



$$\begin{aligned} &^2(y f(2) + 401k^2)y_1 + e_2(x)y_2 + e_3(x)y_3 \\ &(x+1) = \left(\frac{x(x-2)}{2}\right)1 + (x(x-1))0 + \left(\frac{x(x-1)}{2}\right) \\ &)^2 = \left(\frac{(x-1)(x-2)}{2}\right)1 + (x(x-1))0 + \left(\frac{x(x-1)}{2}\right) \\ &f_P(x, y) \\ &)^2(y + 6x + 7)^4 - (y + 7x + 8)^2(y + 9x + 6)^4(x + 1) \\ &1)(x + 6)^4(x + 9)^4 \quad x(x + 8)^4(x + 2)^4 \\ &-9b + \sqrt{3}\sqrt{4a^3 + 27b^2})^{1/3} + 6x)^2(y + 10x + 8)x + 1 \\ &\frac{2^{1/3}3^{2/3}}{x(x+6)^2} \quad (y+9x+ \\ &\frac{(y+8x)^2}{(1-i\sqrt{3})(-9b+\sqrt{3}\sqrt{4a^3+27b^2})^{1/3}} \\ &\frac{1}{3} + \frac{2^{4/3}3^{2/3}x+9}{(y+8x)^2(y+7x+4)^4(y+5} \end{aligned}$$

# Cryptography 2

INGI2347: COMPUTER SYSTEM SECURITY (Spring 2015)

Marco Canini

**UCL**  
Université  
catholique  
de Louvain



# Plan for today

## Lecture 7

- Recap on Symmetric vs Public Key Crypto



- RSA

- Diffie-Hellman

- Authentication

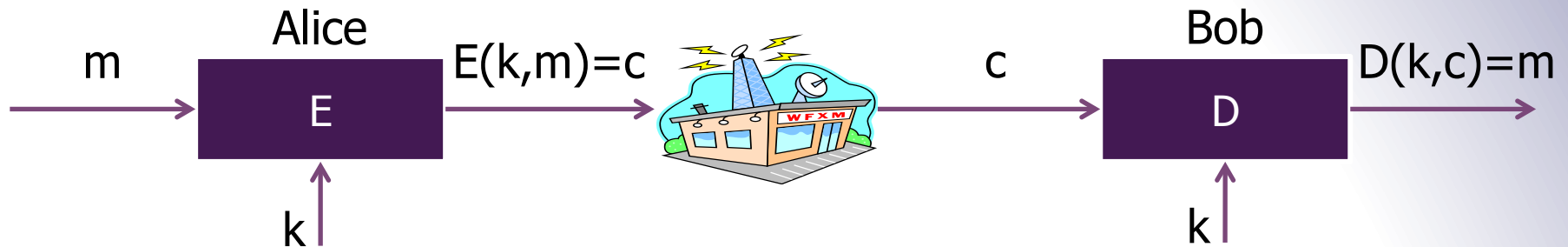
- Generic collision resistance attack

- Integrity

+

# Symmetric Key Encryption

3



$E, D$ : cipher

$k$ : secret key (e.g., 128 bits)

$m, c$ : plaintext, ciphertext

- Same secret key for both encryption and decryption
- Stream ciphers
  - Act on the plaintext one symbol at a time
- Block ciphers
  - Act on the plaintext in blocks of symbols

# Stream Ciphers: The One Time Pad (Vernam 1917)

First example of a “secure” cipher

$$M = C = \{0,1\}^n, \quad K = \{0,1\}^n$$

key = (random bit string as long the message)

$$E(k, m) = k \oplus m$$

$$D(k, c) = k \oplus c$$

msg: 0 1 1 0 1 1 1

key: 1 0 1 1 0 1 0

$\oplus$

---

CT: 1 1 0 1 1 0 1

# One-time vs Many-time Security

**Never use stream cipher key more than once !!**

$$C_1 \leftarrow m_1 \oplus k$$

$$C_2 \leftarrow m_2 \oplus k$$

Eavesdropper does:

