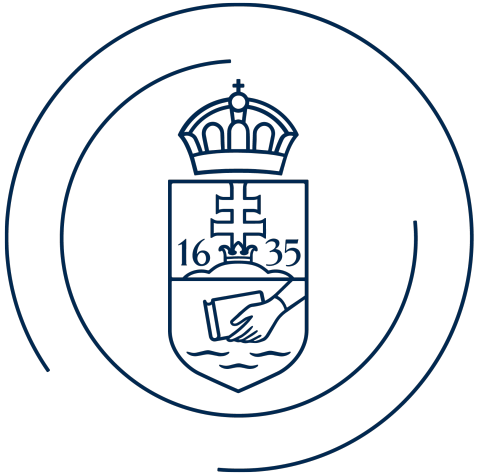


ELTE
EÖTVÖS LORÁND
UNIVERSITY

Cryptography and security -Practice (IPM-18sztKVSZKRBG)

Dr. Norbert Tihanyi

OSCP, OSCE, OSWP, CRTP, CEH, ECES , ISO27001 LA

ntihanyi@inf.elte.hu

@TihanyiNorbert

Minimum requirements

Minimum requirements:

- There will be **5** different **assingments** in this semester.
- Each assignment worth 20 points.
- You have to collect **at least 5 points in all assignments**
- The minimum requirement is **60 points overall**
- Always send your solution to **ntihanyi@inf.elte.hu**
- Attendance is mandatory

Definitions

Definition (Cryptology)

The branch of mathematics encompassing both cryptography and cryptanalysis is **cryptology**

Definition (Cryptography)

Cryptography is the art and science of keeping messages secure.

Definition (Cryptanalysis)

Cryptanalysis is the art and science of breaking ciphertexts.

Definitions

Definition (**Encryption**)

Encryption is the process of converting data (called plaintext) to an unrecognizable or unintelligible (called ciphertext) form.

Definition (**Decryption**)

The conversion of encrypted data into its original form is called **decryption** . It is generally a reverse process of encryption.

Definitions

Definition (Plaintext)

Ordinary readable text before being encrypted into ciphertext or after being decrypted.

Definition (Cipher)

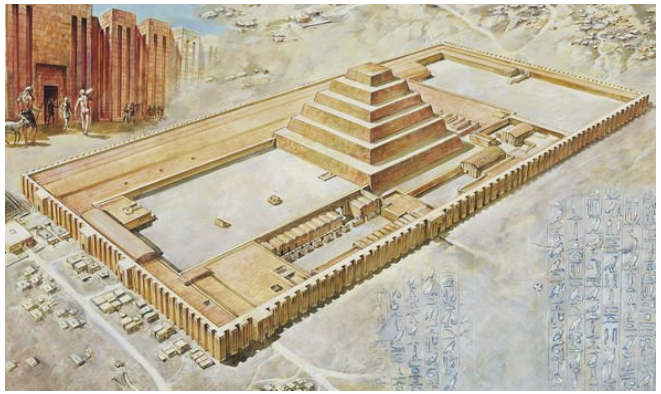
A cryptographic algorithm for encryption and decryption.

Definition (Ciphertext)

Ciphertext is the encrypted form of the message being sent.

Security by obscurity

Security by obscurity is the reliance in security engineering on design or implementation secrecy as the main method of providing security to a system or component. **We try to avoid this concept**. You do not have to kill the designer of an encryption algorithm



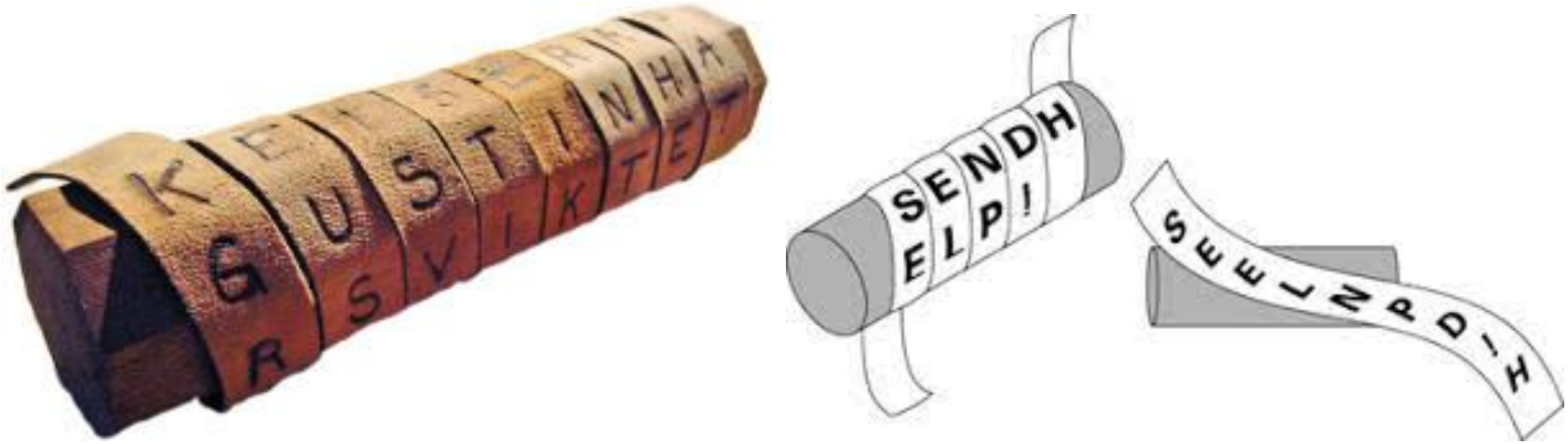
Definition (Kerckhoff's principle -1883)

Kerckhoff's principle is the concept that a cryptographic system should be designed to be secure, even if all its details, except for the key, are publicly known.

Module 0x01
Historical ciphers

Scytale

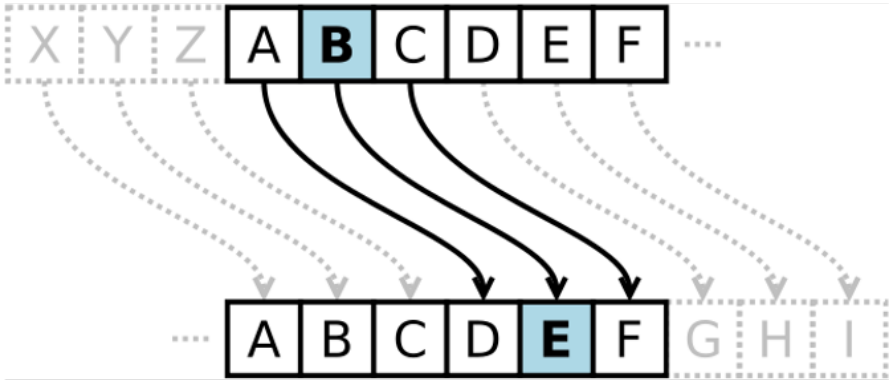
Scytale is a tool used to perform a transposition cipher, consisting of a cylinder with a strip of parchment wound around it on which is written a message.



The recipient uses a rod of the **same diameter** on which the parchment is wrapped to read the message.

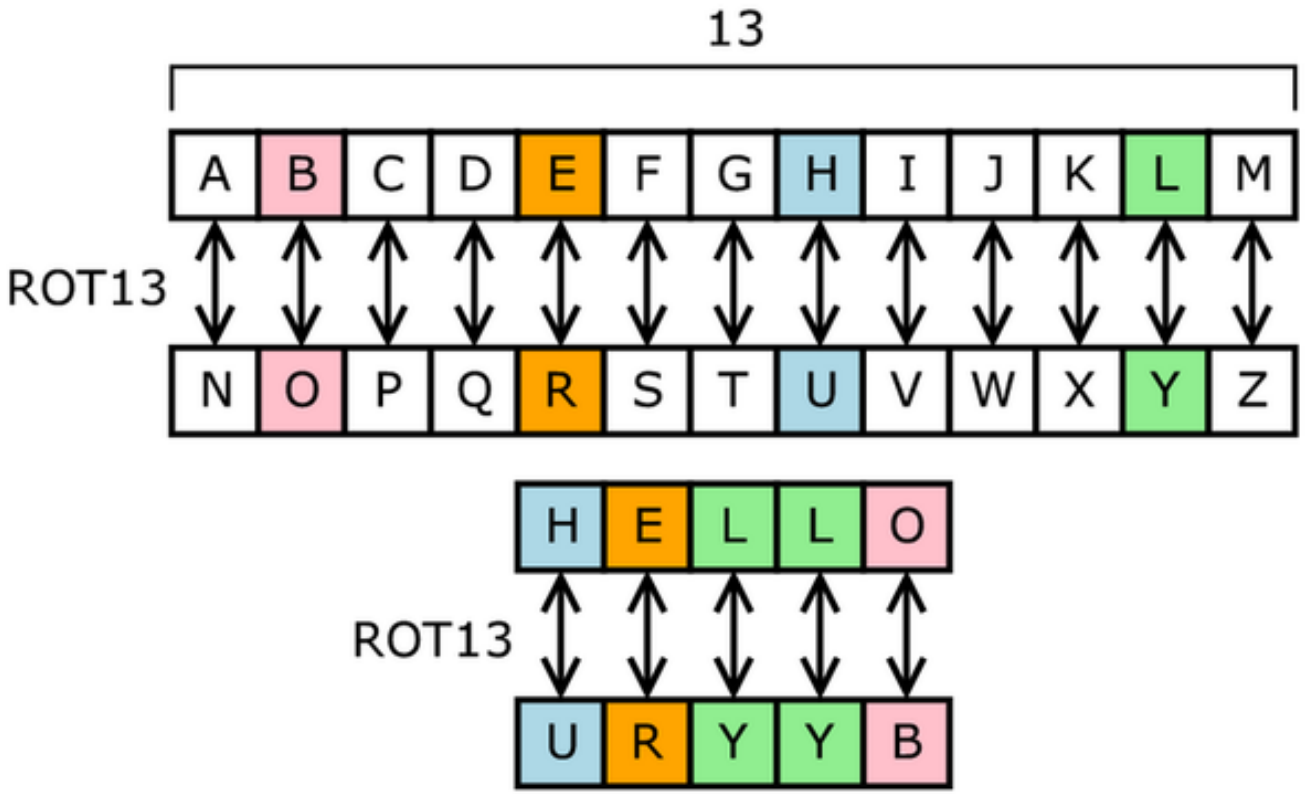
Caesar cipher

- Caesar's cipher, is one of the simplest and most widely known encryption techniques.
- The method is named after Julius Caesar, who used it in his private correspondence.
- It is a type of **substitution cipher** in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet.



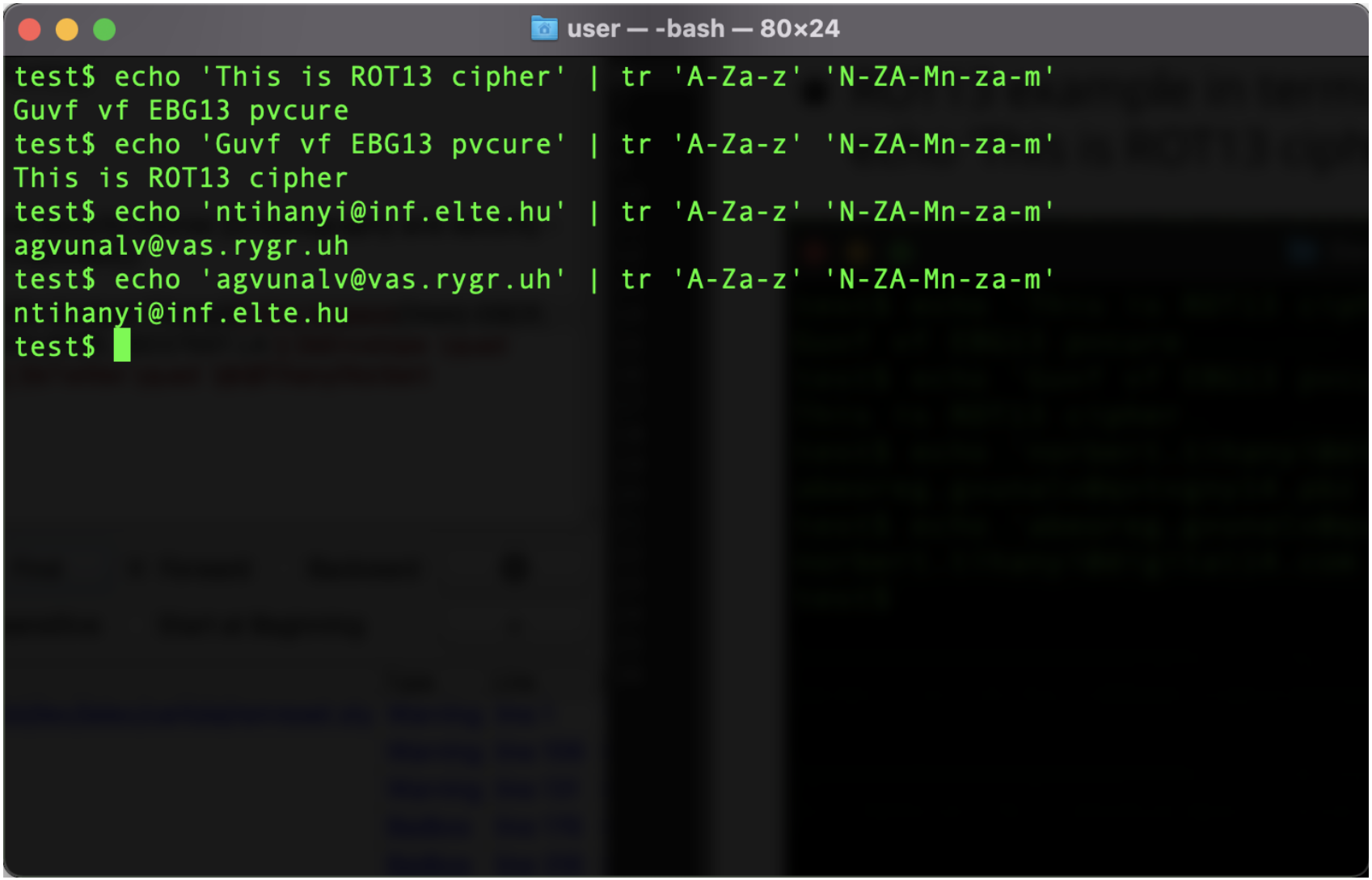
Special Caesar cipher: ROT13

- ROT13 is a special case of the Caesar cipher
- ROT13 is a simple letter substitution cipher that replaces a letter with the 13th letter after it in the alphabet.
- Well known example for a **very weak cipher**



Special Caesar cipher: ROT13

- ROT13 example in terminal:
  echo 'This is ROT13 cipher' | tr 'A-Za-z' 'N-ZA-Mn-za-m'



Challenge 1 - Cracking Substitution cipher

Substitution cipher with unknown shift: Ch nby niqh qbyly C qum vilh Fcpyx u guh qbi mucfyx ni myu Uhx by nifx om iz bcm fczy Ch nby fuhx iz movgulchym Mi qy mucfyx oj ni nby moh Ncff qy ziohx u myu iz alyyh Uhx qy fcpyx vyhyunb nby qupym Ch iol syffiq movgulchy Qy uff fcpy ch u syffiq movgulchy Syffiq movgulchy, syffiq movgulchy Qy uff fcpy ch u syffiq movgulchy Syffiq movgulchy, syffiq movgulchy Uhx iol zlcyhxm uly uff uviulx Guhs gily iz nbyg fcpy hyrn xiil Uhx nby vuhx vyachm ni jfus Qy uff fcpy ch u syffiq movgulchy Syffiq movgulchy, syffiq movgulchy Qy uff fcpy ch u syffiq movgulchy Syffiq movgulchy, syffiq movgulchy

Do you recognize this????

Etaoin shrdlu

Do you recognize this????

Etaoin shrdlu

The most common letters in the English alphabet.

