prime number *p***
  - and a **number α < p**
- Each party generates a public/private key pair

private key for user i: X_i

public key for user i: Y_i

$$\alpha^{X_i} \bmod p$$

Diffie-Hellman exponential key exchange

- Alice has secret key X_A
- Alice has public key Y_A
- Alice computes

- Bob has secret key X_B
- Bob has public key Y_B

$$K = Y_B^{X_A} \bmod p$$

K = (Bob's public key) $^{\textit{(Alice's private key)}}$ *mod p*

Diffie-Hellman exponential key exchange

- Alice has secret key X_A
- Alice has public key Y_A
- Alice computes

- Bob has secret key X_B
- Bob has public key Y_B
- Bob computes

$$K = Y_B^{X_A} \bmod p \qquad K' = Y_A^{X_B} \bmod p$$

K′ = (Alice's public key) $^{\textit{(Bob's private key)}}$ *mod p*

Diffie-Hellman exponential key exchange

- Alice has secret key X_A
- Alice has public key Y_A
- Alice computes

$$K = Y_B^{X_A} \bmod p$$

- expanding:

$$\begin{aligned} K &= Y_B^{X_A} \mod p \\ &= \left(\alpha^{X_B} \mod p\right)^{X_A} \mod p \\ &= \alpha^{X_B X_A} \mod p \end{aligned}$$

- Bob has secret key X_B
- Bob has public key Y_B
- Bob computes

$$K' = Y_A^{X_B} \bmod p$$

- expanding:

$$\begin{aligned} K' &= Y_A^{X_B} \mod p \\ &= \left(\alpha^{X_A} \mod p\right)^{X_B} \mod p \\ &= \alpha^{X_A X_B} \mod p \end{aligned}$$

$$K = K'$$

K is a common key, known *only* to Bob and Alice

Diffie-Hellman example

Suppose $p = 31667$, $\alpha = 7$

Alice picks

$X_A = 18$

Alice's public key is:

$Y_A = 7^{18} \bmod 31667 = 6780$

Bob picks

$X_B = 27$

Bob's public key is:

$Y_B = 7^{27} \bmod 31667 = 22184$

$K = 22184^{18} \bmod 31667$

$K = 14265$

$K = 6780^{27} \bmod 31667$

$K = 14265$

Key distribution problem is solved!

- User maintains private key
- Publishes public key in database ("phonebook")

- Communication begins with key exchange to establish a common key
- Common key can be used to encrypt a session key
  - increase difficulty of breaking common key by reducing the amount of data we encrypt with it
  - session key is valid only for one communication session

RSA: Public Key Cryptography

- Ron **R**ivest, Adi **S**hamir, Leonard **A**dleman created a true public key encryption algorithm in 1977
- Each user generates two keys
  - private key (kept secret)
  - public key
- Difficulty of algorithm based on the difficulty of factoring large numbers
  - keys are functions of a pair of large (~200 digits) prime numbers

RSA algorithm

Generate keys:

- choose two random large prime numbers p, q
- Compute the product $n = pq$
- randomly choose the encryption key, e,
  such that:
  e and $(p - 1)(q - 1)$ are relatively prime
- use the extended Euclidean algorithm to compute the decryption key, d:

  $$ed = 1 \bmod ((p - 1)(q - 1))$$

  $$d = e^{-1} \bmod ((p - 1)(q - 1))$$

- discard p, q

RSA algorithm

Encrypt:

- divide data into numerical blocks < n
- encrypt each block:
  $c = m^e \bmod n$

Decrypt:

$$m = c^d \bmod n$$

Communication with public key algorithms

Different keys for encrypting and decrypting

- no need to worry about key distribution

Communication with public key algorithms

Alice

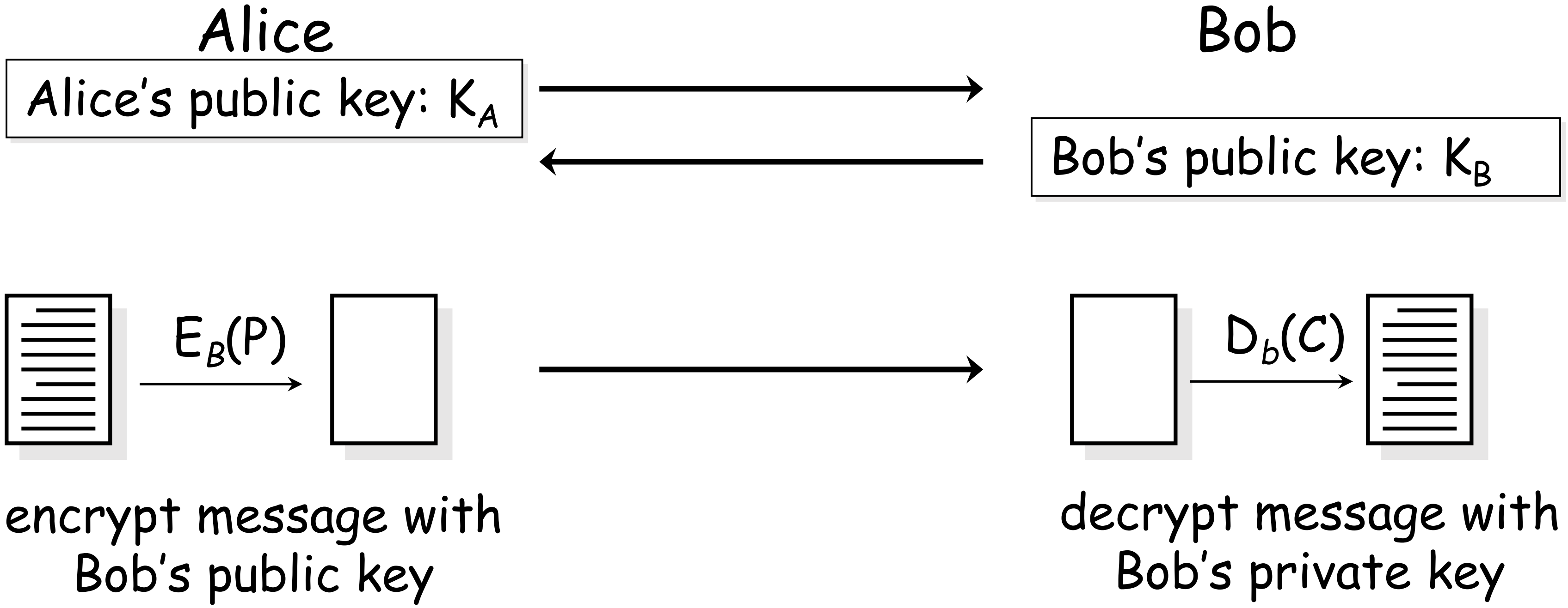
Alice's public key: K_A

Bob