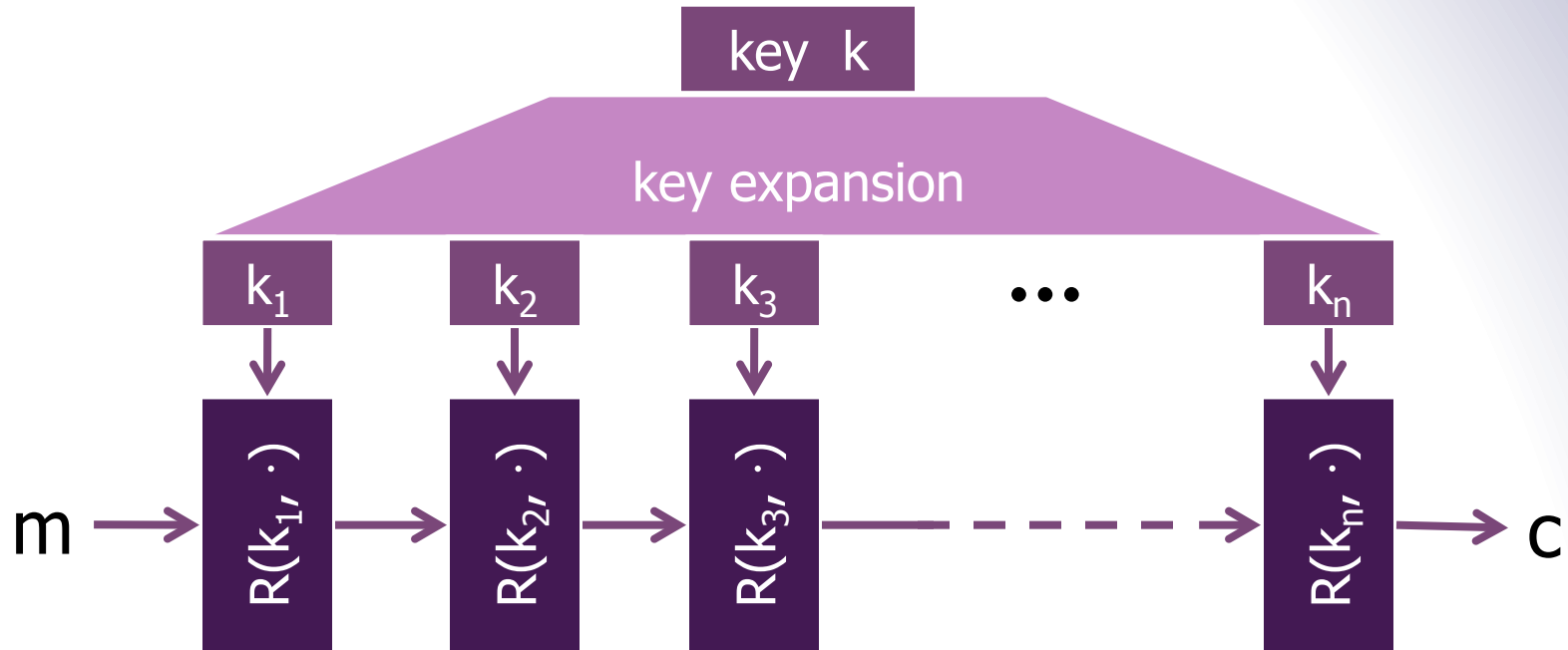
$$C_1 \oplus C_2 \rightarrow m_1 \oplus m_2$$

Enough redundancy in English and ASCII encoding that:

$$m_1 \oplus m_2 \rightarrow m_1, m_2$$



# Block Ciphers Built by Iteration



$R(k, m)$  is called a round function

**for 3DES ( $n=48$ ),      for AES-128 ( $n=10$ )**

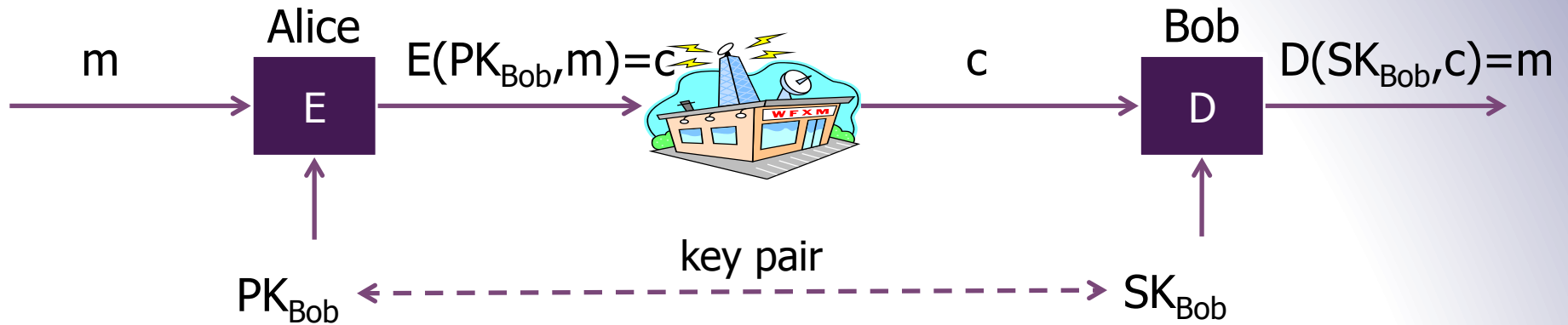
# Problems with Shared Key Crypto

- Compromised key means interceptors can decrypt any ciphertext they've acquired
  - Change keys frequently to limit damage
- Distribution of keys is problematic
  - Keys must be transmitted securely
  - Use couriers?
  - Distribute in pieces over separate channels?
- $O(n)$  keys per user ;  $O(n^2)$  keys in the system
- Online TTP not an ideal solution

+

# Public Key Encryption

8



PK: public key , SK: secret key (e.g., 1024 bits)

Example: Bob generates  $(PK_{Bob}, SK_{Bob})$  and gives  $PK_{Bob}$  to Alice

- Only the private key must be kept secret!
- Interactive applications: session setup
- Non-interactive applications: e.g., email