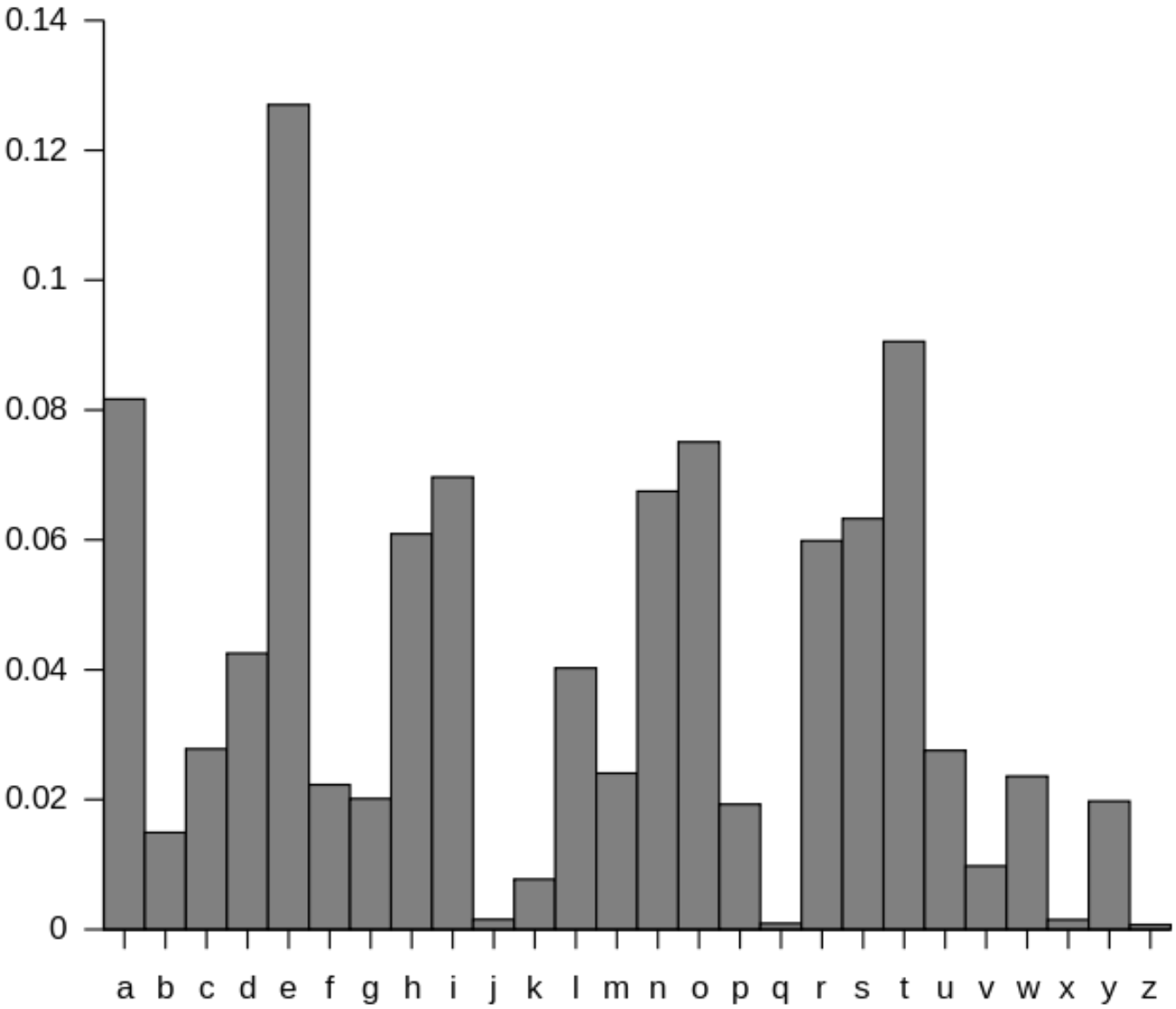
Frequency Analysis

Definition (Frequency Analysis)

Frequency analysis is the study of the frequency of letters or groups of letters in a ciphertext.

- Frequency analysis is based on the fact that, in any given stretch of written language, certain letters and combinations of letters occur with varying frequencies
- In the English language **E**, **T**, **A** and **O** are the most common, while **Z** **Q**, **X** and **J** are rare.

Frequency Analysis



Cracking substitution ciphers

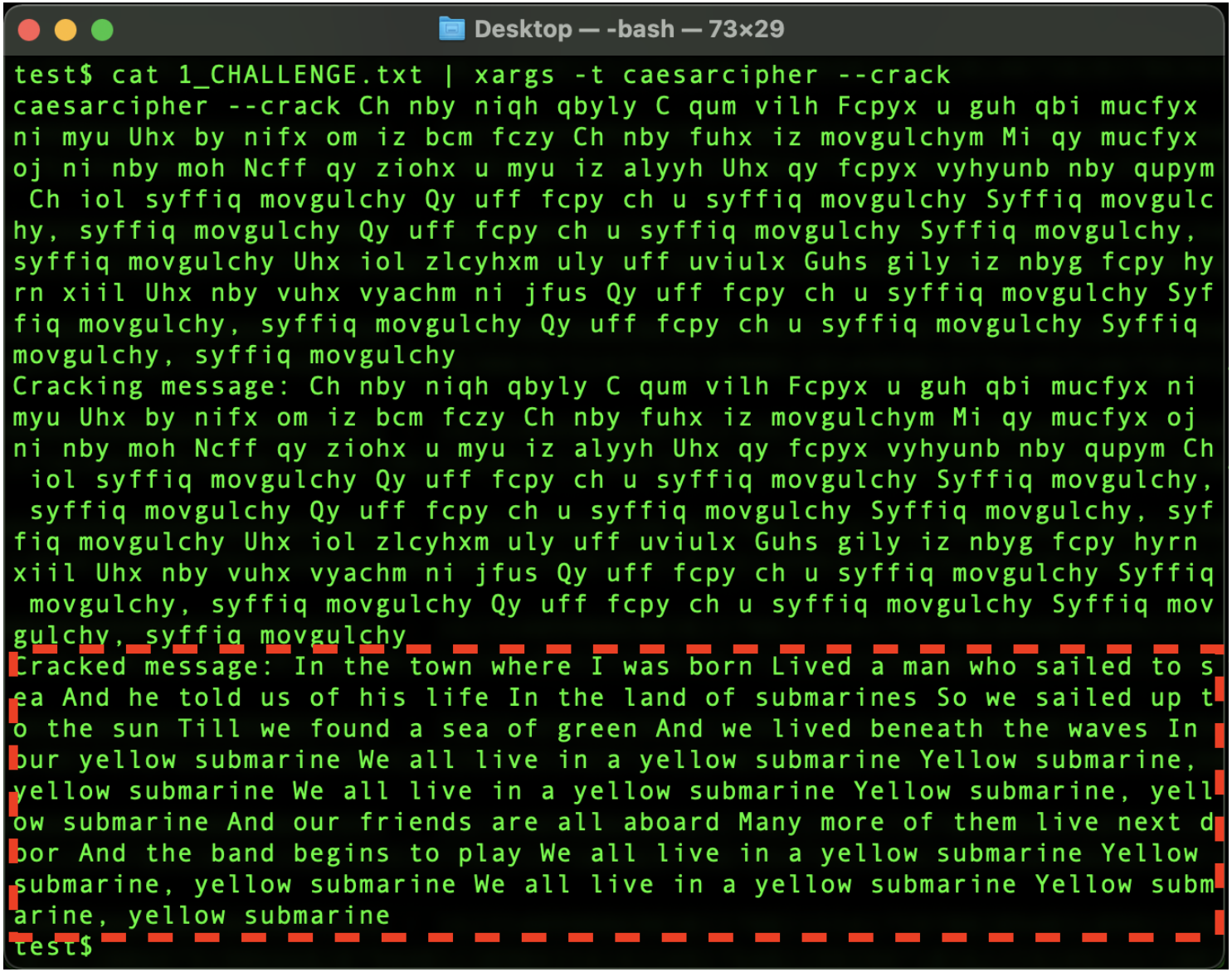
Method1:

- pip install caesarcipher
- **Encoding message:** echo ”test” | xargs -t caesarcipher –encode
- **Encoding File:** cat cleartext.txt | xargs -t caesarcipher –encode
- **Cracking:** cat encrypted.txt | xargs caesarcipher –crack

Method2:

- `https://www.dcode.fr/caesar-cipher`

Cracking Challenge 1



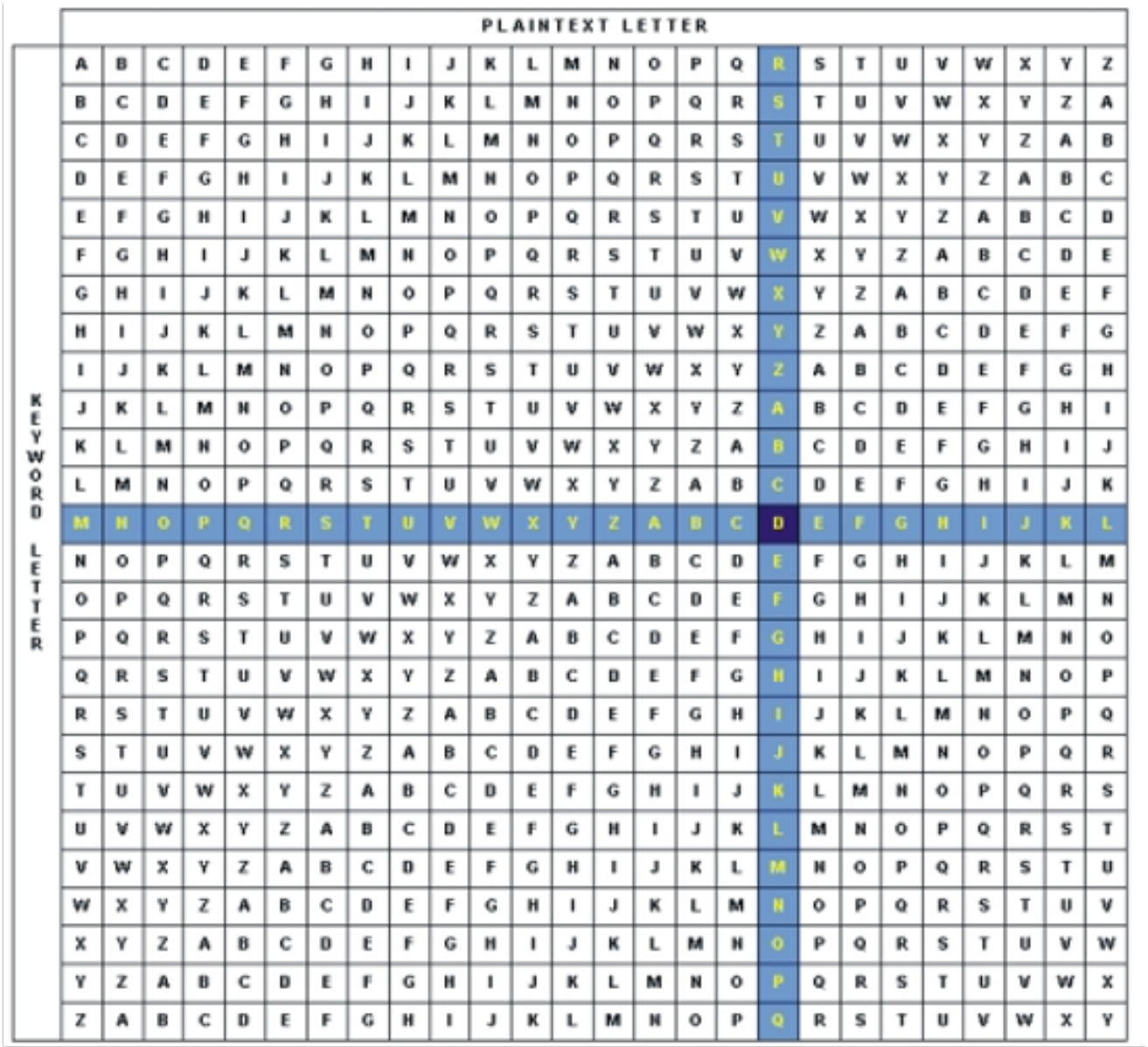
Vigenère cipher

Definition (Vigenère cipher)

The Vigenère cipher is a method of encrypting alphabetic text by using a series of different Caesar ciphers, based on the letters of a keyword.

- It is polyalphabetic substitution cipher.
- First described by Giovan Battista Bellaso in 1553
- Can not be cracked by a simple frequency analysis

Vigenère cipher



KEYWORD LETTER	PLAINTEXT LETTER																									
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Example for Vigenère cipher

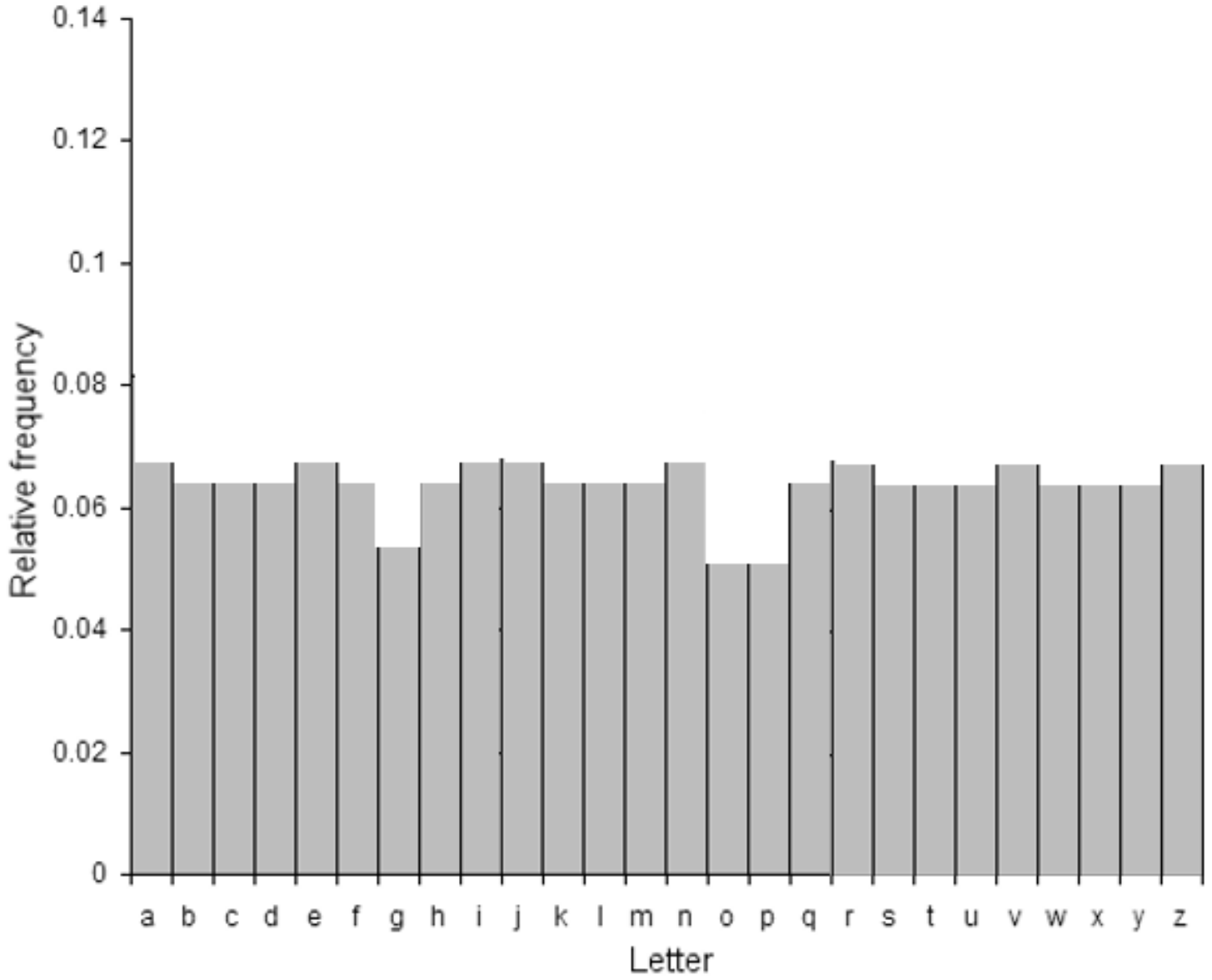
The first secret key is: **CARROT**.
The second secret key is: **SECRET**.
The third secret key is: **ITISAVERYLONGKEY**.

PLAINTEXT : MY NAME IS NORBERT TIHANYI
SECRET KEY : CA RROT CA RROTCAR ROTCARR
CIPHERTEXT: OY ERAX KS EFFUGRK KWACNPZ

PLAINTEXT : MY NAME IS NORBERT TIHANYI
SECRET KEY : SE CRET SE CRETSEC RETSECR
CIPHERTEXT: EC PRQX AW PFVUWVV KMASRAZ

PLAINTEXT : MY NAME IS NORBERT TIHANYI
SECRET KEY : IT ISAV ER YLONGKE YITISAV
CIPHERTEXT: UR VSMZ MJ LZFOKBX RQAIFYD

Frequency Analysis is not working



Challenge 2 - Cracking the Vigenère cipher

Find the Secret key of this Vigenère cipher: lo nkl tpen xbhye J eat vryn Mqvfx d tao ehp mdplfl tp mhh Aol hf nrsd va og blz ljne Jh woe mine ii zucuascqls Tw wf mdplfl uq nr ahf auo Nlsl xm fpoqk a tma pz jyefv Aox zl ljdee vhuebbh ubh dawms Jh rbr zmlmiz zucuascql Wf ilm flce jv a zyosox aucgdyiom Yffovw tcbnuupnf, gemfrd svjmbllue Xm amf opvf qn b shslpe svvphrjve Zyosox aucgdyiom, yffovw tcbnuupnf Ine ixy fsqeoxv hrf ilm uevasl Mbhb tosm og nklm mqvf hhet ewos Uqk tim bbhg iehqnt nr wlbg Wf uos ljde jh d femtox mximbzioy Bllmww toetasqnf, shslpe svvphrjve Xy dsl mqvf cq h yftlpq vbbnirjhh Femtox mximbzioy, bllmww toetasqnf

Cracking Vigenère ciphers

- In 1863, Friedrich **Kasiski** was the first to publish a general method of deciphering Vigenère ciphers.
- Kasiski method, takes advantage of the fact that repeated words are, by chance, sometimes encrypted using the same key letters, leading to repeated groups in the ciphertext.

