Cryptography

Cryptography and computer security

- Cryptography is not the same as security.
- Cryptography is seldom the weakest link or the heart of the matter in security.
  - ***Cryptography is not broken, it is circumvented.*** [attributed to A. Shamir]
  - ***If you think that cryptography is the answer to your problem then you don't understand cryptography and you don't understand your problem.*** [attributed to R. Needham]

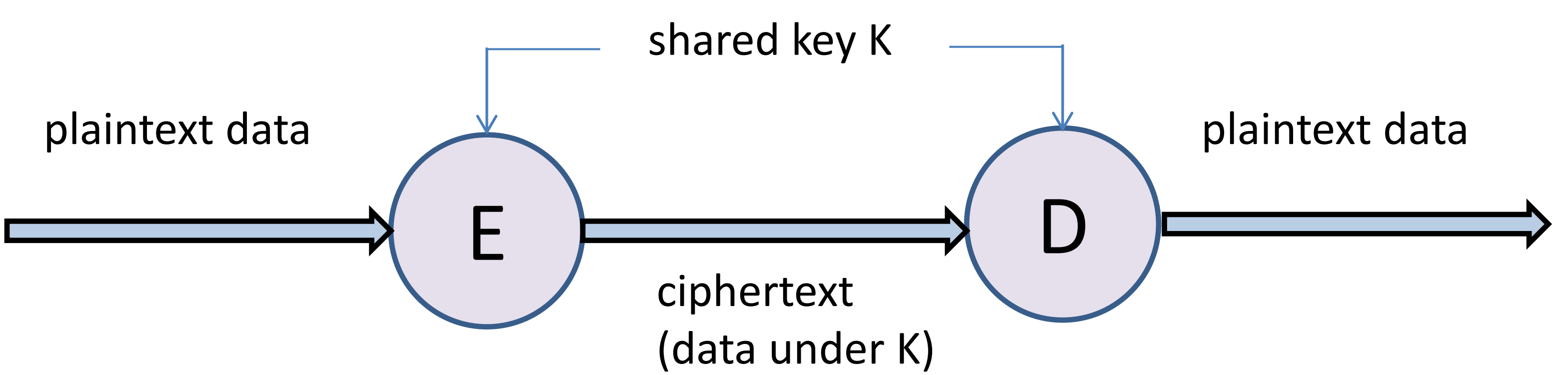
Cryptography and computer security (cont.)

- The applications of cryptography in security are broad and significant.
- They have shaped both fields.
  - Cryptographic constructions are informed by those applications.
  - Many computer systems include special support for cryptography.

Shared-key encryption
(a.k.a. symmetric encryption)

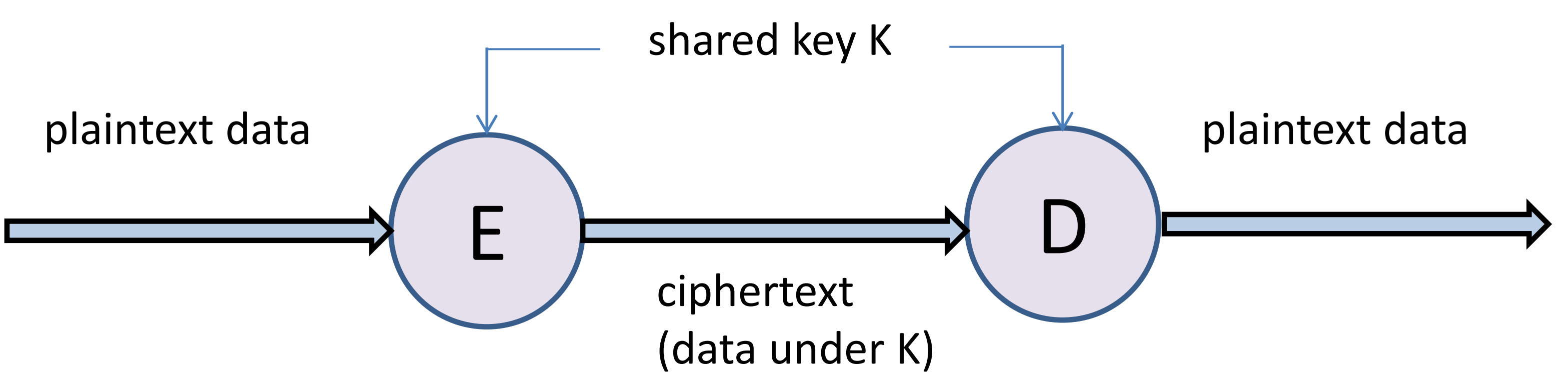
Shared-key encryption

- E and D are algorithms that use a same key K.
  - We write E_K and D_K for the algorithms for a given value of K.



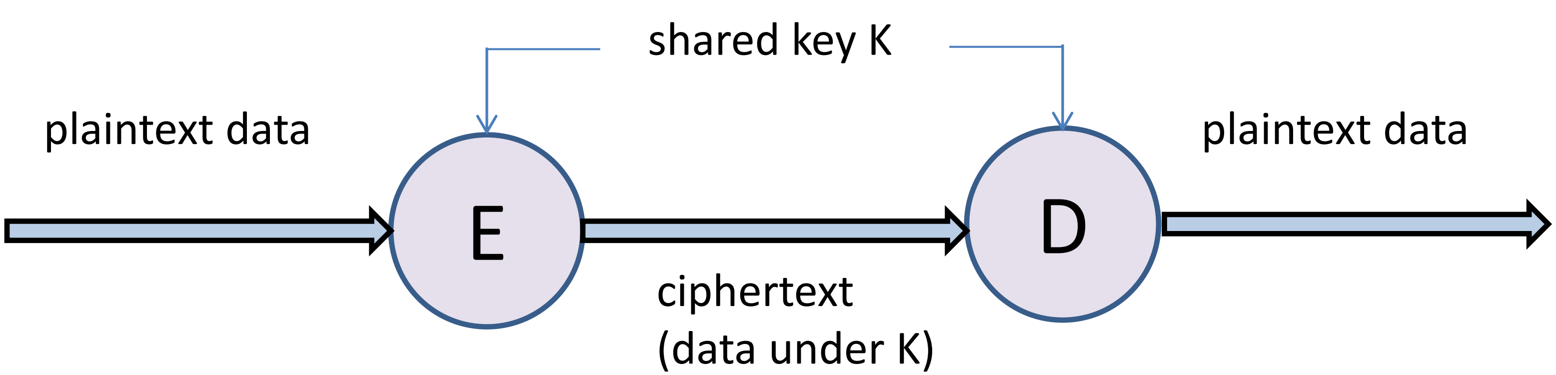
Shared-key encryption

- E and D are algorithms that use a same key K.
  - We write E_K and D_K for the algorithms for a given value of K.
- The main goal is that $E_K(M)$ should conceal M.



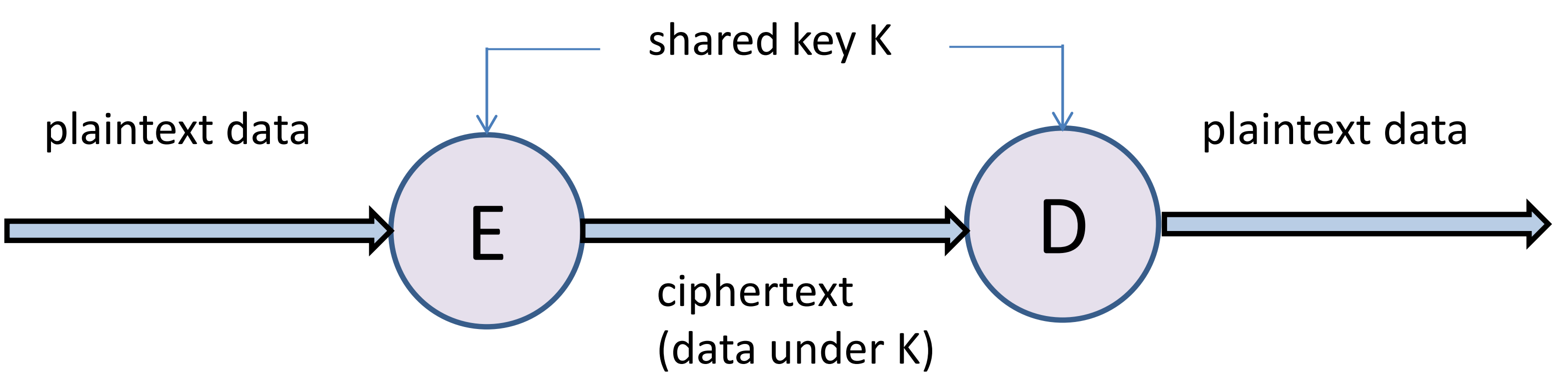
Shared-key encryption

- E and D are algorithms that use a same key K.
  - We write E_K and D_K for the algorithms for a given value of K.
- The main goal is that $E_K(M)$ should conceal M.
- E and D may be public.
- K should be secret.



Shared-key encryption

- E and D are algorithms that use a same key K.
  - We write E_K and D_K for the algorithms for a given value of K.
- The main goal is that $E_K(M)$ should conceal M.
- E and D may be public (*Kerckhoff's principle*).
- K should be secret.



JOURNAL

DES

SCIENCES MILITAIRES.

Janvier 1883.

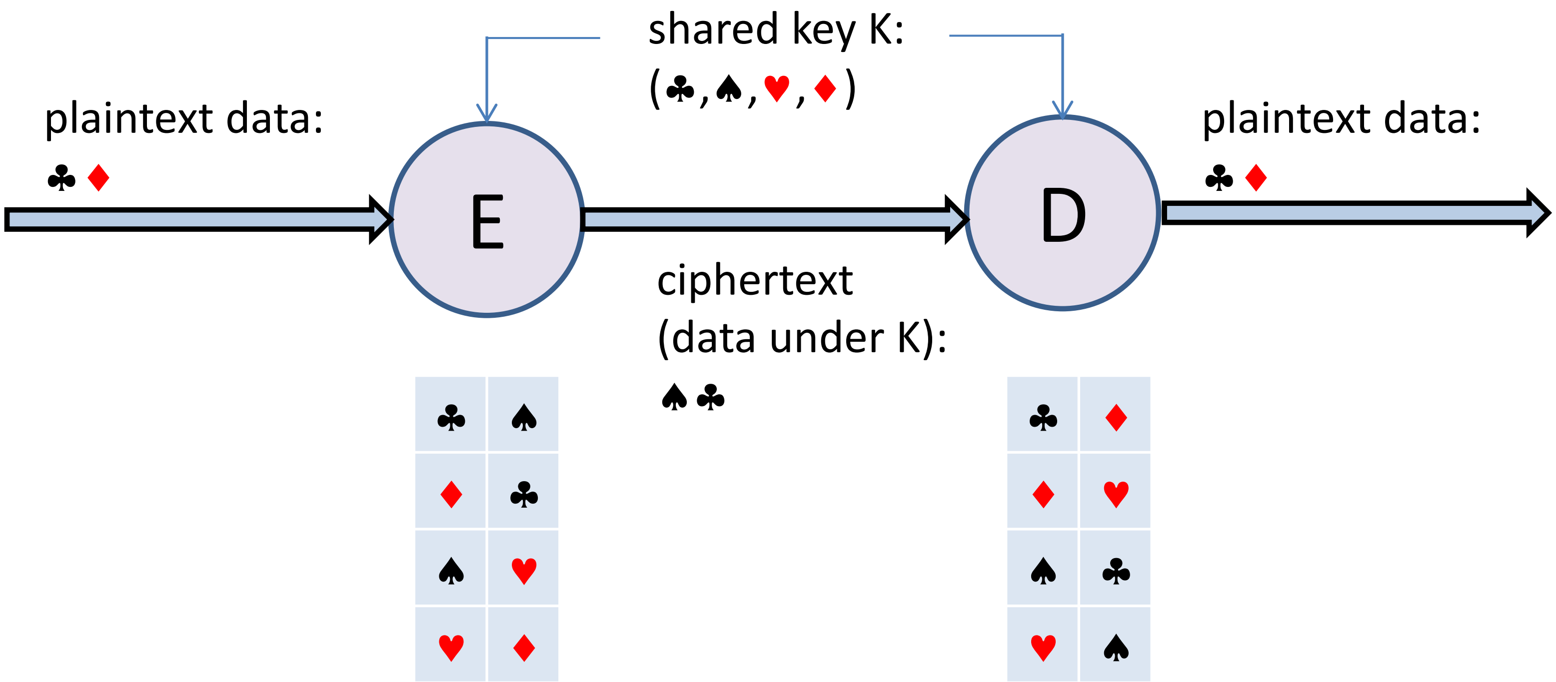
LA CRYPTOGRAPHIE MILITAIRE.