# Establishing a shared secret

**Alice**

$(pk, sk) \leftarrow G()$

"Alice",  $pk$

**Bob**

choose random  
 $x \in \{0,1\}^{128}$

"Bob",  $c \leftarrow E(pk, x)$

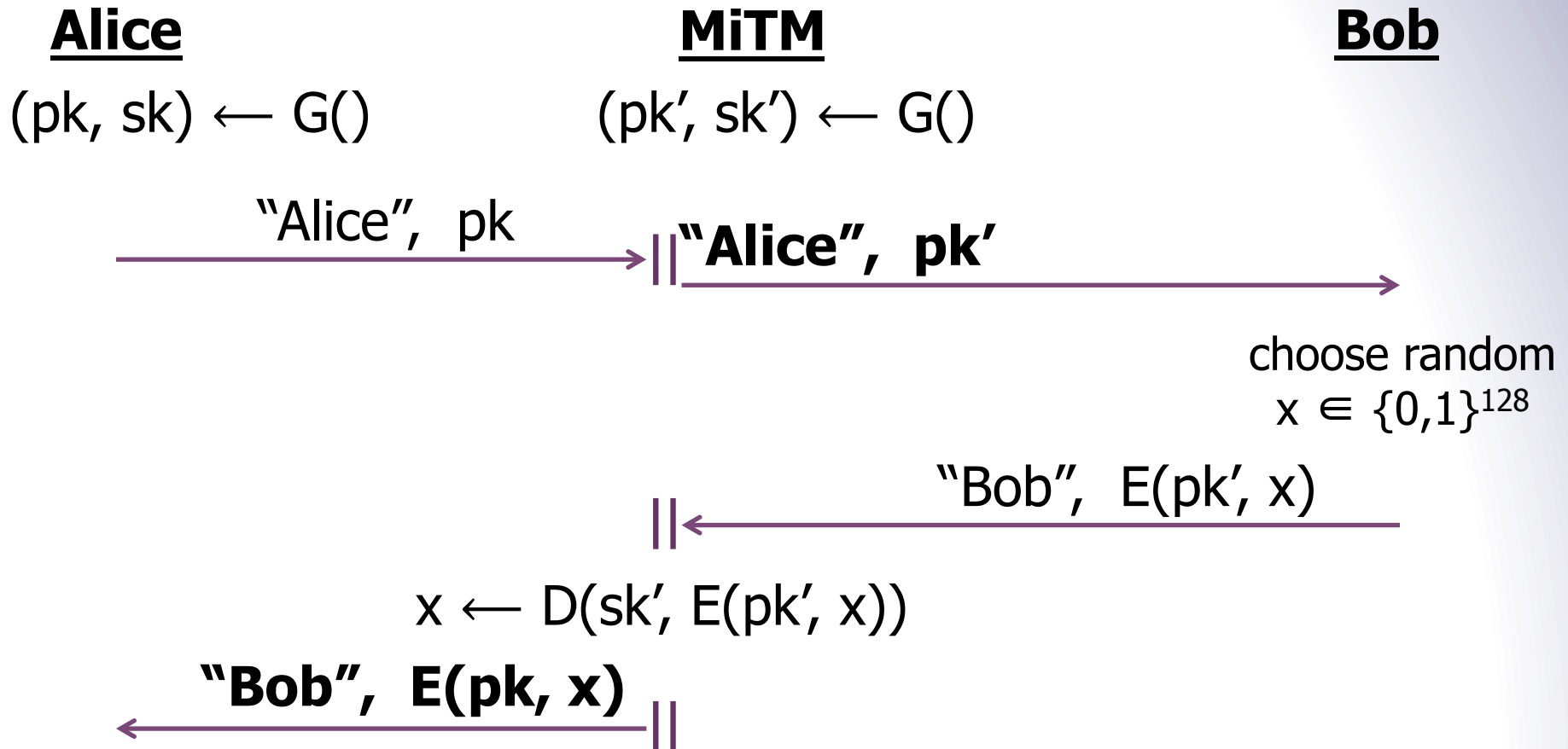
$D(sk, c) \rightarrow x$  shared secret

Note: protocol is vulnerable to man-in-the-middle

+

# Insecure against man in the middle

The protocol is insecure against **active** attacks



# Trade-offs for Public Key Crypto

- More computationally expensive than symmetric (shared) key crypto
  - Algorithms are harder to implement
  - Require more complex machinery
- More formal justification of difficulty
  - Hardness based on complexity-theoretic results
- A principal needs 1 private key and 1 public key
  - Number of keys for pair-wise communication is  $O(n)$

# Model of the attacker

## ■ Ciphertext-only attack

- Attacker has access to cipher text of one or more messages, all of which were encrypted with the same key  $K$
- His goal is to find the corresponding plaintext, or even better  $K$

## ■ Known-plaintext attack

- Attacker has access to one or more plaintext-ciphertext pairs, encrypted with the same key  $K$
- His goal is to determine  $K$ 
  - An example of this is the DES challenge

# Model of the attacker

## ■ Chosen-ciphertext attack (CCA)

- The attacker has access to a **decryption** oracle: he can choose ciphertexts (based on the same key  $K$ ) and get their corresponding plaintext

## ■ Chosen-plaintext attack (CPA)

- The adversary has access to an **encryption** oracle: he can choose plaintexts and get their corresponding ciphertexts, based on the same key  $K$
- More powerful than CCA



+

RSA



# RSA Algorithm

- Ron **R**ivest, Adi **S**hamir, Leonard **A**dleman
  - Proposed in 1979
  - They won the 2002 Turing award for this work
- Has withstood years of cryptanalysis
  - Not a guarantee of security!
  - But a strong vote of confidence
    - Further reading: Twenty years of attacks on the RSA cryptosystem, D. Boneh, Notices of the AMS, 1999
- Hardware implementations:
  - 1000 x slower than DES

# RSA at a High Level

- Public and private key are derived from secret prime numbers
  - Today at least 2048 bits to ensure security (4096 bits is better)
- Plaintext message (a sequence of bits)
  - Treated as a (large!) binary number
- Encryption is modular exponentiation
- To break the encryption, conjectured that one must be able to factor large numbers
  - Not known to be in P (polynomial time algorithms)



