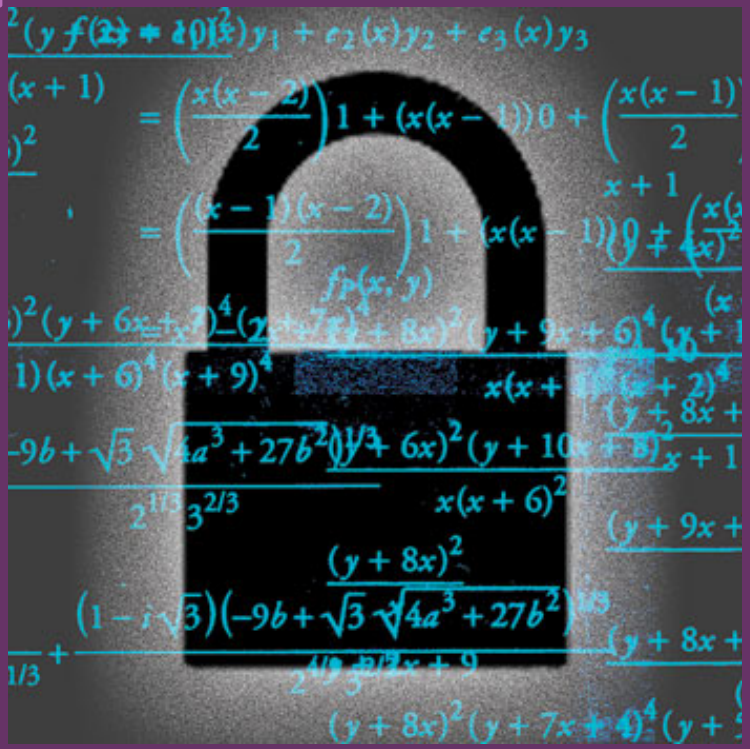


Cryptography 2

INGI2347: COMPUTER SYSTEM SECURITY (Spring 2016)

Marco Canini

Université catholique de Louvain

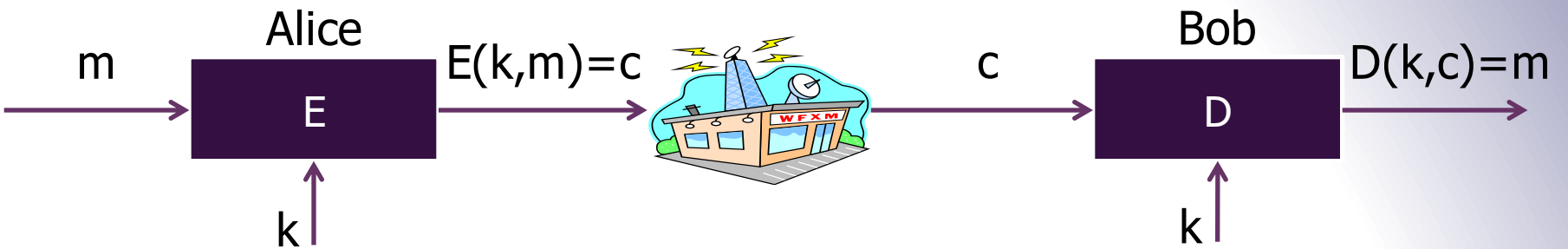
Lecture slides adapted from UCL INGI2347 by Gildas Avoine, Upenn CSE331 by Steve Zdancewic, Stanford Cryptography I@Coursera by Dan Boneh, and CMU 18487 by David Brumley

Plan for today

Lecture 5

- Recap on Symmetric vs Public Key Crypto
- Crypto hash functions
- Generic collision resistance attack
- RSA
- Diffie-Hellman
- Authentication
- Integrity

Symmetric Key Encryption



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 !!