Example with secret key: **AZAZ**

```
————————————————————
PLAINTEXT : NORBERT TIHANYI NORBERT TIHANYI
SECRET KEY : AZAZAZA ZAZAZAZ AZAZAZA ZAZAZAZ
CIPHERTEXT: NNRAEQT SIGAMYH NNRAEQT SIGAMYH
————————————————————
```

- `https://www.dcode.fr/vigenere-cipher`

Challenge 2 - Cracking the Vigenère cipher

Do a frequency analysis for every n.th position of the text:
l. ... ...**n** ....**a**. .... ..**v**fx d tao e**h**p mdplfl tp mhh Aol hf nrsd va og blz ljne Jh woe mine ii zucuascqls Tw wf mdplfl uq nr ahf auo Nlsl xm fpoqk a tma pz jyefv Aox zl ljdee vhuebbh ubh dawms Jh rbr zmlmiz zucuascql Wf ilm flce jv a zyosox aucgdyiom Yffovw tcbnuupnf, gemfrd svjmbllue Xm amf opvf qn b shslpe svvphrjve Zyosox aucgdyiom, yffovw tcbnuupnf Ine ixy fsqeoxv hrf ilm uevasl Mbhb tosm og nklm mqvf hhet ewos Uqk tim bbhg iehqnt nr wlbg Wf uos ljde jh d femtox mximbzioy Bllmww toetasqnf, shslpe svvphrjve Xy dsl mqvf cq h yftlpq vbbnirjhh Femtox mximbzioy, bllmww toetasqnf

Vernam Cipher

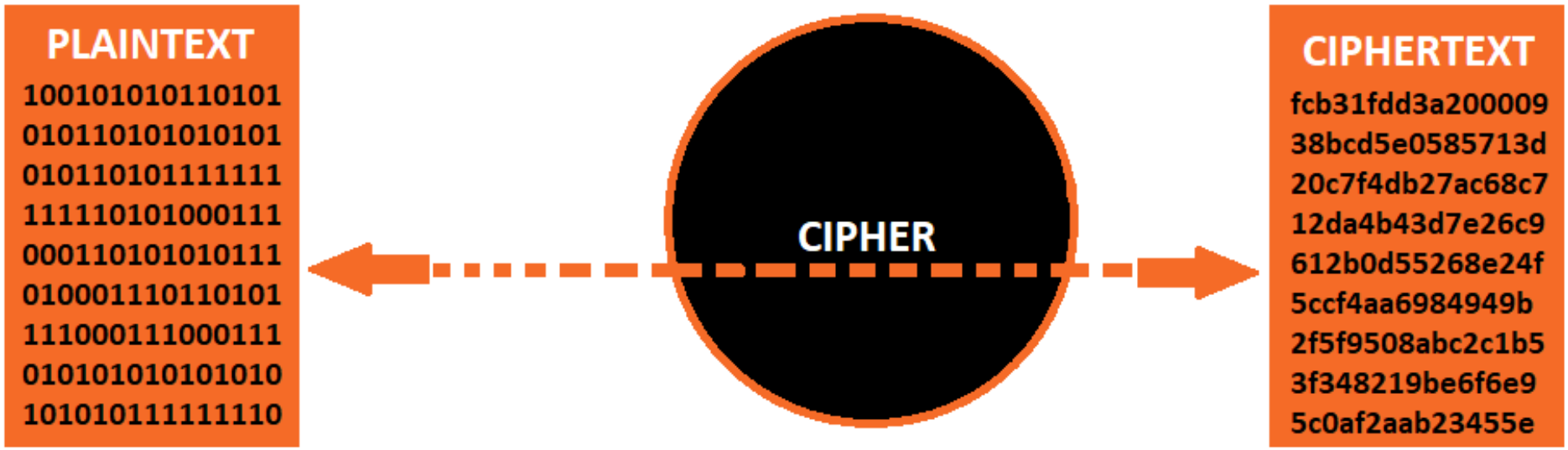
- Invented in 1917 by Gilbert S. Vernam
- It is a polyalphabetic stream cipher.
- The key length is the same size as the message.
- In modern terminology, a Vernam cipher is a symmetrical stream cipher in which the plaintext is combined with a random or pseudorandom stream of data (the ”keystream”) of the same length, to generate the ciphertext, using the Boolean ”exclusive or” (XOR) function

Perfect Secrecy

Is there a perfect secrecy?

Perfect means: The encryption scheme is unbreakable with

- unlimited time
- unlimited computational power



Perfect Secrecy

Definition (Perfect Secrecy)

A cryptosystem has perfect secrecy if for any message x and any encipherment y

$$P(x|y) = P(x) \tag{1}$$

In other words, the message gives the adversary precisely no information about the message contents.

OTP- The unbreakable encryption

In **1949 Claude Shannon** proved that there is a **perfect** and **unbreakable** secrecy. This encryption technique is the so-called OTP (One Time Pad, successor of Vernam cipher). Shannon proved that OTP encryption is unbreakable if and only if:

- **plaintext and secret key is of the same size**
- the secret key has been used only once
- it is assumed that the attacker has no other information about the key
- secret key has the same character set than the plaintext
- **the secret key is true random**

NOTE: In its original form, Vernam's system was vulnerable because the key tape was a loop, which was reused whenever the loop made a full cycle.

Example for Perfect encryption

Different random keys generate the same ciphertext.
Observing only the ciphertext, there is no additional information about the plaintext.

PLAINTEXT : 0xFFFFFFFF
SECRET KEY: 0x34627362 ← random key
CIPHERTEXT: 0xC59D1C9F ← ciphertext

———————————————-

PLAINTEXT : 0x11111111
SECRET KEY: 0xD48C0D8E ← random key
CIPHERTEXT: 0xC59D1C9F ← same ciphertext

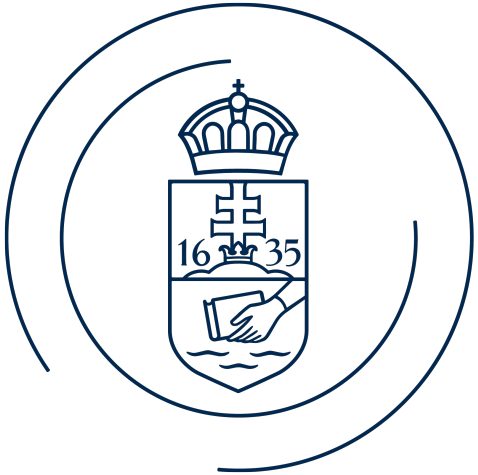
———————————————-

PLAINTEXT : 0xA0A0A0A0
SECRET KEY: 0x653DBC3F ← random key
CIPHERTEXT: 0xC59D1C9F ← same ciphertext

Thank you for your attention!

ntihanyi@inf.elte.hu

@TihanyiNorbert



ELTE
EÖTVÖS LORÁND
UNIVERSITY

Cryptography and security -Practice (IPM-18sztKVSZKRBG)

Dr. Norbert Tihanyi

OSCP, OSCE, OSWP, CRTP, CEH, ECES , ISO27001 LA

ntihanyi@inf.elte.hu

@TihanyiNorbert

Module 0x02
Random Number Generators

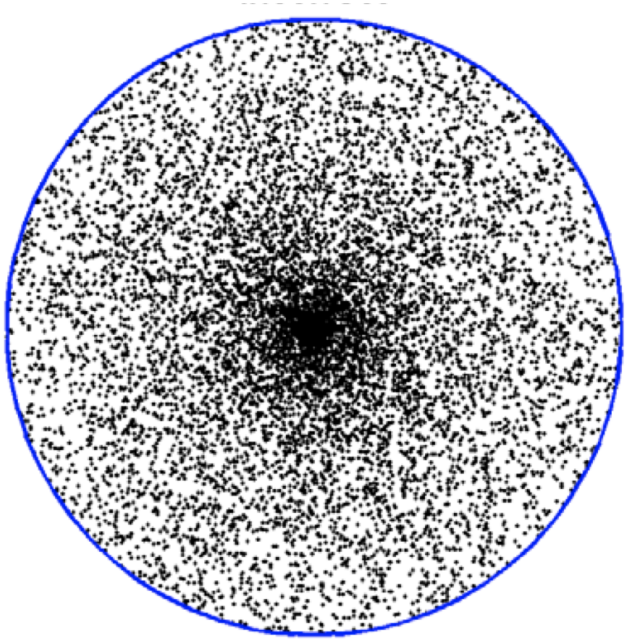
We need true randomness to achieve perfect secrecy.

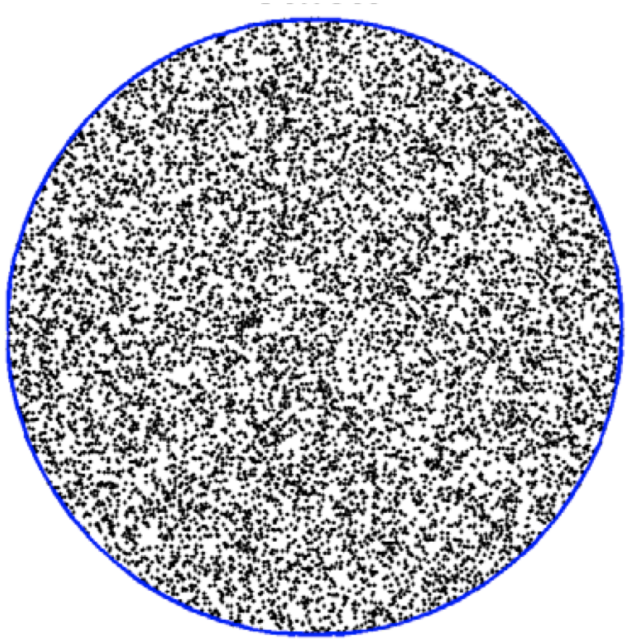
- Can we produce **random numbers** by algorithms?
- Can a computer generate **truly random numbers**?
- Is there any true randomness in physics?

Applications

- Session keys in a Web browser
- Encryption keys (e.g: AES, RSA)
- Lottery drawings
- Online Games
- PIN codes
- OTP tokens

Randomness





Pseudo Random Number Generators

What is a **pseudorandom number generator (PRNG)?**

Definition (PRNG)

$$G : \{0,1\}^k \rightarrow \{0,1\}^n \tag{1}$$

with the following properties:

- $n \gg k$
- G(x) computationally indistinguishable from true random
- **the values are uniformly distributed over a defined interval**
- it is impossible to predict future values based on past or present ones
- there are no correlations between successive numbers

One of the most important usage of random numbers is key generation methods for cryptographic purposes.

Probability Distribution

Definition

A **probability distribution** is a mathematical function that provides the probabilities of occurrence of different possible outcomes in an experiment.

We can pick randomly according to any "probability distribution":

$$\mathcal{P} : S \to \mathcal{R} \tag{2}$$

The probability distribution $\mathcal{P}$ assigns a nonnegative probability to each $x \in S$, such that:

$$\sum_{x \in S} \mathcal{P}(x) = 1 \tag{3}$$

Uniform Distribution

What if we want to pick something **at random**?

Definition

The uniform distribution $\mathcal{U}$ is the probability distribution where everything is picked equally often:

$$\text{If } |S| = n, \text{ then } Pr[x = s] = \frac{1}{n} \quad \forall s \in_{\mathcal{U}} S.$$

Statistical Distance (L^1-distance)

Definition (Statistical Distance)

The statistical distance (L^1-distance) between two probability distributions $\mathcal{P}$ and $\mathcal{E}$ is:

$$d(\mathcal{P}, \mathcal{E}) = \frac{1}{2} \left| \sum_{x \in S} \mathcal{P}(x) - \mathcal{E}(x) \right| \tag{4}$$

One can measure how close a distribution is to the uniform distribution $\mathcal{U}$. If $d(\mathcal{P}, \mathcal{U}) \leq \epsilon$, then we say $\mathcal{P}$ is ϵ-close to **uniform**.

Shannon Entropy

Definition (Entropy)

The entropy is a basic quantity in information theory associated to any random variable, which can be interpreted as the average level of ”information”, or ”uncertainty” inherent in the variable’s possible outcomes.

Definition (Shannon Entropy)

Given a random variable X with possible outcomes x_i, each with probability $P(x_i)$, the entropy $H(X)$ of X can be defined as follows:

$$H(X) = -\sum_{x \in i} P(x_i) \log P(x_i) \tag{5}$$

Coin Toss Example

- Considering a fair coin toss, how many bits of entropy would this contain?
- Such a coin toss has **one bit** of entropy since there are two possible outcomes that occur with equal probability, and learning the actual outcome contains one bit of information.
- In contrast, a coin toss using a coin that has two heads and no tails has zero entropy since the coin will always come up heads, and the outcome can be predicted perfectly.
- However, when biased coins are Introduced things get interesting: there is **less than 1 bit of entropy**

Is there any true random number sequences?

Random sequences (s_1, s_2, s_3, s_4 are generated by coin toss.
Which sequence is random?

- s_1=11111111111111111111111111111111111111
- s_2=11111111111111111111000000000000000000
- s_3=10101010101010101010101010101010101010
- s_4=10001110101100001110101010111000111011111 0

Mathematical definition of randomness

What is the definition of randomness? The mathematical description of randomness was given by Andrey Nikolaevich Kolmogorov in 1963.

Theorem

Kolmogorov-Chaitin (KC) complexity: *Let $d(s)$ denote the minimal description of s in an L universal language. The length of $d(s)$ is the Kolmogorov complexity of s. $KC(s) = |d(s)|$*

Theorem

A string is random if it is impossible to write down shorter than the length of the string itself. ***A random string can not be compressed.***

Is there any true random number sequences?

- s_1=11111111111111111111111111111111111111

  s_1 has a short English-language description, namely ”1 42 times” which consists 10 character. We were able to compress s_1, so it is not random.

- s_4=10001110101100001110101010111000111011111 0

  s_4 **seems** to be random and can not be compressed.

KC is uncomputable function.

Theorem

There is no program that can compute KC(s) for any s.

Let **function** KolmogorovComplexity(string s) which takes as input a string s and returns KC(s). Assume that the length of the following algorithm is 100.000 bits.

```
for i = 1 → ∞ do
    for each all string s of length i do
        if KolmogorovComplexity(s)>= 200.000 then
            return s
        end if
    end for
end for
```

No such program exists: **This is a CONTRADICTION!**

Can we produce true random numbers by a deterministic algorithm?

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin"- John von Neumann

Corollary

There is no any mathematical algorithm that can produce true randomness. *Random numbers generated by computers are not truly random, and we call them* ***pseudo random numbers****.*

Next-bit test

Theorem

*A nonnegative function $f : \mathbb{N} \to \mathbb{R}$ is **negligible** if, for any positive polynomial p, $f \in o(1/p)$.*

Theorem

***Next-bit test:** A sequence of bits passes the next bit test for at any position i in the sequence, if an adversary knows the i first bits, and he cannot predict the $(i+1)$. with probability of success better than $50\% + \epsilon$.*

CSPRNG and Yao's theorem

Criteria for Cryptographically-Secure Pseudo random number generators (CSPRNG):

- pass statistical randomness tests
- resist against well-known cryptographic attacks
- Every CSPRNG should satisfy the next-bit test.

In 1982 Andrew Yao proved the following theorem:

Theorem

Yao's theorem: *If a generator passing the next-bit test will pass all other polynomial-time statistical tests for randomness.*

Theoretical Statistical tests for randomness

Let $E_N = \{e_1, e_2, \ldots, e_N\} \in \{-1, +1\}^N$ represent a finite pseudorandom binary sequence. Sárközy and Mauduit defined the following functions:
The **well-distribution measure** of E_N is defined by

$$W(E_N) = \max_{M,u,v} |U(E_N, M, u, v)| = \max_{M,u,v} \left| \sum_{j=0}^{M-1} e_{u+jv} \right|$$

where the maximum is taken over all M, u, v with $u + (M-1)v \leq N$.

Theoretical Statistical tests for randomness

The **correlation measure of order** k of E_N is defined by

$$C_k(E_N) = \max_{M,D} |V(E_N, M, D)| = \max_{M,D} \left| \sum_{n=1}^{M} e_{n+d_1} \, e_{n+d_2} \ldots e_{n+d_k} \right|$$

where the maximum is taken over all M and $D = (d_1, \ldots, d_k)$ such that $0 \leq d_1 \leq \cdots \leq d_k \leq N - M$.
The goodness of a PRNG is determined by the order of $W(E_N)$ and $C_k(E_N)$.

Statistical tests: NIST and DIEHARD tests

Theoritical proofs can not be used to measure the quality of generated numbers. Statistical tests are needed.

NIST-test: The NIST Test Suite is a statistical package consisting 15 different statistical tests to detect non-randomness in binary sequences produced by pseudorandom or random number generators utilized in cryptographic applications. The package is developed by the National Institute of Standards and Technology (NIST).

DIEHARD test: The DIEHARD test suite is a package of statistical tests for measuring the quality of a random number generators. It was implemented by George Marsalgia and published in 1995.

