

# Introduction to Cryptography

Riccardo Focardi



Università  
Ca' Foscari  
Venezia

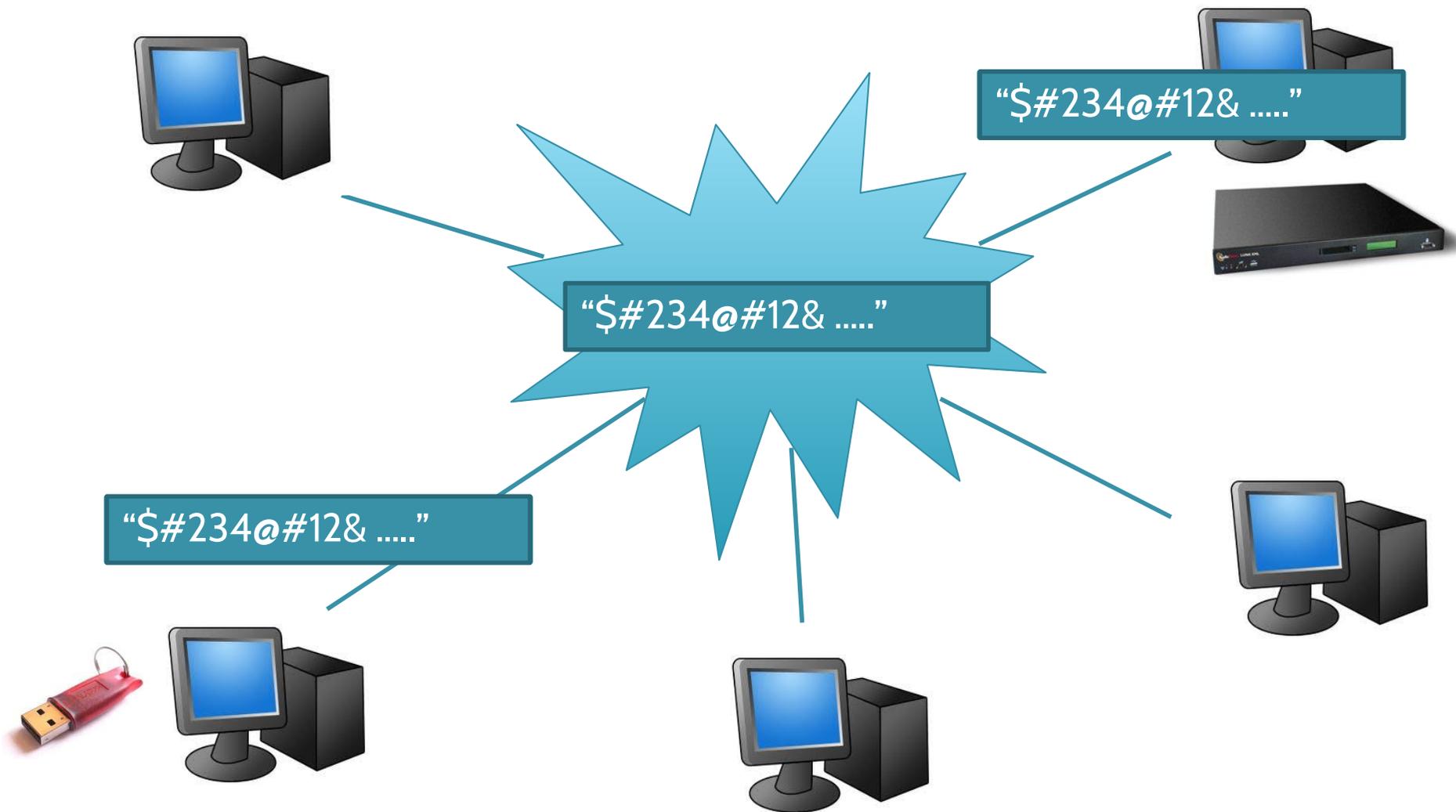


**Università Ca' Foscari Venezia**

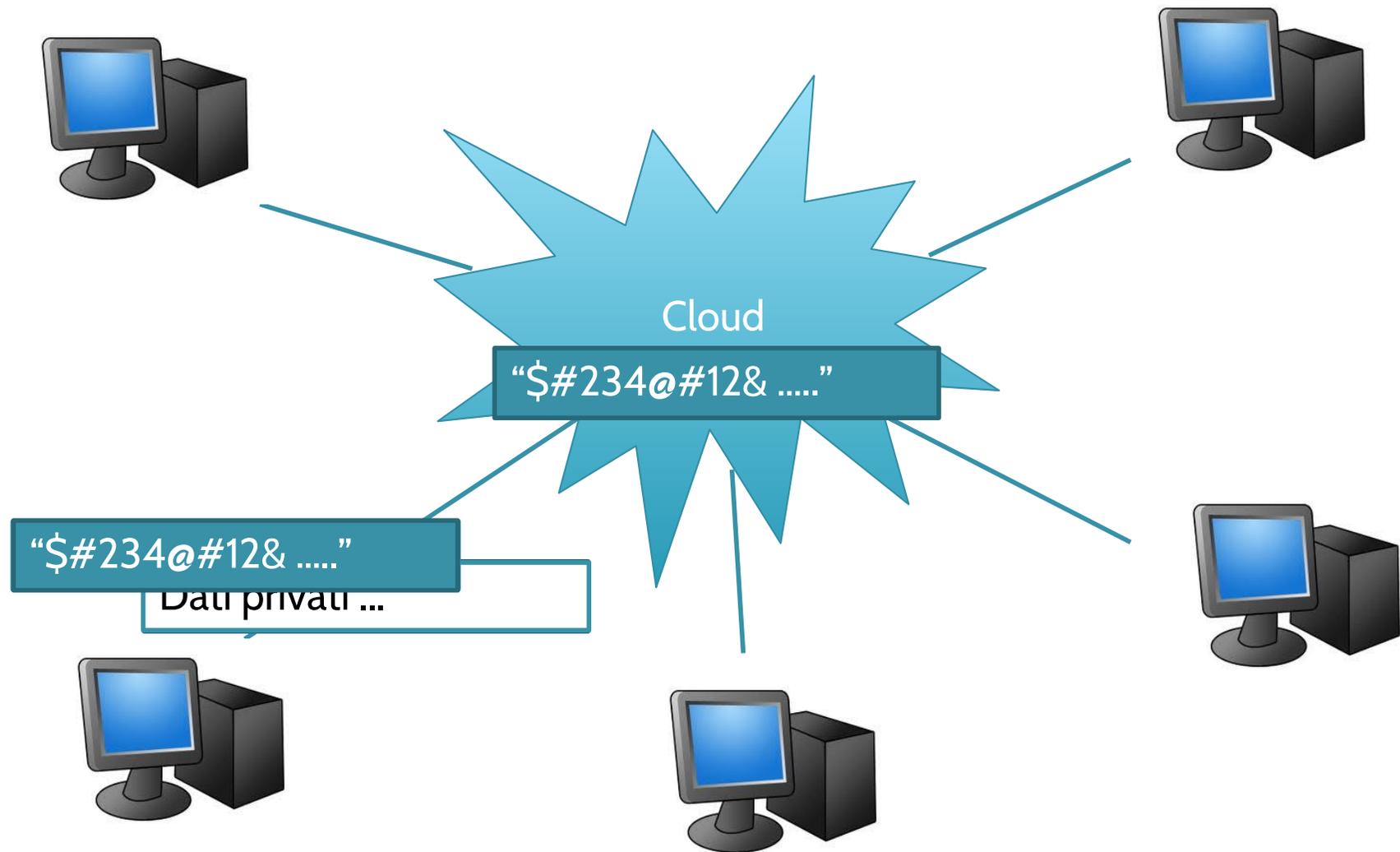
[www.dais.unive.it/~focardi](http://www.dais.unive.it/~focardi)  
[secgroup.dais.unive.it](http://secgroup.dais.unive.it)

**Cryptosense**  
[cryptosense.com](http://cryptosense.com)

# End to end security



# Secure storage in the cloud

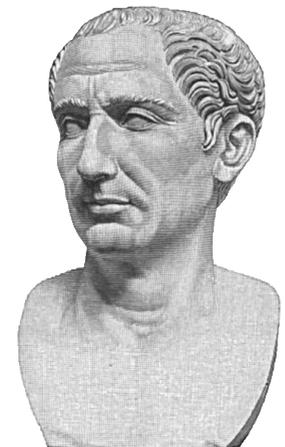


# Classic cryptography

A message is transformed so to make it hard to understand it

**Caesar cipher:** every letter is replaced by the one which is three positions next in the alphabet

GRP XV!



