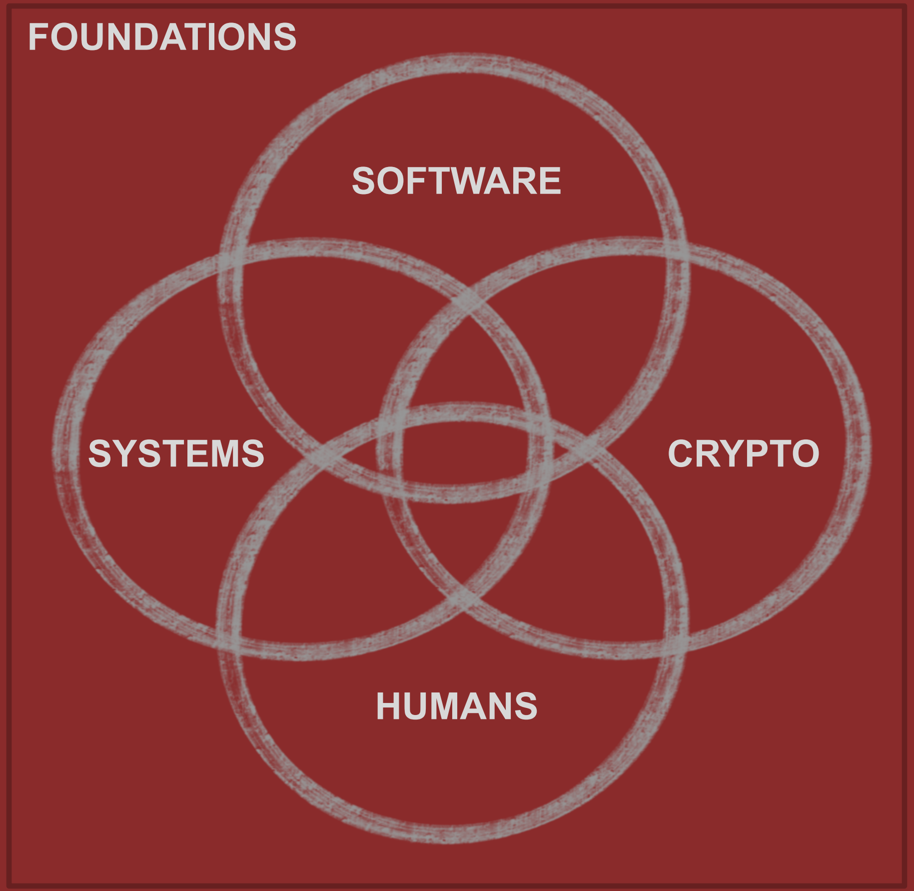
Διάλεξη #17 - Authenticated Encryption and Asymmetric Crypto

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στην Ασφάλεια

Θανάσης Αυγερινός



Huge thank you to David Brumley from Carnegie Mellon University for the guidance and content input while developing this class (lots of slides from Dan Boneh @ Stanford!)

Την προηγούμενη φορά

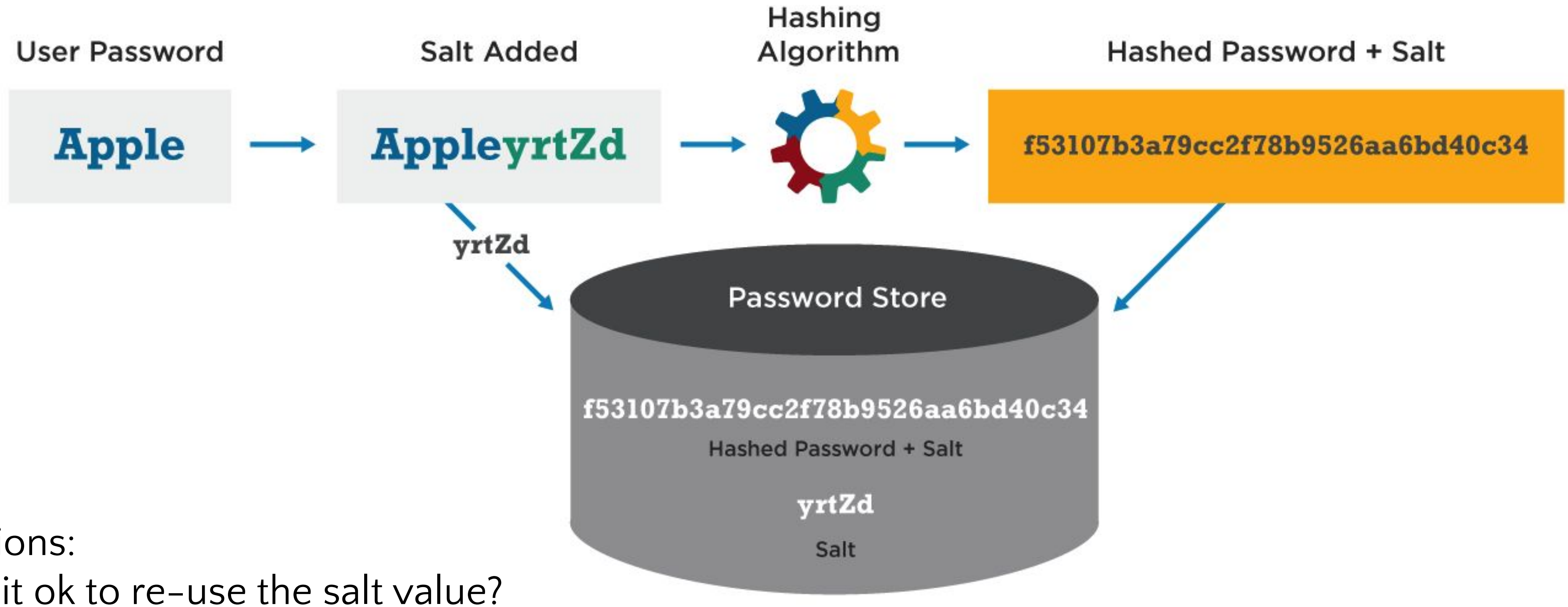
- Hashes Intro
- Hash Constructions
- HMAC
- Hash Tricks/Datastructures