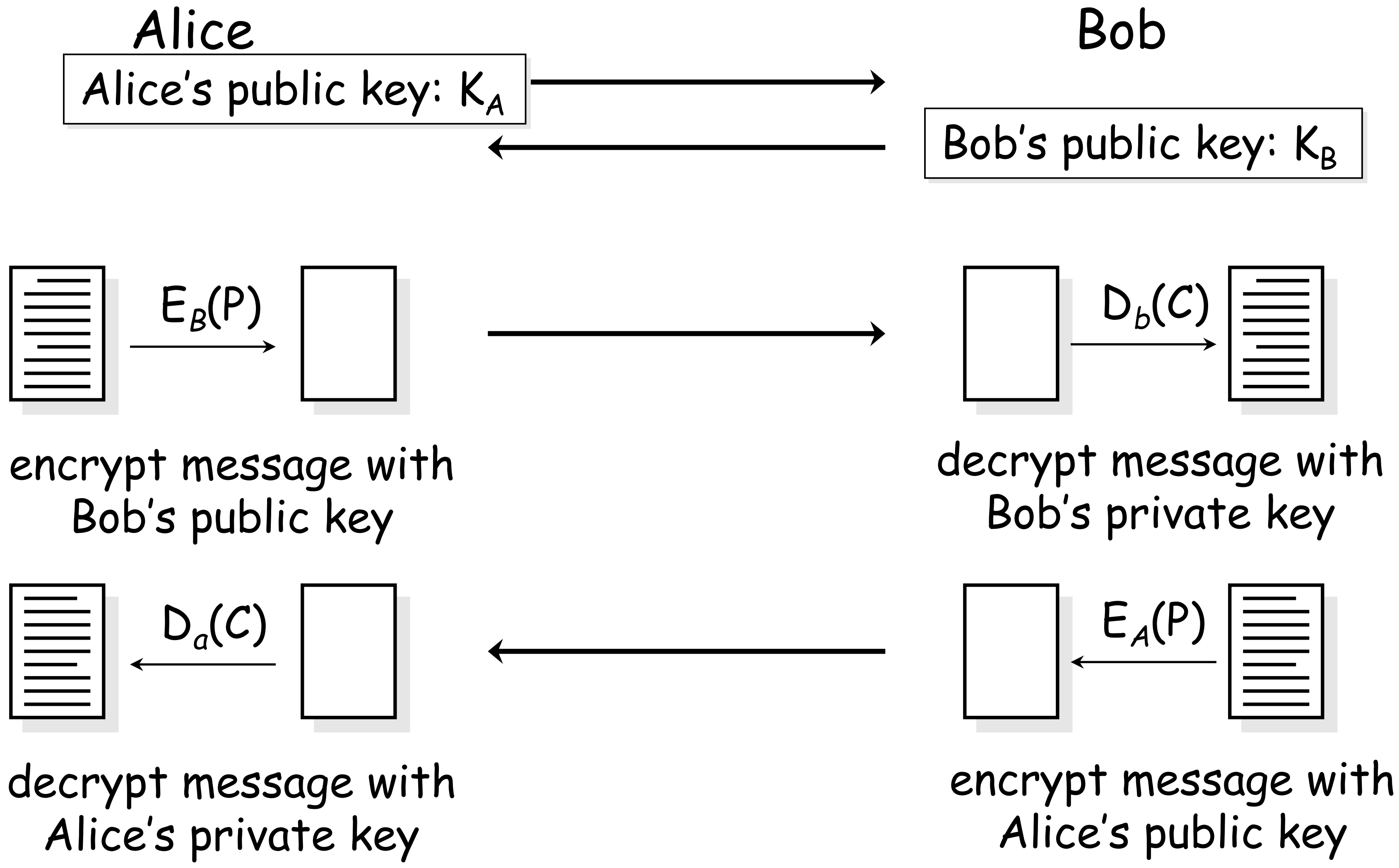
Bob's public key: K_B

exchange public keys
(or look up in a directory/DB)

Communication with public key algorithms



Communication with public key algorithms



Public key woes

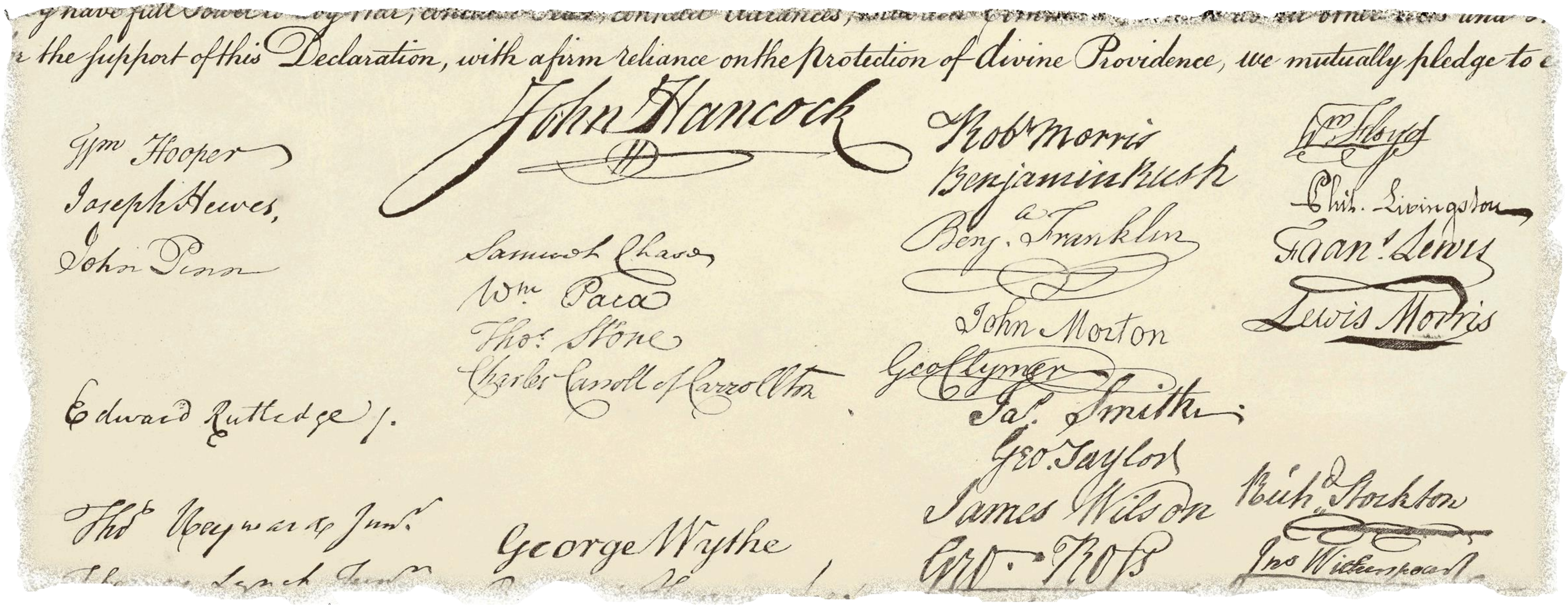
Public key cryptography is great but:

- RSA about 100 times slower than DES in software, 1000 times slower in HW
- Vulnerable to chosen plaintext attack
  - if you know the data is one of *n* messages, just encrypt each message with the recipient's public key and compare
- It's a good idea to reduce the amount of data encrypted with any given key
  - but generating RSA keys is computationally very time consuming

Signatures

We use signatures because a signature is

- Authentic
- Unforgeable
- Not reusable
- Non repudiatable
- Renders document unalterable



Source: http://www.archives.gov/exhibits/charters/declaration.html

Signatures

~~We use signatures because a signature is~~

~~Authentic~~ ~~Unforgeable~~

~~Not reusable~~ ~~Non repudiatable~~

~~Renders document unalterable~~

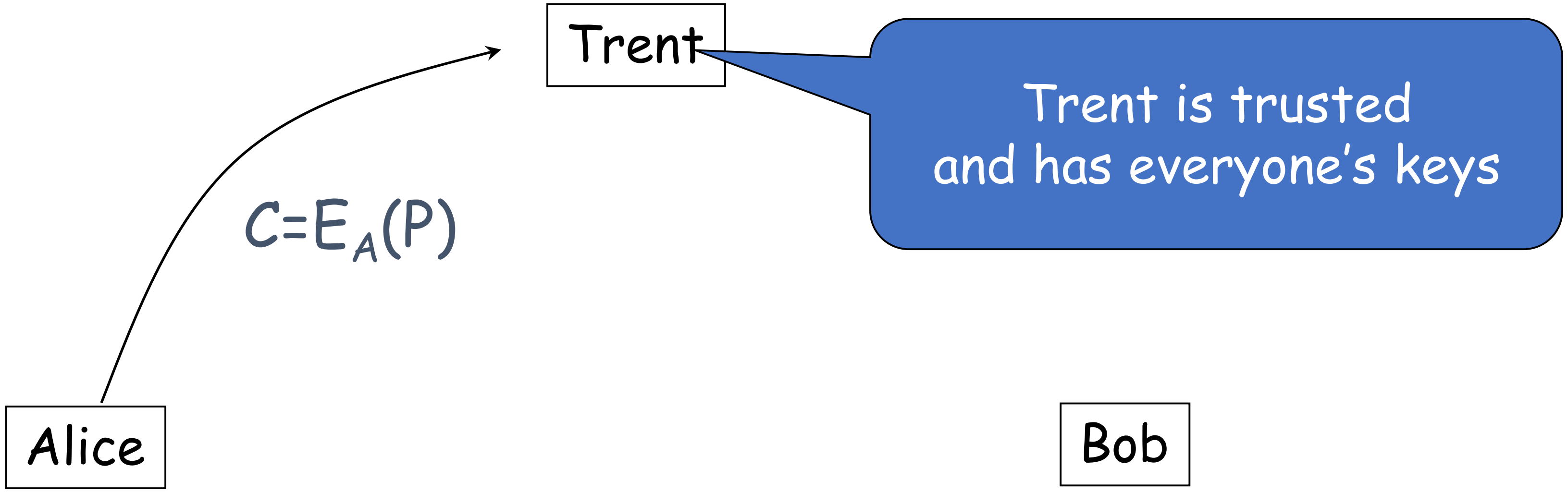
ALL UNTRUE!