# Brute force and cryptanalysis

Caesar cipher only has 26 possible variants: we can **try them all!**

Moreover, equal letters are **encrypted in the same way**: it is easy, for example, to spot vowels and double letters.

# Example 1: Caesar cipher

RPTHGGRXEWGDCANWPHILTCINHXMKPGXPCIHIWP  
ILTRPCTPHXANQGJITUDGRT



# Brute forcing

We try all the 26 possibilities until we find something that makes sense in English:

```
for k in range(26):
    chiave = alfabeto[k:] + alfabeto[:k]
    print (k, d('RPTHGPRXEWGDCANWPHILTCINHXMKPGXPCIHWPILTRPCTPHXANQGJITUDGRT', chiave))
...
0 RPTHGPRXEWGDCANWPHILTCINHXMKPGXPCIHWPILTRPCTPHXANQGJITUDGRT
1 QOSGOFQWDVSFCBZMVOGHKSBHMGWLJOFWOBHGHVOHKSQOBSOGWZMPFIHSTCFQS
...
15 CAESARCIPHERONLYHASTWENTYSIXVARIANTSTHATWECANEASILYBRUTEFORCE
...
25 SQUIQHSYFXUHEDBOXQIJMUDJOIYNLQHYQDJIJXQJMSQDUQIYBORHKJUVEHSU
```



# Example 2: substitution cipher

HMBFSFNCXZEFNBHCWPSHMZFXCNMPNFSYPWWPNKHMIDXWSKQMFENZHMHFZOCNMMPOHKLWCDQZFLSNCBBPKHC  
DKKFYPNOFWPSHPNKHMWPQSKWCQKLZFLSFIHLHFZIMMNBPKHHPEIDKFWKMQWCQKLZFLSBPKHCDKFWPSHKHMICLSAF  
QMNKSHFZKCXZHPEKHFKKHMCXZEFNBFSNCBZMWPNPKMXLNFNZWPNFXXLSFXFCBHPTHPSKHMBCQSKWCQECWDNXDTYLF  
NZKHMICLHFZOCNMFKKHMPQCQZMQSPNFCNCKHMQICFKBHPTHTFDOHKKHQMMOCCZWP SHKHMWPQSKBMMYPKEFZMKH  
MICLSFZKCSMMKHMCMXZEFNTCEMPNMFTHZFLBPKHHPSSYPWWMEAKLFNZHMFXBFLSBMNKZCBNKCHMXAHPETFQQLMPKH  
MQKHMTCPXMZXPNMSCQKHMOWFWFNZHFQACCNFNZKHMSFPXKHFKBFSWDQXMZFOCDNZKHMEFSKKHMSFPXBFSFAFKTH  
MZBPKHWXCDQSFTYSFNZWDQXMZPKXCCYMZXPYMKHMWXFOCWAMQEFNMNKZMWMFKKHMCXZEFNBFSKHPNFNZOFDN  
KBPKHZMMABQPNYXMSPNKHMIFTYCWHPSNMTYKHMICBNIXCKTHMSCWKHMIMNMJCMNKSYPNTFNTMQKHMSDNIQPNO  
SWQCEPKSQMWXMTKPCNCNKHMKQCAPTSMFBMQMCNHPSTHMMYSKHMIXCKTHMSQFNBMXXZCBNKHMSPZMSCWHPSWF  
TMFNZHPSHFNZSHFZKHMZMMATQMFMSZSTFQSWQCEHFNZXPNOHMFJLWPSHCNKHMTQZSIDKNCNMCWKHMSMSTFQSB  
MQMWQMSHKHMLBMQMFSCXZFSMQCSPCNPNFWPSHXMSSZMSMQMJMLKHPNOFICDKHPEBFSCXZMVTMAKHPSMLMS  
FNZKHMLBMQMKHMSFEMTCXCQFSKHMSMFFNZBMQMTTHMMQWDXFNZDNZMWMFKMZSFNKPFOCKHMICLSFPZKCHPEFSK  
HMLTXPEIMZKHMIFNYWQCEBHMQMKHMSYPWWBFSHFDMZDAPTCDXZOCBPKHLCDFOPNBMJMEFZMSCEMECNMLKHM  
XZEFNHFZKFDHKKHMICLKCWPSHFNZKHMICLXCJMZH PENCKHMCXZEFNSFPZLCDQMBPKHFXTDYLICFKSKFLBPKHKKHMEIDK  
QMEMEIMQHCBLCDBMNKMPHKLMSJMNZFLSBPKHCDKWPSHFNZKHMBMTFDOHKIPOCNMSMJMLZFLWCQKHQMMBMM  
YSPQMEMEIMQKHMCXZEFNSFPZPYNCBLCDZPNCKXMFJMEMIMTFDSMLCDZCDIKMZPKBFSFAFAFEFZMEMXMFJMPFEFICLFNZ  
PEDSKCIMLHPEPYNBKHMCXZEFNSFPZPKPSRDPKMNCQEFXHMHFSNKEDTHWFPKHNCNCKHMCXZEFNSFPZIDKBMHFJMHFJMN  
KBMLMSKHMICLSFPZTFNPCWWMQLCDFIMMQCNKHMKMQQFTMFNZKHMBMXXKFYMKHMSKDWWHCEMBHLNCKKHMCXZE  
FNSFPZIMKBMMNWP SHMQEMNKHMLSFKCNKHMKMQQFTMFNZEFNLCKWKMWP SHMQEMNEFZMWDNCWKHMCXZEFNFNZ  
HMBFSNCKFNOQLCKHMQSCWKHMCXZMQWPSHMQEMNXCCYMZFKHPEFNZBMQMSFZIDKKHMLZPNCKSHCBPKFNZKHMLS  
ACYMACXPKMXLFICDKHMTDQQMNKFNZKHMZMAKHSKHM LHFZZQPWKMKZHM PXPNM SFKNZKHMSKMFZLOCCZBMFKH  
MQFNZCWBHFKHMLHFZSMMNKHMSTTTMSSWDXWPSHMQEMNCWKHFZFLBMQMFQMFZLPNFNZHFZIDKTHMQMZKHM  
PQEFQXPNCDFNZTFQQPMZKHMEXFPZWDXXXMNOKHFTQCSSKBCAXFNYSBPKHKBCEMN



# Cryptanalysis of substitution ciphers

Substitution ciphers use random **alphabet permutations**

ABCDEFGHIJKLMNOPQRSTUVWXYZ

KZBARCQHSMNIUWVPJGEOTFDXLY

Since there are

$$26! = 403291461126605635584000000 \approx 2^{88}$$

permutations, we cannot try all of them.

However we can break the cipher through statistical analysis and a dictionary. Try [here](#).