Ανακοινώσεις / Διευκρινίσεις

- Πως λειτουργεί το password salt;

Password Salt

Password Hash Salting



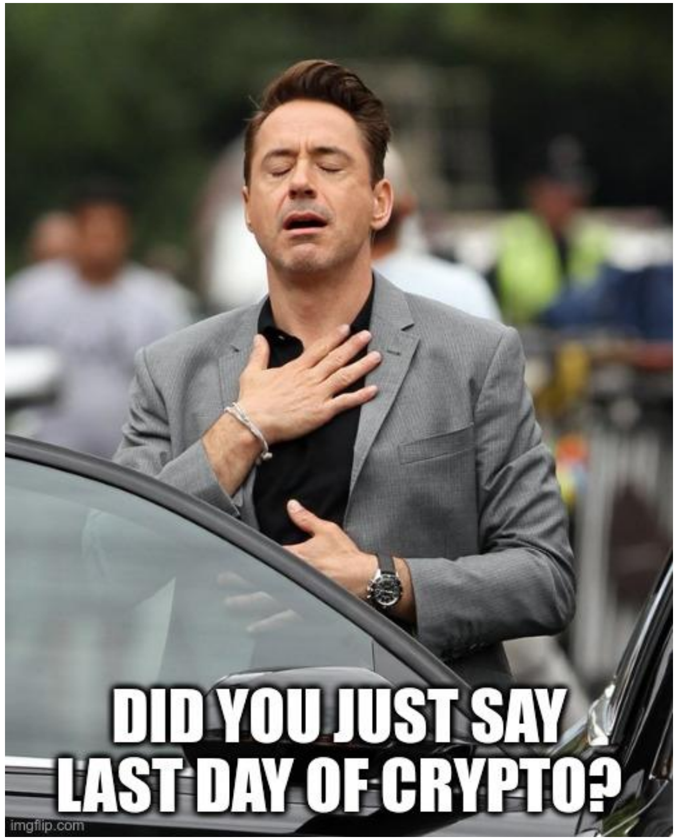
Questions:

1. Is it ok to re-use the salt value?
2. Is it ok to use a 3-bit salt value?

No!

Σήμερα

- Authenticated Encryption (AuthEnc)
- Asymmetric/Public Key Cryptography
  - Merkle's Puzzles
  - Diffie-Hellman
  - RSA



Hopefully!

Authenticated Encryption

Recap: the story so far

Confidentiality: semantic security against a CPA attack

- Encryption secure against **eavesdropping only**

Integrity:

- Existential unforgeability under a chosen message attack
- CBC-MAC, HMAC, *MAC

Can we combine them: encryption secure against **tampering**

- Ensuring both confidentiality and integrity

… but first, some history

Authenticated Encryption (AE): introduced in 2000 [KY'00, BN'00]

Crypto APIs before then: (e.g. MS-CAPI)

- Provide API for CPA-secure encryption (e.g. CBC with rand. IV)
- Provide API for MAC (e.g. HMAC)

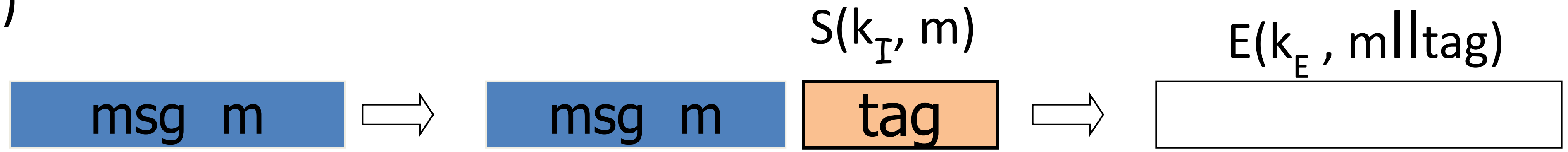
Every project had to combine the two itself without a well defined goal