$$C_1 \leftarrow m_1 \oplus k$$
$$C_2 \leftarrow m_2 \oplus k$$

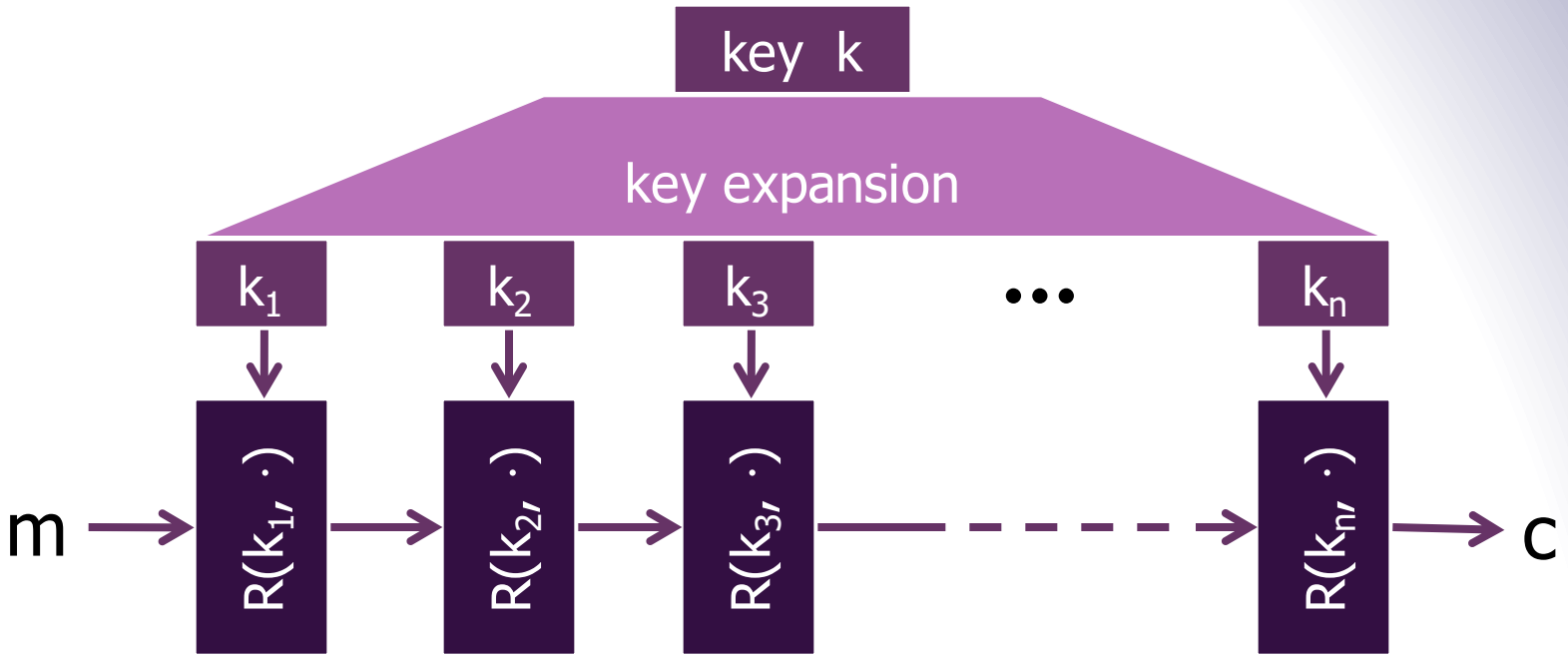
Eavesdropper does:

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

Enough redundancy in English and ASCII encoding that:

$$m_1 \oplus m_2 \rightarrow m_1 \text{ , } m_2$$

Block Ciphers Built by Iteration



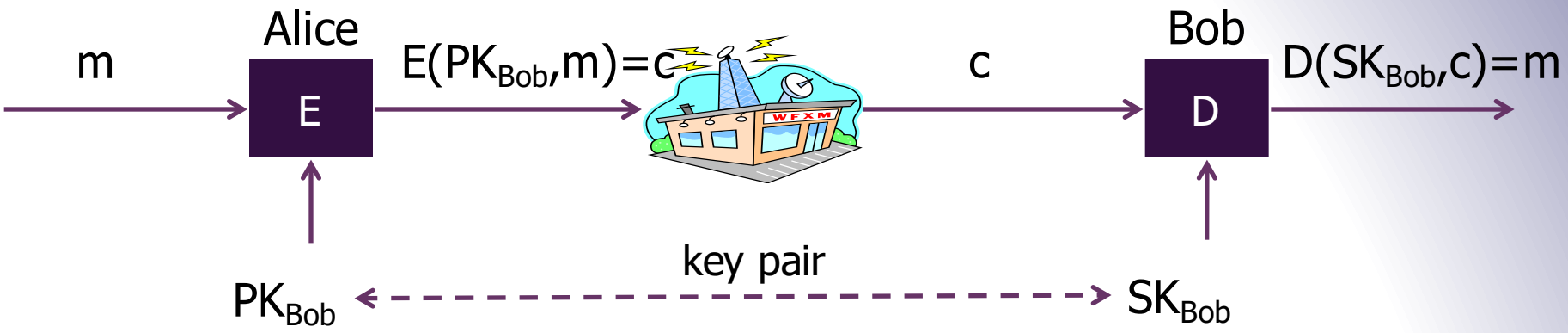
R(k,m) is called a round function

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

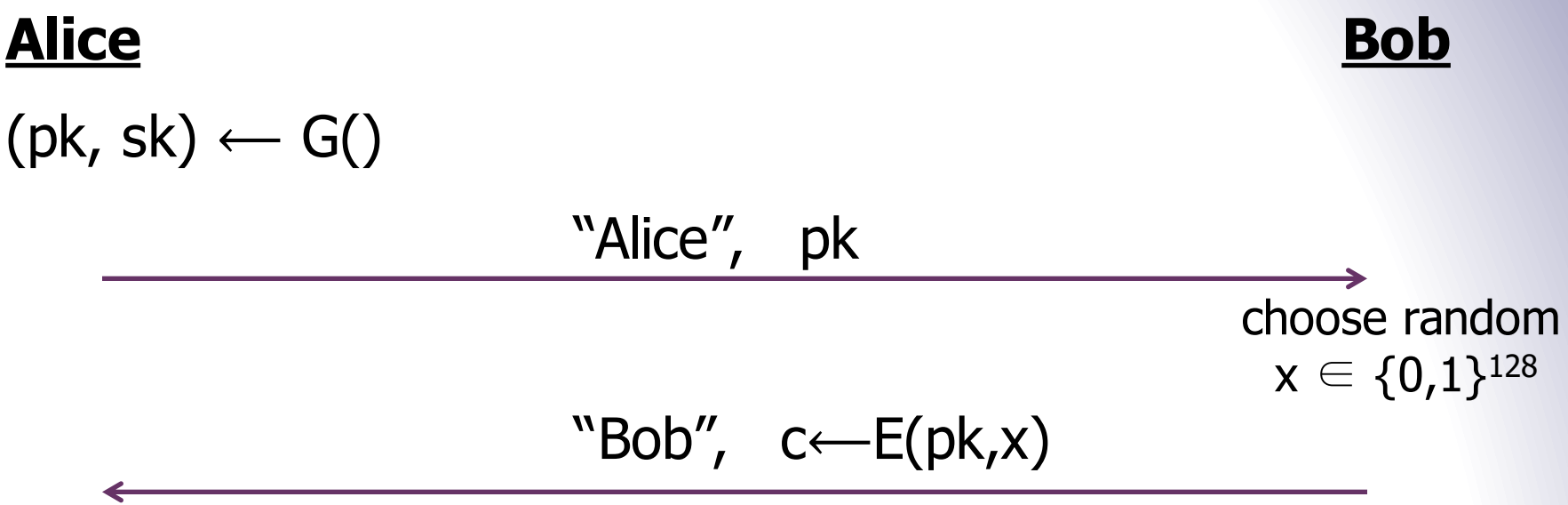


PK: public key , SK: secret key (e.g., 1024 bits)