# RSA Details: Key Generation

- Choose two distinct random prime numbers  $p$  and  $q$
- Compute the **modulus**:  $n = p \cdot q$
- Compute  $\varphi(n) = (p - 1)(q - 1)$ 
  - $\varphi$  is Euler's totient function
  - $\varphi(n)$  counts the positive integers  $\leq n$  that are relatively prime to  $n$
  - $a$  and  $b$  are relatively prime iff their greatest common divisor = 1
    - $\text{GDC}(a, b) = 1$

## Euler's theorem:

- $a^{\varphi(n)} \equiv 1 \pmod{n}$ , for any  $a$  relatively prime with  $n$

# RSA Details: Key Generation

- Choose  $e$  such that  $1 < e < \varphi(n)$  with  $e$  and  $\varphi(n)$  relatively prime
  - $e$  is the **public key exponent** (public key =  $(e, n)$ )
  - Typically small: e.g.  $e = 2^{16} + 1 = 65537$
- Determine  $d \equiv e^{-1} \cdot \text{mod } \varphi(n)$ , that is the multiplicative inverse of  $e$ 
  - $d$  is the **private key exponent** (private key =  $(d, n)$ )
  - We have that  $d \cdot e \equiv 1 \text{ mod } \varphi(n)$



# RSA Details: Key Generation

- Publish  $(e, n)$  as the public key
- Keep  $(d, n)$  as the private key
- $p$ ,  $q$ , and  $\varphi(n)$  must also be kept secret or even thrown away altogether!
  - Why?

# RSA Details: Encryption

- Message  $M$  is turned to an integer  $m$  s.t.  $0 \leq m < n$
- We use the **recipient's public key**  $(e, n)$  to compute:

$$c \equiv m^e \pmod{n}$$

- We use exponentiation by squaring to perform this quickly:

$$m^e \equiv (m^2 \pmod{n})^{(e/2)} \pmod{n} \quad , \text{ if } e \equiv 0 \pmod{2}$$

$$m^e \equiv m (m^2 \pmod{n})^{((e-1)/2)} \pmod{n} \quad , \text{ else}$$

# RSA Details: Encryption Example

## ■ Scaled-down example

- (explicit form of the one on Wikipedia):

$$\begin{aligned} 65^{17} &\equiv 65 (65^2 \bmod 3233)^8 && \equiv 65 \cdot 992^8 \\ &\equiv 65 (992^2 \bmod 3233)^4 && \equiv 65 \cdot 1232^4 \\ &\equiv 65 (1232^2 \bmod 3233)^2 && \equiv 65 \cdot 1547^2 \\ &\equiv 65 (1547^2 \bmod 3233) && \equiv 65 \cdot 789 \\ &\equiv 2790 \pmod{3233} \end{aligned}$$

# RSA Details: Decryption

- The recipient uses its private key  $(d, n)$  to compute:

$$m \equiv c^d \pmod{n}$$

- This works. Why?

$$c^d \pmod{n} \equiv (m^e \pmod{n})^d \pmod{n}$$

$$\equiv m^{(e \cdot d)} \pmod{n}$$

$$\equiv m^1 \pmod{n}$$

- Last step works thanks to Euler's theorem and Fermat's Little Theorem

# RSA Details: Miscellaneous

- How to encrypt long messages ( $m > n$ )?
  - Use a mode of encryption such as Cipher Block Chaining (CBC)?
  - **Too expensive!**
  - Use hybrid encryption: encrypt a symmetric key with RSA, then use this to **encrypt the bulk data**
- How would one do signature with RSA?
  - Sign the message by applying the decryption alg. with the private key
  - For long messages, hash the message first, then sign the hash value

# RSA Details: Miscellaneous

- The “1024” bits (or 2048, or 4096, ...) is the size of the **modulus**  $n$
- Does that mean “1024-bit security”, like with block ciphers?

- **No!**

cipher key size

80 bits

128 bits

256 bits (AES)

modulus size

1024 bits

3072 bits

**15360** bits

- RSA is not CCA-secure (see exercises), but it is never used as explained here!



# Trapdoor functions (TDF)

**Def:** a trapdoor func.  $X \rightarrow Y$  is a triple of efficient algs.  $(G, F, F^{-1})$

- $G()$ : randomized alg. outputs key pair  $(pk, sk)$
- $F(pk, \cdot)$ : det. alg. that defines a func.  $X \rightarrow Y$
- $F^{-1}(sk, \cdot)$ : defines a func.  $Y \rightarrow X$  that inverts  $F(pk, \cdot)$

Security:  $F(pk, \cdot)$  is one-way without  $sk$

# Public-key encryption from TDFs

- $(G, F, F^{-1})$ : secure TDF  $X \rightarrow Y$
- $(E_s, D_s)$ : symm. auth. encryption with keys in  $K$
- $H: X \rightarrow K$  a hash function

We construct a pub-key enc. system  $(G, E, D)$ :

Key generation  $G$ : same as  $G$  for TDF

# Public-key encryption from TDFs

- $(G, F, F^{-1})$ : secure TDF  $X \rightarrow Y$
- $(E_s, D_s)$ : symm. auth. encryption with keys in  $K$
- $H: X \rightarrow K$  a hash function

**$E(pk, m)$**  :

$x \xleftarrow{R} X, \quad y \leftarrow F(pk, x)$   
 $k \leftarrow H(x), \quad c \leftarrow E_s(k, m)$   
output  $(y, c)$