NIST SP 800-22 test evaluation

PRNG output:

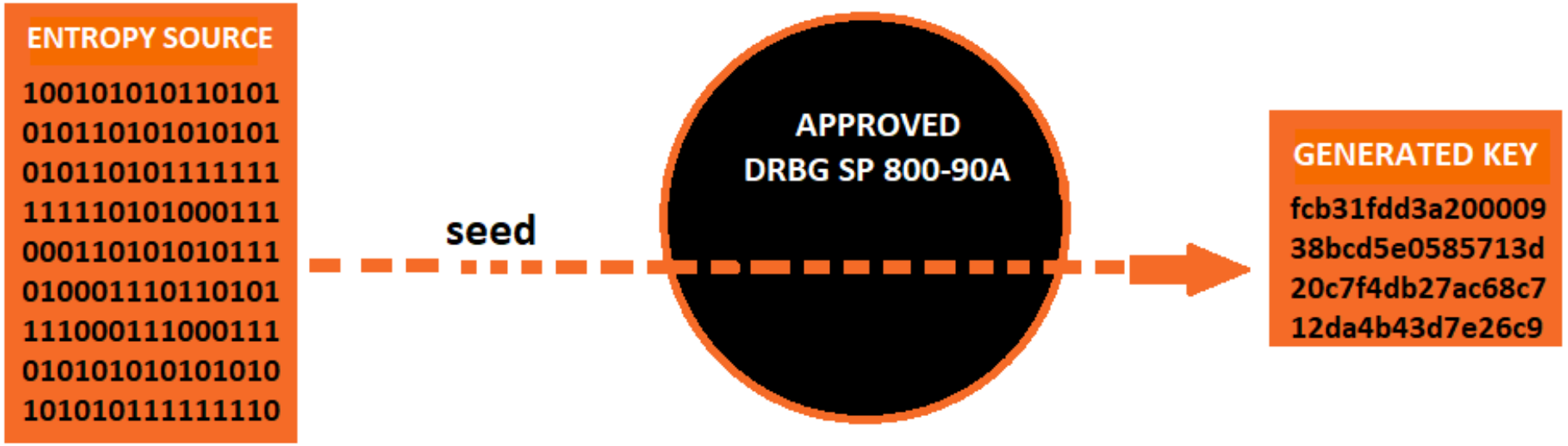
C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-value	Prop	Test name
24	34	30	31	25	30	26	33	30	37	0.828458	297/300	Frequency
34	29	32	27	21	26	20	35	31	45	0.068287	296/300	BlockFrequency
29	32	36	31	31	24	31	27	29	30	0.964295	298/300	CumulativeSums
26	30	31	30	33	27	24	38	28	33	0.840081	295/300	CumulativeSums
32	22	25	35	36	25	32	30	41	22	0.198690	300/300	Runs
34	34	34	31	37	23	33	23	29	22	0.437274	298/300	LongestRun
22	26	23	35	35	32	34	39	29	25	0.334538	296/300	Rank
33	28	31	29	33	30	34	31	25	26	0.973936	296/300	FFT
30	30	21	35	40	29	29	28	25	33	0.514124	296/300	NonOverLappingTemp
43	30	24	29	34	27	32	22	31	28	0.339799	296/300	NonOverLappingTemp
.	.	.	.	.	.	.	.	.	.	.	.	NonOverLappingTemp
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	NonOverLappingTemp
22	21	30	44	38	32	23	31	35	24	0.043745	295/300	OverlappingTemp
32	44	35	29	28	26	20	23	34	29	0.132132	298/300	Universal
28	27	18	36	40	26	33	37	23	32	0.122325	297/300	ApproximateEntropy

List of approved CSPRNG's

As of 2020 the following algorithms are cryptographically secure pseudo random number generators:

- **BlumBlumShub** invented by L. Blum, M. Blum and M. Shub in 1986
- **YARROW** invented by John Kelsey, Bruce Schneier, and Niels Ferguson in 1999
- **Fortuna** invented by Bruce Schneier and Niels Ferguson in 2003
- **ISAAC** based on RC4. Invented by Robert J. Jenkins Jr. in 1993.
- **AES-CTR DRBG**
- **ANSI X9.17 standard**

FIPS PRNG requirements



- The entropy source is referred to as a “Non-Deterministic Random Number Generator” (NDRNG) in FIPS parlance
- The entropy contained in the NDRNG output must be a minimum of 112 bits
- The entropy source code be a hardware or software mechanism

Entropy Example #1 - FIPS

- Min-entropy is: **4.23 bits / 8 bits**
- Seed size in bits: **192 bits**
- Maximum key generated: **128 bits**
- Determine entropy in seed: Min entropy as ratio * seed size

$$4.23/8 * 192 = 101.52 \quad (112 < 101.52) \tag{6}$$

Result: No Entropy Caveat. 101.52 bits does not meet the minimum requirement of 112 bits. **This generator can not be used**

Entropy Example #2 - FIPS

- Min-entropy is: **4.8 bits / 8 bits**
- Seed size in bits: **420 bits**
- Maximum key generated: **256 bits**
- Determine entropy in seed: Min entropy as ratio * seed size

$$4.8/8 * 420 = 252 \quad (112 < 252 < 256) \tag{7}$$

Result: The module generates cryptographic keys whose strengths are modified by available entropy.

Entropy Example #3 - FIPS

- Min-entropy is: **7.2 bits / 8 bits**
- Seed size in bits: **300 bits**
- Maximum key generated: **256 bits**
- Determine entropy in seed: Min entropy as ratio * seed size

$$7.2/8 * 300 = 270 \quad (256 < 270) \tag{8}$$

Result: No Entropy Caveat required , the entropy contained in the seed is greater than the maximum key size generated

Linear Congruential Generators

Definition

LCG generators are dfined by the following recursive formula:

$$X_{n+1} \equiv (aX_n + c)(\mathbf{mod}\, m)$$

where

- a is the multiplier ($0 < a < m$)
- X_0 is a starting seed value
- c is the increment value ($0 \leq c < m$)
- $m > 0$ is the modulus

It is konwn that the period of a general LCG is at **most m**.

Linear Congruential Generators

Linear congruential generators (LCG) are one of the best known pseudorandom number generators. Few example without loss of generality:

- ANSI C, C++
- Microsoft Visual/Quick C/C++
- Microsoft Visual Basic (6 and earlier)
- **Java's java.util.Random**
- Turbo Pascal, Borladn Delphi
- Apple CarbonLib
- C++11's minstd_rand

Challenge - The odds one out

Challenge: Present an effective method how to find the AES encrypted file.

- There are 1000 files: (data1.rnd, . . . , data1000.rnd).
- One file is a TrueCrypt container with AES-Twofish-Serpent 256 bit encryption.
- 999 file were generated by a linear congruential pseudo-random number generator (LCG) with the following parameters:
  $a = 1234$, $c = 6678010$, $m = 78980$ and X_0 is unknown.

Challenge - LCG PRNG cracking

n, m, c and the *seed* are 256 bits unknown parameters.
Challenge: Recover n, m, c only observing the PRNG output.

```
def LCG(seed,r):
    state = seed
    file= open("1_LCG.rnd","wb")
    for _ in range(r):
        state = (state * m  + c) % n
        out ='%x' % state
        out=out.zfill(64)
        file.write(out.decode('hex'))
```

Challenge - LCG PRNG cracking

```
xen1thLabs$ python 1_LCG_GEN.py

------------LCG PRNG with 256 bit modulus generator-----------------------

[x] Generating secret seed           :  589540581937093483188345850773367272
[x] Generating secret modulus        :  105824616689578035826403877229023301
[x] Generating secret multiplier     :  967735057446414092936996317004303023
[x] Generating secret increment      :  660888282852935790470035654575456497
[x] Generating 10MB random output...
xen1thLabs$ python 1_LCG_CRACK.py

------------256 bits LCG PRNG Cracker-----------------------

[x] Reading output file    :  LCG.rnd
[x] Recovered modulus      :  1058246166895780358264038772290233014282772319 8
[x] Recovered multiplier   :  967735057446414092936996317004303023575972295 40
[x] Recovered incrementer  :  660888282852935790470035654575456497956207608 95
xen1thLabs$
```

Mersenne twister

Definition

- The Mersenne Twister is a pseudorandom number generator (PRNG)
- It has a very long period $2^{19937} - 1$
- **k-distributed** to 32-bit accuracy for every $1 \leq k \leq 623$
- Really fast
- Python random.Random() is based on MT.

Challenge - Where is the problem?

Generate random number using python:

```
def PYTHON_rnd(c,n):
        file= open("python.rnd","wb")
        RND = random.Random()
        RND.seed(c)
        for i in range(0,n):
                out =hex(RND.getrandbits(32)).lstrip("0x").
                rstrip("L").upper().zfill(8).decode('hex')
                file.write(out)
```

Challenge - MERSENNE cracking

```
xen1thLabs$ python 2_MERSENNE_GEN.py

------------Mersenne twister generator-----------------------

[x] Generating secret seed              :  10f9d3d7efc1614a8a9138c146
[x] Generating 10MB random output...
xen1thLabs$ python 2_MERSENNE_CRACK.py

------------Mersenne Twister Cracker-------------------------

[x] Reading output file     :  MERSENNE.rnd
[x] Prediction              :  0xbf63e445
[x] Actual state from file  :  0x2cd5409e
[x] Next state from file    :  0xbf63e445
xen1thLabs$
```

LFSR (Linear-feedback shift register)

Definition

- A linear-feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state.
- The most commonly used linear function of single bits is exclusive-or (XOR).
- An LFSR with a well-chosen feedback function can produce a sequence of bits that appears random and has a **very long cycle**.
- The **Berlekamp–Massey** algorithm is an algorithm that will find the shortest linear feedback shift register (LFSR) for a given binary output sequence.

Cracking LFSR algorithm

Binary LFSRs configurations can be expressed as linear functions using matrices in $\mathbb{F}_2$

$$\begin{pmatrix} a_k \\ a_{k+1} \\ a_{k+2} \\ \vdots \\ a_{k+n-1} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_0 & c_1 & c_2 & \cdots & c_{n-1} \end{pmatrix} \begin{pmatrix} a_{k-1} \\ a_k \\ a_{k+1} \\ \vdots \\ a_{k+n-2} \end{pmatrix} =$$

$$\begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_0 & c_1 & c_2 & \cdots & c_{n-1} \end{pmatrix}^k \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

Challenge - LFSR cracking

LFSR random number generator:

```
def LFSR(st, k, n):
    while (n > 0):
        yield st & 0xff
        for _ in range(8):
            c, s = k, st
            b = 0
            while c:
                b ^= c & 1 * s & 1
                c >>= 1 ; s >>= 1
            st = st >> 1 | b << 127
        n -= 1
```

LFSR (Linear-feedback shift register)

```
xen1thLabs$ time python 3_LFSR_GEN.py

------------LFSR PRNG generator------------------------

[x] Generating secret seed          :  0xc58bc5ebec118521d28159b951c861f1
[x] Generating secret key           :  0x50f4071ca186f85481a5b21c6389ae17
[x] Generating 10MB random output...

real    0m0.426s
user    0m0.401s
sys     0m0.021s
xen1thLabs$ python 3_LFSR_CRACK.py

------------LFSR PRNG Cracker ------------------------

[x] Reading file            : LFSR.rnd
[*] Recovering secret seed  : 0xc58bc5ebec118521d28159b951c861f1
[*] Recovering secret key   : 0x50f4071ca186f85481a5b21c6389ae17
xen1thLabs$
```

Challenge - AES-SEC PRNG

INPUT: c: counter, n: number of 128 bits blocks should be generated , k: a 128 bits random AES key
OUTPUT: $n *$ 128-bits random number: $C_0, C_1, \ldots, C_i$

```
1: procedure AES-SEC(c, n, k)
2:     for i = 0 → n do
3:         C_i ← E_k(c)
4:         c ← c + 1
5:         k ← C_i
6:         print (C_i)
7:     end for
8: end procedure
```

* where $E_k()$ denotes the AES encryption with k key.

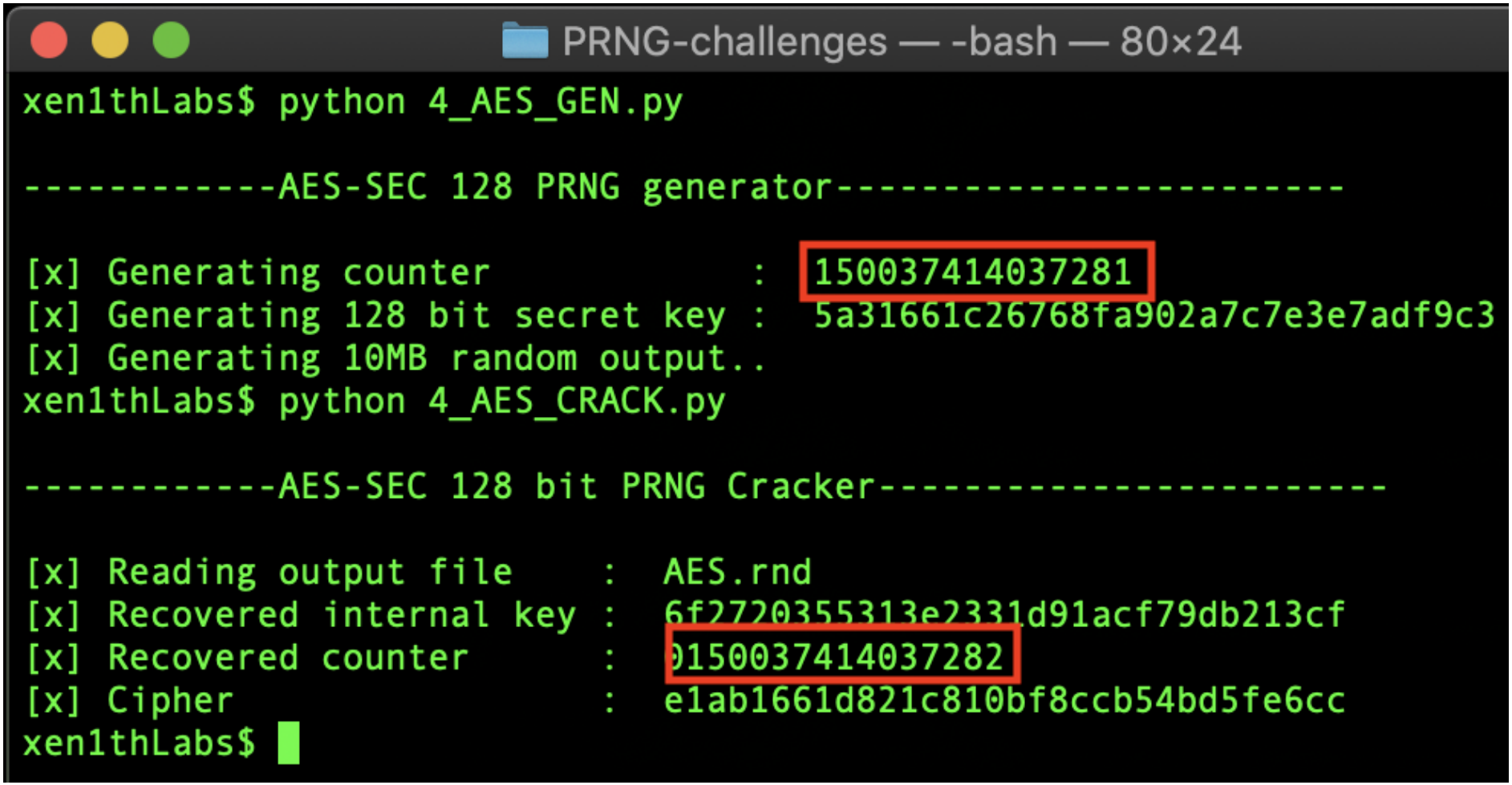
Challenge - AES-SEC PRNG

AES-SEC random number generator:

```
def AES_SEC(c,n,k):

        file= open("AES.rnd","wb")
        for i in range(0,n):
                counter=str(c).zfill(16)
                Ci= AES.new(k, AES.MODE_ECB)
                Ci=Ci.encrypt(counter)
                file.write(Ci)
                c=c+1
                k=Ci.encode('hex')
```

Cracking AES-SEC algorithm

Predicting the next counter:



Can we trust in the NIST test ultimately?

Is it possible to hide a backdoor to a well-known encryption algorithm?

- The source code of a popular encryption algorithm (e.g: AES) is well-known by security experts.
- A white-box audit would identify the malicious activity.

For instance, the appropriate implementation of an algorithm can be verified by test vectors:

Encryption key: 2b7e151628aed2a6abf7158809cf4f3c
IV: 7649abac8119b246cee98e9b12e9197d
Test vector: ae2d8a571e03ac9c9eb76fac45af8e51
Cipher text: 5086cb9b507219ee95db113a917678b2

A test vector is a POC that no modification has been made in the algorithm.

What about PRNG's?

- There are no test vectors
- The main test methods are statistical tests. These tests can be used to evaluate the output of a PRNG (e.g: NIST SP 800-22 test, DIEHARD test, etc.)

Main idea: Modify the algorithm in such a way that the modified one is still passing all statistical tests.