« La cryptographie est un auxiliaire puissant de la tactique militaire. »
(Général LEWAL, *Études de guerre.*)

Source: www.petitcolas.net/fabien/kerckhoffs/

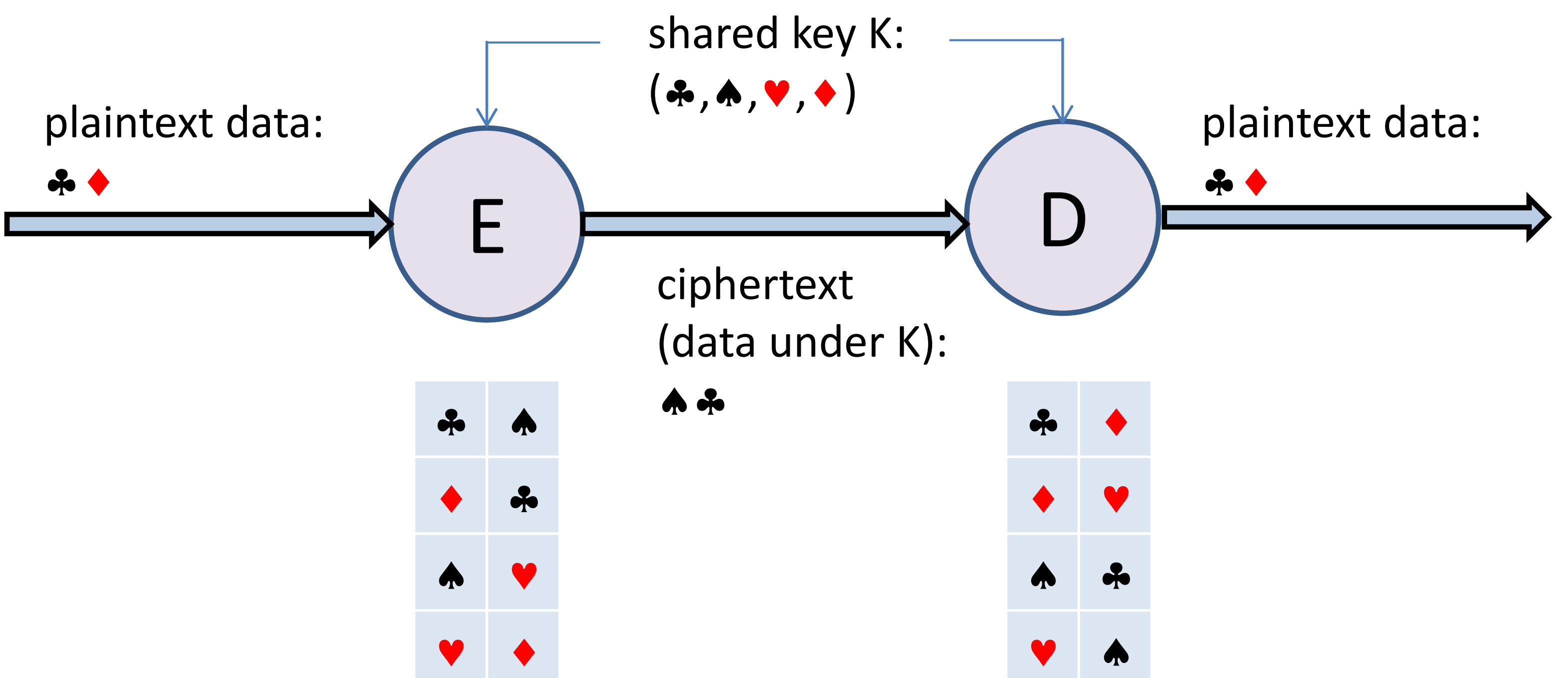
Shared-key encryption methods

- Substitution ciphers:



Shared-key encryption methods

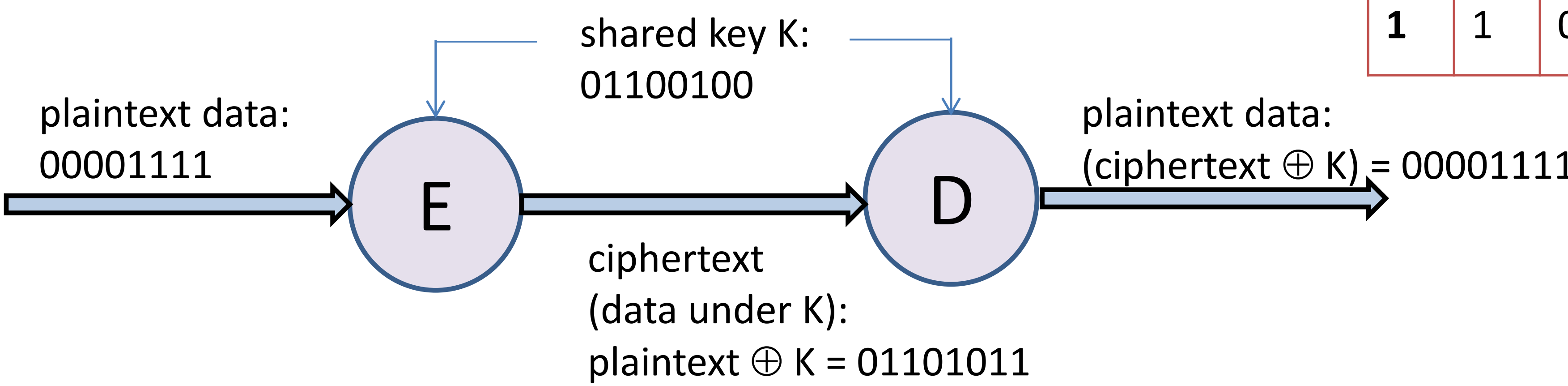
- Substitution ciphers:
  - easy to understand and to run,
  - also easy to break.



Shared-key encryption methods

- XOR (⊕) with one-time pads:

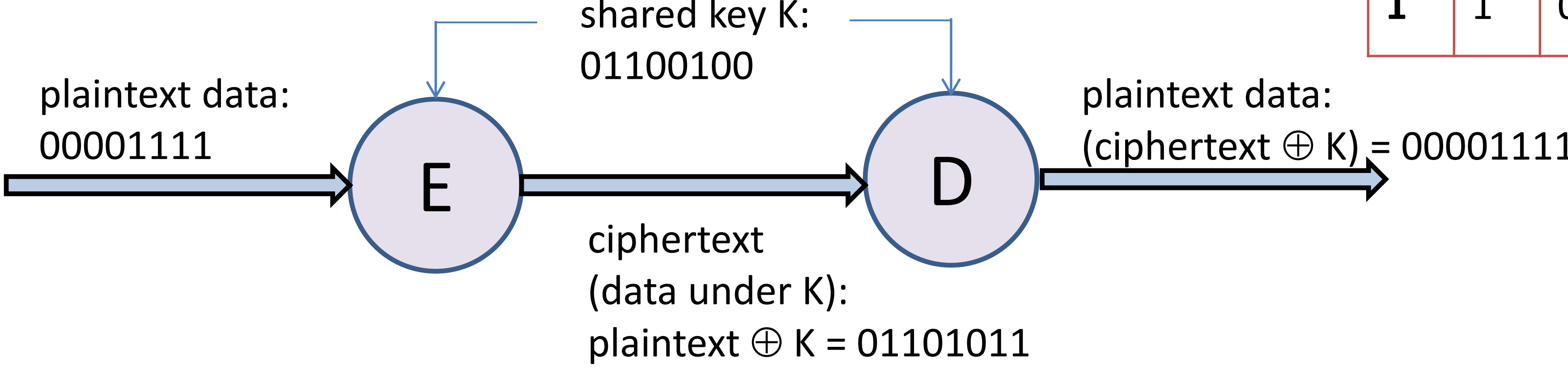
⊕	**0**	**1**
0	0	1
1	1	0



Shared-key encryption methods

- XOR (⊕) with one-time pads:
  - easy to understand, just a little harder to run,
  - hard to deploy: each key K can be used only once
    (for otherwise an attacker can get the XOR of two plaintexts M and N from their ciphertexts: $(M \oplus K) \oplus (N \oplus K) = (M \oplus N)$),
  - impossible to break if K is truly random.

⊕	**0**	**1**
0	0	1
1	1	0



Shared-key encryption methods

- Modern methods:
  - easier to deploy,
  - hard to break (we believe),
  - harder to understand and moderately hard to run (computers are needed),
  - still often based on fast low-level operations (e.g., XORs, shifts):
    a few thousand operations are typically needed for the smallest messages ($\Rightarrow$ ~ microseconds).