# What is a cipher?

A cipher is defined through two functions

1. **Encryption**: given a **plaintext** and a **key** returns a **ciphertext**

$$E_{K_1}(X) = Y$$

2. **Decryption**: given a **ciphertext** and a **key** returns a **plaintext**

$$D_{K_2}(Y) = X$$

# Symmetric and asymmetric ciphers

Decrypting the encryption of  $X$  we obtain  $X$ :

$$D_{K_2}(E_{K_1}(X)) = X$$

When  $K_1=K_2$  we have a **symmetric key cipher**

When  $K_1 \neq K_2$  we have an **asymmetric key cipher**

**Security:** it should be **unfeasible** to compute  $X$  or  $K_2$  from  $Y$  even knowing other pairs  $(X_1, Y_1), \dots, (X_n, Y_n)$

# Modern cryptography: AES

Modern ciphers are very complex and use keys of at least 128 bits:

about  $3.40282367 \times 10^{38}$  different keys

**Example:** we can use openSSL to experiment:

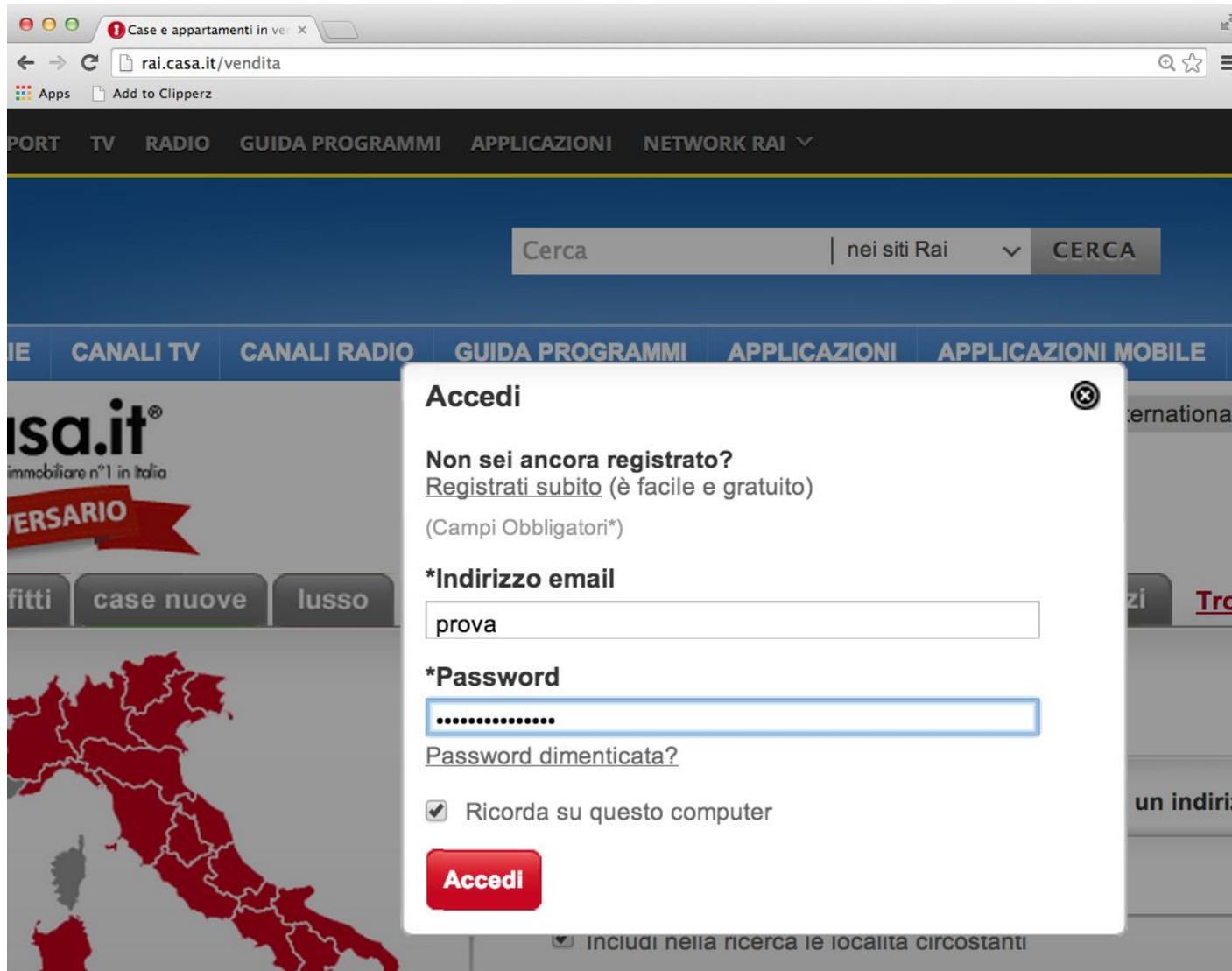
```
$ openssl rand -hex 16
ca8b7f7e66ab27302f7527df300f0fdf
$ ca8b7f7e66ab27302f7527df300f0fdf | hexdump
0000000 e3 d4 af 79 69 fa 02 31 db 58 2a f5 e3 33 13 1e
0000010
```

# Key size

In the [ENISA report 2014](#) we find the following:

<b>Cipher</b>	<b>legacy</b>	<b>near term</b>	<b>long term</b>
Symmetric key	80	128	256
RSA	1024	3072	15360

# Cryptography on the web



# http: no protection!

The screenshot shows a Wireshark window with a 'Follow TCP Stream' dialog open. The dialog displays the raw content of an HTTP POST request, which is unencrypted. The request body contains sensitive information such as a username and password. The status bar at the bottom indicates that 28 packets are displayed, with 7 marked.

Stream Content

```
POST /login_verify.ds HTTP/1.1
Host: rai.casa.it
Connection: keep-alive
Content-Length: 55
Origin: http://rai.casa.it
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/34.0.1847.137 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: */*
Referer: http://rai.casa.it/vendita
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,it;q=0.6
Cookie: NSC_etbqqgbsn=e2451c313660;
lmdstok=aWQjZmY4MDgxODE0NWZlOTdmMzAxNDYyMGQyMGNiMDRiZjg6MzU0ODE5NDA2MDMxNTpiYzc4YzU5Yzk
5NjgxNTRlMmEyNDI5NmU0MTNmNzU0YQ; s_nr=1400710641395;
JSESSIONID=0A004D54AE602C9391470186C21E7579;
__utma=135384818.195807481.1400709972.1400709972.1400750729.2;
__utmb=135384818.2.10.1400750729; __utmc=135384818;
__utmz=135384818.1400709972.1.1.utmcsr=sipra-rai|utmccn=institutional|utmcmd=cobrand-
tab|utmctt=logo-link; s_cc=true; _stc=raicobweb; s_sq=%5B%5BB%5D%5D

username=prova&password=passwordsegreta&rememberMe=true...5...@c.-L..K.
%Z...M..lk.,.|W.nQ?n...|....
.|?
```

Entire conversation (1297 bytes)

Find Save As Print ASCII EBCDIC Hex Dump C Arrays  Raw

Help Filter Out This Stream Close

File: "/var/folders/b\_/6833\_... : Packets: 28 Displayed: 7 Marked: 0 Dropped: 0 Profile: Default



# https: communication is encrypted

The screenshot shows the Wireshark interface with a packet list on the left and a packet details pane on the right. The main window displays the 'Stream Content' for a selected packet (No. 533). The data is shown as a sequence of garbled, non-readable characters, indicating that the communication is encrypted. The packet list shows packets 530 through 542, and the details pane shows 'Internet Protocol Version 4' and 'Hypertext Transfer Protocol'.