Can we do better with digital signatures?

Digital signatures - arbitrated protocol

Arbitrated protocol using symmetric encryption

- turn to trusted third party (arbiter) to authenticate messages



Alice encrypts message for *herself* and sends it to Trent

Arbitrated protocol

Trent

$P = D_A(C)$

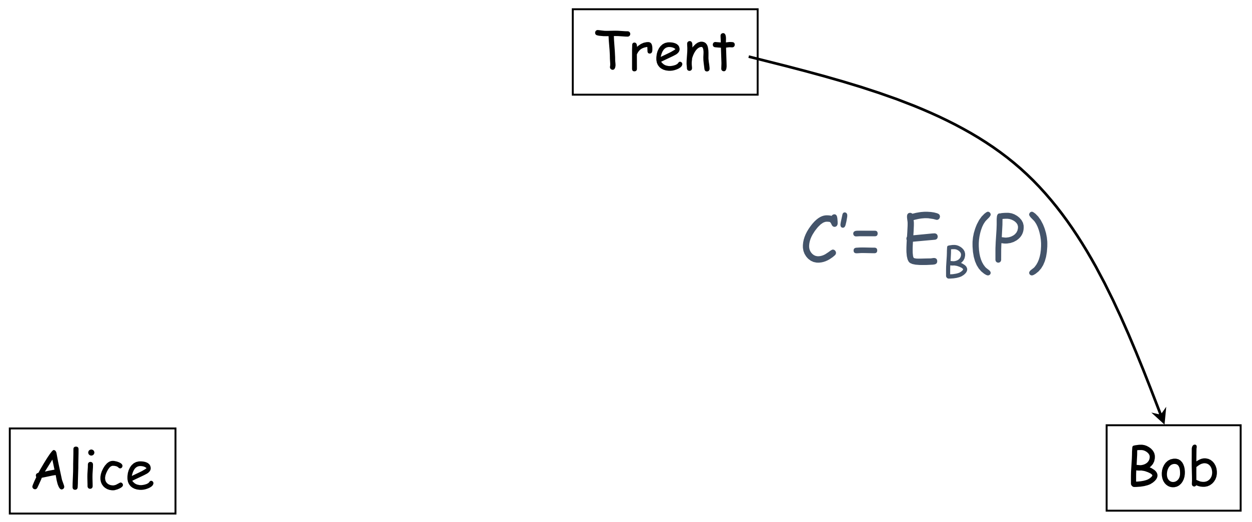
Alice

Bob

Trent receives Alice's message and decrypts it with Alice's key

- this authenticates that it came from Alice
- he may choose to log a hash of the message to create a record of the transmission