Concerns (summary)

- Security
- Key distribution
- Execution complexity

Some themes (summary)

1. Attackers with certain capabilities and information (e.g., some ciphertexts)
2. One-way computation (e.g., encryption)
3. Randomness (e.g., of keys)

1. Types of attacks

- Ciphertext only
- Known plaintext
- Chosen plaintext
- Chosen ciphertext

Some practical chosen-ciphertext attacks [Bleichenbacher, Vaudenay, and others]

Encryption without authentication is often useless and even risky:

Ciphertext

Format ok / not ok

… … …

(some number of attempts, e.g., just 256)

Ciphertext

Format ok / not ok

Server

(knows the key)

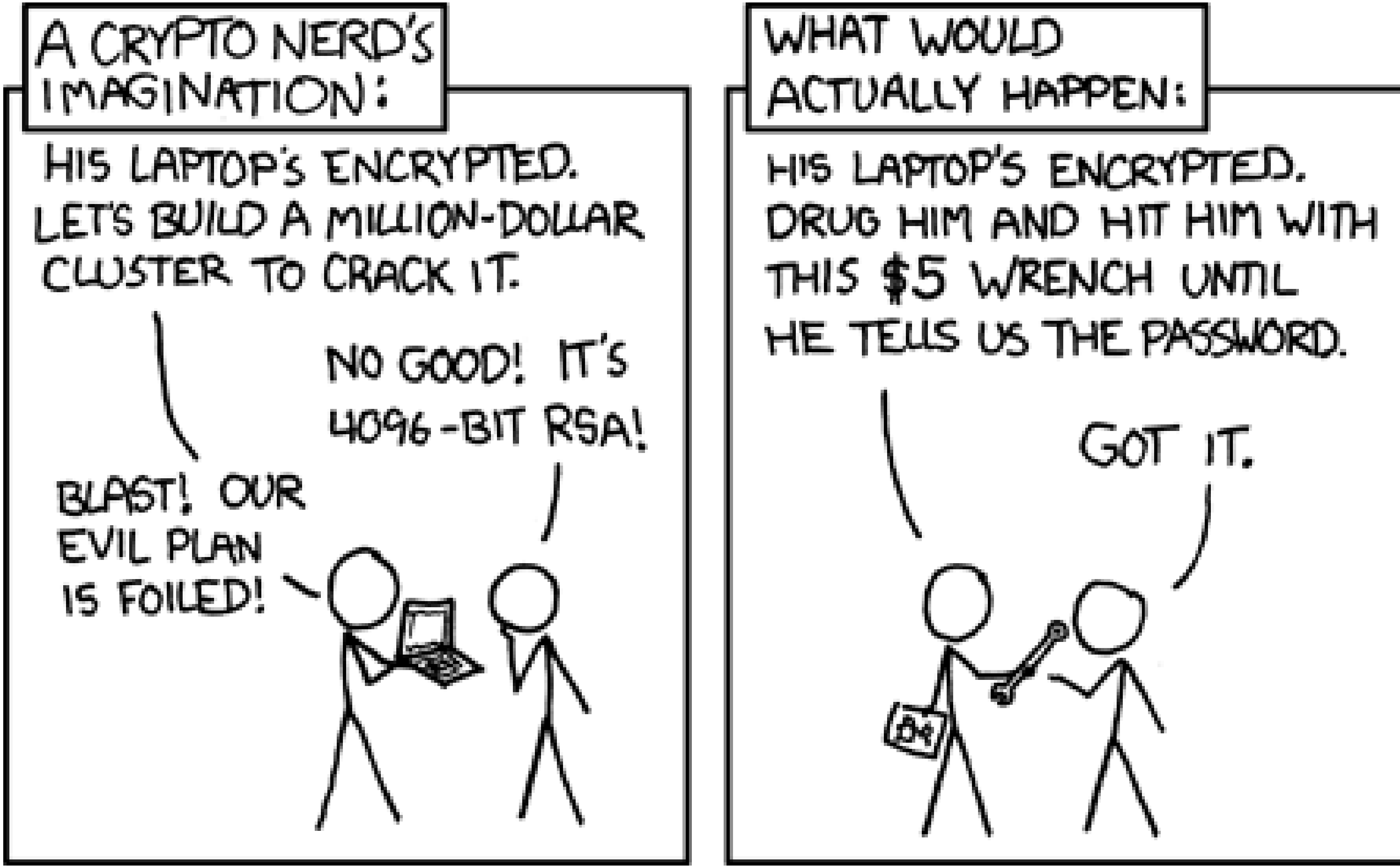
Eventually the attacker can deduce the key!

Recent attacks exploit “oracles” for the correctness of padding [Duong & Rizzo].

1. Types of attacks (cont.)

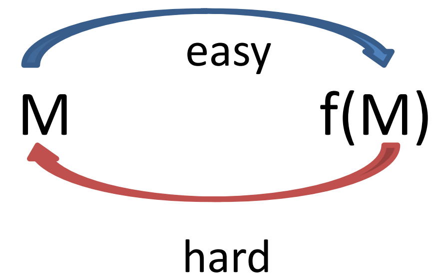
- Ciphertext only
- Known plaintext
- Chosen plaintext
- Chosen ciphertext

- Obtaining key material somehow, e.g., via
  – a software flaw (e.g., buffer overflow),
  – side-channels (e.g., power analysis),
  – social engineering or “rubber-hose cryptanalysis”.



Source: xkcd.com

2. One-way functions



f is a one-way function if:

- given M, it is easy to compute f(M);
- for most M, given f(M) it is hard to find M or any M’ such that f(M) = f(M’).

Examples:

- Multiplication is (believed to be) a one-way function on sufficiently large prime numbers.
- If E_K is a good encryption function and K is secret, then E_K must be one-way.

3. Randomness

- Good (pseudo)random numbers are crucial.
  - With them, we have at least the one-time pad.
  - Without, keys are bad, algorithms are worthless.

3. Randomness

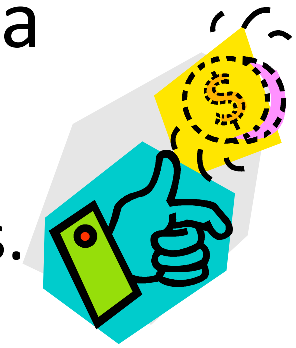
- Good (pseudo)random numbers are crucial.
  - With them, we have at least the one-time pad.
  - Without, keys are bad, algorithms are worthless.
- Some sources rely on physical phenomena (noisy diodes, air turbulence on disks).
  - Such sources may be slow and yield patterns.

  $\Rightarrow$ *Spread and stretch the randomness.*

3. Randomness

- Good (pseudo)random numbers are crucial.
  - With them, we have at least the one-time pad.
  - Without, keys are bad, algorithms are worthless.
- Some sources rely on physical phenomena (noisy diodes, air turbulence on disks).
  - Such sources may be slow and yield patterns.

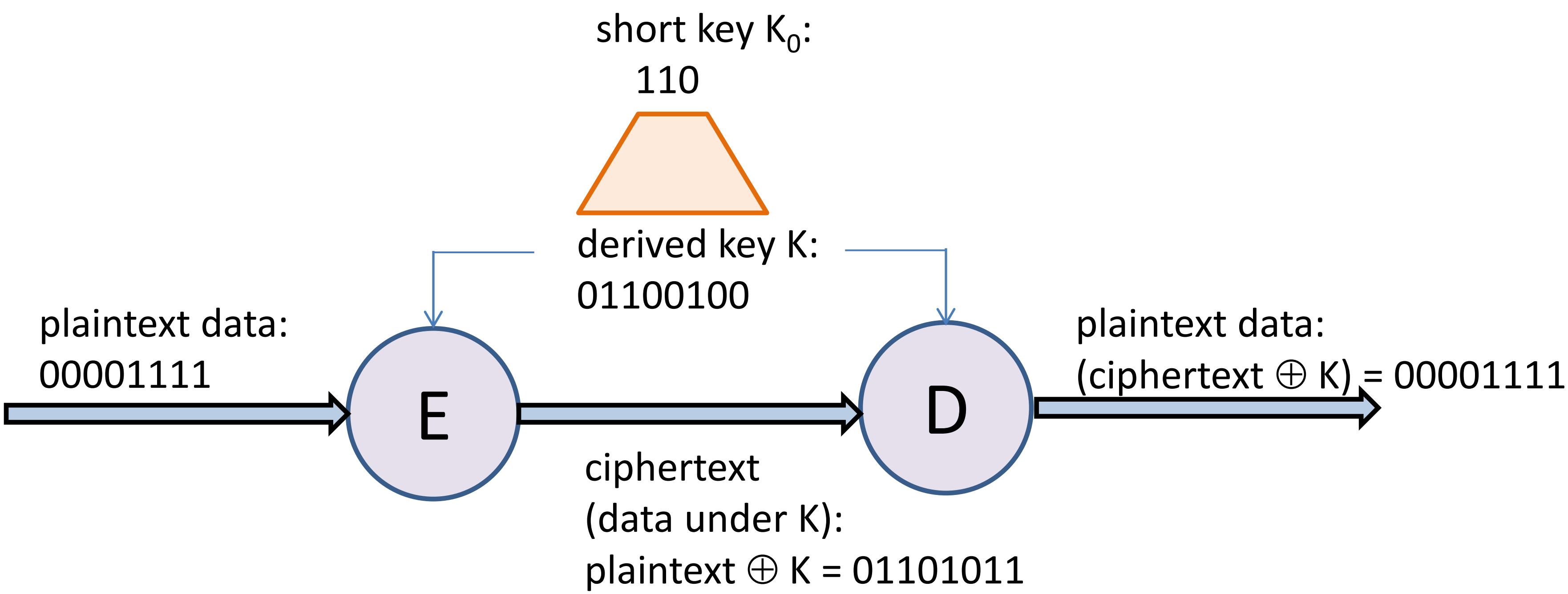
  $\Rightarrow$ *Spread and stretch the randomness.*



Theorem [Håstad et al.]: Pseudorandom generators can be constructed from one-way functions. (The converse is true too, and easier.)

Approximating the one-time pad: stream ciphers (e.g., RC4, SEAL)

- Start with a fixed-size key K_0 (maybe random).
- Stretch it into a key K as long as the plaintext.
- Then XOR.



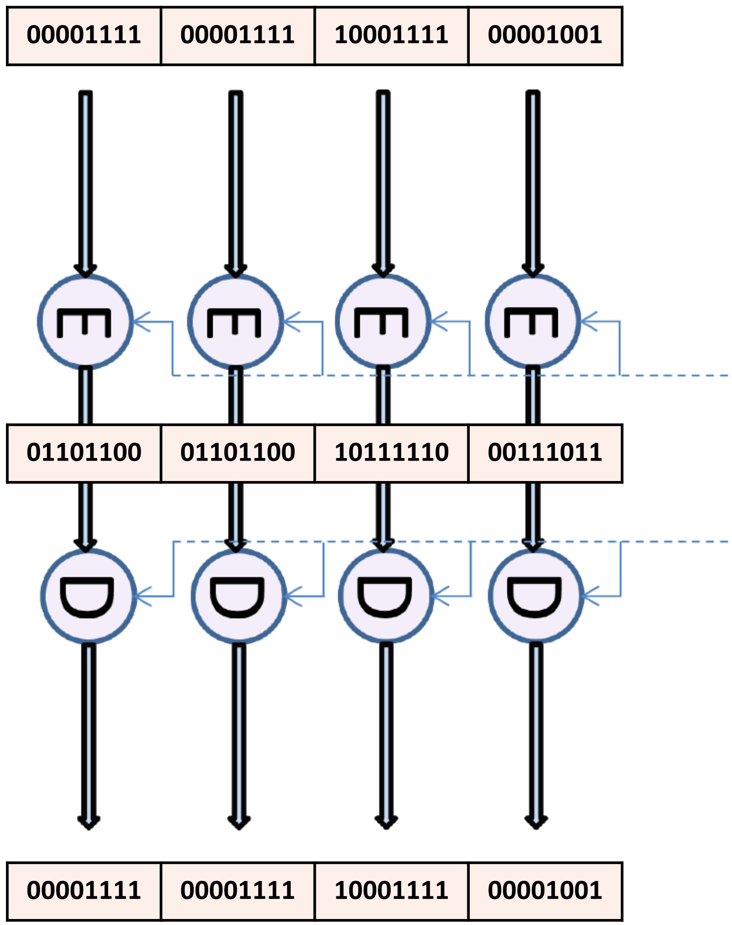
Another approach: block ciphers (e.g., AES)

- Block ciphers apply keys of fixed length to plaintext blocks of fixed length.
- They are extended to longer message by various *modes of operation*.
  - ECB (electronic code book): long plaintexts are encrypted block by block, each independently.
  - CBC (cipher block chaining): encryptions are chained.