- Not all combinations provide AE …

Combining MAC and ENC (CCA)

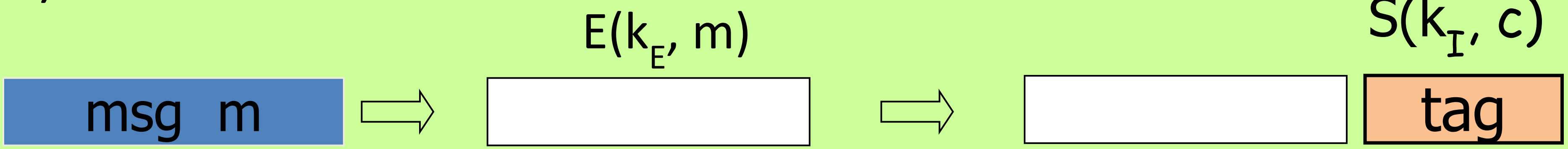
Encryption key k_E.   MAC key = k_I

Option 1: (SSL)

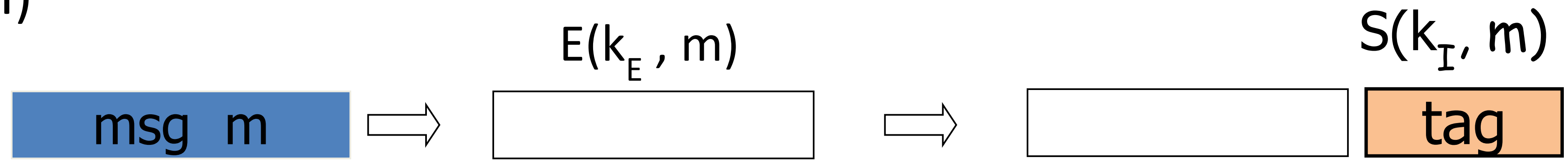


Option 2: (IPsec)

always correct



Option 3: (SSH)



A.E. Theorems

Let (E,D) be CPA secure cipher and (S,V) secure MAC. Then:

1. **Encrypt-then-MAC**: always provides A.E.

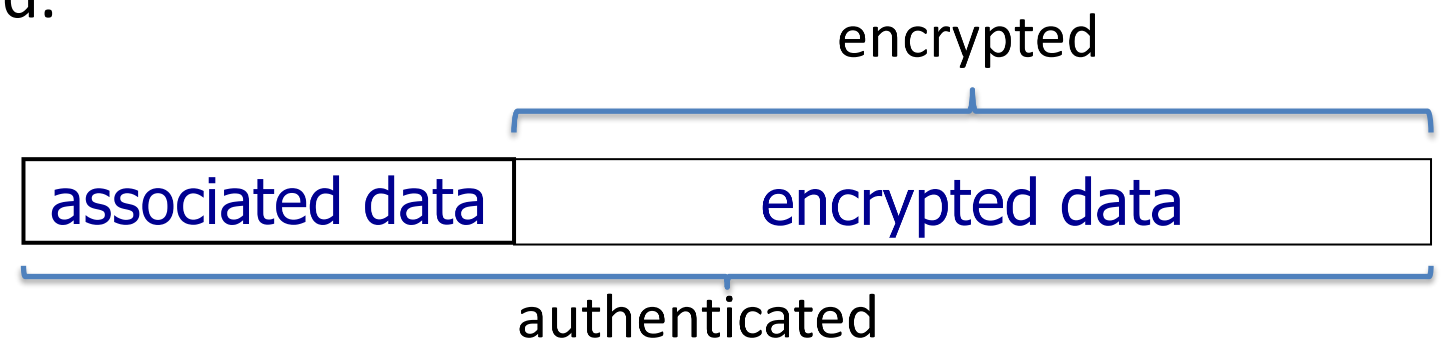
1. **MAC-then-encrypt**: may be insecure against CCA attacks

   however: when (E,D) is rand-CTR mode or rand-CBC
   M-then-E provides A.E.

Standards (at a high level)

- **GCM**: CTR mode encryption then CW-MAC
  (accelerated via Intel's PCLMULQDQ instruction)
- **CCM**: CBC-MAC then CTR mode encryption (802.11i)
- **EAX**: CTR mode encryption then CMAC

All support AEAD: (auth. enc. with associated data). All are nonce-based.