Arbitrated protocol



Trent now encrypts the message for Bob and sends it to Bob

Arbitrated protocol

Trent

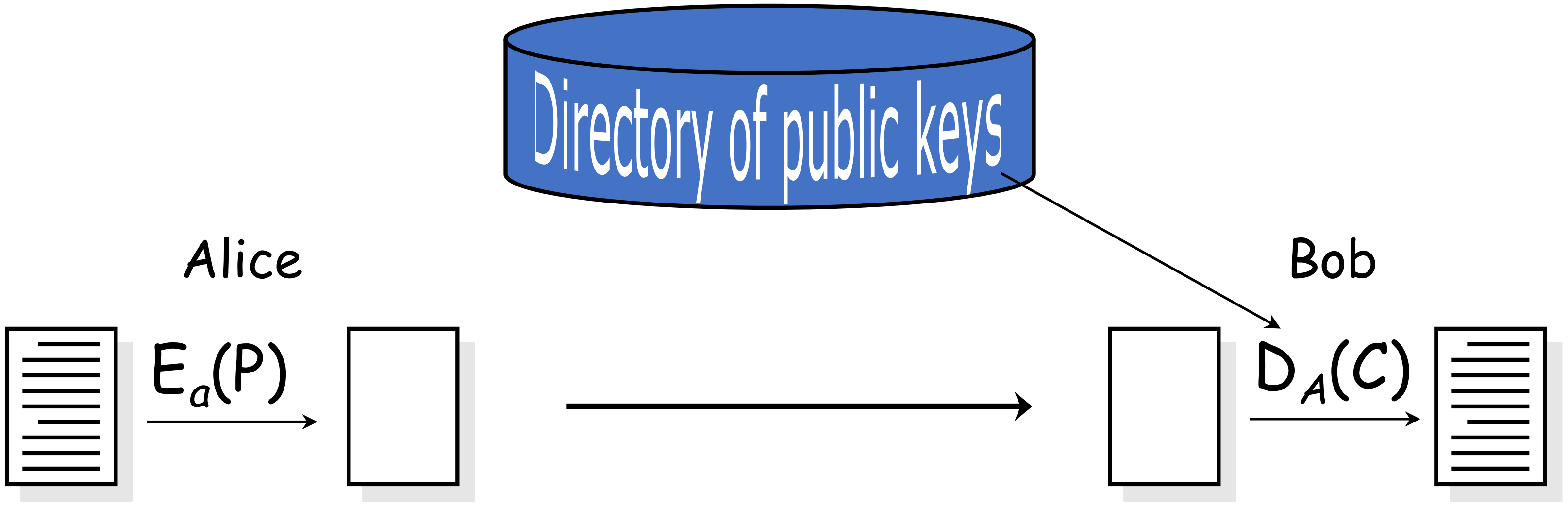
Alice

Bob $P' = D_B(C')$

Bob receives the message and decrypts it
- it *must* have come from Trent since only Trent and Bob have Bob's key
- if the message says it's from Alice, it must be - we trust Trent

Digital signatures - public key cryptography

Encrypting a message with a private key is the same as signing!



encrypt message with Alice's private key

decrypt message with Alice's public key

Digital signatures - public key cryptography

- What if Alice was sending Bob binary data?
    - Bob might have a hard time knowing whether the decryption was successful or not
- Public key encryption is considerably slower than symmetric encryption
    - what if the message is very large?
- What if we don’t want to hide the message, yet want a valid signature?

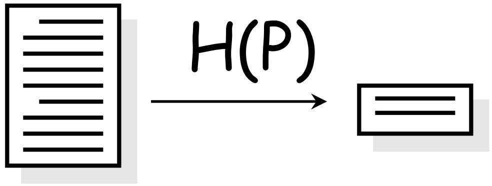
Digital signatures - public key cryptography

- Create a hash of the message
- Encrypt the hash and send it with the message
- Validate the hash by decrypting it and comparing it with the hash of the received message
- The signature is now a distinct entity from the message