  …

ECB

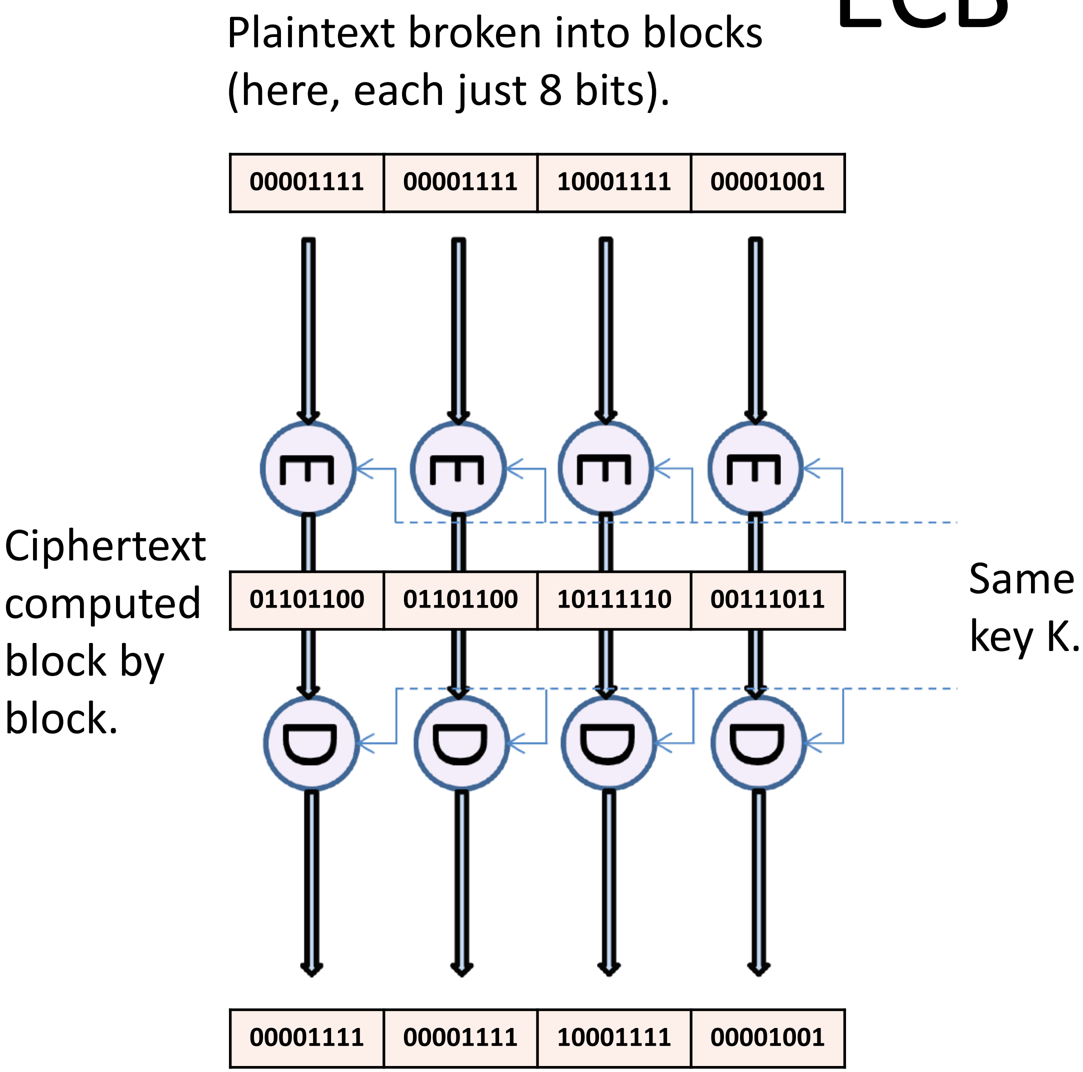
Plaintext broken into blocks
(here, each just 8 bits).



Ciphertext computed block by block.

Same key K.

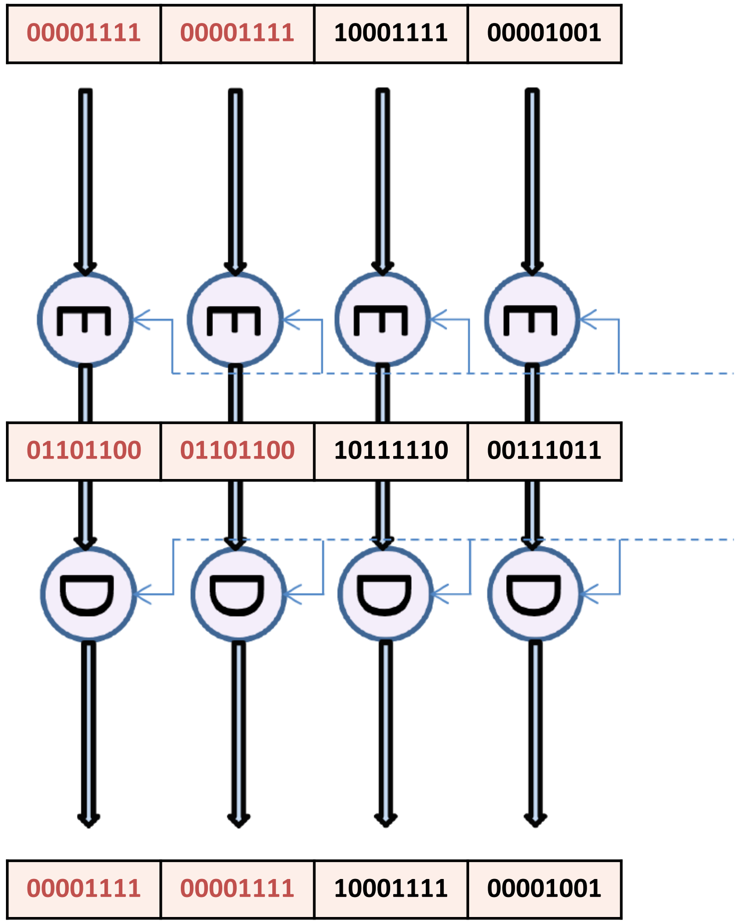
ECB



- Blocks can be exchanged (no integrity).

ECB

Plaintext broken into blocks (here, each just 8 bits).



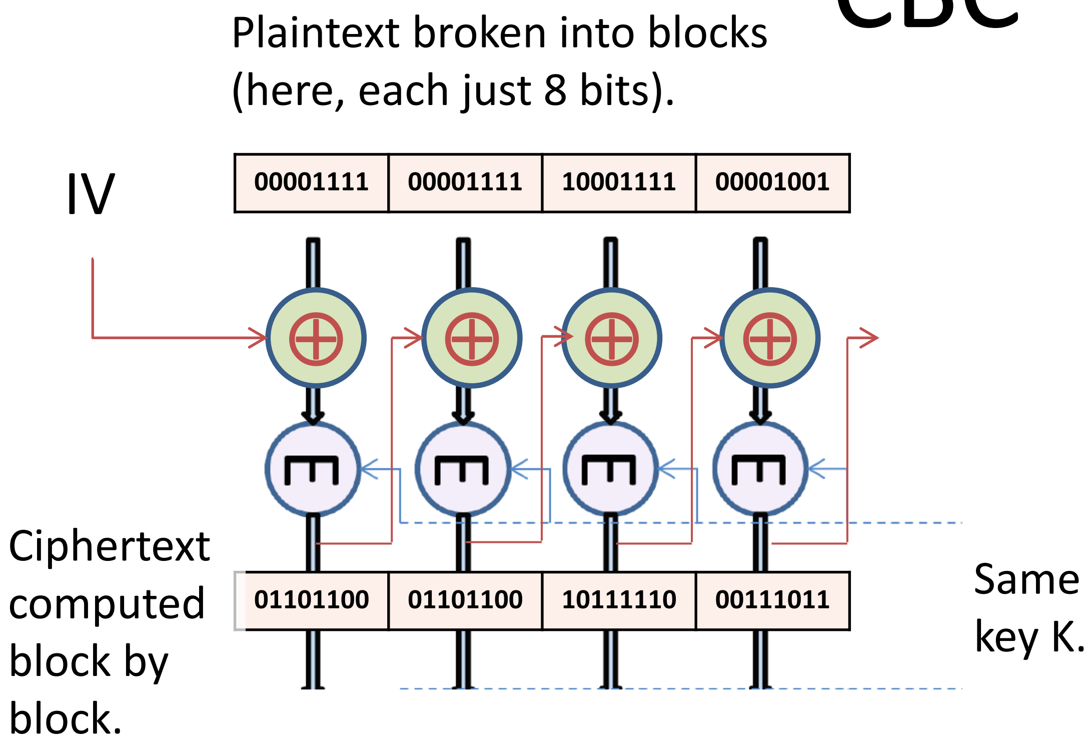
Ciphertext computed block by block.

Same key K.

- Blocks can be exchanged (no integrity).
- Equalities between blocks leak (no secrecy).

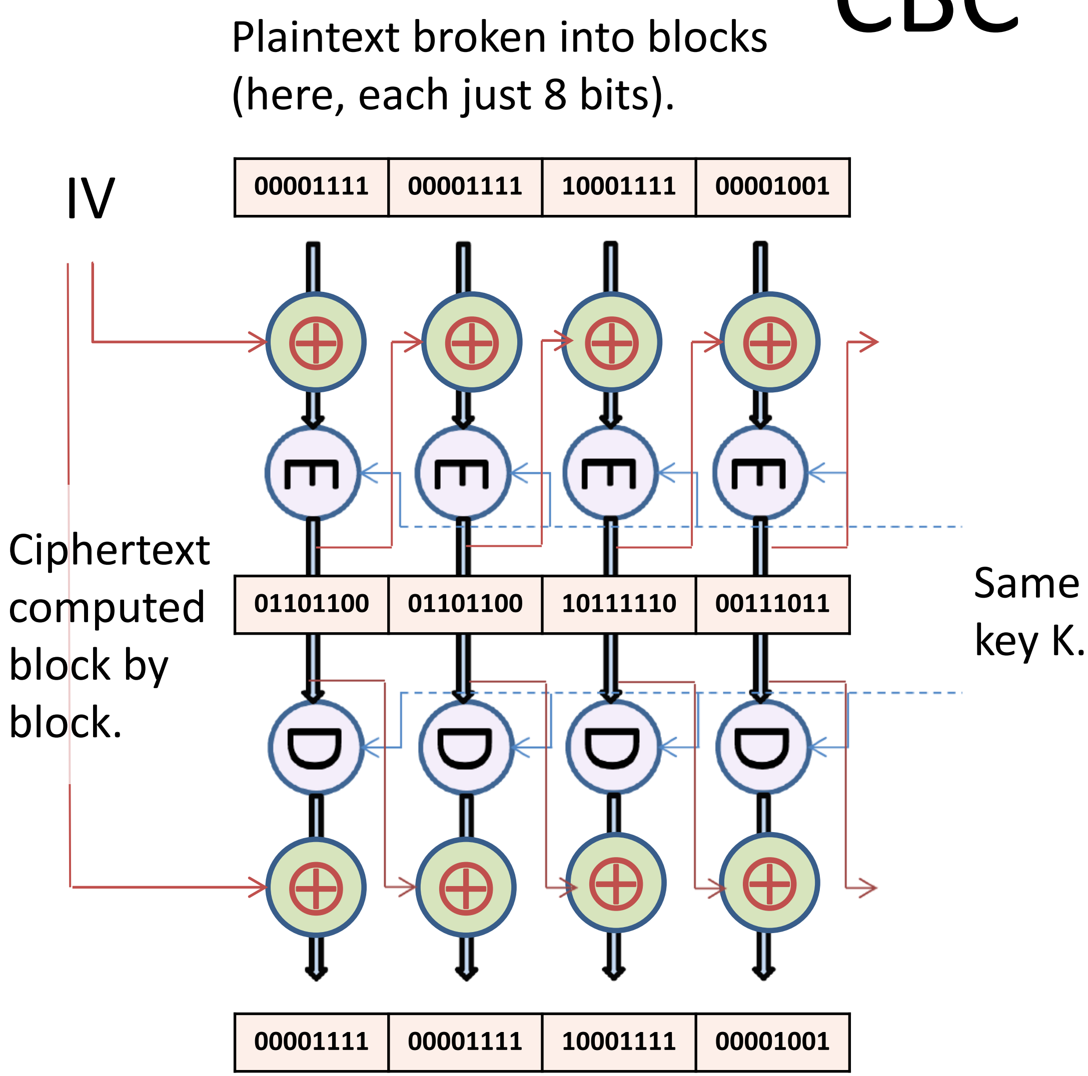
⇒ *Not generally a good mode!*

CBC



- Each plaintext block is first XORed with the previous ciphertext block.
- The first is XORed with an Initialization Vector (IV).