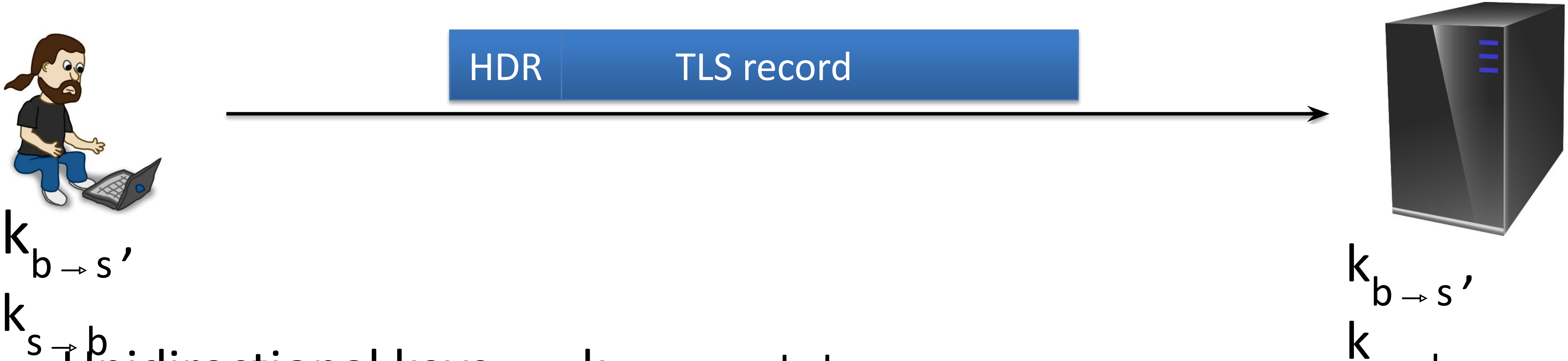


An example API (OpenSSL)

int **AES_GCM_Init**(AES_GCM_CTX *ain,
    unsigned char ***nonce**,   unsigned long noncelen,
    unsigned char ***key**,   unsigned int klen)

int **AES_GCM_EncryptUpdate**(AES_GCM_CTX *a,
    unsigned char ***aad**,   unsigned long aadlen,
    unsigned char ***data**,   unsigned long datalen,
    unsigned char ***out**,   unsigned long *outlen)

The TLS Record Protocol (TLS 1.2)



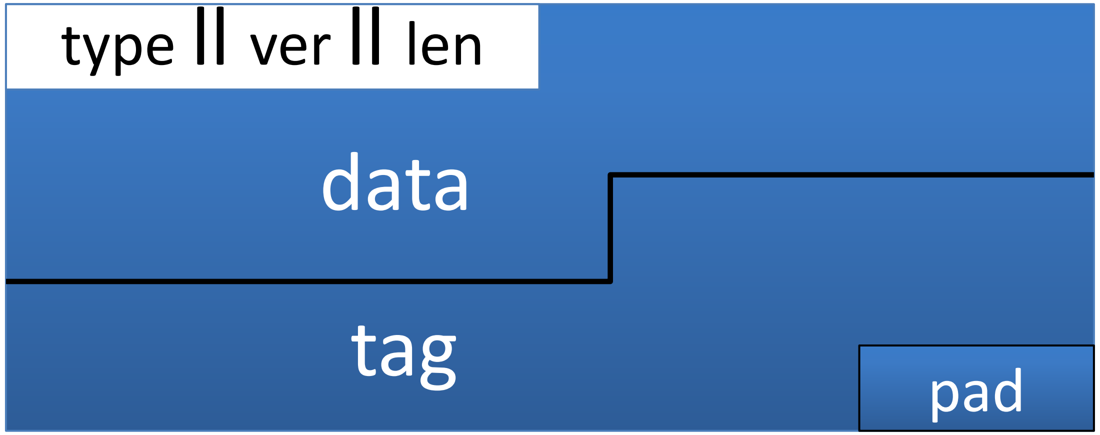
Unidirectional keys: $k_{b \to s}$ and $k_{s \to b}$

Stateful encryption:

- Each side maintains two 64-bit counters: $ctr_{b \to s}$, $ctr_{s \to b}$
- Init. to 0 when session started. ctr++ for every record.
- Purpose: replay defense

TLS record: encryption (CBC AES-128, HMAC-SHA1)

$k_{b \to s} = (k_{mac}, k_{enc})$



Browser side **enc($k_{b \to s}$, data, $ctr_{b \to s}$) :**

step 1: tag ← S(k_{mac} , [$++ctr_{b \to s}$ ll header ll data])

step 2: pad [header ll data ll tag] to AES block size

step 3: CBC encrypt with k_{enc} and new random IV

step 4: prepend header

TLS record: decryption (CBC AES-128, HMAC-SHA1)

Server side **dec($k_{b\to s}$, record, ctr$_{b\to s}$)** :

step 1: CBC decrypt record using k_{enc}

step 2: check pad format: send bad_record_mac if invalid

step 3: check tag on [++$ctr_{b\to s}$ ll header ll data]

send bad_record_mac if invalid

Provides authenticated encryption

(provided no other info. is leaked during decryption)

Bugs in older versions (prior to TLS 1.1)

IV for CBC is predictable: (chained IV)

IV for next record is last ciphertext block of current record.