No.	Time
530	11.02
531	11.02
532	11.02
533	11.03
534	11.03
535	11.03
536	11.03
537	11.03
538	11.03
539	11.03
540	11.03
541	11.03
542	11.08

Stream Content

```
%J.....4.....z]U.  
c.\.b..fN...A.[$.S.#k.n.R.....=(.....>.....%  
^..x,...yw...<.....Xj.....v.z  
+..c.*Xw.w..b7...~.d..x.f..6.;.v.';.....!b.^..h.pR...o.v...?yA..Ut)t.M(!?  
Nv..xR.7...:EK...b..m~.h..R.<..o.X...m.&:"...&.MP.@.....'.m.x<.  
\..U#...8..k.I5/'..1.'-.....s&..l.....E.^).....N...['.....p..}  
R.....g.....}4..D..{.b(..6.....H.O..{.T~a..#i..ag...0h...-Z.....q...!  
*..C...@q..  
Wlj.H.Q.X7Y.P..&.*.....)}...8.=&..aA.[.(...y..[Q.....q  
$lv.@..".u...l...f6.....S.H.)E.G4...  
+...i.Wz..e...qG..Z...#d.....H...w.....{PG.X6.y..|...u..t.S#p..*9K..{/..  
+y.....0...P1D.....l.jG..J.p.;.....  
-.....i...f..T...mF.z.n(H.....r..P.H..v.....e...  
.....3MO,..t./L...].i.KA.d...Q.....!  
E...S...o1...m.g.....un...y...Q..W...vIY.....n...M.....'S.t:.Ng.{l  
(3y#.<&.."Y.L.....z...%i.....z#\'.|.?<.....3.>e..  
(=.4...c.CF2...R.  
..|x...D?.9i.2.g.a.d...3t.&6..#...Nu.n#...(...U..My,GalZS.....<|. [...z9.#  
D6n..O.C:r.}....q....g(  
F4p66.f....^.....|...U.K.!.[.d[m  
.....*.....#.  
..gi..B.....;...T..v[.....A.V.&.K8..  
gyL.....+.@.....}.....B;&.3uj....'FF*2.[.zk....4.@w./m..%E-....#.7.A4...L..yv|  
n.R.C.....n2j..tS.kP.<8}N..._[H1HR.o.l.s.@.In...~  
...]P.....f.S&NEWQ.y.....j.....
```

Entire conversation (17211 bytes)

Find Save As Print ASCII EBCDIC Hex Dump C Arrays Raw

Help Filter Out This Stream Close

# Cryptography ... of things!



# Cryptography in banks

Payments, ATMs, money transfers, ...

## Hardware Security Module (HSM)

Costs about 5k-20k € for a market of 200M € a year



# ... but things can go wrong

Many attacks on cryptographic systems in the last years:

- R. Verdult, F. D. Garcia and B. Ege.  
**Dismantling Megamos Crypto: Wirelessly Lockpicking a Vehicle Immobilizer.** USENIX Security 2013
- R. Bardou, R. Focardi, Y. Kawamoto, L. Simionato, G. Steel, J. Tsay.  
**Efficient Padding Oracle Attacks on Cryptographic Hardware.** CRYPTO 2012
- M. Bortolozzo, M. Centenaro, R. Focardi, G. Steel.  
**Attacking and fixing PKCS#11 security tokens.** ACM CCS 2010
- F. D. Garcia, P. van Rossum, R. Verdult and R. Wichers Schreur.  
**Wirelessly Pickpocketing a Mifare Classic Card.** IEEE S&P 2009



# Smartcards and crypto tokens

Brand	Device Model	Supported Functionality						Attacks found				Tk	
		s	as	cobj	chan	w	ws	wd	rs	ru	su		
Aladdin	eToken PRO	✓	✓	✓	✓	✓	✓	✓					wd
Athena	ASEKey	✓	✓	✓									
Bull	Trustway RCI	✓	✓	✓	✓	✓	✓	✓					wd
Eutron	Crypto Id. ITSEC		✓	✓									
Feitian	StorePass2000	✓	✓	✓	✓	✓	✓	✓	✓	✓			rs
Feitian	ePass2000	✓	✓	✓	✓	✓	✓	✓	✓	✓			rs
Feitian	ePass3003Auto	✓	✓	✓	✓	✓	✓	✓	✓	✓			rs
Gemalto	SEG		✓		✓								
MXI	Stealth MXP Bio	✓	✓		✓								
RSA	SecurID 800	✓	✓	✓	✓				✓	✓	✓		rs
SafeNet	iKey 2032	✓	✓	✓		✓							
Sata	DKey	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		rs
ACS	ACOS5	✓	✓	✓	✓								
Athena	ASE Smartcard	✓	✓	✓									
Gemalto	Cyberflex V2	✓	✓	✓		✓	✓	✓					wd
Gemalto	SafeSite V1		✓		✓								
Gemalto	SafeSite V2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		rs
Siemens	CardOS V4.3 B	✓	✓	✓		✓					✓		ru



# Real attacks!



20 February 2013

**35 000 000 € stolen from ATMs in less than 10 hours**

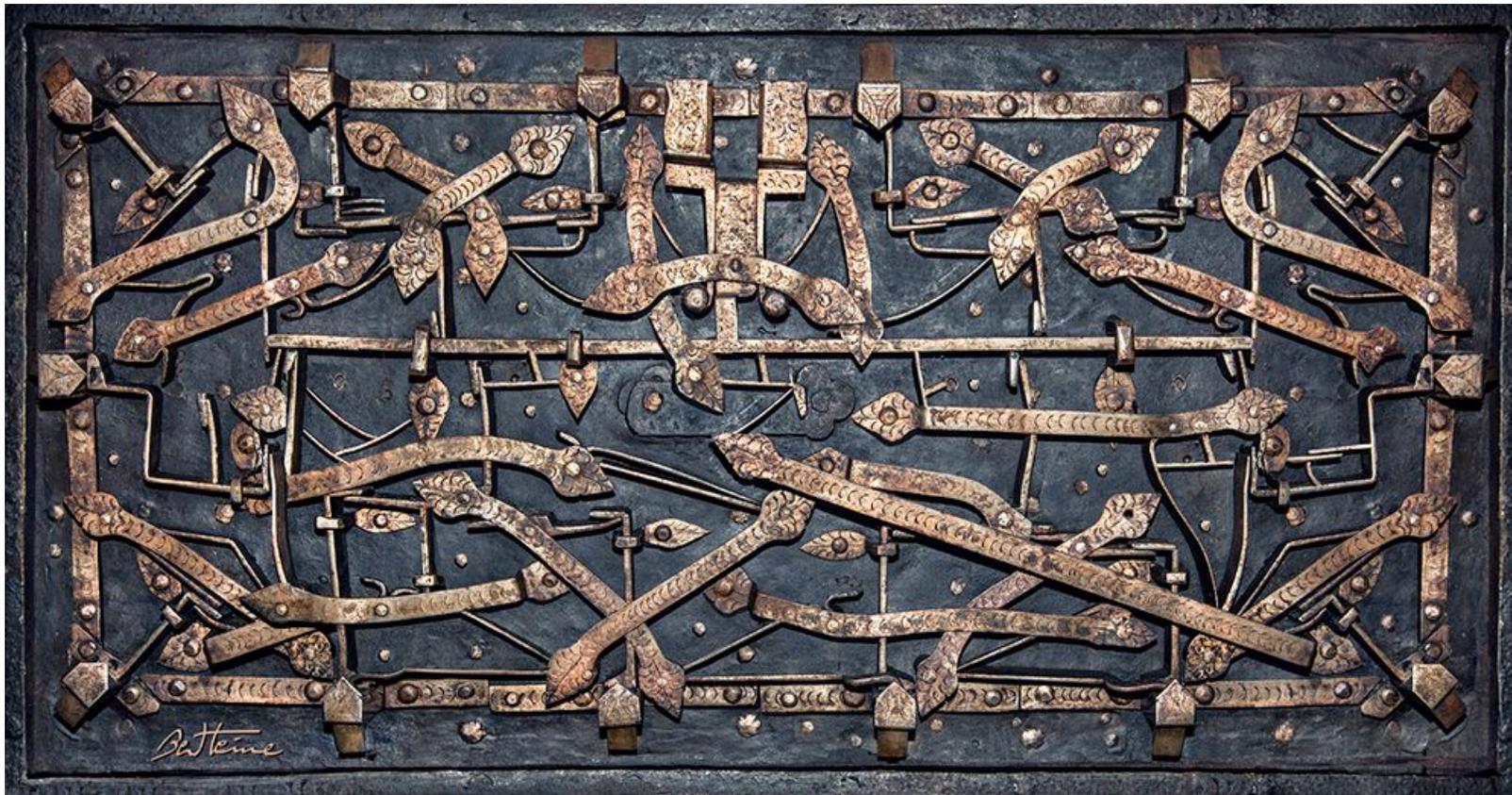


Università  
Ca' Foscari  
Venezia

# People think crypto look like this ...



... but it is more like this!