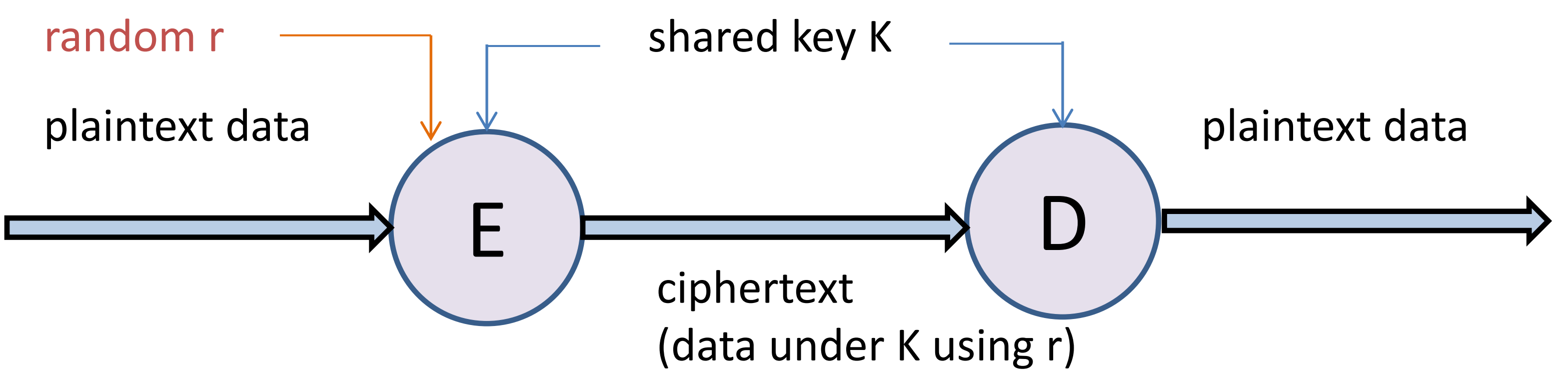
CBC



- Each plaintext block is first XORed with the previous ciphertext block.
- The first is XORed with an Initialization Vector (IV).

Probabilistic encryption

- Encryption can be randomized. That is, it may take a random number as a third argument.
- Thus, two encryptions of a plaintext with a key need not be identical.



One construction (from a non-probabilistic system (E,D)):

$E'_{K,r}(M)$ = pair of r and $E_K(M \oplus r)$

$D'_K(N)$ = (first element of N $\oplus$ D_K(second element of N))

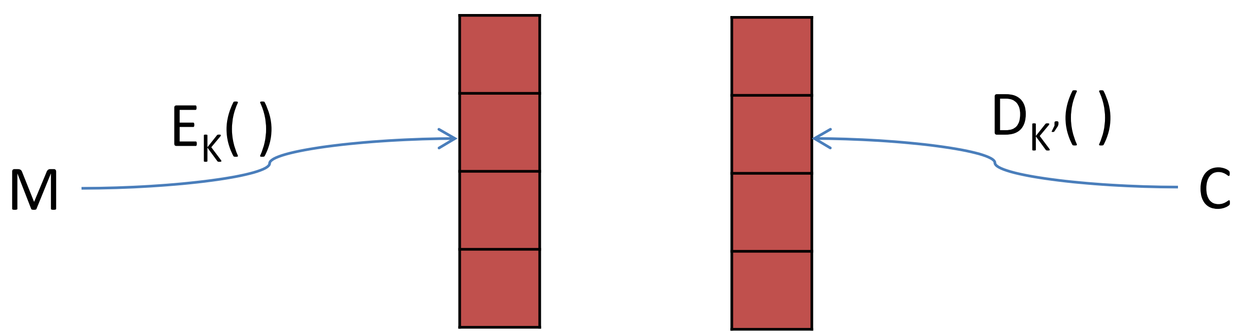
Multiple encryption

- So we know how to go from short messages to long messages. *Can we also go from short keys to long keys, and get stronger encryption?*
- A first idea is to nest two encryptions, as in $E_{K2}(E_{K1}(M))$, with different keys K_1 and K_2.
  - The hope is that the result will be as strong as if we had a longer key…
  - E.g., if K_1 and K_2 have length n, and breaking the encryption takes time 2^n , then breaking the double encryption should take time 2^{2n} … ???

A known-plaintext attack on double encryption

Given M and $C = E_{K2}(E_{K1}(M))$, find K_1 and K_2:

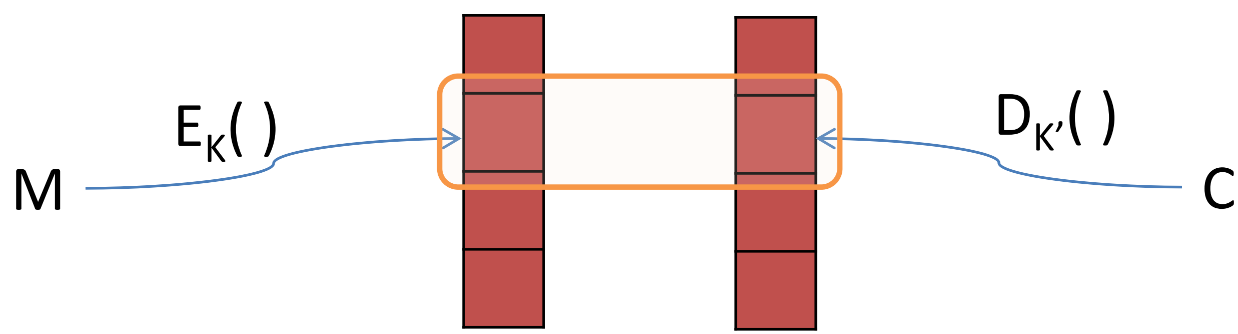
- Build a sorted table of pairs $(E_K(M), K)$ for all K, and a sorted table of pairs $(D_{K'}(C), K')$ for all K'.
- If $(E_K(M), K)$ and $(D_{K'}(C), K')$ are such that $E_K(M) = D_{K'}(C)$, consider that (K, K') is a candidate.



A known-plaintext attack on double encryption

Given M and $C = E_{K2}(E_{K1}(M))$, find K_1 and K_2:

- Build a sorted table of pairs $(E_K(M), K)$ for all K, and a sorted table of pairs $(D_{K'}(C), K')$ for all K'.
- If $(E_K(M), K)$ and $(D_{K'}(C), K')$ are such that $E_K(M) = D_{K'}(C)$, consider that (K, K') is a candidate.



A known-plaintext attack on double encryption

Given M and C = $E_{K2}(E_{K1}(M))$, find K_1 and K_2:

- Build a sorted table of pairs ($E_K(M)$, K) for all K, and a sorted table of pairs ($D_{K'}(C)$, K') for all K'.
- If ($E_K(M)$, K) and ($D_{K'}(C)$, K') are such that $E_K(M) = D_{K'}(C)$, consider that (K, K') is a candidate.
- There should be only one or few candidates. All but one can be discarded by checking a few other plaintext/ciphertext pairs.

Time: a fixed number of iterations over the key space (so, more like 2^{n+1} than 2^{2n}).

Perspectives

- It is easier and safer to rely on encryption schemes with variable key lengths by design.
- But some techniques with multiple encryption are strong. (This is not easy to prove.)

- *Not all "intuitive" techniques work as well as we might hope.*

$\Longrightarrow$ *"Don't do this at home."*

Public-key encryption
(a.k.a. asymmetric encryption)

Public-key encryption

- ***Public-key encryption*** generalizes shared-key encryption:
  - Each principal has a secret key SK for decrypting.
  - The inverse of the secret key is a public key PK for encrypting, with the property $D_{SK}(E_{PK}(M)) = M$.
- It usually relies on more mathematics, and it is usually slower (~ milliseconds).
- Key-distribution services need to know and transmit only public keys.

RSA

Encryption key:

- a modulus $N = pq$, where p and q are two (randomly chosen, large) primes,
- an exponent e that has no factors in common with $p - 1$ or $q - 1$.

$E_{(N,\, e)}(M) = M^e \bmod N$

Decryption key: the factors p and q.

$D_{(p,\, q)}(C) = C^d \bmod N$ where d is chosen so that

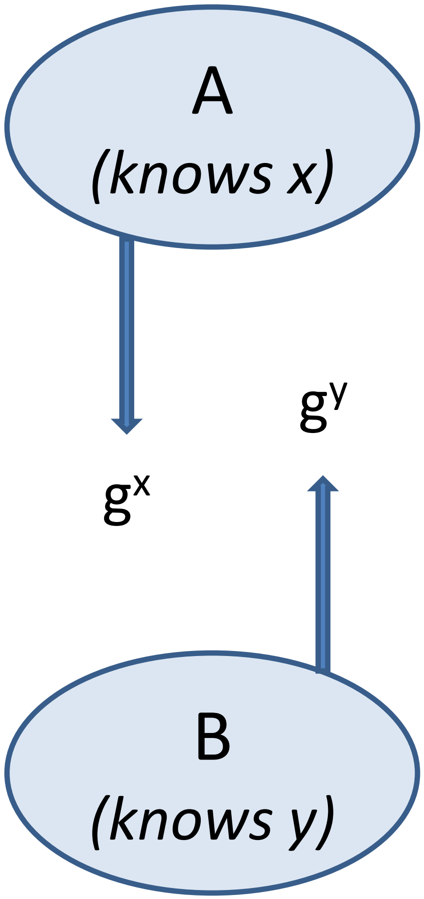
$ed = 1 \bmod (p - 1)(q - 1)$

RSA (cont.)