Predictable output

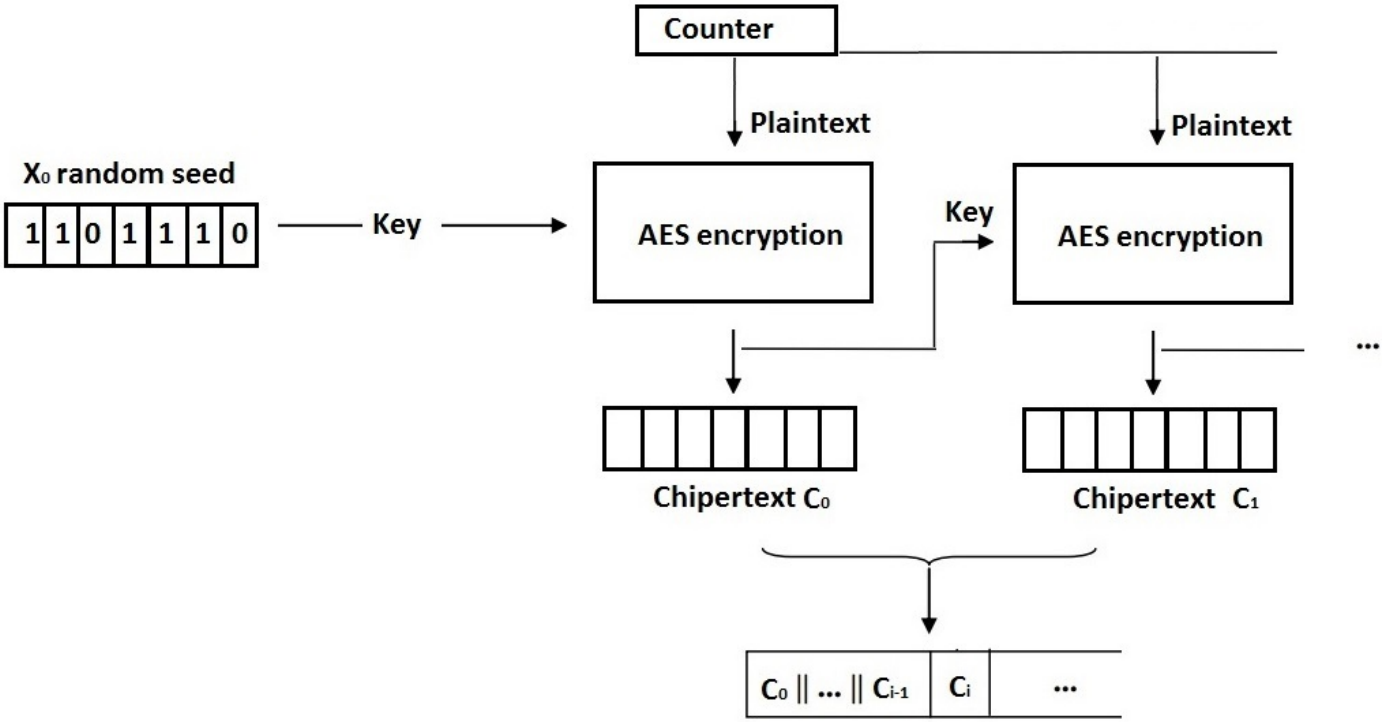
Let A_{PRNG} define a pseudorandom number generator algorithm. Let b_i denote an i bit length binary sequence produced by A_{PRNG} and let T_n denote the n_{th} NIST test. Furthermore, let us define the following function:

$$\varphi(A_{PRNG}, b_i) = \begin{cases} 0 & \text{if } A_{PRNG} \text{ pass } T_n \, \forall 1 \leq n \leq 15\,, \\ k/15 & \text{if } A_{PRNG} \text{ fail for k different } T_n \end{cases}$$

Theorem (Norbert Tihanyi 2015, Studia Math)

There exist a polynomial-time A_{PRNG} algorithm with 2^m period where $\phi(A_{PRNG}, b_i) = 0$ and observing $2m-1$ consecutive bits from the output of A_{PRNG} an adversary can predict the next-bit (2m) with 100% success.

Proof: SEA- Self Encryption AES

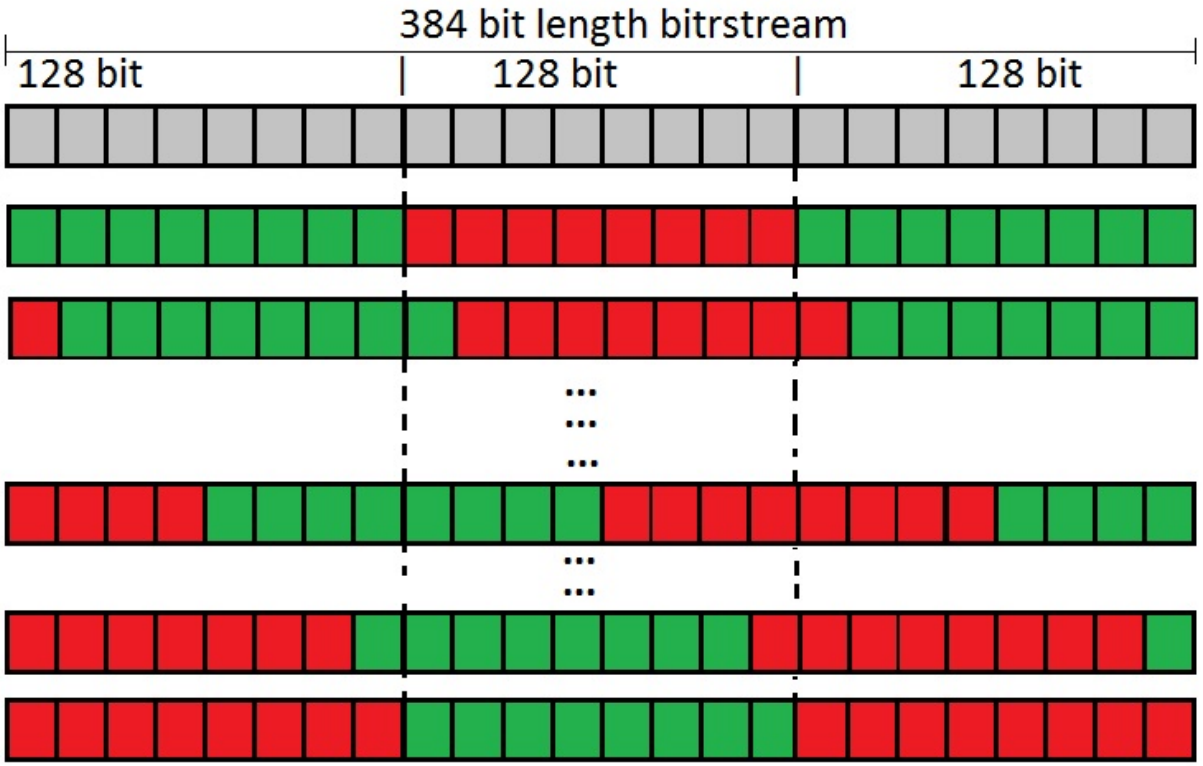


$C_{i+1} = E_k(c)$ where $k = C_i$. Because $C_i = E_k(c)$ and the c counter is different for every C_i, hence there will be no loop in the algorithm if $C_i = C_{i+x}$ occur. The cycle length of SEA is at least 2^{128} bits with the usage of AES-128.

Observing only 383 bits...

Corollary

Consecutive C_i and C_{i+1} (256 bit) always can be found in a 383 $(2 * 128 + 127)$ length bit stream. It is possible to determine the next-bit of SEA observing maximum 383 bits from the output.



Secure SEA -Non-Secure SEA

Unpredictable algorithm

Algorithm 1 Secure SEA

1: **procedure** SSEA(s, n, k)
2: $D \leftarrow$ Date/time
3: $s \leftarrow E_k(D) \oplus E_k(s)$
4: **for** $i = 0 \rightarrow n$ **do**
5: $C_i \leftarrow E_k(s)$
6: $s \leftarrow s + 1$
7: $k \leftarrow s$
8: **end for**
9: **end procedure**

Predictable algorithm

Algorithm 2 Simple SEA

1: **procedure** SSEA(s, n, k)
2: $D \leftarrow$ Date/time
3: $s \leftarrow E_k(D) \oplus E_k(s)$
4: **for** $i = 0 \rightarrow n$ **do**
5: $C_i \leftarrow E_k(s)$
6: $s \leftarrow s + 1$
7: $k \leftarrow C_i$
8: **end for**
9: **end procedure**

Output of SEA can not be distinguished from random oracle by statistical tests .

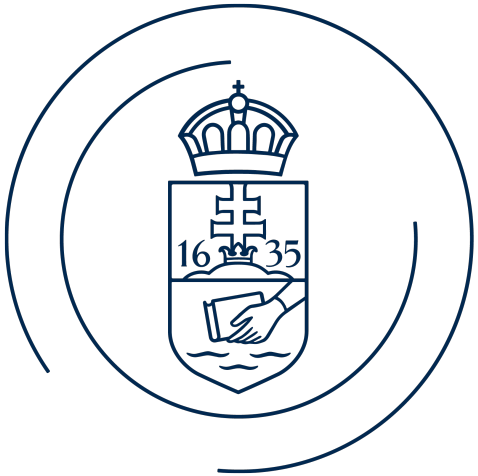
NIST test result for SEA

P-value	PROPORTION	TEST
0.122325	50/50	*Frequency*
0.616305	49/50	*BlockFrequency*
0.739918	50/50	*CumulativeSums*
0.883171	49/50	*Runs*
.	.	.
0.075719	50/50	*Rank*

Thank you for your attention!

 ntihanyi@inf.elte.hu

 @TihanyiNorbert



ELTE
EÖTVÖS LORÁND
UNIVERSITY

Cryptography and security -Practice (IPM-18sztKVSZKRBG)

Dr. Norbert Tihanyi

OSCP, OSCE, OSWP, CRTP, CEH, ECES , ISO27001 LA

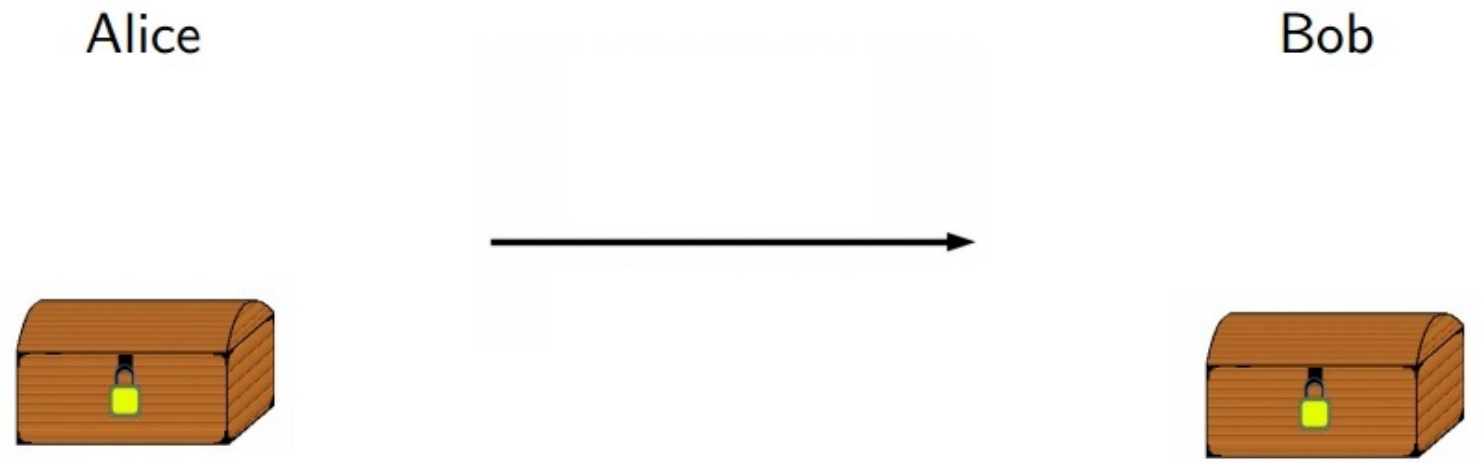
ntihanyi@inf.elte.hu

@TihanyiNorbert

Module 0x03
Public Key Infrastructure

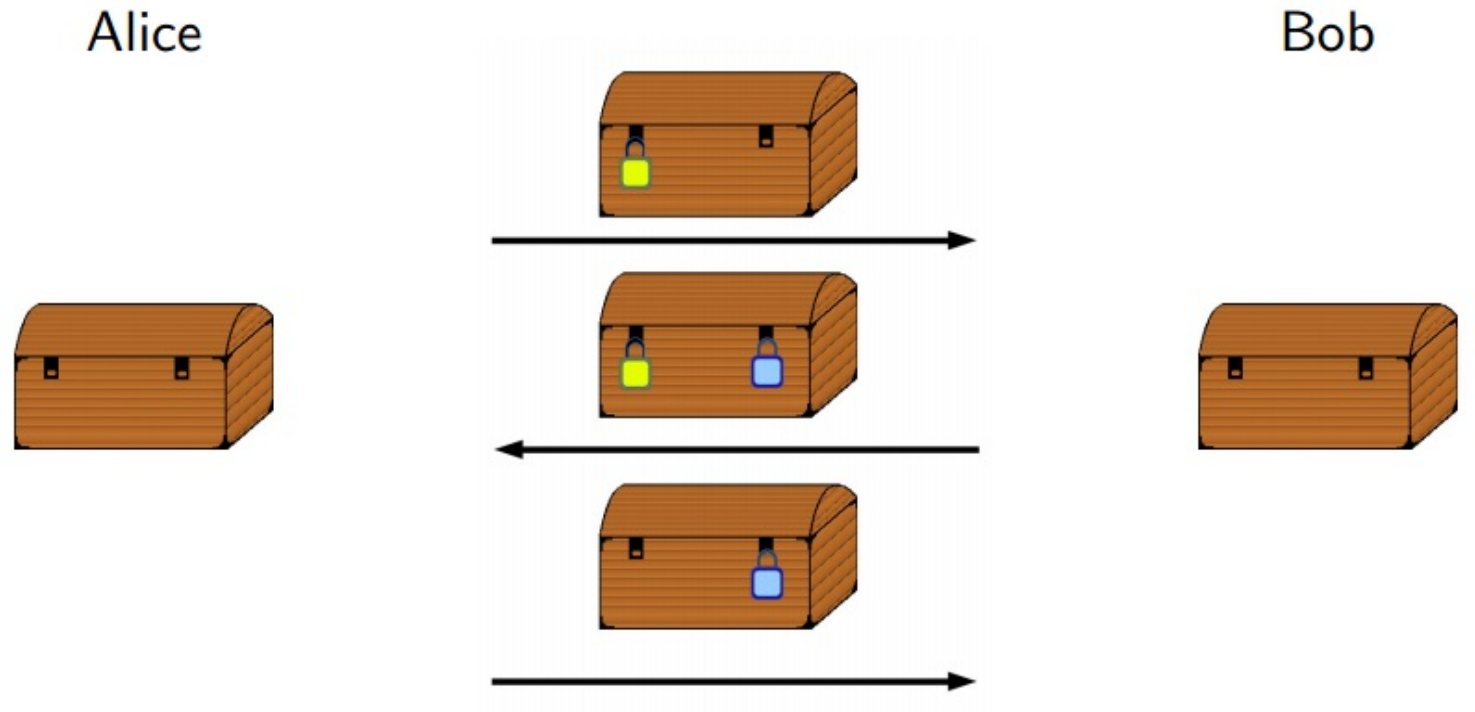
The Key distribution problem

Alice want to send a secret to Bob, but they don't share a common key



- Alice can't send the key with the box (In this case the key would be compromised)
- The key is uniqe, and Bob do not have Alices's key

Secure Communication via insecure channel



- Alice puts a secret in a box, which she locks with her own lock.
- Bob adds his own lock to this box, so that now the box has two locks.
- Alice, knowing that the box is secure with Bob's lock, then takes her own lock off the box (with her key).
- Bob then removes his lock and receives the secret

HTTPS communication

Diffie-Hellman Key Exchange Protocol

In a landmark paper in 1976 , W. Diffie and M.E. Hellman introduced an algorithm for exchanging a symmetric key in public. This is the so-called Diffie-Hellman key exchange protocol.

Generators, primitive roots

- Let $\mathbb{Z}_n$ be the set of integers modulo n, where the addition and multiplication are perfomed modulo n.
- $\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n : gcd(a, n) = 1\}$
- $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$
- The group $\mathbb{Z}_P^*$ for a prime P is *cyclic*. This means that there is some number $g \in \mathbb{Z}_P^*$ such that $\mathbb{Z}_P^* = \{1, g, g^2, g^3, \ldots, g^{P-2}\}$. (*primitive roots*)
- g is called a *generator* for the group.

Primitive root - Example

Definition

Let G be a group, usually with $|G| < \infty$, and let $a, b \in G$. If $a^x = b$ for some integer x, then we call x a discrete logarithm of b, base a.

The number 3 is a primitive root modulo 7:

$3^1 \equiv 3 \mod 7$ $\quad$ $3^2 \equiv 2 \mod 7$

$3^3 \equiv 6 \mod 7$ $\quad$ $3^4 \equiv 4 \mod 7$

$3^5 \equiv 5 \mod 7$ $\quad$ $3^6 \equiv 1 \mod 7$

3, 2, 6, 4, 5, 1 form a rearrangement of all nonzero remainders modulo 7, implying that 3 is indeed a primitive root modulo 7.

Discrete Logarithm Problem (DLP)

Definition (Discret Logarithm Problem)

Given a group G, a generator g of the group and an element h of G, to find the discrete logarithm to the base g of h in the group G.