Digital signatures - public key cryptography

Alice

Bob

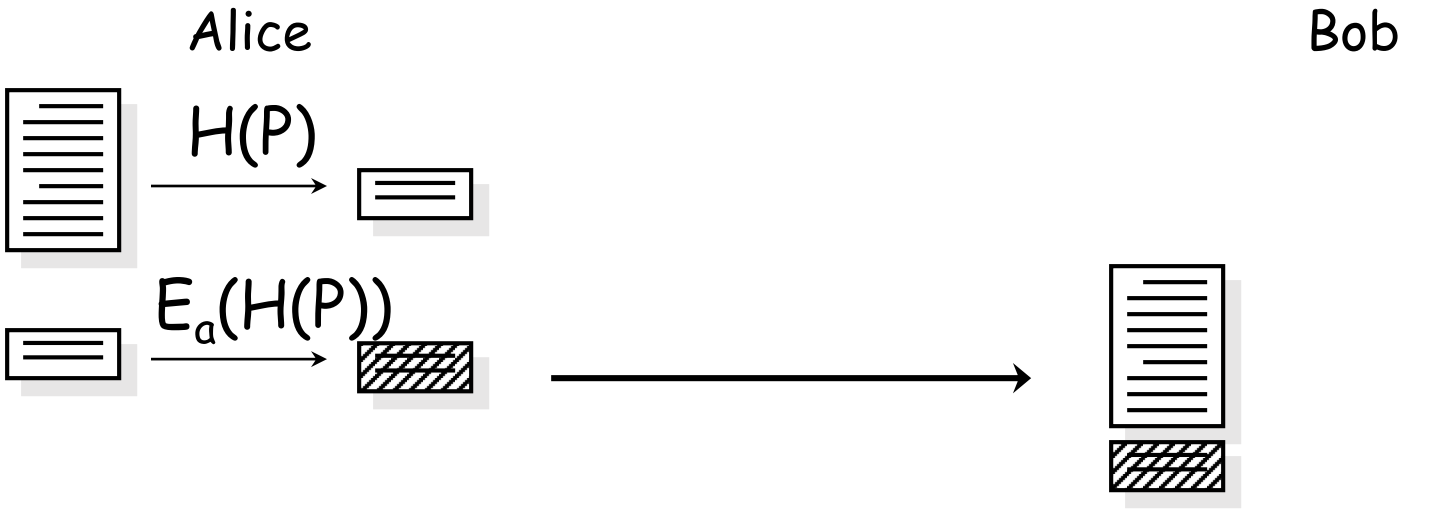


Alice generates a hash of the message

Digital signatures - public key cryptography

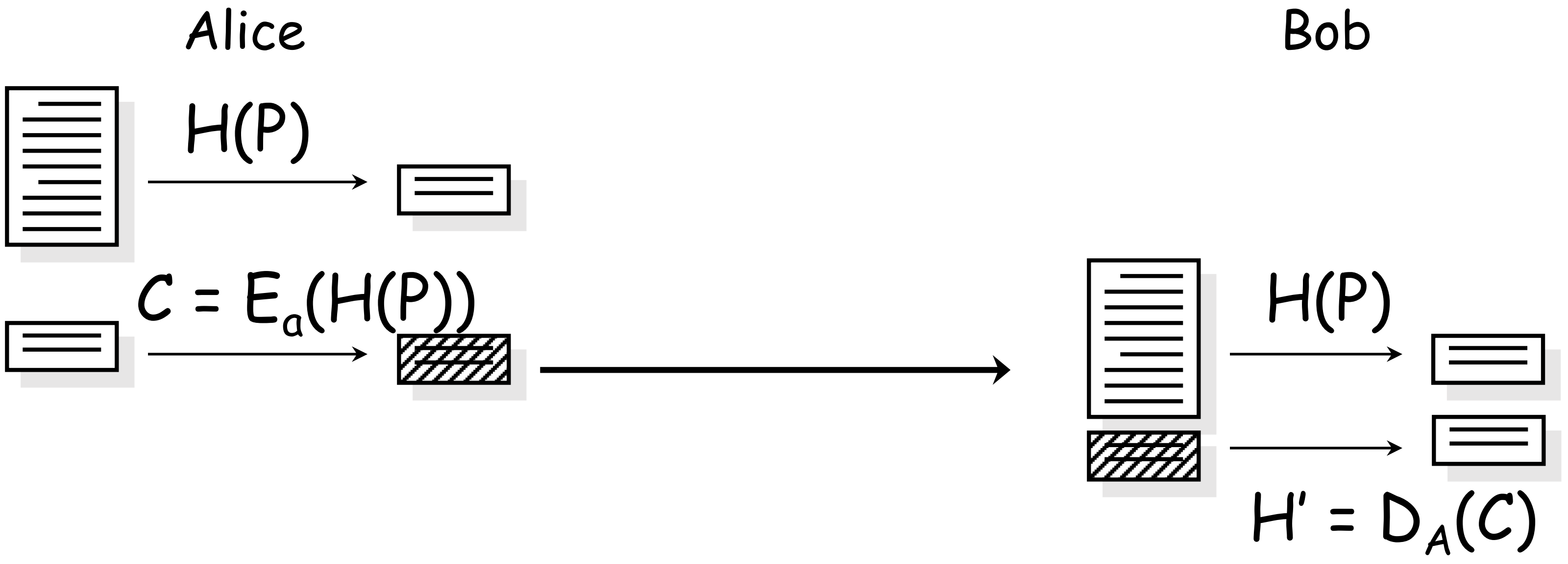
Alice encrypts the hash with her private key