With a little number theory:

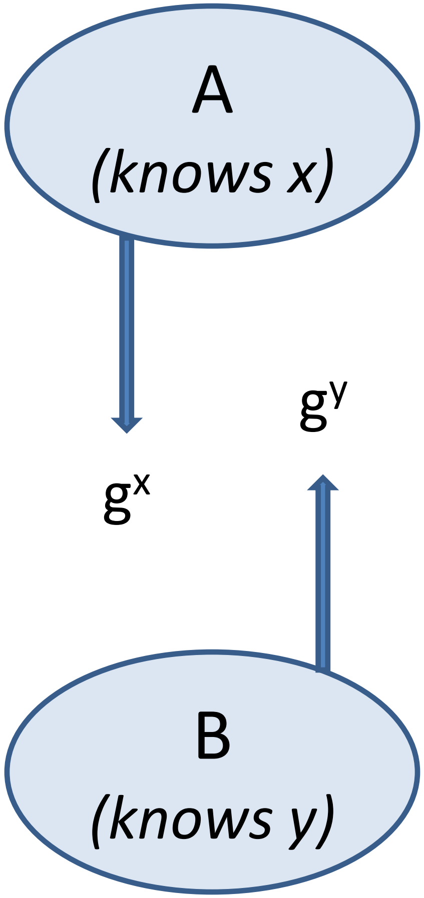
- d can be found efficiently: given e, p, and q, one can use the GCD algorithm to find d and k such that $ed + k(p - 1)(q - 1) = 1$.
- $C^d = M^{ed} = M^{1-k(p-1)(q-1)} = M \bmod N$.

Diffie-Hellman



- Let p be a prime and g a generator of $\mathbf{Z}_p^*$ (chosen with a little care).
- A invents x and publishes g^x mod p.
  B invents y and publishes g^y mod p.
  - x and y serve as secret keys.
  - g^x mod p and g^y mod p serve as public keys.

Diffie-Hellman



- Let p be a prime and g a generator of $\mathbf{Z}_p^*$ (chosen with a little care).
- A invents x and publishes g^x mod p.
  B invents y and publishes g^y mod p.
  - x and y serve as secret keys.
  - g^x mod p and g^y mod p serve as public keys.
- Both A and B can compute g^{xy} mod p.
  - It is a shared secret (but not authenticated).
  - From g^{xy}, A and B can for example compute keys.

Homomorphic encryption

A property of pure RSA

Given

- $E_{(N,\, e)}(M_1) = M_1^{\,e} \bmod N$
- $E_{(N,\, e)}(M_2) = M_2^{\,e} \bmod N$

anyone can compute

- $E_{(N,\, e)}(M_1 M_2) = (M_1 M_2)^e \bmod N$
  $= E_{(N,\, e)}(M_1) E_{(N,\, e)}(M_2) \bmod N.$

(This homomorphism is often false in standards based on RSA, but holds for pure RSA.)

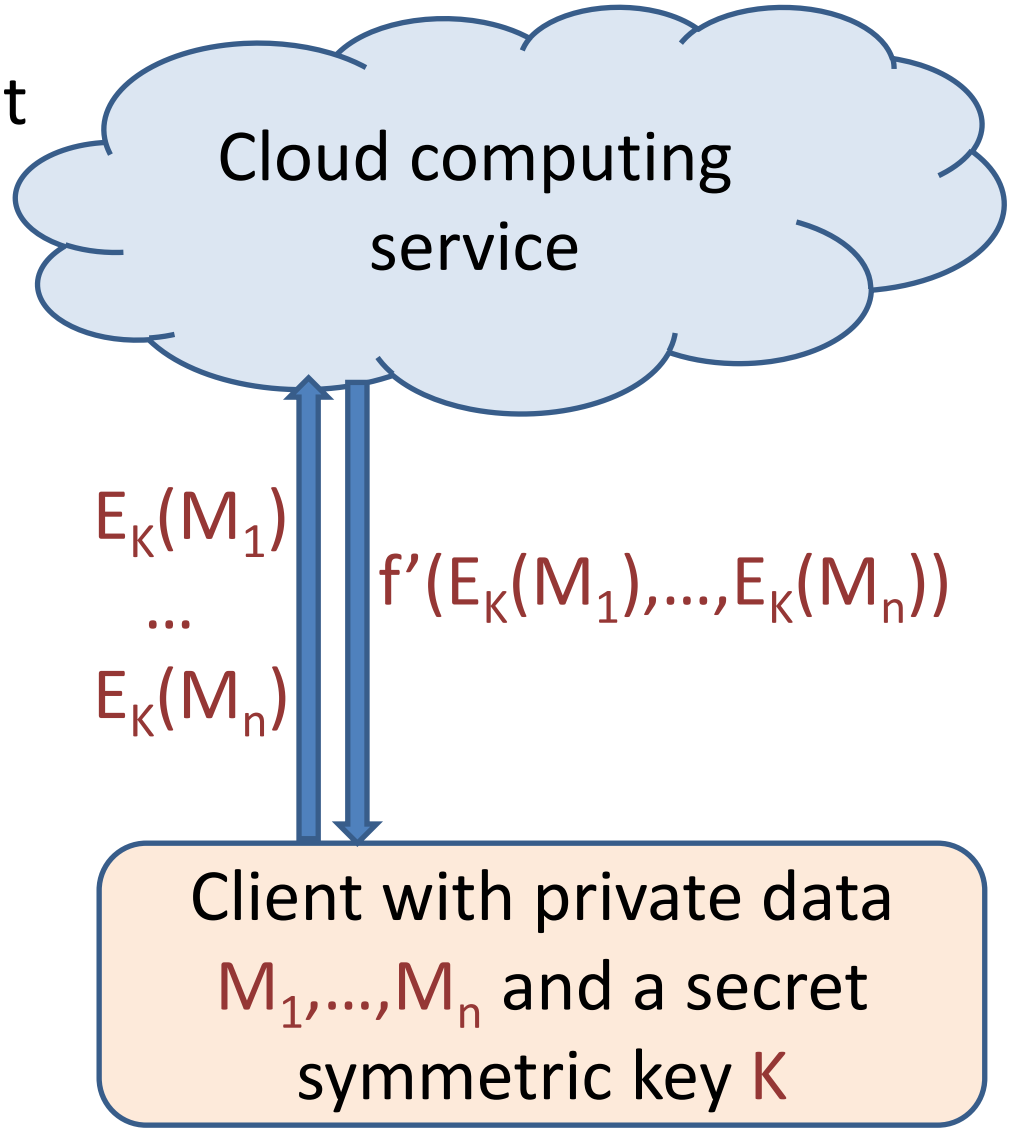
Homomorphic encryption (more generally)

- An encryption scheme is ***fully homomorphic*** if, for any function f on plaintexts, there is a function f' on ciphertexts such that $f(M_1,\ldots,M_n) = D_{SK}(f'(E_{PK}(M_1),\ldots,E_{PK}(M_n)))$ or, in the symmetric case, $f(M_1,\ldots,M_n) = D_K(f'(E_K(M_1),\ldots,E_K(M_n)))$.

The existence of such schemes was a big open problem, recently solved by C. Gentry. Costs seem to be measured in seconds and minutes.

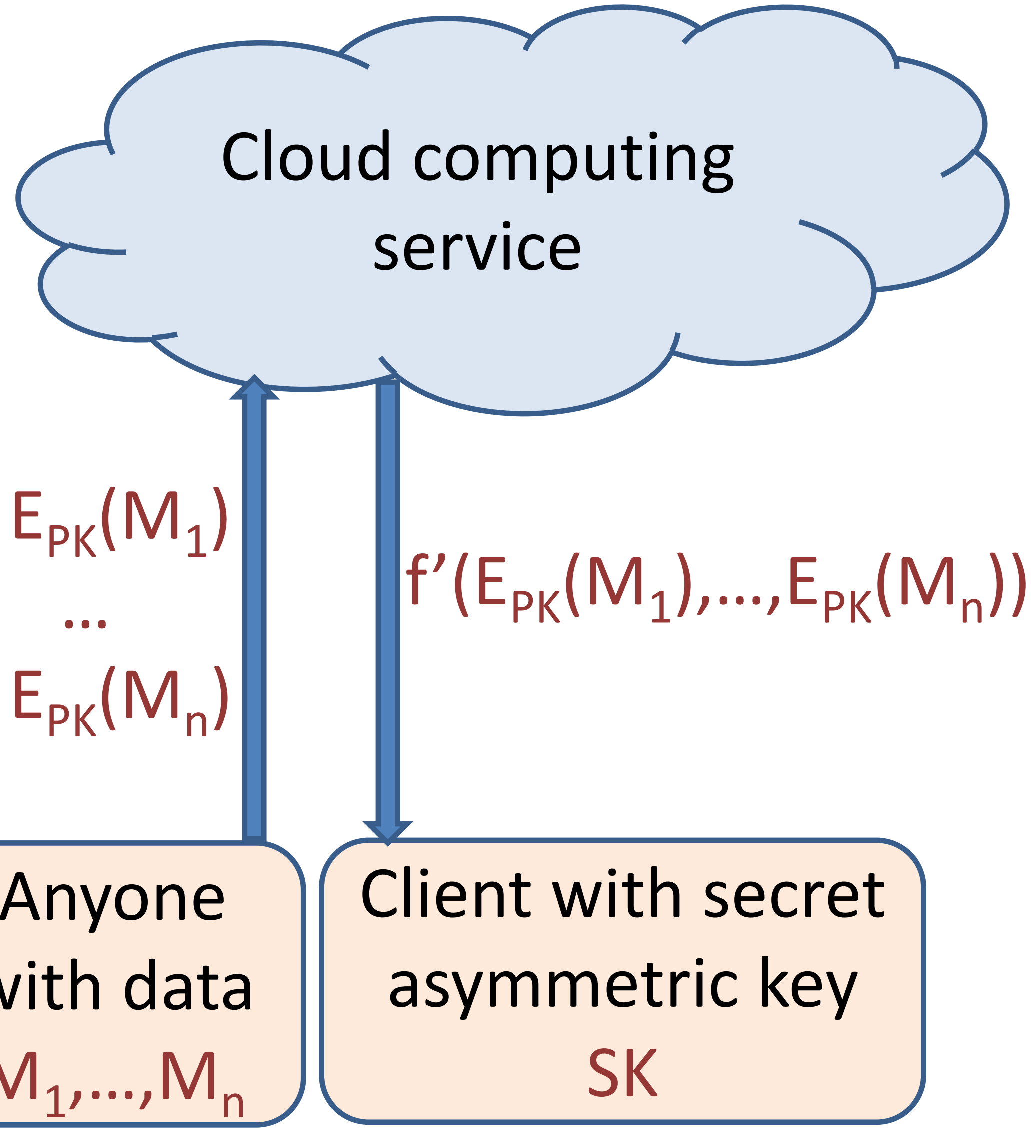
Homomorphic encryption and the clouds

The cloud can help a client in computing f without seeing plaintext data.



Homomorphic encryption and the clouds

Public-key versions allow more generality.