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

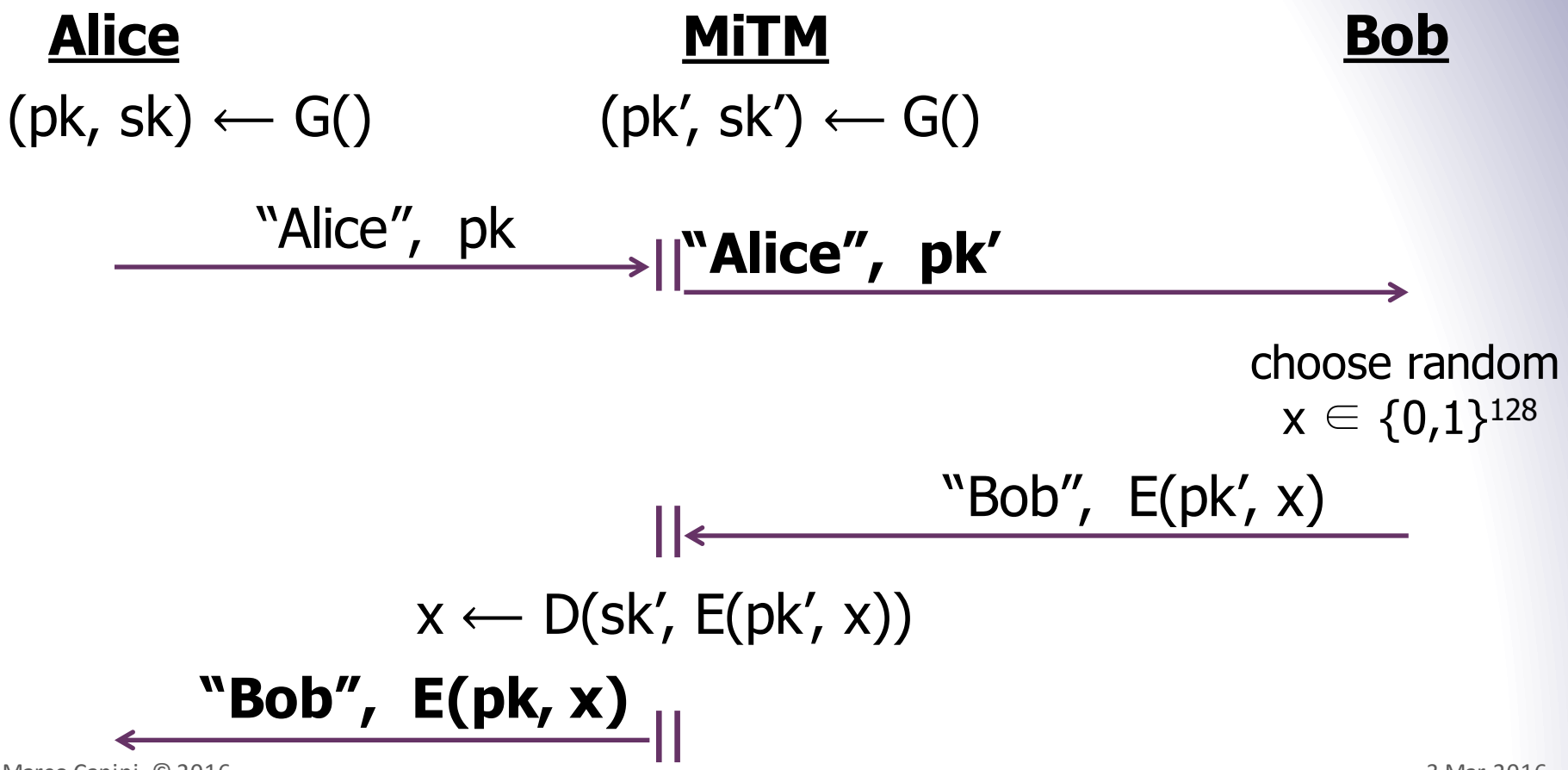
Establishing a shared secret



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 ciphertext 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

Auguste Kerckhoffs

- A cryptosystem should be secure even if **everything** about the system, except the secret key, **is public knowledge**.