Computing $h \to g^h$ is easy. Computing $g^h \to h$ is **hard**.
Discrete logarithm problem is not always hard. The hardness of finding discrete logarithms depends on the groups. For example, a popular choice of groups for discrete logarithm based crypto-systems is $\mathbb{Z}_P^*$ where p is a prime number. However, if $p-1$ is a product of small primes, then the Pohlig−Hellman algorithm can solve the discrete logarithm problem in this group very efficiently.

Diffie-Hellman Key Exchange Protocol

The Diffie-Hellman protocol:

- Alice chooses prime P at random and finds a generator g.
- Alice chooses $X \leftarrow_R \{0, 1, \ldots, P-2\}$ and sends P, g and $\hat{X} = g^X \pmod{P}$ to Bob.
- Bob chooses $Y \leftarrow_R \{0, 1, \ldots, P-2\}$ and sends $\hat{Y} = g^Y \pmod{P}$ to Alice.
- Alice and Bob both compute $k = (g^X)^Y = (g^Y)^X \pmod{P}$. Alice does that by computing $\hat{Y}^X$ and Bob does this by computing $\hat{X}^Y$.
- They then use k as a key to exchange messages using a private key encryption scheme.

The RSA algorithm

The RSA algorithm is invented in 1977 by Ron Rivest, Adi Shamir and Leonard Adleman. RSA is the most common asymmetric cryptosystem nowadays.

- Let p and q two large prime numbers. Let $N = p \cdot q$.
- Calculate the φ function. $\varphi(N) = (p-1)(q-1)$.
- Choose an odd $e \in \mathbb{N}$, such that $1 < e < \varphi(N)$, and $gcd(\varphi(N), e) = 1$.
- Calculate the multiplicative inverse of e mod $\varphi(N)$, so calculate $ed \equiv 1 \mod \varphi(N)$.

The secret key is (d,N) and the public key is (e,N). **There is not known efficient algorithm that can factorize a large number to its prime factors**.

Factoring large number

According to Jevons (1874, p. 123), ”Can the reader say what two numbers multiplied together will produce the number 8616460799? I think it unlikely that anyone but myself will ever know.”

$$8616460799 = 89681x96079$$

Published factorizations include those by Lehmer (1903) and Golomb (1996).

Breaking a 2048 bit modulus in polynomial time?

- Let N be the following 2048 bit modulus:

18288555059064136749641790235966046816151902195376350587472799010089331214280648563180987764008880912824138327852185177758309666737535598494017671220209321075393350589076624777915491904935815407466443450756622382829697619461680267524143610575740343470708102619224093270459810845049652599071062964498510820317014503624995318937297539456252183497919945901972745299021815120246428766125247731547244080477055689347928643817775783169934163296735469364027651150794889282048589925137570749102458356158897066497265248398663779685483528766819329025900164360281342714500147083328086609575803873558733222874601085724990528052440 9

Where is the problem?

```
from Crypto.Util import number
# Generate 1024 bit P prime
x=number.getStrongPrime(1024)
while True:
    x=x+2
    if number.isPrime(x)==True:
        P=x
        break
# Generate 1024 bit Q prime
while True:
    x=x+2
        if number.isPrime(x)==True:
            Q=x
            break
N=P*Q
print N
```

Fermat factorization

Fermat's method works best when there is a factor near the square-root of N. It is well-known that if $|p - q|$ is small N can be factored in polynomial time. In 2009 the following result proved by Robert Erra and Christophe Grenier:

Theorem

One can factor N in a polynomial time if $|p - q| < N^{5/18}$. A better result can be achieved by using the fact that the Newton polygon of $P(x, y)$ is in fact a lower triangle: one can indeed factor N in a polynomial time if $|p - q| < N^{1/3}$.

Pollard's $p - 1$

In 1974 JOHN POLLARD invented a special-purpose integer factorization algorithm based on algebraic-groups. This algorithm is the so called *Pollard's* $p - 1$ algorithm. He proved that for the successful factorization it is necessary but not sufficient that $P - 1$ has only small prime factors.

In our particular case $P - 1$ has no larger factors than 100 000. Openssl library modified to generate ”bad primes”.

Pseudocode of Pollard's $p-1$

Algorithm 1 POLLARD $p-1$

select a smoothness bound B ▷ not necessary explicitly
while the time limit is not reached **do**
    $L \leftarrow \prod_{\text{primes } q \leq B} q^{\lfloor \log_q B \rfloor}$
    $c \leftarrow$ a random integer comprime to N
    $d \leftarrow \gcd(c^L, N)$ ▷ using modular arithmetic
    if $1 < d < N$ **then return** d
    end if
    if $d = 1$ **then** select a larger B
    else ▷ $d = N$
        select a smaller B
    end if
end while

Strong primes

In cryptography a prime number p is strong if:

- p is sufficiently large
- $p \pm 1$ and $q \pm 1$ have large prime factors
- In some cases, some additional conditions may be included.

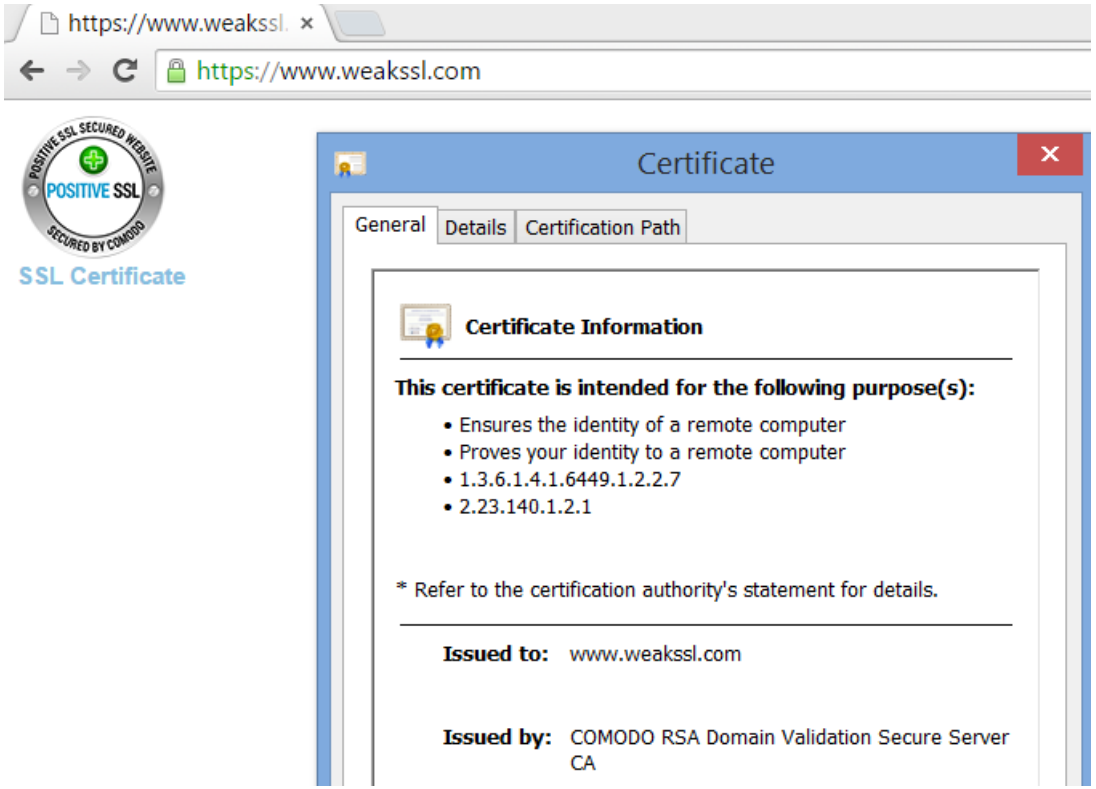
Are Strong Primes needed for RSA?

"...We argue that, contrary to common belief, it is unnecessary to use strong primes in the RSA cryptosystem. That is, by using strong primes one gains a negligible increase in security over what is obtained merely by using "random" primes of the same size... ***...use of strong primes provides no additional protection against factoring attacks, because Lenstra's method of factoring based on elliptic curves (ECM...)*** - Ron Rivest & Robert Silverman, 2001

Can we trust in HTTPS ultimately?

Example HTTPS: www.weakssl.com

Certificate issued by COMODO for weakssl.com



Example HTTPS: www.weakssl.com

Certificate issued by COMODO for weakssl.com

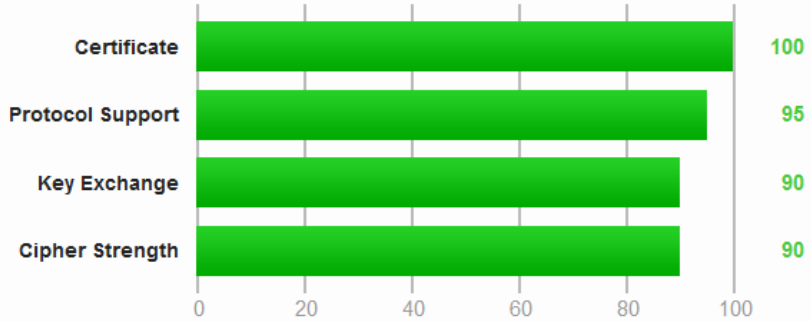
Perfect configuration

QUALYS® SSL LABS

Summary

Overall Rating

A+



Visit our documentation page for more information, configuration guides, and books. Known issues are documented here.

This server supports TLS_FALLBACK_SCSV to prevent protocol downgrade attacks.

This server supports HTTP Strict Transport Security with long duration. Grade set to A+. MORE INFO »

Prime generation method?

Prime generation method:

1: **while** $P \neq$prime **do**
2: $P \leftarrow 2$
3: **while** BinaryLength($P < 1024$) **do**
4: $P \leftarrow P \cdot nextprime(random(100000))$
5: **end while**
6: $P \leftarrow P + 1$
7: **end while**

Pollard $p-1$ algorithm

Algorithm 2 POLLARD $p-1$

1: select a smoothness bound B ▷ not necessary explicitly
2: **while** the time limit is not reached **do**
3: $L \leftarrow \prod_{\text{primes } q \leq B} q^{\lfloor \log_q B \rfloor}$
4: $c \leftarrow$ a random integer comprime to N
5: $d \leftarrow \gcd(c^L, N)$ ▷ using modular arithmetic
6: **if** $1 < d < N$ **then return** d
7: **end if**
8: **if** $d = 1$ **then** select a larger B
9: **else** ▷ $d = N$
10: select a smaller B
11: **end if**
12: **end while**

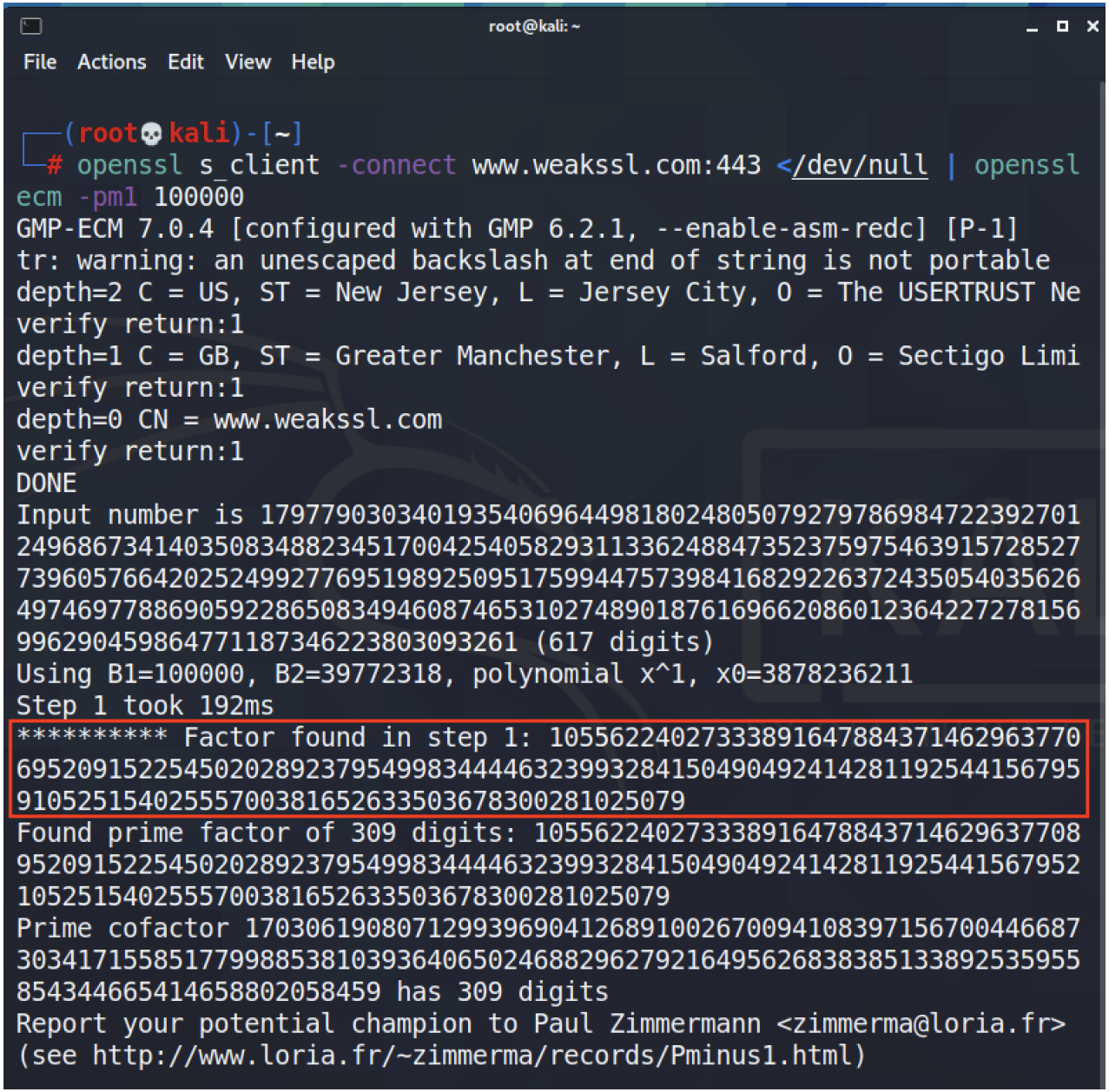
Recovering P and Q from 2048 bits N modulus

command:

openssl s_client -connect www.weakssl.com:443 </dev/null | openssl x509 -noout -modulus

```
root@weakssl:~# openssl s_client  -connect www.weakssl.com:443 </dev/null  |  op
enssl x509 -noout -modulus
depth=3 C = SE, O = AddTrust AB, OU = AddTrust External TTP Network, CN = AddTru
st External CA Root
verify error:num=19:self signed certificate in certificate chain
verify return:0
Modulus=8E6994A179BF0D66327670641E08F57B14533BBD47D14C919C2BBA454E74E4C9647A966B
B01AA8DD5C287B50BF6DEA123E8B81C0B681CE69E0F7B7833232B852C9A2BDED515FFA150972A453
D1A59DEF05DF52EBFE80D810949792688C4163D80DA73D8CBEDBEA45134CD73E3ABD5FEA0362BF9E
C3F67A5EAAEED3560667AC018CE4A002B88803F9EC5A5193FB6DFF9FA53307E866E29E19FF4855B5
90EEE98089DC709A1CDBE55DEBE58E56B1773C2FC3BC8F712CEC7923FA02C8F8BA37A63444B7D9D3
11E8FA7AC0AE1EE35AA99871086653E7394170EB43112AE3ED1D80C76E48A1EEDFB97F1AE72B39F7
739203C67B20C39B6AD9D01546C1FA27D8F7510D
DONE
root@weakssl:~#
```

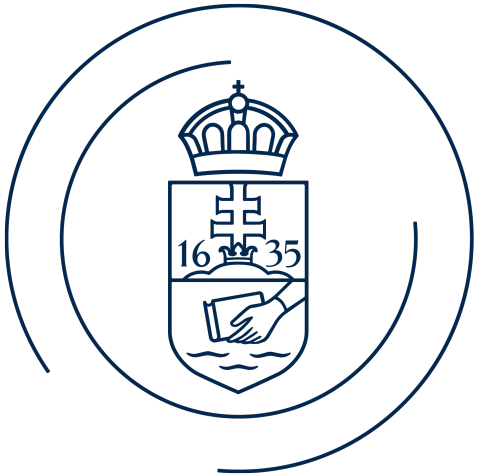
Factorization in few secs



Thank you for your attention!

ntihanyi@inf.elte.hu

@TihanyiNorbert



ELTE
EÖTVÖS LORÁND
UNIVERSITY

Cryptography and security -Practice (IPM-18sztKVSZKRBG)