**$D(sk, (y, c))$**  :

$x \leftarrow F^{-1}(sk, y),$   
 $k \leftarrow H(x), \quad m \leftarrow D_s(k, c)$   
output  $m$



# Diffie-Hellman Key Exchange

# Diffie-Hellman Key Exchange

- Problem with shared-key systems:

Distributing the shared key

- Suppose that Alice and Bob want to agree on a secret (i.e. a key)
  - Communication link is public
  - They don't already share any secrets

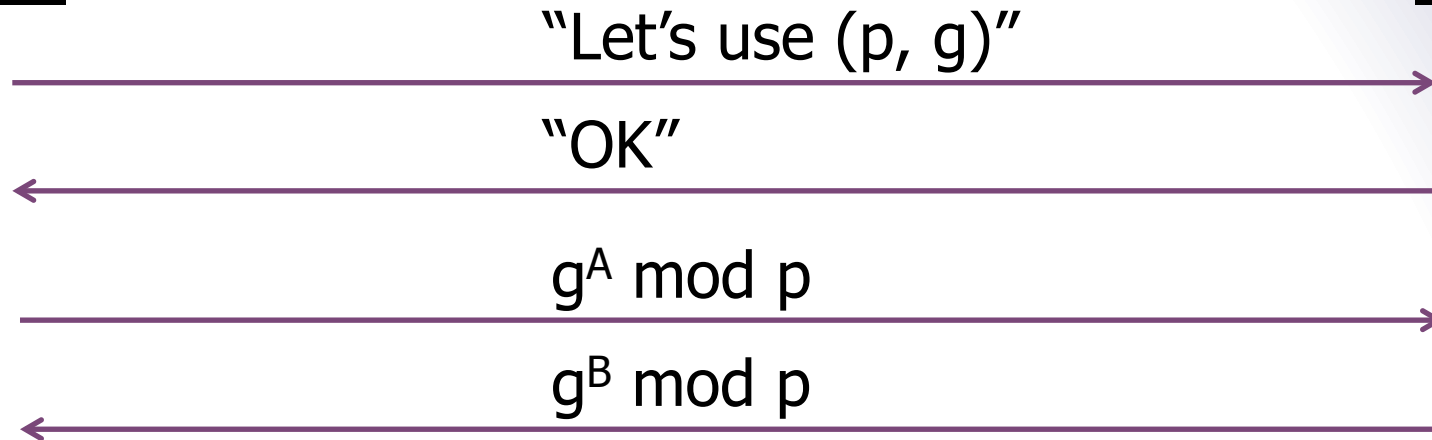
# Diffie-Hellman Key Exchange

- Choose a prime  $p$  (publicly known)
  - Should be about 512 bits or more
- Pick  $g < p$  (also public)
  - $g$  must be a *primitive root* of  $p$
  - A primitive root generates the finite field  $p$
  - Every  $n$  in  $\{1, 2, \dots, p-1\}$  can be written as  $g^k \bmod p$
  - Example: 2 is a primitive root of 5
  - $2^0 = 1$        $2^1 = 2$        $2^2 = 4$        $2^3 = 3 \pmod{5}$
- Intuitively means that it's hard to take logarithms base  $g$  because there are many candidates

# Diffie-Hellman Protocol

Alice

Bob



1. Alice & Bob decide on a public prime  $p$  and primitive root  $g$
2. Alice chooses secret number  $A$       Bob chooses secret number  $B$
3. Alice sends Bob  **$g^A \bmod p$**       Bob sends Alice  **$g^B \bmod p$**
4. The shared secret is  **$g^{AB} \bmod p$**

Note: security against eavesdropping only (vulnerable to man-in-the-middle)

# Diffie-Hellman Details

- Alice computes  $g^{AB} \bmod p$  because she knows A:

$$g^{AB} \bmod p = (g^B \bmod p)^A \bmod p$$

- An eavesdropper gets  $g^A \bmod p$  and  $g^B \bmod p$ 
  - They can easily calculate  $g^{A+B} \bmod p$  but that doesn't help
  - The problem of computing discrete logarithms (to recover A from  $g^A \bmod p$ ) is hard



# Diffie-Hellman Example

- Alice and Bob agree that  $p=71$  and  $g=7$
- Alice selects a private key  $A=5$  and calculates a public key
$$g^A \equiv 7^5 \equiv 51 \pmod{71} \quad ; \text{ she sends this to Bob}$$
- Bob selects a private key  $B=12$  and calculates a public key
$$g^B \equiv 7^{12} \equiv 4 \pmod{71} \quad ; \text{ he sends this to Alice}$$
- Alice calculates the shared secret:

$$S \equiv (g^B)^A \equiv 4^5 \equiv 30 \pmod{71}$$

- Bob calculates the shared secret:

$$S \equiv (g^A)^B \equiv 51^{12} \equiv 30 \pmod{71}$$

# Why Does It Work?

- Security is provided by the difficulty of calculating discrete logarithms
- Feasibility is provided by
  - The ability to find large primes
  - The ability to find primitive roots for large primes
  - The ability to do efficient modular arithmetic
- Correctness is an immediate consequence of basic facts about modular arithmetic



# Authentication



# Authenticated channel

- You should always expect a **man-in-the-middle**
  - e.g. on the internet, your messages go through many intermediaries
- Solution: Use an authenticated channel
  - For instance, Alice and Bob have certificates that contain a public key, and exchange them prior to the DH exchange
  - They use them to authenticate the values in the DH phase
  - More on that in the SSL/TLS lecture