+

Crypto hash functions

Hash Algorithms

- Take a variable length string
- Produce a fixed length digest

hash

$$h : \{0,1\}^{*} \rightarrow \{0,1\}^{n}$$

- (Non-cryptographic) Examples:
  - Parity (or byte-wise XOR)
  - CRC
- Realistic Example:
  - The NIST Secure Hash Algorithm (SHA-1) takes a message of less than 2^{64} bits and produces a digest of 160 bits

Cryptographic Hashes

- Create a hard-to-invert summary of input data
- Like a check-sum or error detection code
  - Uses a cryptographic algorithm internally
  - More expensive to compute
- Sometimes called a Message Digest
- Examples:
  - Secure Hash Algorithm (SHA)
    - SHA-1: 160 bits SHA-256: 256 bits SHA-512: 512 bits
  - Message Digest (MD4, MD5)

Desired Properties

- One way hash function
  - Given a hash value y, it should be **infeasible to find** m s.t. $\mathrm{h}(m) = y$
- Collision resistance
  - It should be **infeasible to find two different messages** m_1 **and** m_2 s.t. $\mathrm{h}(m_1) = \mathrm{h}(m_2)$
- Random oracle property
  - $\mathrm{h}(m)$ **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

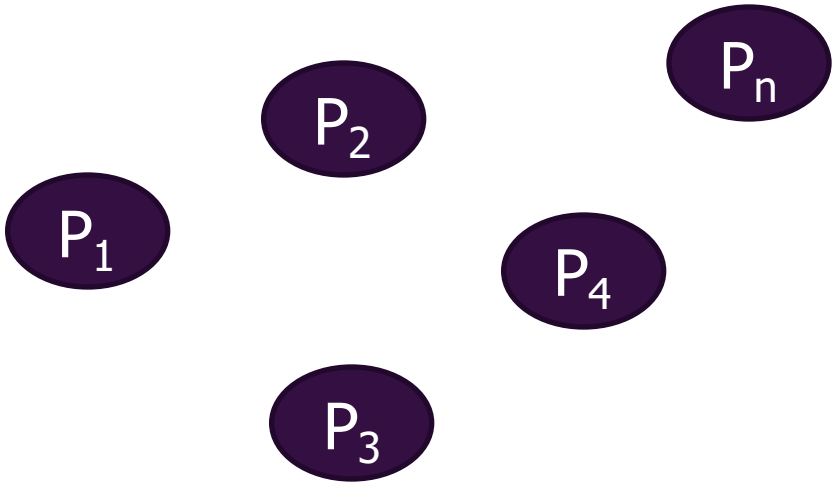
+

Collision resistance

Generic birthday attack

Birthday Paradox

Put n people in a room. What is the probability that 2 of them have the same birthday?



$PR[P_i = P_j] > 50\%$ with 23 people.
(Think: n^2 different pairs)

Birthday Paradox Rule of Thumb

Given B possibilities, and random samples x_1, …, x_n, $PR[x_i = x_j] \approx 50\%$ when $n = B^{1/2}$

Generic attack on C.R. functions

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

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

Algorithm:

1. Choose $\mathbf{2^{n/2}}$ random messages in M: $m_1, \ldots, m_{2^{n/2}}$
   (distinct with high prob.)
2. For i = 1, …, $2^{n/2}$ 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

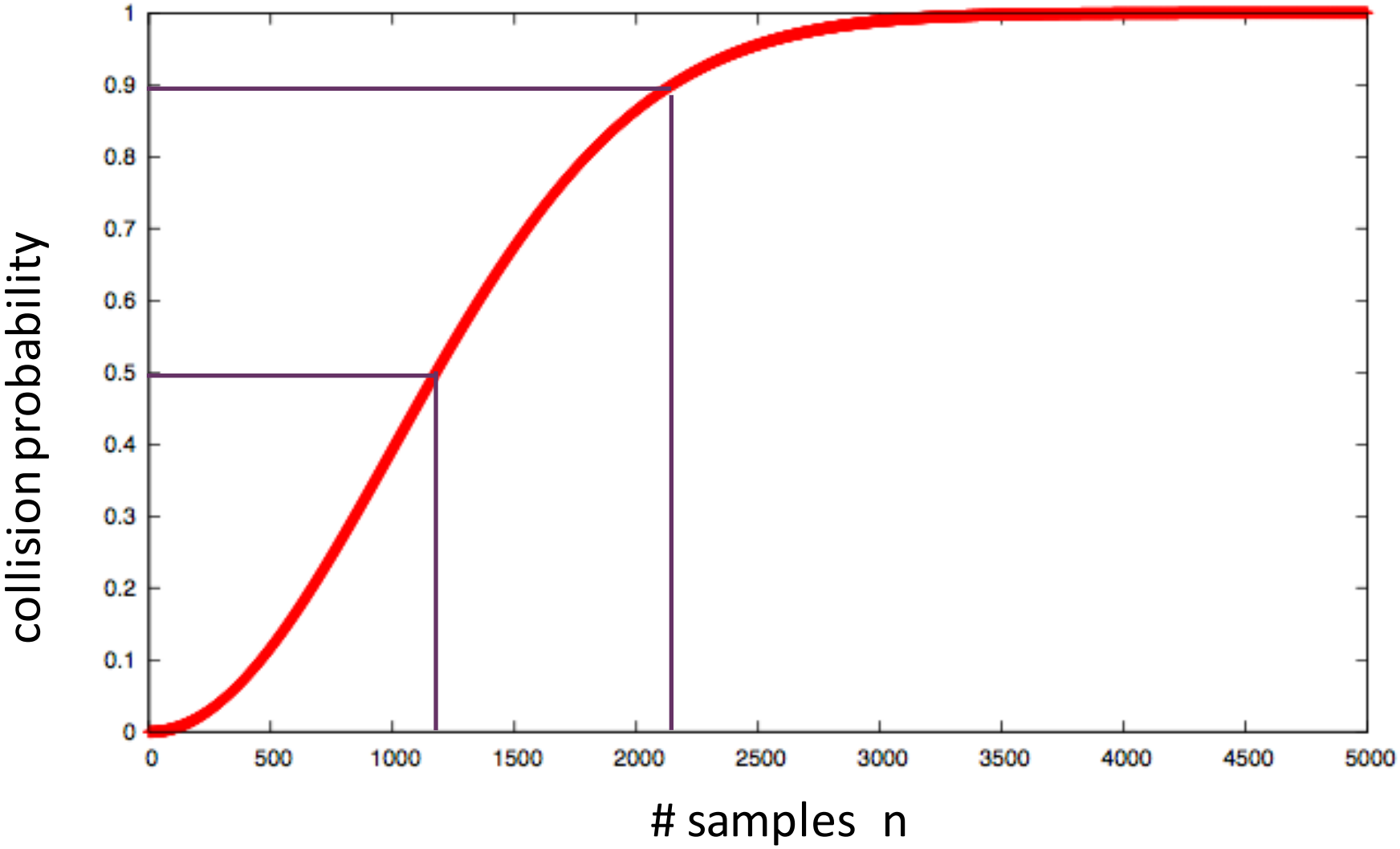
Let $r_1, \ldots, r_n \in \{1,\ldots,B\}$ be indep. identically distributed integers.