16th Century, Citadel of Dinant, Belgium.

Photo © 2016 [Ben Heine](#)



Università  
Ca' Foscari  
Venezia

# Cryptographic vulnerabilities

Crypto mechanisms **are not** equally secure

**Vulnerabilities in applications** can reveal keys or downgrade to less secure mechanisms

Improvements in **technology** and **cryptanalysis** require better crypto

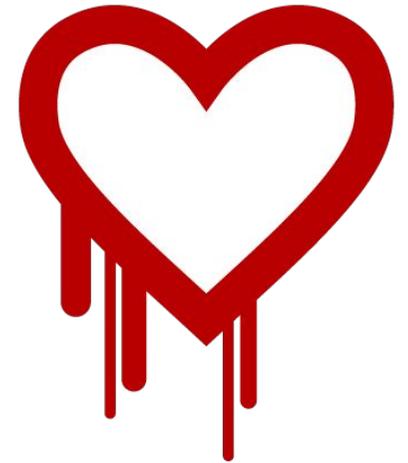
**The configuration and management** of cryptographic systems is complex and error prone

# Heartbleed

Vulnerability in OpenSSL, the protocol underneath https

An *over-read* allows for accessing process memory where **server keys** are stored

Once those keys are leaked it is possible to mount a **MITM attack** and intercept the whole Web session



<http://heartbleed.com/>

# Modes of operation

Needed when:

- Data is bigger than the block size
- We need to encrypt a stream with a block cipher

# Example: AES ECB

ECB is a mode of operation that splits long messages into blocks of 16 bytes (the size of AES block)

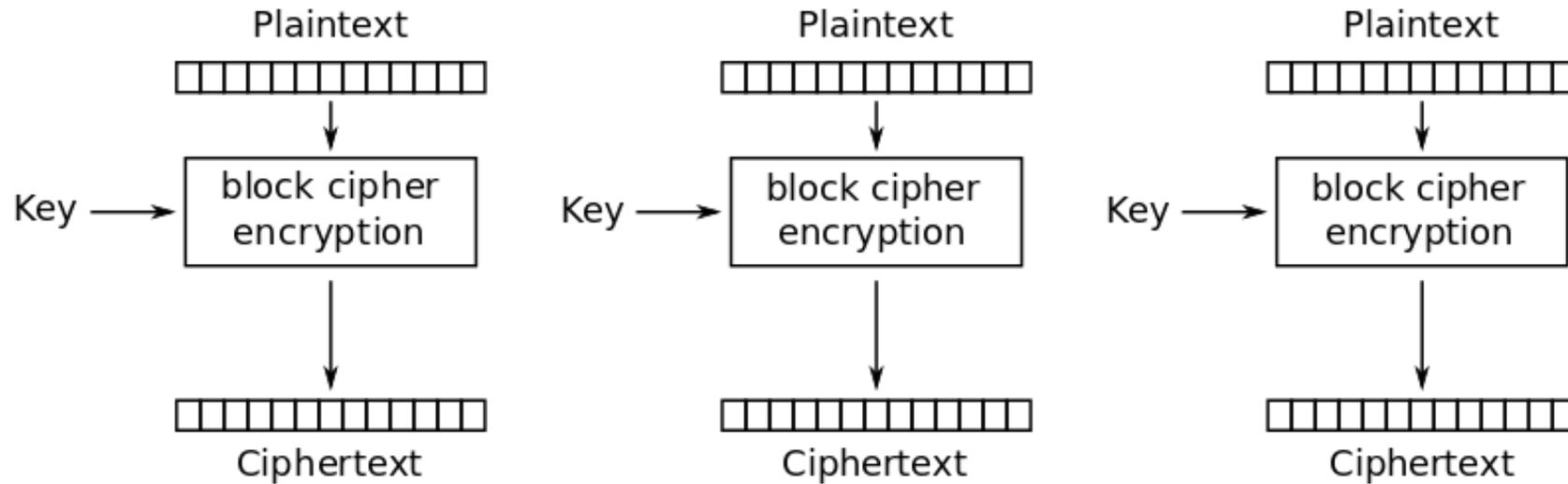
Blocks are encrypted **independently** under the same key

... not so different from substitution ciphers!

```
openssl enc -e -aes-128-ecb -K deccbe6da45d0d0fc57aad310d934ffe -in  
LogoBig-tail.ppm -out LogoBig-tail-enc.ppm
```