Digital signatures - public key cryptography



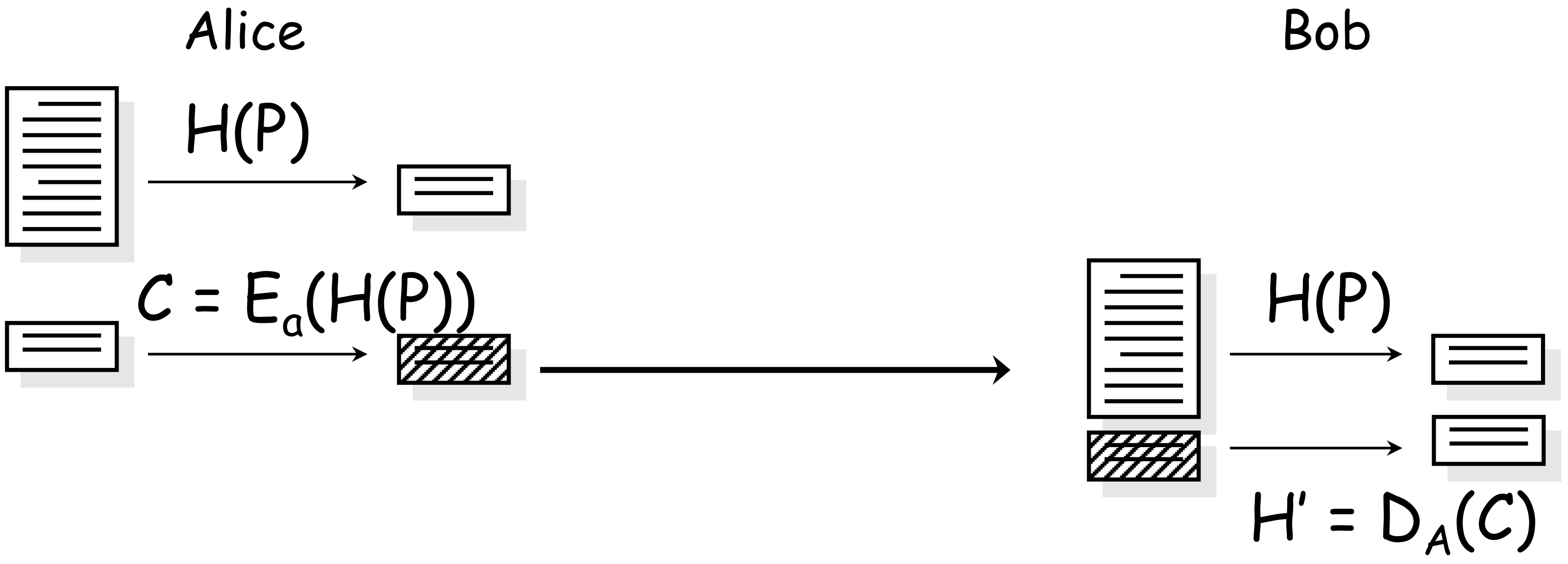
Alice sends Bob the message and the encrypted hash

Digital signatures - public key cryptography



1. Bob decrypts the has using Alice's public key
2. Bob computes the hash of the message sent by Alice

Digital signatures - public key cryptography



If the hashes match

- the encrypted hash *must* have been generated by Alice
- the signature is valid

Demo of public/privacy keys and digital signature

- Public / Private Key Pairs
  - https://andersbrownworth.com/blockchain/public-private-keys/keys

- Digital signatures
  - https://andersbrownworth.com/blockchain/public-private-keys/signatures

Cryptographic toolbox

- Symmetric encryption
- Public key encryption
- One-way hash functions
- Random number generators
    - Nonces, session keys

Examples

- Key exchange
  - Public key cryptography
- Key exchange + secure communication
  - Public key + symmetric cryptography
- Authentication
  - Nonce + encryption
- Message authentication codes
  - Hashes
- Digital signature
  - Hash + encryption