Thm: we get at least two occurrences of the same number with probability:

$$p(n,B) \approx 1 - e^{\frac{-n^2}{2B}}$$

when $\mathbf{n} = 1.2 \times \mathbf{B^{1/2}}$ then $\Pr[\ \exists i \neq j: \ r_i = r_j\ ] \geq \frac{1}{2}$

$B=10^6$


collision probability
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
samples n

Generic attack

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

1. Choose $\mathbf{2^{n/2}}$ random elements in M: $m_1, \ldots, m_{2^{n/2}}$
2. For i = 1, …, $2^{n/2}$ 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.

If H: M-> $\{0,1\}^n$, then Pr[collision] ~ ½ with $2^{n/2}$ hashes

Expected number of iteration:

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

Sample C.R. hash functions:

Crypto++ 5.6.0 [Wei Dai]

AMD Opteron, 2.2 GHz (Linux)

	function	digest size (bits)	Speed (MB/sec)	generic attack time
NIST standards	SHA-1	160	153	2^{80}
NIST standards	SHA-256	256	111	2^{128}
	SHA-512	512	99	2^{256}
	Whirlpool	512	57	2^{256}

* best known collision finder for SHA-1 requires 2^{51} hash evaluations

RSA

RSA Algorithm

- Ron **R**ivest, Adi **S**hamir, Leonard **A**dleman
  - Proposed in 1977
  - 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
- Very widely used:
  - SSL/TLS: certificates and key exchange
  - Secure email and file systems

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)

Modular arithmetic facts

- Let $n = p \cdot q$ where p and q are prime numbers
  - a and b are relatively prime iff their greatest common divisor = 1
    - $\mathrm{GDC}(a, b) = 1$
- Num. of pos. integers < n that are relatively prime to n
- $\varphi(n) = (p-1)(q-1) = \mathrm{n - p - q + 1}$
  - φ is Euler's totient function

$a \equiv b \pmod{n}$
is equivalent to
$a \bmod n = b \bmod n$

Euler's theorem:

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

RSA Details: Key Generation

- Choose two distinct random prime numbers p and q
- Compute the **modulus**: $n = p \cdot q$
- 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 $d \equiv e^{-1} \pmod{\varphi(n)}$, that is the modular multiplicative inverse of e (modulo $\varphi(n)$)
  - Solve for d given $d \cdot e \equiv 1 \pmod{\varphi(n)}$
  - d is the **private key exponent** (private key $= (d, n)$)

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 ciphertext:

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

- We use modular exponentiation by squaring to compute this efficiently:

$$m^e \equiv (m^2 \bmod n)^{(e/2)} \pmod{n}, \text{ if } e \text{ is even}$$

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

RSA Details: Encryption Example

- Choose two distinct prime numbers, such as $p = 61$ and $q = 53$
- Compute $n = p \cdot q$ giving $n = 61 \times 53 = 3233$
- Compute the totient of the product as $\varphi(n) = (p - 1)(q - 1)$ giving $\varphi(3233) = (61 - 1)(53 - 1) = 3120$
- Choose any number $1 < e < 3120$ that is relative prime to 3120
  Let $e = 17$
  - Choosing a prime number for e leaves us only to check that e is not a divisor of 3120
- Compute d, the modular multiplicative inverse of $e \ (mod \ \varphi(n))$ yielding $d \cdot e \bmod \varphi(n) = 1 \quad \Rightarrow \quad d = 2753$
- The **public key** is $(n = 3233, e = 17)$
- The **private key** is $(n = 3233, d = 2753)$

RSA Details: Encryption Example

- To encrypt $m = 65$, we calculate $c = 65^{17} \pmod{3233} = 2790$
- Via modular exponentiation by squaring:
  - Note: each congruence is $\pmod{3233}$, omitted below

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

RSA Details: Decryption

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

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

- This works. Why?

$$\begin{aligned} c^d \bmod n &\equiv (m^e \bmod n)^d \pmod{n} \\ &\equiv m^{(e \cdot d)} \pmod{n} \\ &\equiv m^{(\mathrm{k} \cdot \varphi(n) + 1)} \pmod{n} \\ &\equiv m \cdot (m^{\varphi(n)})^{\mathrm{k}} \pmod{n} \\ &\equiv m^1 \pmod{n} \end{aligned}$$