# ECB



Electronic Codebook (ECB) mode encryption

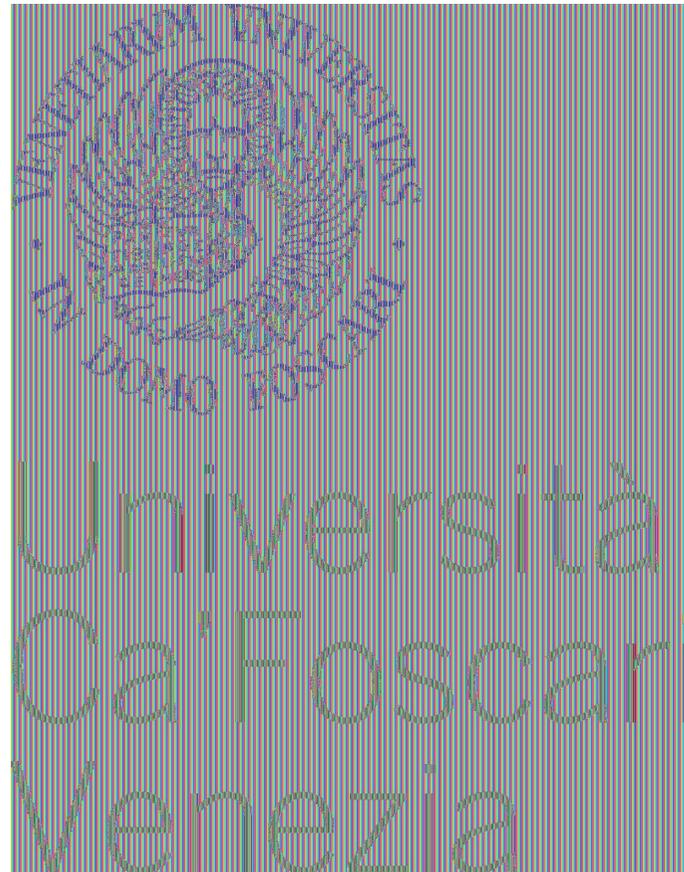
Poor confidentiality and integrity

# The unsatisfactory result



Università  
Ca' Foscari  
Venezia

plaintext



ciphertext



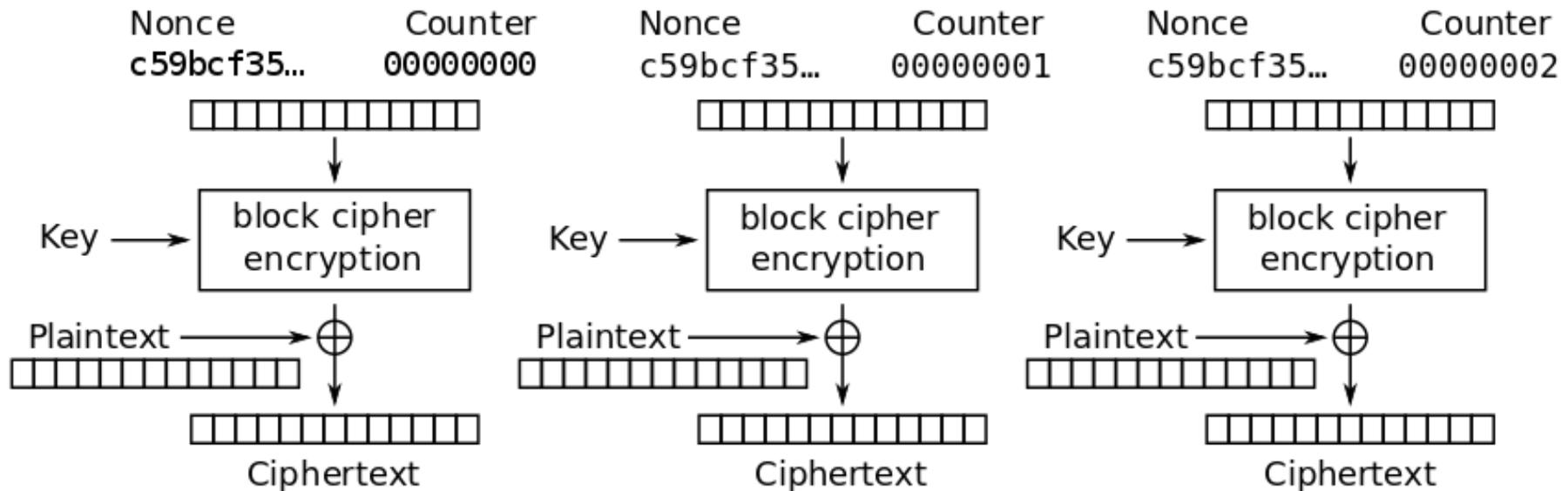
Università  
Ca' Foscari  
Venezia

# Chosen plaintext attack in ECB

If an attacker can prepend arbitrary prefix to the plaintext he can bruteforce blocks byte after byte

- prepend 15 known bytes
- bruteforce byte 16
- iterate over all bytes

# CTR: stream ciphers



Counter (CTR) mode encryption

Nonce is fundamental for security!



# Challenge

## ciphertext 1:

```
8f079a817d1dfa5bb2b1e069b0f4027abc65db6d130e6f3c154611d
165d66b0a23424734790df0769cc3c4f4f289e784ac0cc5cab7e47c
5c1a
```

## ciphertext 2:

```
9f0a92807d33fb1ab7a9ad36e5cd4064a320da7a56122e21004c42c
46d93214b28595b777612e46c9dc3c4eefedde88ee31c97c1b1e834
135c
```

## Leaked plaintext:

```
Dear Graham, I'll be happy to participate in the
training
```

**A CTR with fixed nonce has been used  
... how would you break the other ciphertext?**



# Solution

P1, P2 plaintexts and C1, C2 corresponding ciphertext

Same nonce means same key K

$$P1 \oplus K = C1$$

$$P2 \oplus K = C2$$

thus

$$P1 \oplus P2 = C1 \oplus C2$$

$$P2 = P1 \oplus C1 \oplus C2$$



# Padding oracle attacks

An attack that exploits **padding errors**

We have a padding oracle when

1. an application **exhibits padding errors** while decrypting a ciphertext
2. the **attacker can choose the ciphertext** (chosen ciphertext attack)

**Example:** key unwrapping in security devices

# PKCS5 / PKCS7 Padding

if we need 5 bytes we add 05 05 05 05 05

Possible paddings:

01

02 02

03 03 03

04 04 04 04

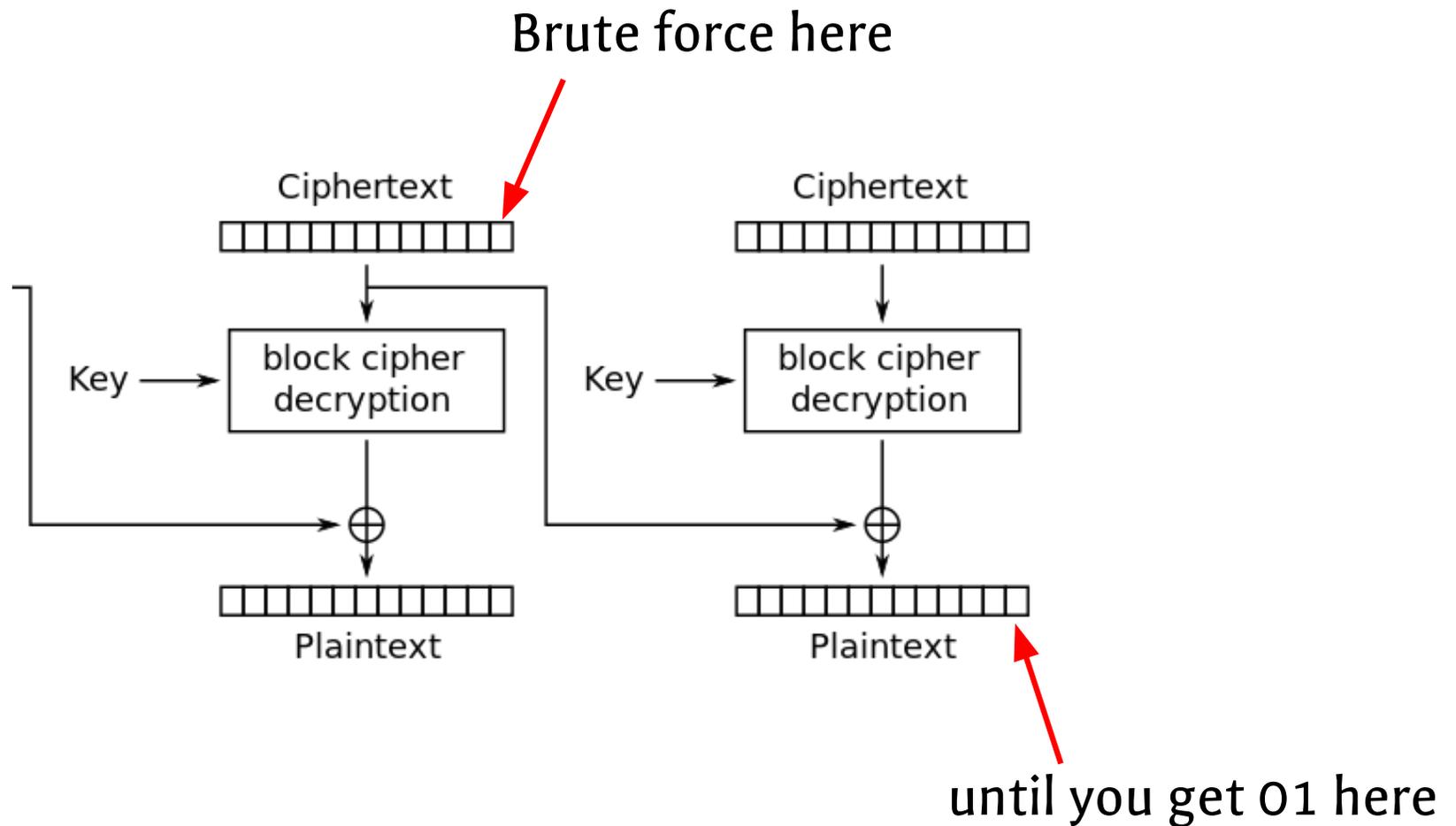
05 05 05 05 05

...