Not CPA secure. (a practical exploit: BEAST attack)

Padding oracle: during decryption

if pad is invalid send decryption failed alert

if mac is invalid send bad_record_mac alert

⇒ attacker learns info. about plaintext (various attacks possible)

Lesson: when decryption fails, do not explain why

Leaking the length

The TLS header leaks the length of TLS records

- Lengths can also be inferred by observing network traffic

For many web applications, leaking lengths reveals sensitive info:

- In tax preparation sites, lengths indicate the type of return being filed which leaks information about the user's income
- In healthcare sites, lengths leaks what page the user is viewing
- In Google maps, lengths leaks the location being requested

No easy solution

Asymmetric / Public Key Cryptography

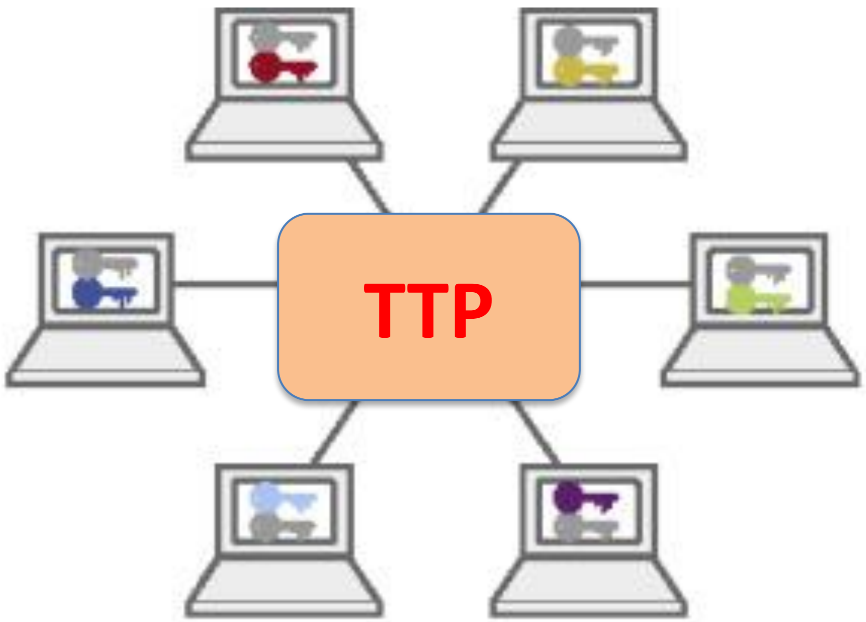
Key management

Problem: n users. Storing mutual secret keys is difficult

Total: O(n) keys per user

A better solution

Online Trusted 3rd Party (TTP)



Generating keys: a toy protocol

Alice wants a shared key with Bob. Eavesdropping security only.

Bob (k_B) **Alice** (k_A) **TTP**

"Alice wants key with Bob" (Alice → TTP)

choose random k_{AB}

$E(K_A, \text{"A, B"} \,||\, K_{AB})$ (TTP → Alice)

$\text{ticket} = E(K_B, \text{"A, B"} \,||\, K_{AB})$

ticket (Alice → Bob)

k_{AB} k_{AB}

(E,D) a CPA-secure cipher

Generating keys: a toy protocol

Alice wants a shared key with Bob. Eavesdropping security only.

Eavesdropper sees: $E(k_A, \text{"A, B"} \parallel k_{AB})$; $E(k_B, \text{"A, B"} \parallel k_{AB})$

(E,D) is CPA-secure $\Rightarrow$
eavesdropper learns nothing about k_{AB}

Note: TTP needed for every key exchange, knows all session keys.

(basis of Kerberos system)

Toy protocol: insecure against active attacks

Example: insecure against replay attacks

Attacker records session between Alice and merchant Bob

- For example a book order

Attacker replays session to Bob

- Bob thinks Alice is ordering another copy of book

Key question

Can we generate shared keys without an **online** trusted 3rd party?

Answer: yes!