  - 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

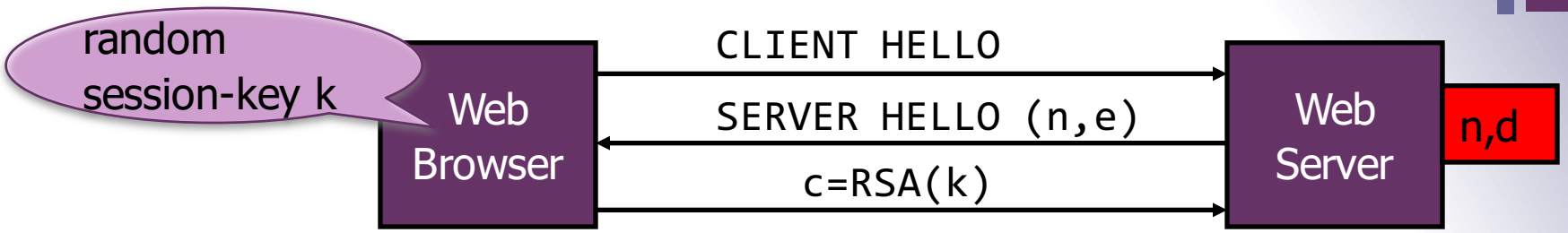
RSA Details: Miscellaneous

- 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	modulus size
80 bits	1024 bits
128 bits	3072 bits
256 bits (AES)	**15360** bits

- RSA as described thus far is not secure, but it is never used as explained before!

A simple attack on textbook RSA



Suppose k is 64 bits: $k \in \{0,\ldots,2^{64}\}$. Eve sees: $c = k^e$

If $\mathbf{k = k_1 \cdot k_2}$ where $k_1, k_2 < 2^{34}$ (prob. ≈20%) then $\mathbf{c/k_1^e = k_2^e}$

Step 1: build table: $c/1^e, c/2^e, c/3^e, \ldots, c/2^{34e}$. time: 2^{34}

Step 2: for $k_2 = 0,\ldots, 2^{34}$ test if k_2^e is in table. time: 2^{34}

Output matching (k_1, k_2). Total attack time: $\approx 2^{40} << 2^{64}$

Trapdoor functions (TDF)

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

- $G()$: randomized alg. outputs key pair (pk, sk)
- $F(pk, \cdot)$: deter. alg. that defines a func. $X \longrightarrow Y$
- $F^{-1}(sk, \cdot)$: defines a func. $Y \longrightarrow X$ 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 $X \longrightarrow Y$
- (E_s, D_s) : symm. auth. encryption with keys in K
- H: $X \longrightarrow K$ 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 $X \longrightarrow Y$
- (E_s, D_s) : symm. auth. encryption with keys in K
- H: $X \longrightarrow K$ a hash function

E(pk, m) :

$x \xleftarrow{Rnd} X$, $y \leftarrow F(pk, x)$

$k \leftarrow H(x)$, $c \leftarrow E_s(k, m)$

output (y, c)

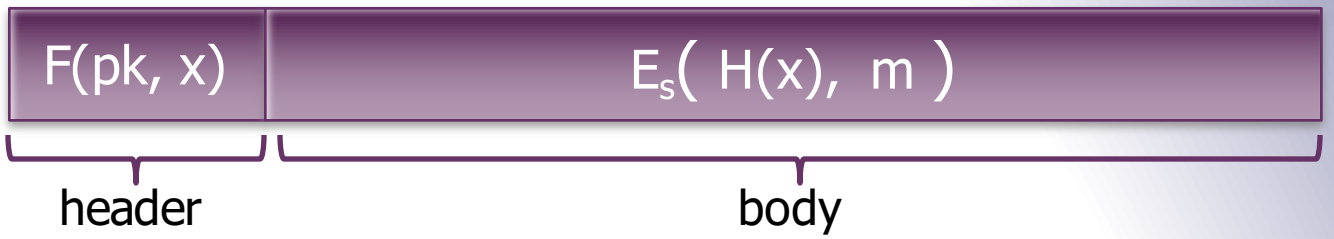
D(sk, (y,c)) :

$x \leftarrow F^{-1}(sk, y)$,

$k \leftarrow H(x)$, $m \leftarrow D_s(k, c)$

output m

In pictures:



Security Theorem:

If **(G, F, F^{-1})** is a secure TDF,
(E_s, D_s) provides auth. enc.
and **H:** $X \longrightarrow K$ is a "random oracle"
then **(G,E,D)** is CCA secure.

Diffie-Hellman Key Exchange

Diffie-Hellman Key Exchange

- Whitfield **Diffie** and Martin **Hellman**
  - Proposed in 1976 in a seminal paper "New Directions in Cryptography"
  - They won the 2015 Turing award for this work

Diffie-Hellman Key Exchange

- Problem with shared-key systems:

  Distributing the shared key

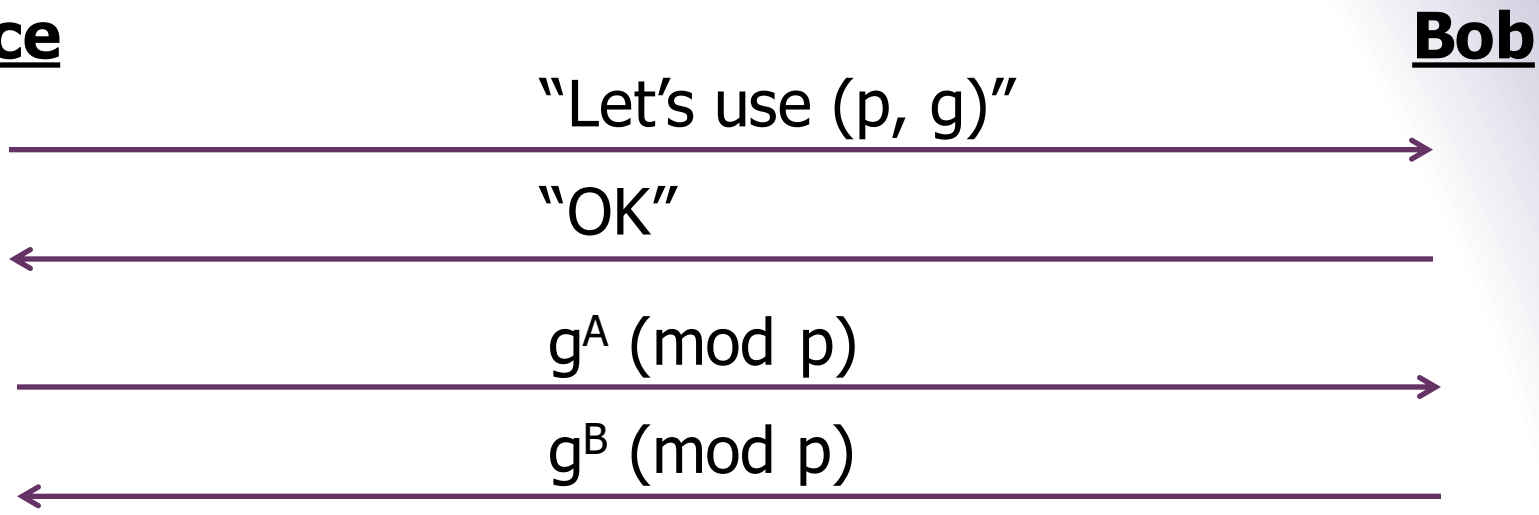
- Suppose that Alice and Bob want to agree on a secret (i.e. a key)
  - Communication link is public
  - They don't already share any secrets

- For now: security against eavesdropping only!

Diffie-Hellman Protocol

Fix a large prime p (e.g. 600 digits ~ 2000 bits)

Fix an integer g in {1, …, p}



1. Alice & Bob decide on p and g (chosen once and fixed forever)
2. Alice chooses secret number **A** Bob chooses secret number **B**
3. Alice sends Bob $\mathbf{g^A \pmod p}$ Bob sends Alice $\mathbf{g^B \pmod p}$
4. The shared secret is $\mathbf{g^{AB} \pmod p}$

Diffie-Hellman Protocol

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

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

- Bob computes $g^{AB} \pmod p$ because he knows B:

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

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

Diffie-Hellman Key Details

- p is a prime (publicly known)
  - Should be at least 600 bits or more
- g < p (also public) is a *primitive root* of p
  - A primitive root generates the finite field p
  - Every n in {1, 2, ..., p-1} can be written as g^k mod p
  - Example: 2 is a primitive root of 5
  - $2^0 = 1$ $2^1 = 2$ $2^2 = 4$ $2^3 = 3$ (mod 5)
  - Intuitively means that it's hard to take logarithms base g because there are many candidates

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

  $g^A \equiv 7^5 \equiv 51 \pmod{71}$; she sends this to Bob
- Bob selects a private key B=12 and calculates a public key

  $g^B \equiv 7^{12} \equiv 4 \pmod{71}$; 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

Insecure against man-in-the-middle

As described, the protocol is insecure against **active** attacks

Alice **MiTM** **Bob**

Now try to construct the attack

+

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

+

Integrity

Message Authentication Codes

Message Integrity

Goal: **integrity**, no confidentiality

Examples:

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

Message Integrity: MACs

Generate tag:
tag ← S(k, m)

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

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

Integrity requires a secret key

Generate tag:
tag ← CRC(m)

Verify tag:
V(m, tag) = ?'yes'

Attacker can easily modify m and re-compute CRC

CRC designed to detect **random**, not malicious errors

Secure MACs

- Attacker's power: chosen message attack
  - for $m_1, m_2, \ldots, m_q$ attacker is given $t_i \leftarrow S(k, m_i)$

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

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

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

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

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 $|Y| = 2^{80}$

tag $\leftarrow$ F(k,m)

accept msg if
tag = F(k,m)