Dr. Norbert Tihanyi

OSCP, OSCE, OSWP, CRTP, CEH, ECES , ISO27001 LA

ntihanyi@inf.elte.hu

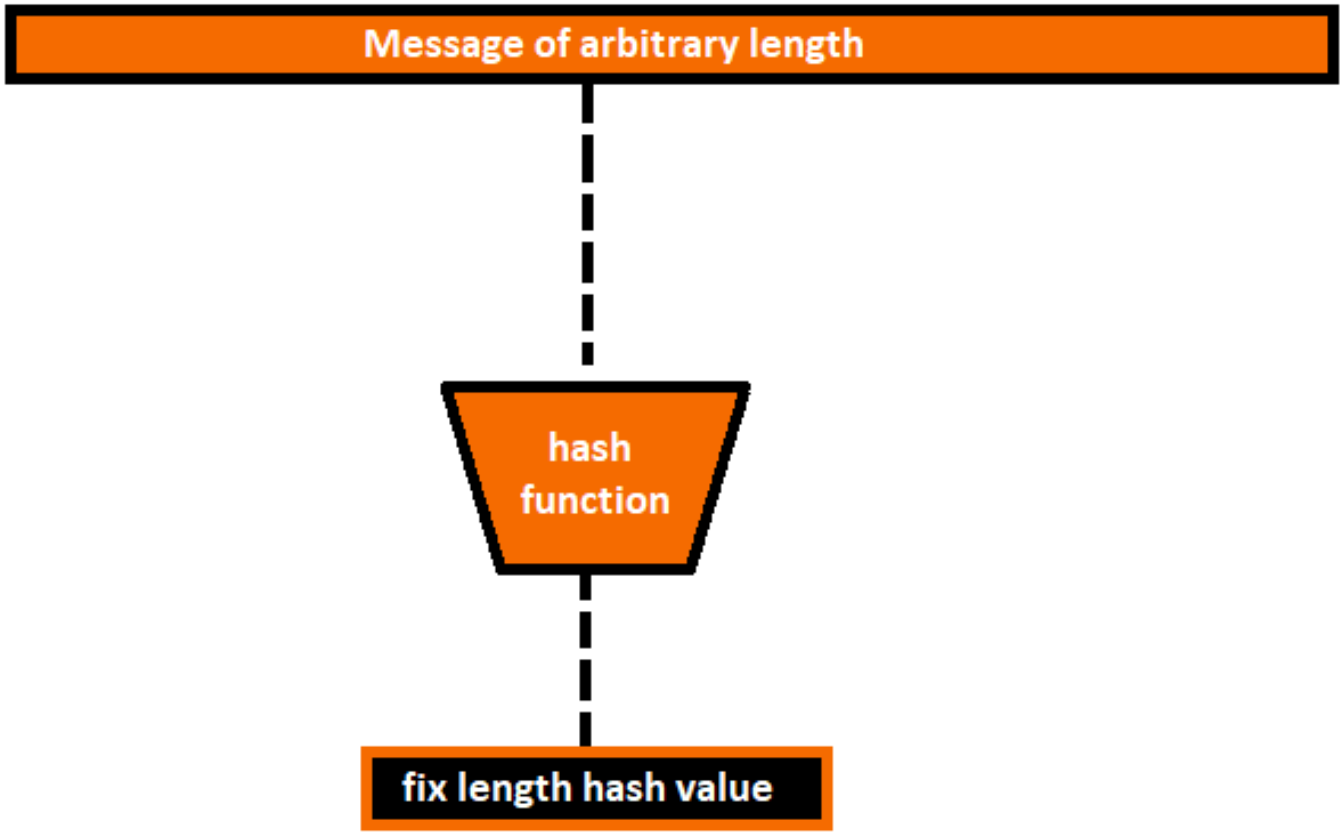
@TihanyiNorbert

Module 0x04
Hash Functions

What is hash function?

Definition

A **hash function** is a computationally efficient function mapping binary strings of arbitrary length to binary strings of some fixed length, called *hash-values*. $\mathbf{H} : \{0,1\}^* \rightarrow \{0,1\}^k$



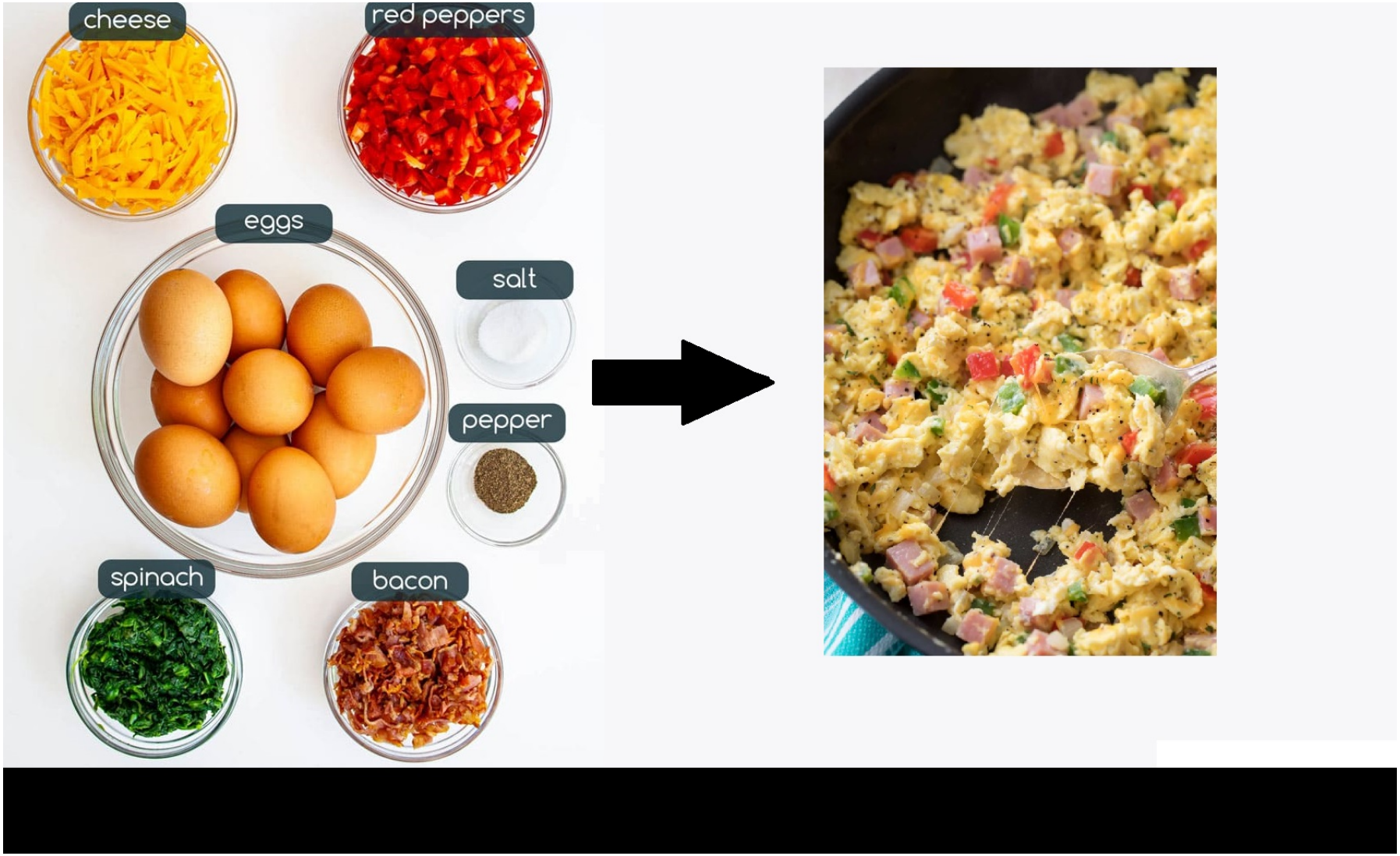
What is Cryptographic hash function?

Definition (Cryptographic Hash Function)

A cryptographic hash function is a mathematical algorithm that maps data of arbitrary size to a bit string of a fixed size (a hash function) which is designed to also be **one-way function**, that is, a function which is **infeasible to invert**.

The input data is often called the **message**, and the output (the hash value or hash) is often called the **message digest** or simply the digest.

One-way : infeasible to invert



Properties of cryptographic hash functions

The ideal cryptographic hash function has four main properties:

Definition (Properties of cryptographic hash functions)

- it is quick to compute the hash value for any given message
- it is infeasible to generate a message from its hash value except by trying all possible messages
- a small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value
- it is infeasible to find two different messages with the same hash value

Avalanche effect

Definition (Avalanche effect)

An important and desirable feature of a good hash function is the non-correlation of input and output, or so-called **avalanche effect**, which means that a small change in the input results in a significant change in the output, making it statistically indistinguishable from random.

Example:

- MD5(**123456**) $->$ `e10adc3949ba59abbe56e057f20f883e`
- MD5(**1234567**) $->$ `fcea920f7412b5da7be0cf42b8c93759`

Cryptanalysis attacks

A cryptographic hash function must be able to withstand all known types of cryptanalytic attack:

- **Pre-image resistance:** Given a hash value h. It should be difficult to find any message m such that $h = H(m)$.
- **Second pre-image resistance:** Given an input m_1. It should be difficult to find different input m_2 such that $H(m_1) = H(m_2)$.
- **Collision resistance:** It should be difficult to find two different messages m_1 and m_2 such that $H(m_1) = H(m_2)$. Such a pair is called a cryptographic hash collision.

Birthday Paradox

What's the probability that two people on the football/soccer pitch share a birthday? (day and month)

Birthday Paradox

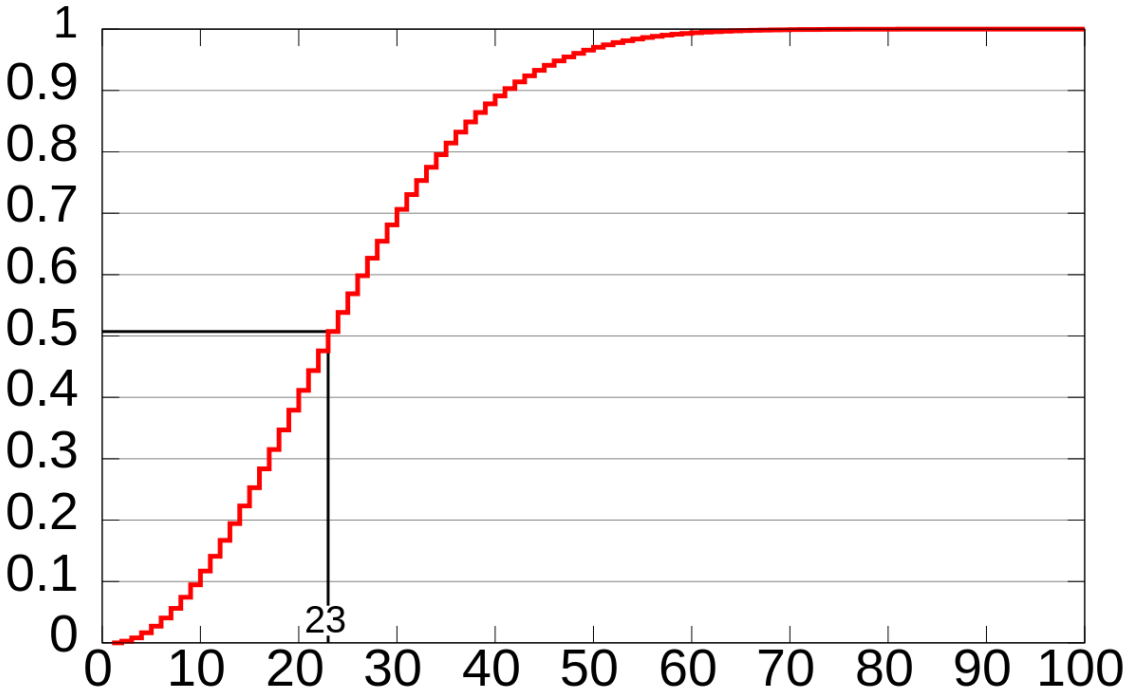
In probability theory, the birthday paradox concerns the probability that, in a set of n randomly chosen people, some pair of them will have the same birthday.

The probability reaches 100% when the number of people reaches 367 (including February 29)

99.9% probability is reached with just 70 people, and 50% probability with 23 people.

Birthday Paradox

$$P(A) = 1 - \frac{366!}{366^N(366-N)!}$$



Probability distribution

Birthday Attack

A birthday attack is a type of cryptographic attack that exploits the mathematics behind the birthday problem in probability theory.

Definition (Birthday Attack)

A collision in a H function can be found with probability $1/2$ by computing $2^{n/2}$ hash value.

MD5 Cryptanalysis I

MD5 is a widely used cryptographic hash function producing a 128-bit hash value designed by Ronald Rivest in 1991.

MD5 is not collision resistant:

- **1993:** pseudo-collision in the compression function (IV based attack) by B. den Boer and A. Bosselaers.
- **1996:** semi-free start collision by H. Dobbertin
- **2004:** MD5CRK, a distributed effort using birthday attack launched by Jean-Luc Cooke
- **2004:** analytical hash collision attack within 1 hour by Xiaoyun Wang et al.

MD5 cryptanalysis II

- **2005:** In March 2005, Xiaoyun Wang and Hongbo Yu of Shandong University in China published an article in which they describe an algorithm that can find two different sequences of 128 bytes with the same MD5 hash.

```
d131dd02c5e6eec4693d9a0698aff95c 2fcab58712467eab4004583eb8fb7f89
55ad340609f4b30283e488832571415a 085125e8f7cdc99fd91dbdf280373c5b
d8823e3156348f5bae6dacd436c919c6 dd53e2b487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080a80d1e c69821bcb6a8839396f9652b6ff72a70
```

and

```
d131dd02c5e6eec4693d9a0698aff95c 2fcab50712467eab4004583eb8fb7f89
55ad340609f4b30283e4888325f1415a 085125e8f7cdc99fd91dbd7280373c5b
d8823e3156348f5bae6dacd436c919c6 dd53e23487da03fd02396306d248cda0
e99f33420f577ee8ce54b67080280d1e c69821bcb6a8839396f965ab6ff72a70
```

MD5 cryptanalysis III

- **2007:** Marc Stevens, Arjen K. Lenstra, and Benne de Weger used an improved version of Wang and Yu's attack known as the chosen prefix collision method to produce two executable files with the same MD5 hash, but different behaviors.

hello.exe MD5 Sum: cdc47d670159eef60916ca03a9d4a007
erase.exe MD5 Sum: cdc47d670159eef60916ca03a9d4a007

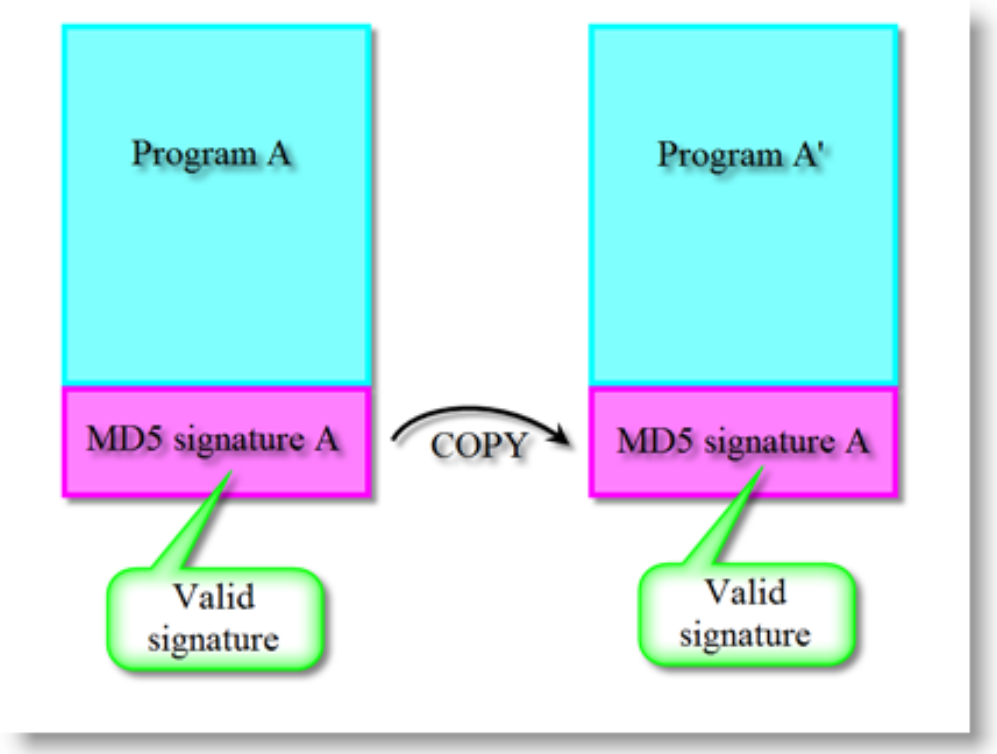
MD5 cryptanalysis IV

- **2007:** Fast Collision Finding program based on Marc Steven's thesis work.

```
C:\fastcoll>fastcoll_v1.0.0.5.exe -o md51 md52
MD5 collision generator v1.5
by Marc Stevens (http://www.win.tue.nl/hashclash/)

Using output filenames: 'md51' and 'md52'
Using initial value: 0123456789abcdeffedcba9876543210

Generating first block: .....
Generating second block: W.
Running time: 1.666 s
```