Starting point of public-key cryptography:

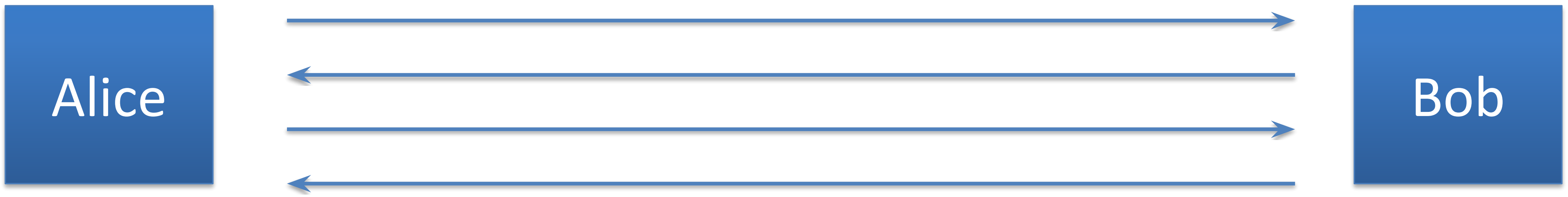
- Merkle (1974), Diffie-Hellman (1976), RSA (1977)
- More recently: ID-based enc. (BF 2001), Functional enc. (BSW 2011)

Merkle Puzzles

Key exchange without an online TTP?

Goal: Alice and Bob want shared key, unknown to eavesdropper

- For now: security against eavesdropping only (no tampering)



eavesdropper ??

Can this be done using generic symmetric crypto?

Merkle Puzzles (1974)

Answer: yes, but very inefficient

<u>Main tool</u>: puzzles

- Problems that can be solved with some effort
- Example: E(k,m) a symmetric cipher with $k \in \{0,1\}^{128}$
  - **puzzle(P) = E(P, "message")** where $P = 0^{96} \,||\, b_1 \ldots b_{32}$
  - Goal: find P by trying all 2^{32} possibilities

Merkle puzzles

Alice: prepare 2^{32} puzzles

- For $i=1, \ldots, 2^{32}$ choose random $\mathbf{P_i} \in \{0,1\}^{32}$ and $\mathbf{x_i}, \mathbf{k_i} \in \{0,1\}^{128}$

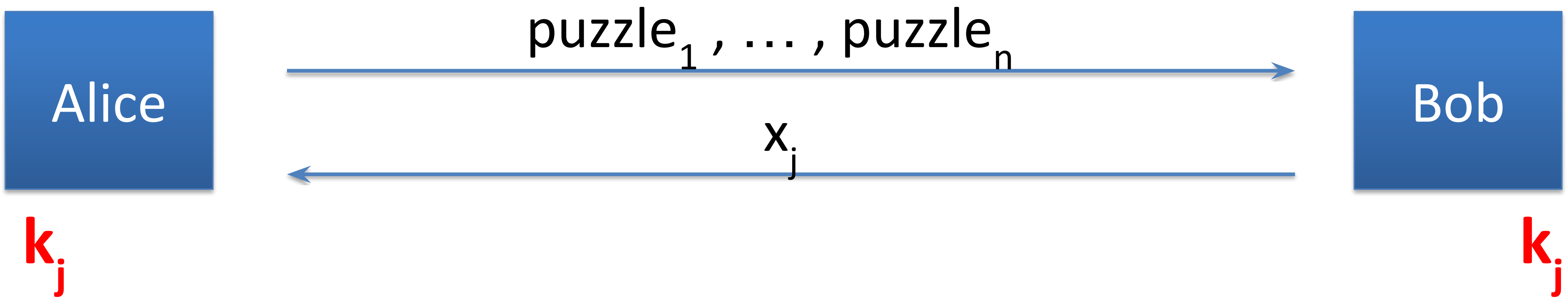
  set $puzzle_i \leftarrow E(\, 0^{96} \,\|\, \mathbf{P_i}\, , \text{ "Puzzle \# } \mathbf{x_i}\text{"} \,\|\, \mathbf{k_i} \,)$
- Send $puzzle_1, \ldots, puzzle_{2^{\wedge}32}$ to Bob

Bob: choose a random $puzzle_j$ and solve it. Obtain (x_j, k_j).

- Send x_j to Alice

Alice: lookup puzzle with number x_j. Use k_j as shared secret

In a figure



Alice’s work: O(n) (prepare n puzzles)

Bob’s work: O(n) (solve one puzzle)

Eavesdropper’s work: (e.g. 2^{64} time)

Impossibility Result

Can we achieve a better gap using a general symmetric cipher?

Answer: unknown

But: roughly speaking,

quadratic gap is best possible if we treat cipher as

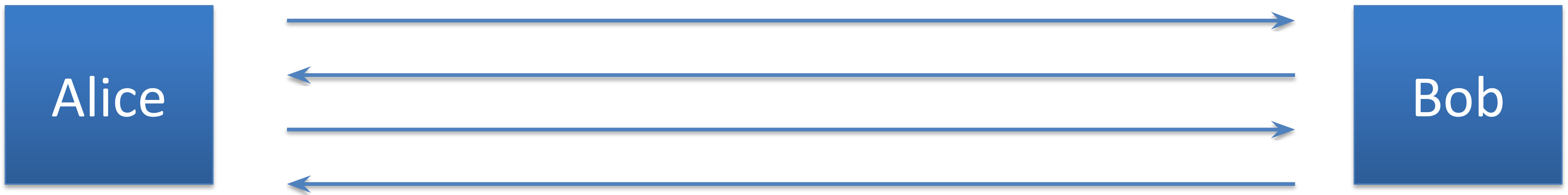
a black box oracle [IR'89, BM'09]

The
Diffie-Hellman (DH)
Protocol

Key exchange without an online TTP?