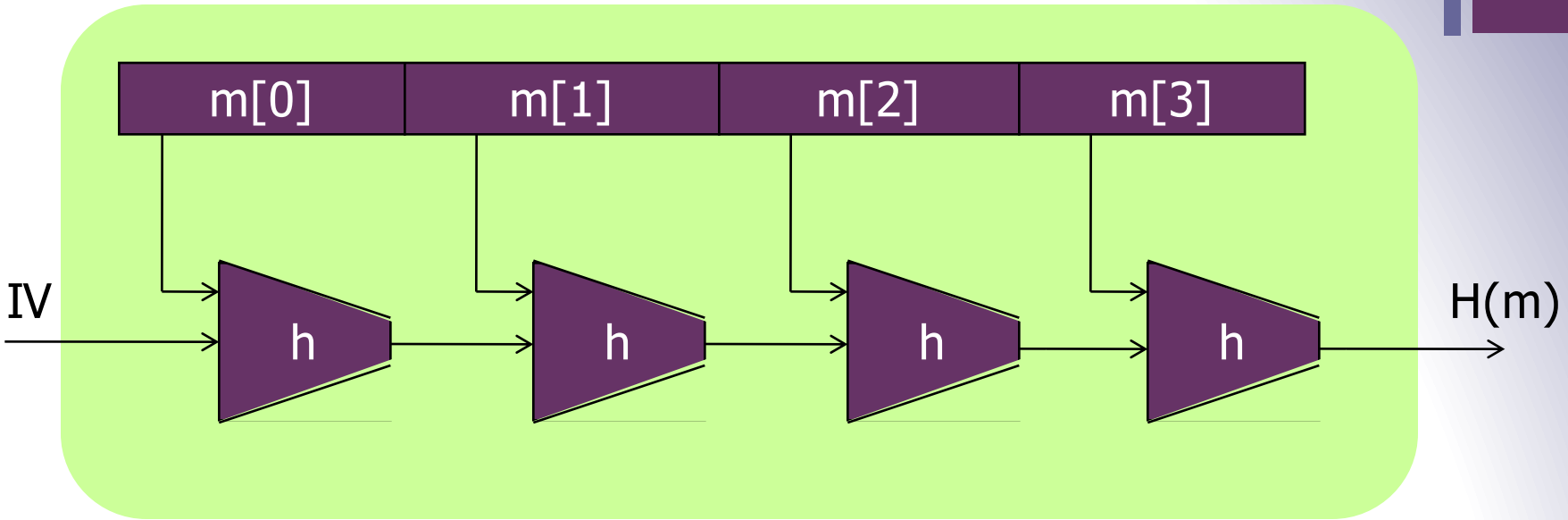
Standardized method: HMAC (Hash-MAC)

- 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

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

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

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

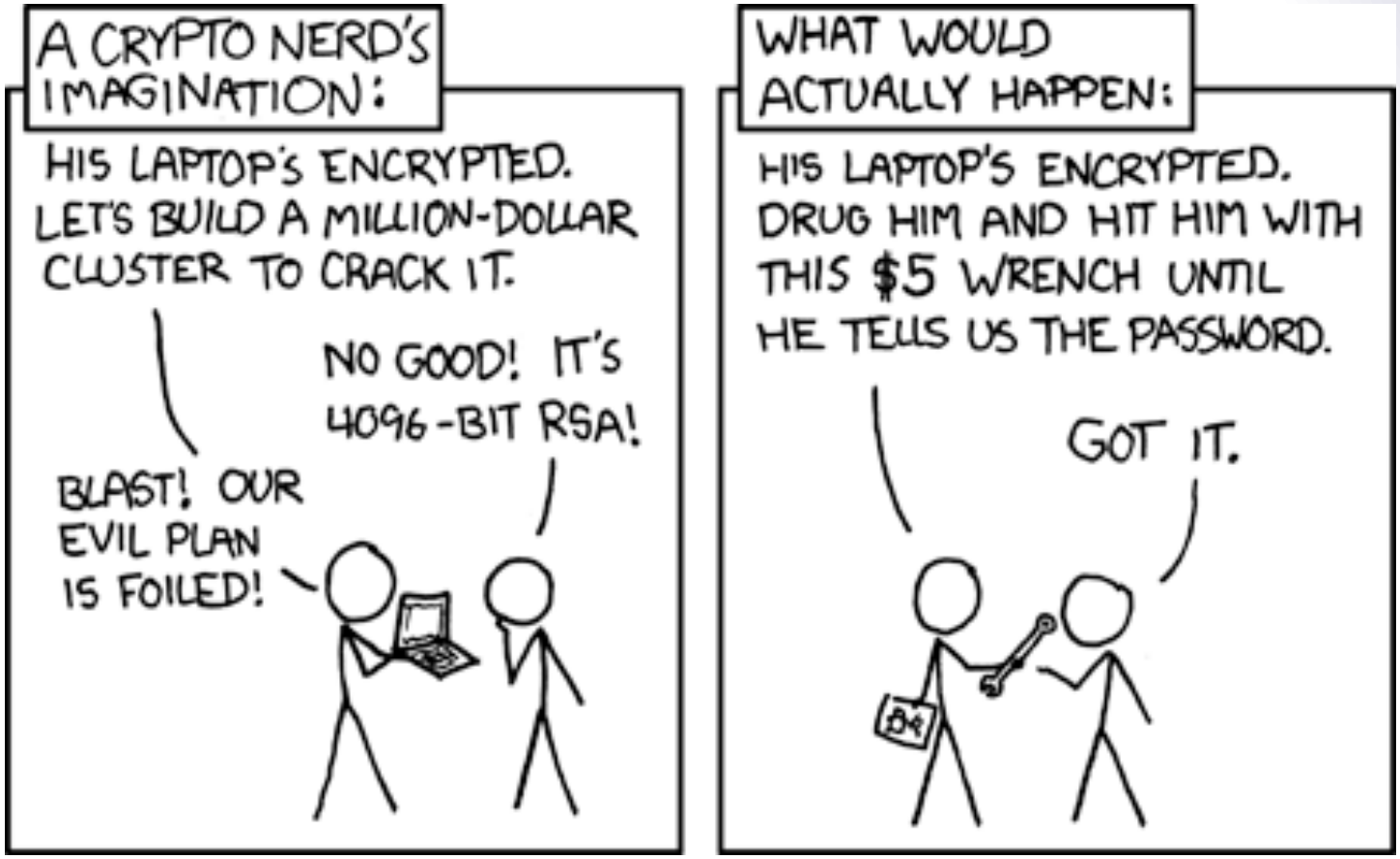
An Insecure MAC Construction

- 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 "extension" attack is possible
- An adversary can compute $t' = H(k \,||\, m \,||\, \text{padding} \,||\, m')$ for any 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