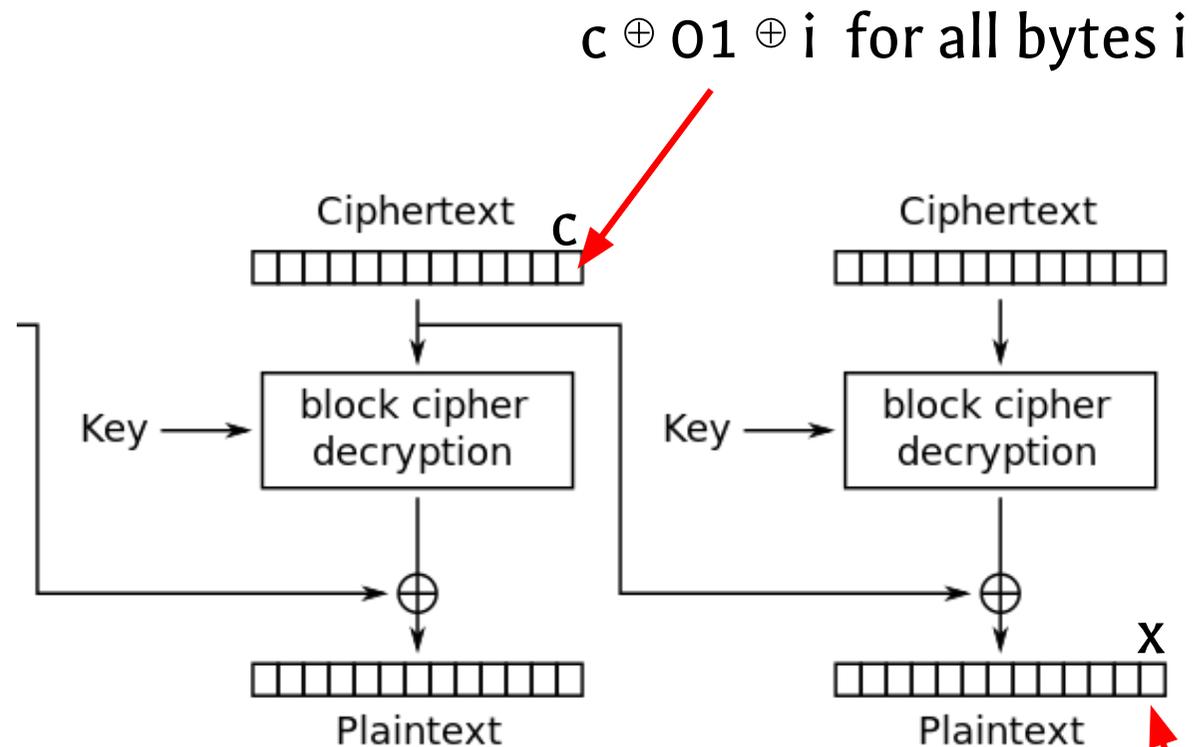
(PKCS5 is for 8 byte block size only)



# Padding oracle attack on CBC



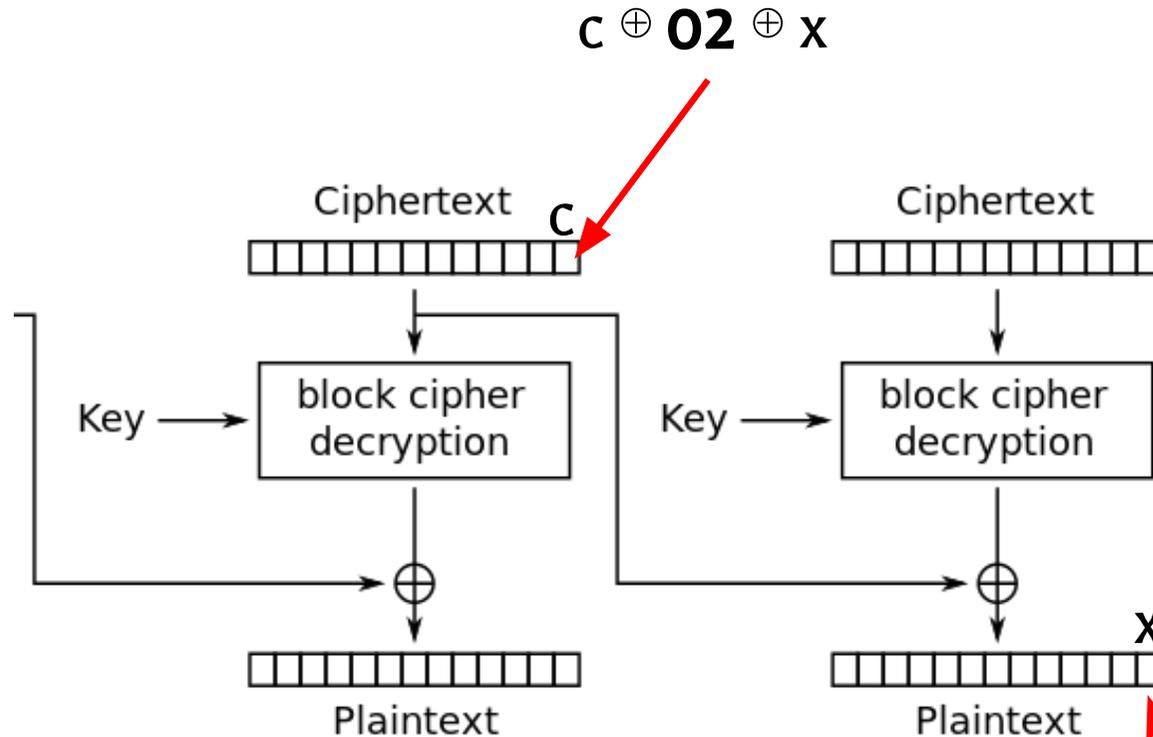
# Padding oracle attack on CBC



$01 == x \oplus 01 \oplus i$   
which implies  $x == i$ !