# Collision resistance

Generic birthday attack

# Cryptographic Hashes

- Create a hard-to-invert summary of input data

$$h: \{0,1\}^* \xrightarrow{\text{hash}} \{0,1\}^n$$

- Sometimes called a Message Digest
- Examples:
  - Secure Hash Algorithm (SHA)
  - Message Digest (MD4, MD5)

# Desired Properties

## ■ One way hash function

- Given a hash value  $y$ , it should be infeasible to find  $m$  s.t.  $h(m)=y$

## ■ Collision resistance

- It should be infeasible to find two different messages  $m_1$  and  $m_2$  s.t.  $h(m_1)=h(m_2)$

## ■ Random oracle property

- $h(m)$  is indistinguishable from a random  $n$ -bit value
- Attacker must spend a lot of effort to be able to modify the message without altering the hash value

# Generic attack on C.R. functions

Let  $H: M \rightarrow \{0,1\}^n$  be a hash function ( $|M| \gg 2^n$ )

Generic alg. to find a collision **in time**  $O(2^{n/2})$  hashes

Algorithm:

1. Choose  $2^{n/2}$  random messages in  $M$ :  $m_1, \dots, m_{2^{n/2}}$   
(distinct w.h.p.)
2. For  $i = 1, \dots, 2^{n/2}$  compute  $t_i = H(m_i) \in \{0,1\}^n$
3. Look for a collision ( $t_i = t_j$ ).  
If not found, got back to step 1.

How well will this work?



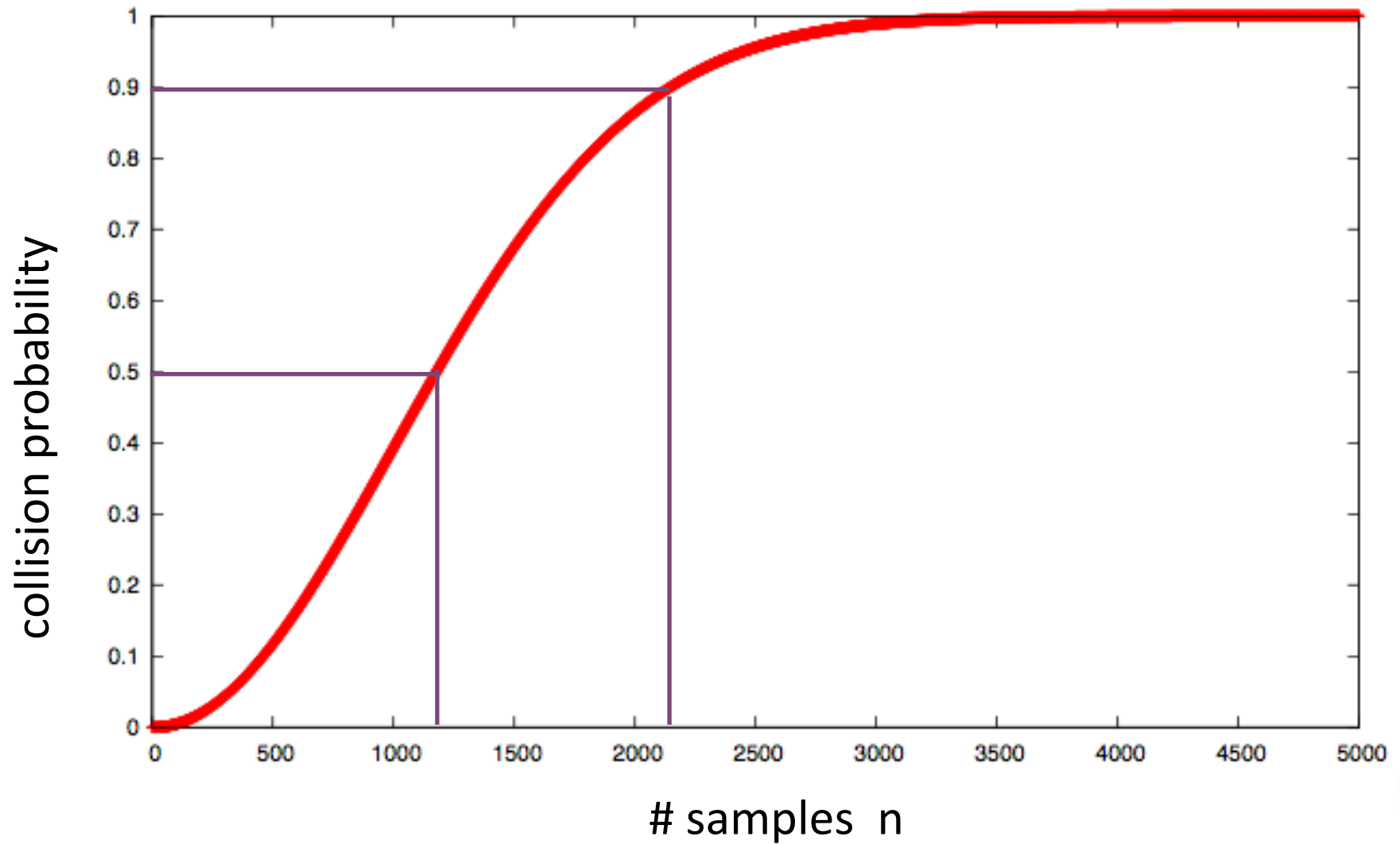
# The birthday paradox

- In a group of **23** people, the probability to have at least two people with the same birthday is about **50%**
- Theorem: If we pick  $\theta \sqrt{N}$  independently and uniformly distributed random numbers in  $\{1, 2, \dots, N\}$ , we get at least two occurrences of the same number with probability:

$$1 - \frac{N!}{N^{\theta\sqrt{N}} (N - \theta\sqrt{N})!} \xrightarrow{N \rightarrow +\infty} 1 - e^{-\frac{\theta^2}{2}}$$