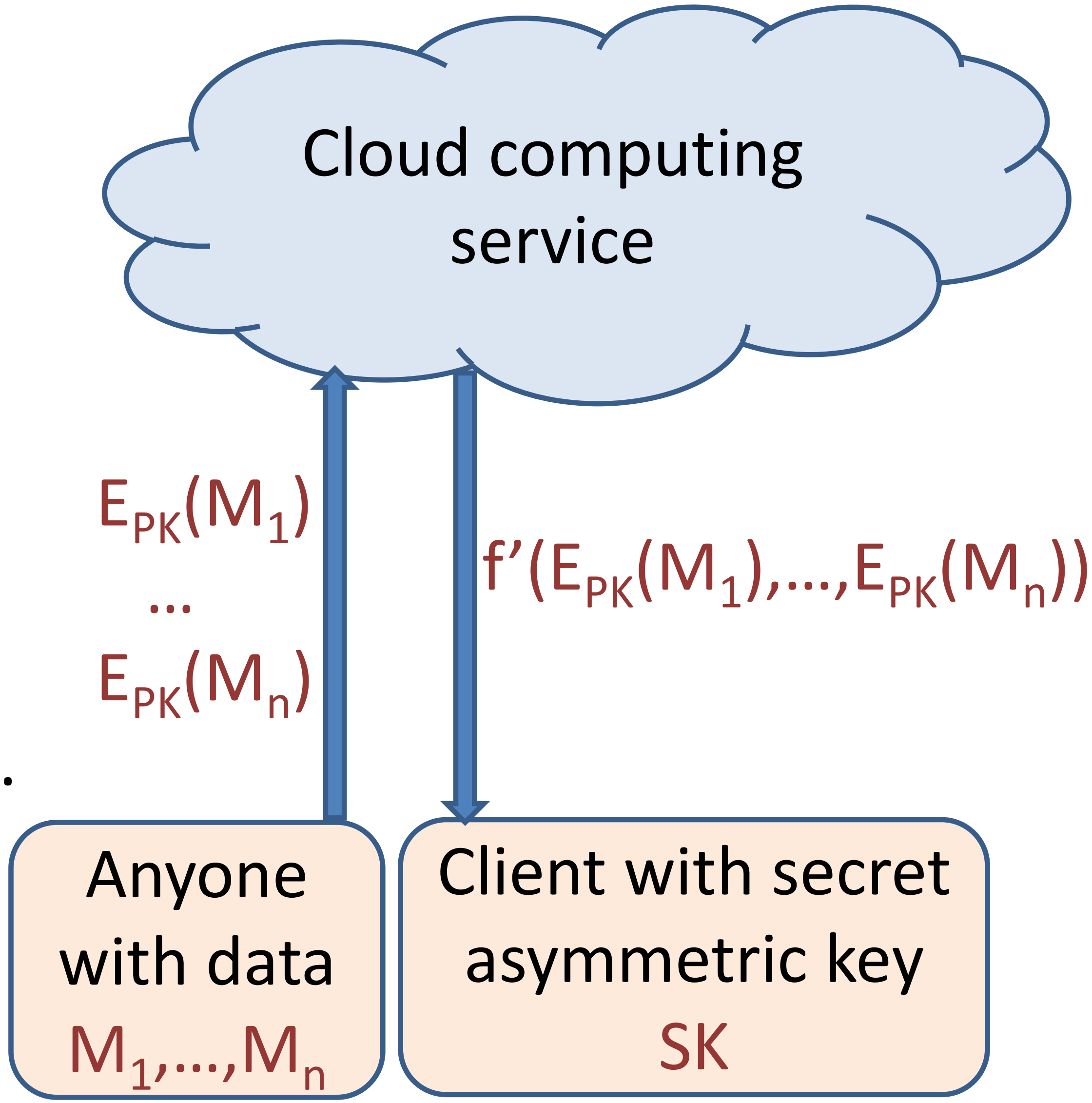
Homomorphic encryption and the clouds

Applications?

- Searches on private data.
- Any analysis of private data.

This has caused much excitement, but is not yet practical in general.

For some applications, special methods may be faster.



Hashes, MACs, and signatures

One-way hash functions (e.g., *hopefully* SHA-2)

f is ***collision-resistant*** if it is hard to find distinct M and N such that f(M)=f(N).

f is a ***one-way hash function*** (or ***cryptographic hash function***) if:

- f is collision-resistant,
- f is one-way,
- f(M) is of fixed size.

An example application: user authentication [Needham, 1967]

Using a one-way hash function f, a principal can recognize M without knowing it in advance.

For user authentication, this means that passwords do not need to be stored in cleartext.

Alice

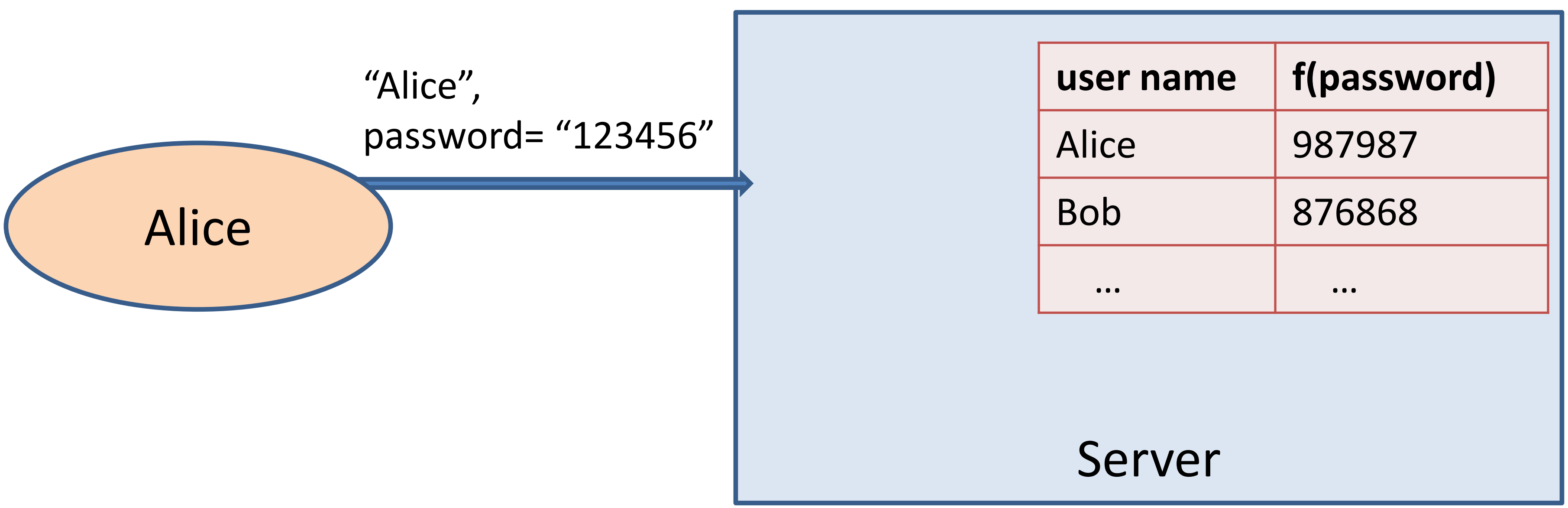
user name	f(password)
Alice	987987
Bob	876868
…	…

Server

An example application: user authentication [Needham, 1967]

Using a one-way hash function f, a principal can recognize M without knowing it in advance.

For user authentication, this means that passwords do not need to be stored in cleartext.

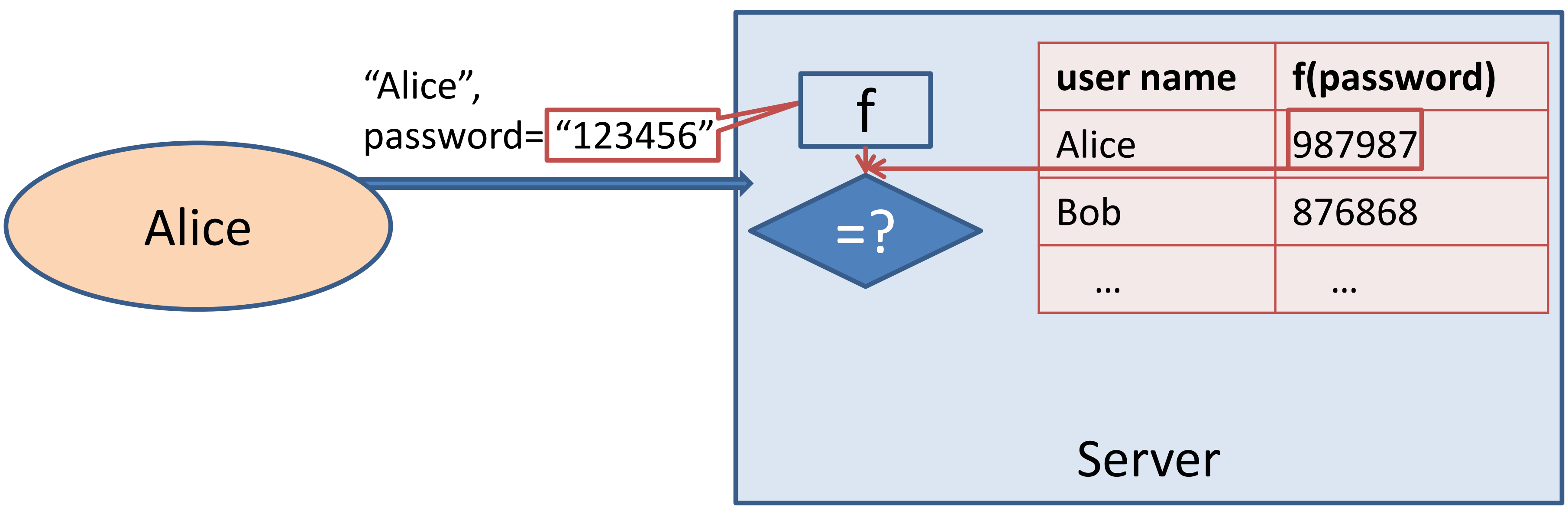


user name	f(password)
Alice	987987
Bob	876868
…	…

An example application: user authentication [Needham, 1967]

Using a one-way hash function f, a principal can recognize M without knowing it in advance.

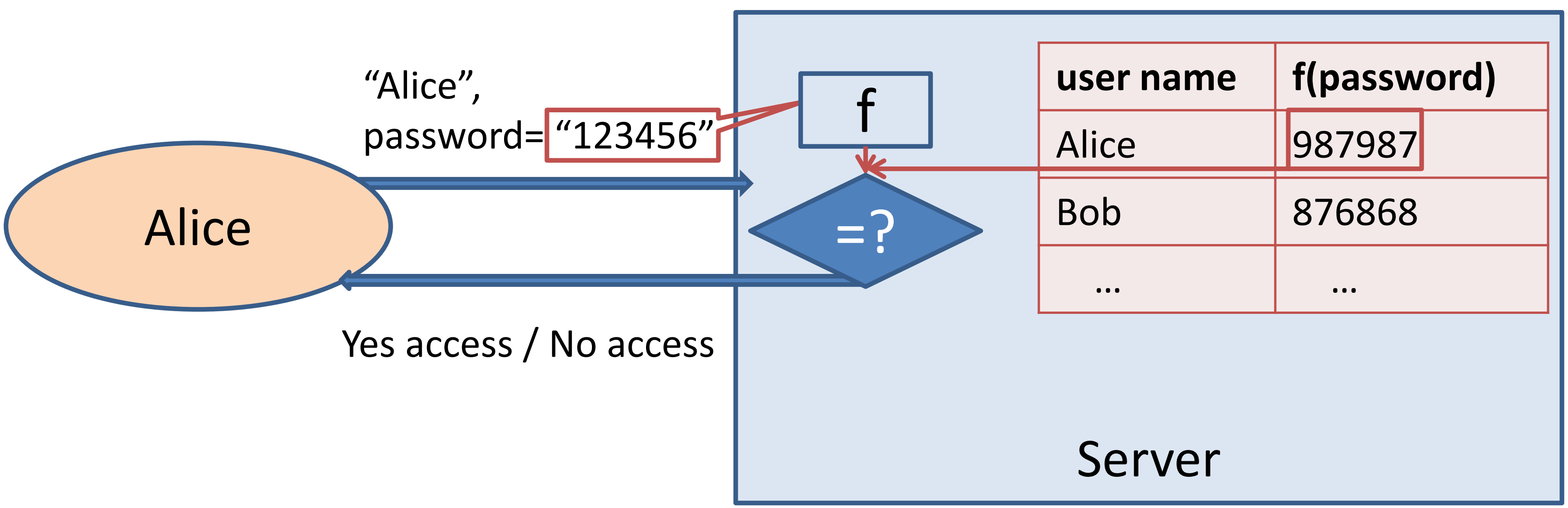
For user authentication, this means that passwords do not need to be stored in cleartext.