Goal: Alice and Bob want shared secret, unknown to eavesdropper

- For now: security against eavesdropping only (no tampering)



eavesdropper ??

Can this be done with an exponential gap?

The Diffie-Hellman protocol (informally)

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

Fix an integer g in {1, …, p}

<u>Alice</u> **<u>Bob</u>**

choose random **a** in {1,…,p-1} choose random **b** in {1,…,p-1}

$A = g^a \pmod p$ →

← $B = g^b \pmod p$

$\mathbf{B^a} \pmod p = (g^b)^a = \mathbf{k_{AB} = g^{ab}} \pmod p$

Security (much more on this later)

Eavesdropper sees: p, g, $A=g^a \pmod p$, and $B=g^b \pmod p$

Can she compute $g^{ab} \pmod p$??

More generally: define $DH_g(g^a, g^b) = g^{ab} \pmod p$

How hard is the DH function mod p?

How hard is the DH function mod p?

Suppose prime p is n bits long.

Best known algorithm (GNFS): run time exp($\tilde{O}(\sqrt[3]{n})$)

cipher key size	modulus size	Elliptic Curve size
80 bits	1024 bits	160 bits
128 bits	3072 bits	256 bits
256 bits (AES)	**15360** bits	512 bits

As a result: slow transition away from (mod p) to elliptic curves

Elliptic curve
Diffie-Hellman

Insecure against man-in-the-middle

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

Alice | **MiTM** | **Bob**

Alice → MiTM: $A = g^{a} \pmod{p}$

MiTM → Bob: $A' = g^{a'} \pmod{p}$

Bob → MiTM: $B = g^{b} \pmod{p}$

MiTM → Alice: $B' = g^{b'} \pmod{p}$

Alice: $\text{Key} = g^{ab'} \pmod{p}$

Bob: $\text{Key} = g^{a'b} \pmod{p}$

Public Key Cryptography

Establishing a shared secret

Goal: Alice and Bob want shared secret, unknown to eavesdropper

- For now: security against eavesdropping only (no tampering)

This segment: a different approach

Public key encryption

Def: a public-key encryption system is a triple of algs. (G, E, D)

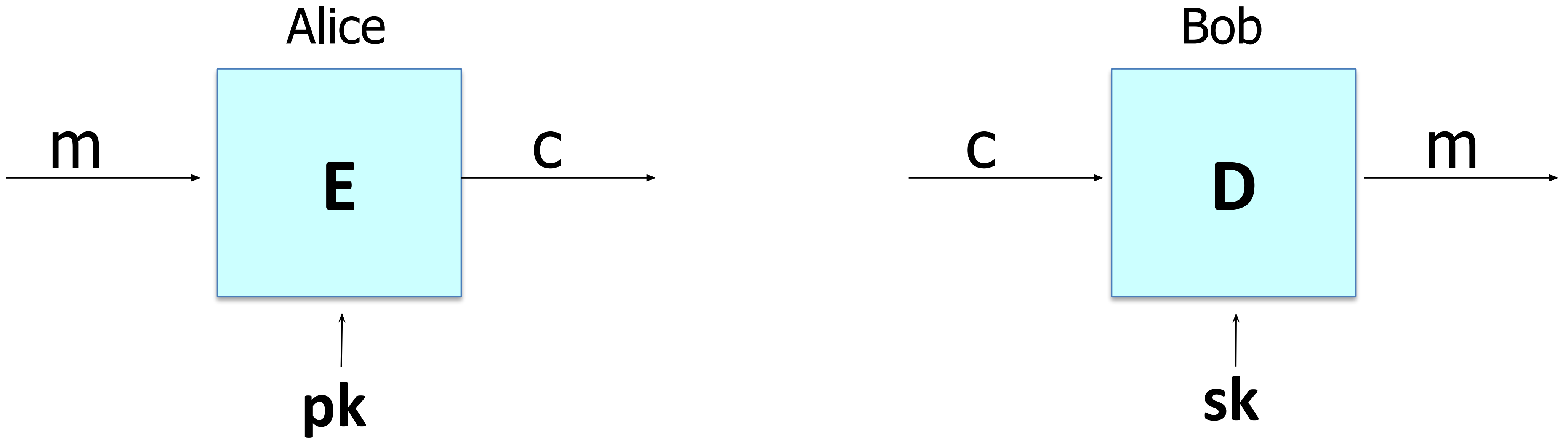
- G(): randomized alg. outputs a key pair (pk, sk)
- E(pk, m): randomized alg. that takes $m \in M$ and outputs $c \in C$
- D(sk,c): det. alg. that takes $c \in C$ and outputs $m \in M$ or $\perp$