$N=10^6$

42



# Generic attack

$H: M \rightarrow \{0,1\}^n$  . Collision finding algorithm:

1. Choose  $2^{n/2}$  random elements in  $M$ :  $m_1, \dots, m_{2^{n/2}}$
2. For  $i = 1, \dots, 2^{n/2}$  compute  $t_i = H(m_i) \in \{0,1\}^n$
3. Look for a collision ( $t_i = t_j$ ).  
If not found, got back to step 1.

Expected number of iteration  $\approx 2$

Running time:  $O(2^{n/2})$  (space  $O(2^{n/2})$ )



# Integrity



# Message Integrity

Goal:     **integrity**,   no confidentiality

Examples:

- Protecting public binaries on disk
- Protecting banner ads on web pages

# Message Integrity: MAC



**Generate tag:**  
 $\text{tag} \leftarrow S(k, m)$

**Verify tag:**  
 $V(k, m, \text{tag}) \stackrel{?}{=} \text{'yes'}$

Def: **MAC**  $I=(S,V)$  defined over  $(K,M,T)$  is a pair of algs

- $S(k,m)$  outputs  $t$  in  $T$
- $V(k,m,t)$  outputs 'yes' or 'no'

Consistency:

$$\forall k \in K, \forall m \in M: \quad V(k, m, S(k, m)) = \text{'yes'}$$

# Secure MACs

- Attacker information: chosen message attack
  - for  $m_1, m_2, \dots, m_q$  attacker is given  $t_i \leftarrow S(k, m_i)$

- Attacker's goal: existential forgery.
  - produce some **new** valid message/tag pair  $(m, t)$ .

$$(m, t) \notin \{ (m_1, t_1), \dots, (m_q, t_q) \}$$

---

$\Rightarrow$  attacker cannot produce a valid tag for a new message

$\Rightarrow$  given  $(m, t)$  attacker cannot even produce  $(m, t')$  for  $t' \neq t$

+

48

## Secure PRF $\Rightarrow$ Secure MAC

For a Pseudo Random Function  $F: K \times X \rightarrow Y$  define a MAC  $I_F = (S, V)$  as:

- $S(k, m) := F(k, m)$
- $V(k, m, t)$ : output 'yes' if  $t = F(k, m)$  and 'no' otherwise.

$\Rightarrow I_F$  is secure as long as  $|Y|$  is large, say  $|Y| = 2^{80}$





# Standardized method: HMAC (Hash-MAC)

- Most widely used MAC on the Internet
  - Proposed by Bellare, Canetti, Krawczyk in 1996
  - Provably secure
  - Standards: FIPS 198-1, RFC 2104, ISO 9797-2
- Builds a MAC out of a hash function

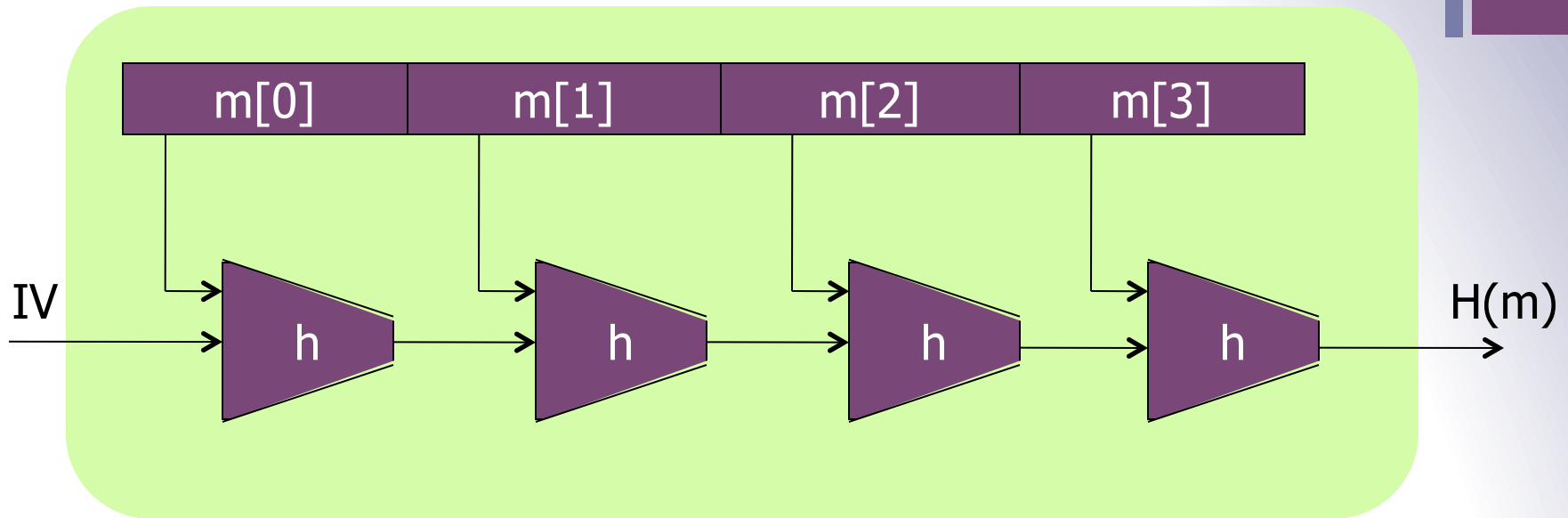
$$\text{HMAC: } S(k, m) = H(k \oplus \text{opad} \parallel H(k \oplus \text{ipad} \parallel m))$$