An example application: user authentication [Needham, 1967]

Using a one-way hash function f, a principal can recognize M without knowing it in advance.

For user authentication, this means that passwords do not need to be stored in cleartext.



An example application: user authentication [Needham, 1967]

Not always done perfectly…

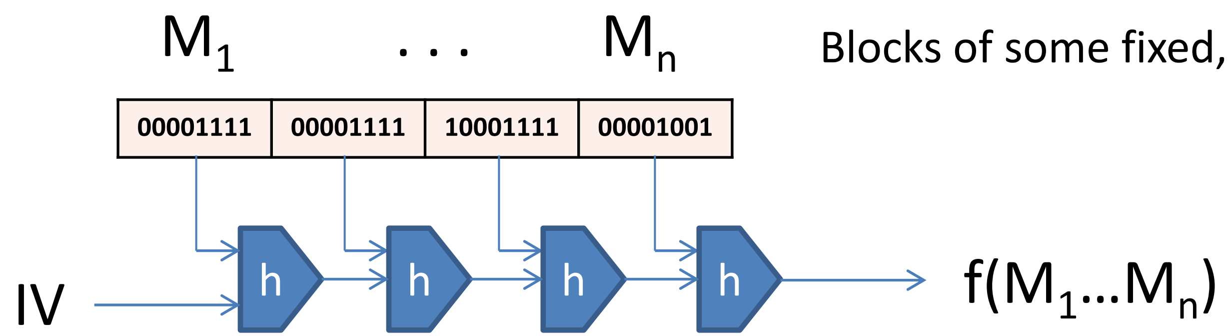
Amazon.com Security Flaw Accepts Passwords That Are Close, But Not Exact

By Dylan Tweney January 28, 2011 | 3:56 pm | Categories: Glitches and Bugs

One-way hash functions: the Merkle–Damgård construction

One-way hash functions are often defined by iterating a basic compression function h:

- $f(M_1) = h(IV, M_1)$
- $f(M_1 \ldots M_{i+1}) = h(f(M_1 \ldots M_i), M_{i+1})$ for $i = 1..(n-1)$.



Blocks of some fixed, small size.

One strengthening: add the length as a last block.

Message authentication codes or MACs

- Two principals know a key K.
- Both principals apply a function MAC_K for signing and for checking signatures:
  - To sign M, append $MAC_K(M)$.
  - To verify a signature N of M, check $N = MAC_K(M)$.

Message authentication codes or MACs: unforgeability

$MAC_K(M)$ should be easy to compute from K and M, but hard without knowing K.

More precisely:

- Given $MAC_K(M_1)$, . . ., $MAC_K(M_n)$ (but not K), it is hard to compute $MAC_K(M)$, for a new M.
- So $MAC_K(M_i)$ should not leak K, but it may reveal M_i.

Constructing MACs

- Typically, MACs are based on hash functions and on encryption functions.
- For example, given a one-way hash function f, we may try to set: $MAC_K(M) = f(KM)$.

Here KM is
the concatenation
of K and M.

Constructing MACs

- Typically, MACs are based on hash functions and on encryption functions.
- For example, given a one-way hash function f, we may try to set: $MAC_K(M) = f(KM)$.
  But this is subject to an ***extension attack***:
  $MAC_K(M_1 \ldots M_{n+1}) = h(MAC_K(M_1 \ldots M_n), M_{n+1})$ if f is defined from the compression function h.

Constructing MACs

- Typically, MACs are based on hash functions and on encryption functions.
- For example, given a one-way hash function f, we may try to set: $MAC_K(M)= f(KM)$.
  But this is subject to an ***extension attack***:
  $MAC_K(M_1 \ldots M_{n+1}) = h(MAC_K(M_1 \ldots M_n), M_{n+1})$ if f is defined from the compression function h.
- There are better ideas, for example:
  $MAC_K(M) = f(K\, f(KM))$ [see Krawczyk et al.'s HMAC]

Public-key signatures (e.g., RSA)

- Each principal has a secret key for signing.
- The inverse of the secret key is a public key for checking signatures.

A bit of theory: defining secure shared-key encryption

“Syntax”

An encryption scheme consists of algorithms:

$$K : \text{Parameter} \times \text{Coins} \to \text{Key}$$

$$E : \text{Key} \times \text{String} \times \text{Coins} \to \text{Ciphertext}$$

$$D : \text{Key} \times \text{String} \to \text{Plaintext}$$

where Parameter = 1* (numbers in unary) and Key, Plaintext, Ciphertext $\subseteq$ String.

Functionality

For all $\eta \in$ Parameter, $k \in K(\eta)$, $r \in$ Coins:

1) If $m \in$ Plaintext, then $D_k(E_k(m,r)) = m$.
2) If $m \notin$ Plaintext, then $D_k(E_k(m,r)) = \mathbf{0}$ where $\mathbf{0} \in$ Plaintext (a fixed string).

Security (1)

Idea:
An adversary cannot recover the key k from $E_k(m,r)$ (with or without knowledge of m and r).

Security (1)

Idea:

An adversary cannot recover the key k from $E_k(m,r)$ (with or without knowledge of m and r).

Objection:

The definition may be a little too strong:
The adversary may succeed only with low probability or after a lot of work.

Security (2)

Another objection:

The definition is satisfied by a scheme where keys are well protected but where $E_k(m,r) = m$!

Security (2)

Another objection:

The definition is satisfied by a scheme where keys are well protected but where $E_k(m,r) = m$!

$\Rightarrow$ So we will need to focus more on plaintexts and ciphertexts (rather than just on keys).

Security (3)

Idea:

Encryptions look random.

Security (3)

Idea:

Encryptions look random.

Objection:

Starting every ciphertext with a particular fixed string (e.g., “hello”) may be perfectly fine, even if encryptions don’t look random.

Security (4)

Idea:

An adversary cannot recover the plaintext m from $E_k(m,r)$ (without knowledge of k, but with or without knowledge of r and possibly other plaintexts and ciphertexts).

Security (4)

Idea:

An adversary cannot recover the plaintext m from $E_k(m,r)$ (without knowledge of k, but with or without knowledge of r and possibly other plaintexts and ciphertexts).

Objection:

This adversary may be able to gather a lot of information about m, for example a part of m.

Security (5)

Idea:

Whatever is efficiently computable from the ciphertext is also efficiently computable without the ciphertext.

Security (5)

Idea:

Whatever is efficiently computable from the ciphertext is also efficiently computable without the ciphertext.

Objection:

But what about the ciphertext itself, and the length of the plaintext?

Security (6)

Idea:

It is hard to distinguish a “real” encryption $E_k(m,r)$ from an encryption of 0s, namely $E_k(0^{|m|},r)$ where $|m|$ is the length of m.

Security (6)

Idea:

It is hard to distinguish a "real" encryption $E_k(m,r)$ from an encryption of 0s, namely $E_k(0^{|m|},r)$ where $|m|$ is the length of m.

Objection:

We also want, e.g., that $(E_k(m,r), E_k(m',r'))$ and $(E_k(0^{|m|},r), E_k(0^{|m'|},r'))$ be hard to distinguish, and even if the adversary picks m' dynamically.

Security (7)

Idea:

It is hard to distinguish a "real" encryption oracle $E_k(.)$ from an oracle $E_k(0^{|\cdot|})$ that encrypts 0s (both with fresh randomness r for each encryption).

This idea is the one that we will explore further (for security against chosen-plaintext attacks).

Negligible

A function f : Parameter $\rightarrow$ ***R*** is ***negligible*** if,

for all $c > 0$,

there exists b such that,

for all $\eta \geq b$,

$f(\eta) \leq \eta^{-c}$.

(In other words, f is below every inverse polynomial.)

Adversaries

An ***adversary*** is an algorithm A with access to an oracle.

If this oracle is the function f(.), we then write $A^{f(.)}$ for the system in which, whenever A queries the oracle with input x, it receives f(x).

Secure encryption

Given an encryption scheme and an adversary A, the ***advantage*** of A at η is the difference of probabilities:

$\Pr[k \in K(\eta) : A^{E_k(.)}(\eta) = 1] - \Pr[k \in K(\eta) : A^{E_k(0^{|.|})}(\eta) = 1]$

Secure encryption

Given an encryption scheme and an adversary A, the ***advantage*** of A at η is the difference of probabilities:

$\Pr[k \in K(\eta) : A^{Ek(.)}(\eta) = 1] - \Pr[k \in K(\eta) : A^{Ek(0^{|.|})}(\eta) = 1]$

The encryption scheme is ***secure*** (or ***"semantically secure"***) if this advantage is negligible for all polynomial-time adversaries A.

Some support for this definition

This definition might be counterintuitive.

(E.g., why are plaintexts in clear?)

Still:

- It is equivalent to other definitions.
- It implies expected properties.

Example

Suppose that there is a function g that recovers the first bit of m from its encryption.
Then we can construct an adversary A:

- A calls its oracle with a string of 1s.
- A runs g on the output.
- A returns the result.
- A’s advantage is 1, which is not negligible.

Deterministic encryption

By this definition, a deterministic encryption scheme cannot be secure.

The following A gets a high advantage:

- A calls its oracle twice, with two different plaintexts of the same length.
- A returns 1 if the results are different.

Deterministic encryption reveals equalities between plaintexts.

Going further

Nothing here says that:

- encryption provides integrity,
- encryption hides plaintext size,
- …

Some stronger definitions do.

Going further (cont.)

This approach is not specific to shared-key encryption and chosen-plaintext attacks.

- It applies to other cryptographic operations.
- It yields precise definitions and proofs.

Closing comments

Cryptography summary

	Encryption (for secrecy)	**Signatures** (for authenticity)
Symmetric a.k.a. *shared key*	The same key is used for encrypting and decrypting.	The same key is used for signing and checking signatures.
Asymmetric a.k.a. *public key*	The public key is used for encrypting. The corresponding secret key is used for decrypting.	The secret key is used for signing. The corresponding public key is used for checking signatures.

It is not safe, in general, to assume anything else !!!

In particular: Decryption success/failure may not be evident.
Encryptions may not look random, and may not provide integrity.

Reading

- "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", by Rivest, Shamir, and Adleman
  http://people.csail.mit.edu/rivest/RivestShamirAdleman-AMethodForObtainingDigitalSignaturesAndPublicKeyCryptosystems.pdf
- "Why Cryptosystems Fail", by Ross Anderson
  http://www.cl.cam.ac.uk/~rja14/Papers/wcf.pdf
- "Computing Arbitrary Functions on Encrypted Data", by Craig Gentry
  http://dl.acm.org/citation.cfm?id=1666444

Homework 5 (due November 15)

Exercise 1:

After reading the paper by Rivest et al., factor the number 7031 using the fact that its totient $\varphi(7031)$ is 6864.

Homework 5

Exercise 2:

Suppose that an attacker wishes to decrypt a given ciphertext C, and for this purpose can obtain the decryptions of any other ciphertexts C_1, …, C_n. Briefly show that fully homomorphic encryption, as defined in section 2.2 of Gentry's paper, is not secure against this attack. (This is a chosen-ciphertext attack, outside the class of attacks considered in the definition of semantic security in section 2.3.)