Consistency: $\forall$(pk, sk) output by G :

$$\forall m \in M: \quad D(sk, E(pk, m)) = m$$

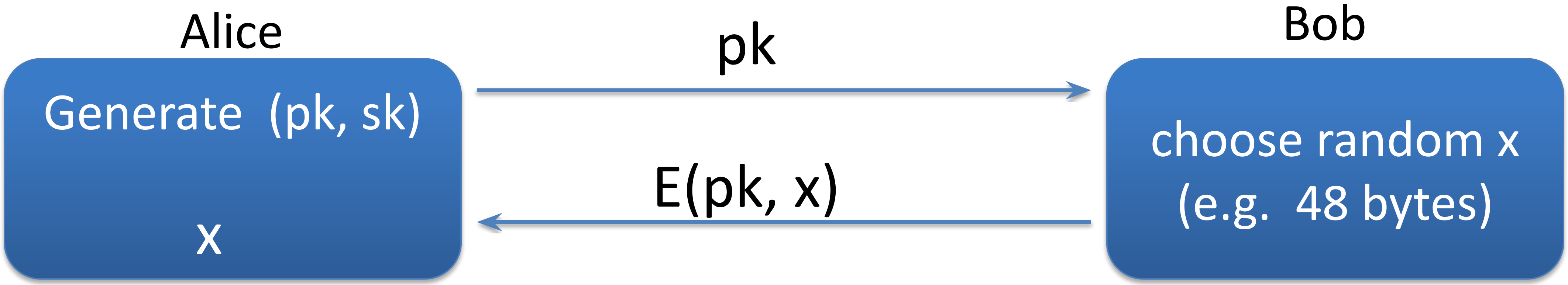
Public key encryption

Bob: generates (PK, SK) and gives PK to Alice



Applications

Session setup (for now, only eavesdropping security)



Non-interactive applications: (e.g. Email)

- Bob sends email to Alice encrypted using pk_{alice}
- Note: Bob needs pk_{alice} (public key management)

Trapdoor functions (TDF)

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

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

More precisely: $\forall (pk, sk)$ output by G

$$\forall x \in X: \quad F^{-1}(sk, F(pk, x)) = x$$

The RSA trapdoor permutation

First published: Scientific American, Aug. 1977.

Very widely used:

– SSL/TLS: certificates and key-exchange

– Secure e-mail and file systems

… many others

The RSA trapdoor permutation

G(): choose random primes p,q ≈1024 bits. Set **N=pq**.

choose integers **e , d** s.t. **$e \cdot d = 1 \pmod{\phi(N)}$ where $\phi(N) = (p-1)(q-1)$**

output pk = (N, e) , sk = (N, d)

F(pk, x): $\mathbb{Z}_N^* \to \mathbb{Z}_N^*$; **RSA(x) = x^e** (in Z_N)

F^{-1}(sk, y) = y^d ; y^d = **$RSA(x)^d$** = x^{ed} = $x^{k\phi(N)+1}$ = $\left(x^{\phi(N)}\right)^k \cdot$ **x** = x

The RSA Assumption

RSA assumption: RSA is one-way permutation

For all efficient algs. A:

$$\Pr\left[\; A(N,e,y) = y^{1/e} \;\right] < \text{negligible}$$

where $p,q \xleftarrow{R}$ n-bit primes, $N \leftarrow pq$, $y \xleftarrow{R} Z_N^*$

Review: RSA pub-key encryption (ISO std)

(E_s, D_s): symmetric enc. scheme providing auth. encryption.

H: $Z_N \rightarrow K$ where K is key space of (E_s,D_s)

- **G**(): generate RSA params: pk = (N,e), sk = (N,d)
- **E**(pk, m): (1) choose random x in Z_N

  (2) $y \leftarrow RSA(x) = x^e$, $k \leftarrow H(x)$

  (3) output $(y, E_s(k,m))$
- **D**(sk, (y, c)): output $D_s(H(RSA^{-1}(y)), c)$

Textbook RSA is insecure

Textbook RSA encryption:

- public key: **(N,e)** Encrypt: $\mathbf{c} \leftarrow \mathbf{m}^{\mathbf{e}}$ (in Z_N)
- secret key: **(N,d)** Decrypt: $\mathbf{c}^{\mathbf{d}} \rightarrow \mathbf{m}$

Insecure cryptosystem !!

- Is not semantically secure and many attacks exist

⇒ The RSA trapdoor permutation is not an encryption scheme !

Ευχαριστώ και καλή μέρα εύχομαι!

Keep hacking!