- Maintains performance of the original hash function
- Examples:
  - HMAC-SHA256:  $H = \text{SHA256}$  ; output is 256 bits
  - HMAC-SHA1-96:  $H = \text{SHA1}$  ; output truncated to 96 bits

+

50

# SHA-256: Merkle-Damgard



$h(t, m[i])$ : compression function

Thm 1: if  $h$  is collision resistant then so is  $H$

“Thm 2”: if  $h$  is a PRF then HMAC is a PRF

# An Insecure MAC Construction

- Let us define  $t = S(m, k) = H(k || m)$
- Because of the way typical hash function are implemented (up to SHA-2), the "Merkle-Damgård" construction, an attack is possible
- An adversary can compute  $t' = H(k || m || \text{padding} || m')$  without knowing  $m$
- She can therefore send  $m', t'$  instead of  $m, t$



# Things To Remember

## ■ Cryptography is:

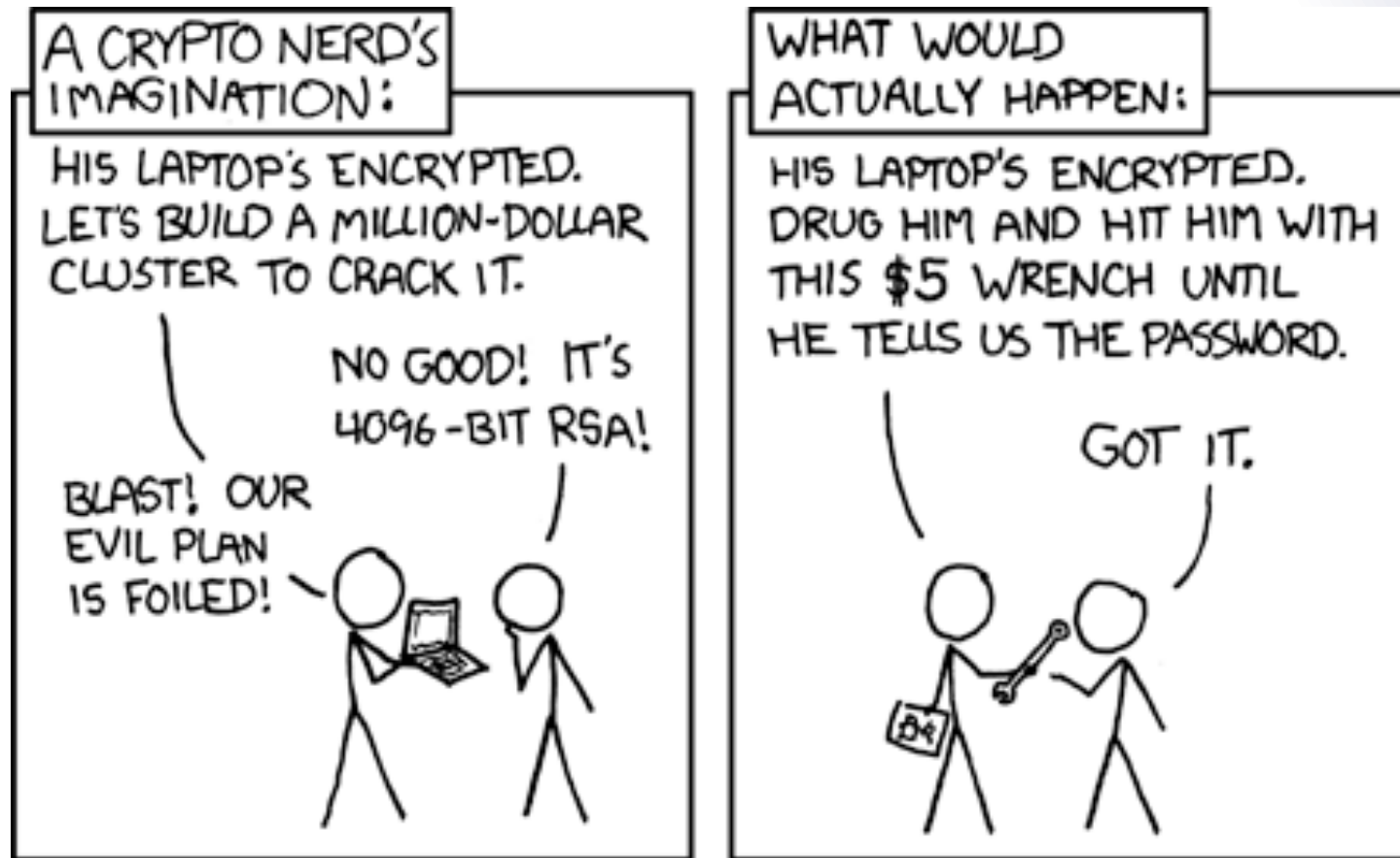
- A tremendous tool
- The basis for many security mechanisms

## ■ Cryptography is **NOT**:

- The solution to all security problems
- Reliable unless implemented and used properly
- Something you should try to invent yourself



# Any questions?



# Stay tuned



Next time you will learn about

## Network vulnerabilities