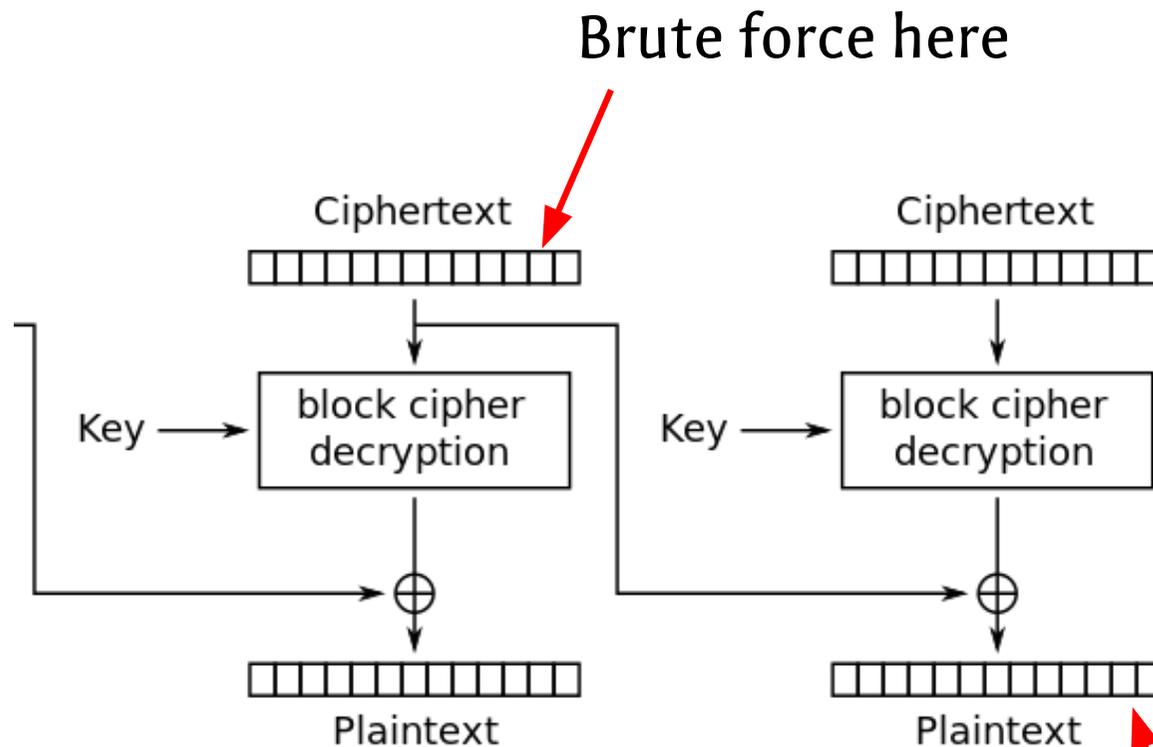
# Iterating for the next byte



We get **02** here



# Iterating for the next byte

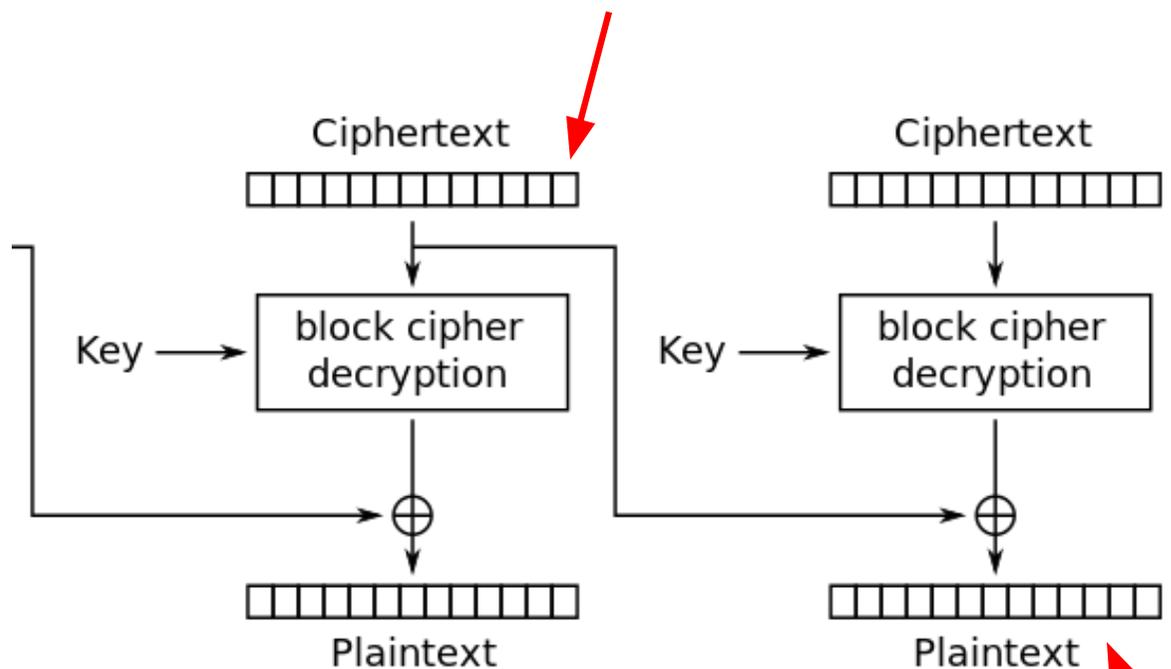


until you get 02 02 here



# What if it is correctly padded?

Brute force: we get two “yes” answers

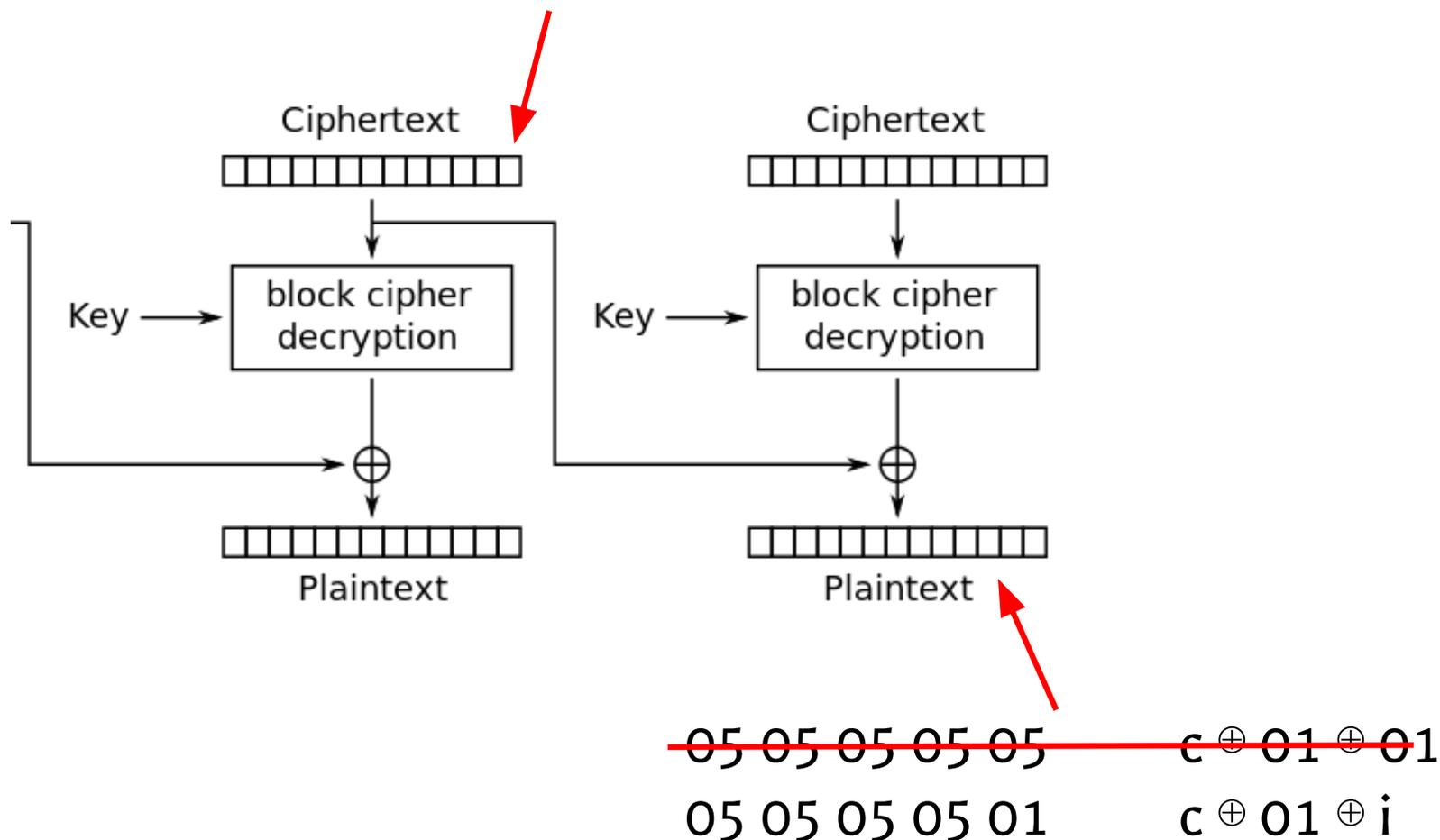


05 05 05 05 05  
05 05 05 05 01

$c \oplus 01 \oplus 01$   
 $c \oplus 01 \oplus i$

# What if it is correctly padded?

Brute force: we get two “yes” answers



# Key Management

## RSA SecurID Breach (March 2011)

- Seed values for devices **stored insecurely**, compromised after phishing breach.
- **40M devices replaced**, big companies breached, massive brand damage.



# Sophisticated attacks on crypto

May 2012, sophisticated attack on Iranian nuclear programme named **FLAME**

- **A fake certificate** using an *MD5* collision was used to install the malware, bypassing software update check
- The *MD5* collision method used was **different from the one publicly known**

# Conclusion

Cryptography is complex

Cyber criminals use sophisticated attacks against crypto

It is important to understand what security level is provided by the various mechanisms

Have a look at [www.cryptosense.com](http://www.cryptosense.com) !