MD5 cryptanalysis V

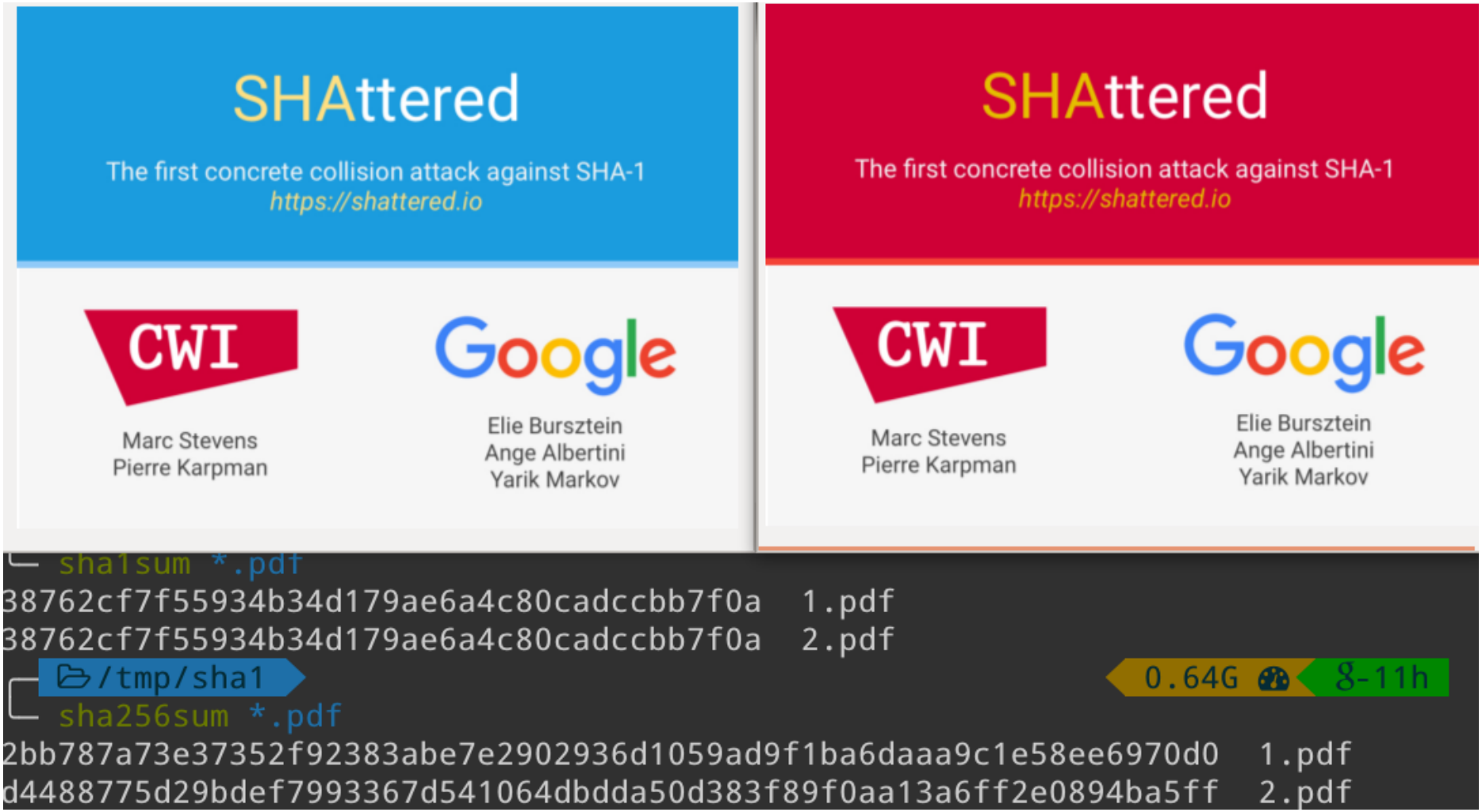
- **2009:** Didier Stevens used the evilize program to create two different programs with the same Authenticode digital signature. (Authenticode is Microsoft's code signing mechanism),



MD5 cryptanalysis VI

- **2010:** Tao Xie and Dengguo Feng constructed the first single-block collision for MD5 consisting of two 64-byte messages that have the same MD5 hash.
- **2012:** Single-block collision attack on MD5, Marc Stevens (md5-1block-collision-attack.exe)
- **2013:** A 2013 attack by Xie Tao, Fanbao Liu, and Dengguo Feng breaks MD5 collision resistance in 2^{18} (2-block attack) time. This attack runs in less than a second on a regular computer.

First collision for SHA1 by Google - 2017



Facts about hash functions

- MD5, SHA1 is considered cryptographically **NOT** secure hash functions
- SHA2, SHA3 (Keccak) is cryptographically secure hash functions
- For password hashing PBKDF2 (Password Based Key Derivation Function), Bcrypt or Scrypt **should be** used.
- In 2013 a long-term Password Hashing Competition was announced to choose a new, standard algorithm for password hashing. On 20 July 2015 **Argon2** was selected as the final PHC winner,

What is length extension attack

Theorem

Length extension attack is a type of attack where an attacker can use $H(message_1)$ *and the length of* $message_1$ *to calculate* $H(message_1||message_2)$

This attack can be used against **Merkle–Damgard** based hash functions, such as:

- MD4,MD5
- RIPEMD-160
- SHA1
- SHA2-256, SHA2-512
- WHIRLPOOL

The SHA-3 (Keccak) algorithm is not susceptible to this attack.

Length extension attack: POC python code

./sha1.py secret e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4
./sha1-len-attack.py e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4
6 ” ADVANCED IT Security”
msg: 80000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000
00000000003020414456414*e*4345442049542053656375726974 79
adf51eec44e5fd8bae82992a181596bd6708914b

Verifiying the output:

Hex representation of secret is 736563726574
./sha1.py -x 73656372657480000000000000000000000000000000
000000000000000000000000000000000000000000000000000000
000000000000000000000000000302041445641*e*4345442049542
05365637572697479
adf51eec44e5fd8bae82992a181596bd6708914b

Challenge - Hash cracking challenge

Given a 128-bit hash function based on MD5 algorithm.
An unknown password was hashed by this 128-bit hash function.
The output is **76edaf93c13a7e339ce6642ecf21c188**.
Find an appropriate X string where

$$ENCRYPT(X) = 76edaf93c13a7e339ce6642ecf21c188.$$

Challenge - Hash cracking challenge

The hash function pseudocode is the following:

Algorithm 1 -128 bit hash function

```
1: procedure ENCRYPT(password)
2:     hash ← MD5(password)                    ▷ Calculating MD5 hash
3:     l ← SORT(hash)                          ▷ Sorting of the input string
4:     l ← l[1 : 12]                           ▷ The first 12 characters are used
5:     l ← l + "0xFF"                          ▷ Append 0xFF string to l
6:     hash ← MD5(l)
7:     RETURN(hash)
8: end procedure
```

Challenge - Hash cracking challenge

Example for password=”hello”.

---

procedure ENCRYPT(*password*)
    hash $\leftarrow$ MD5(*password*)
        ▷ hash= 5d41402abc4b2a76b9719d911017c592
    l $\leftarrow$ SORT(*hash*)
        ▷ l= 00111112224445567779999aabbbccdd
    l $\leftarrow$ *l*[1 : 12] ▷ l=001111122244
    l $\leftarrow$ *l* + ”0*xFF*” ▷ l=0011111222440xFF
    hash $\leftarrow$ MD5(*l*)
        ▷ hash=bc25563a49a79ab56656ce2f4a3d82b0
    RETURN(*hash*)
end procedure

Challenge: Python code of the hash fucntion

Python code of the hash function:

```
import md5
def encrypt(password):
      hash = md5.new(password).hexdigest()
      l = list(hash)
      l.sort()
      return md5.new(''.join(l)[:12]+"0xFF").hexdigest()
print encrypt("hello")
```

For password =”hello” the hash checksum is **bc25563a49a79ab56656ce2f4a3d82b0**.

Challenge - Solution in python

```
import string,random,md5,time,os,socket, select
password_hash="76edaf93c13a7e339ce6642ecf21c188"
def encrypt(password):
      hash = md5.new(password).hexdigest()
      l = list(hash)
      l.sort()
      return md5.new(''.join(l)[:12]+"0xFF").hexdigest()

def id_generator(size=8, chars=string.ascii_uppercase +
string.ascii_lowercase + string.digits):
         return ''.join(random.choice(chars) for x
         in range(size))
while 1:
         text = id_generator()
         if encrypt(text)==password_hash:
                  print text
                  exit()
```

Challenge - hash cracking Solutions

Iti is possible to find collision in an average PC within few seconds. The following strings are solutions for our challenge:

- c9o9k8Ro
- rCFClH5y
- yR5ATjYq
- ...

e.g:

```
print encrypt("8rhKX4OE")
```

76edaf93c13a7e339ce6642ecf21c188.
It is possible to find solutions with only 4 characters too. e.g: **9yO2, fUtz**

MySQL 3.23 hash Collision

Iti is possible to find collision in the MySQL 3.23 hash function within few seconds. Attack based on the following publication: *Cryptanalysis of T-Function-Based Hash Functions., F. Muller and T. Peyrin -ICISC 2006*

6.1 Breaking the compression function

The compression function h is described by the following equations :

$$\begin{aligned} n_1^i &= n_1^{i-1} \oplus (((n_1^{i-1} \wedge 63) + add^{i-1}) \cdot c^i + (n_1^{i-1} \ll 8)) \\ n_2^i &= (n_2^{i-1} \ll 8) \oplus n_1^{i-1} \\ add^i &= add^{i-1} + c^i \end{aligned}$$

where the incoming chaining value $(n_1^{i-1}, n_2^{i-1}, add^{i-1})$ and the output (n_1^i, n_2^i, add^i) are composed with 3 words of length 32 bits and c^i is the password character to be hashed.

MySQL 3.23 hash Collision

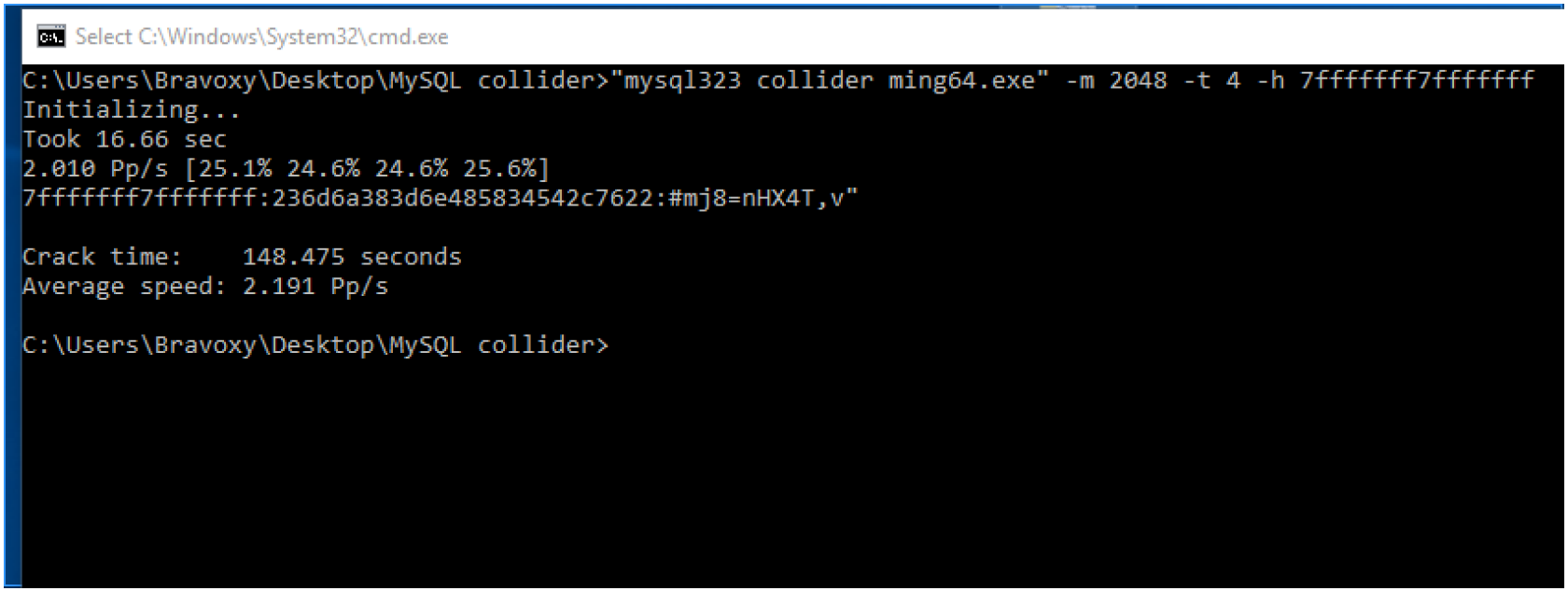
In order to find collision one have to reverse the following function:

```
h1[0] = 0x50305735;
h2[0] = 0x12345671;
sum[0] = 7;

h1[i+1] = h1[i] ^ (((h1[i] & 63) + sum[i]) *
password[i] + (h1[i] << 8));
h2[i+1] = h2[i] + ((h2[i] << 8) ^ h1[i+1]);
sum[i+1] = sum[i] + ch[i];
```

”sum” and ”password” are unknown.

MySQL 3.23 hash Collision

POC windows binary from tobtu.com:



So we have ENCRYPT(**#mj8=nHX4T,v”**)=7fffffff7fffffff.

Data Breaches relating to passwords

Most common usage of hash functions: password storage

- **September 2014:** 4.93 million Gmail usernames and passwords leaked to a Russian Bitcoin forum.
- **March 2014:** 145 million eBay accounts were compromised in massive hack including email addresses and encrypted passwords.
- **September 2013:** Over 150 million breached records from Adobe have surfaced online.
- **April 2013:** 50 million usernames and passwords lost as LivingSocial site was hacked.

Analyzing cleartext passwords

Analysis of 32 million cleartext password leaked from RockYou shows the following character type distributions:

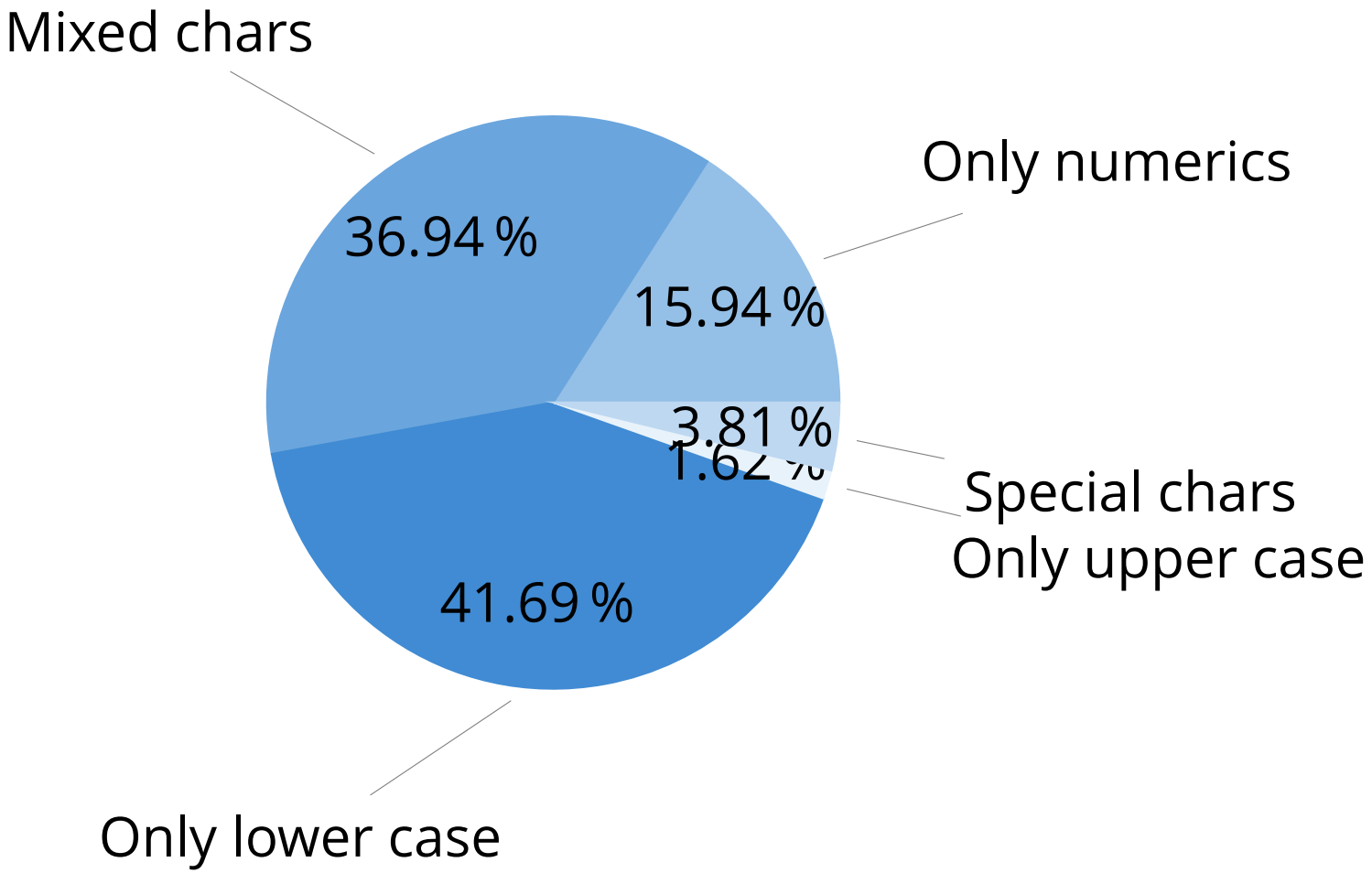
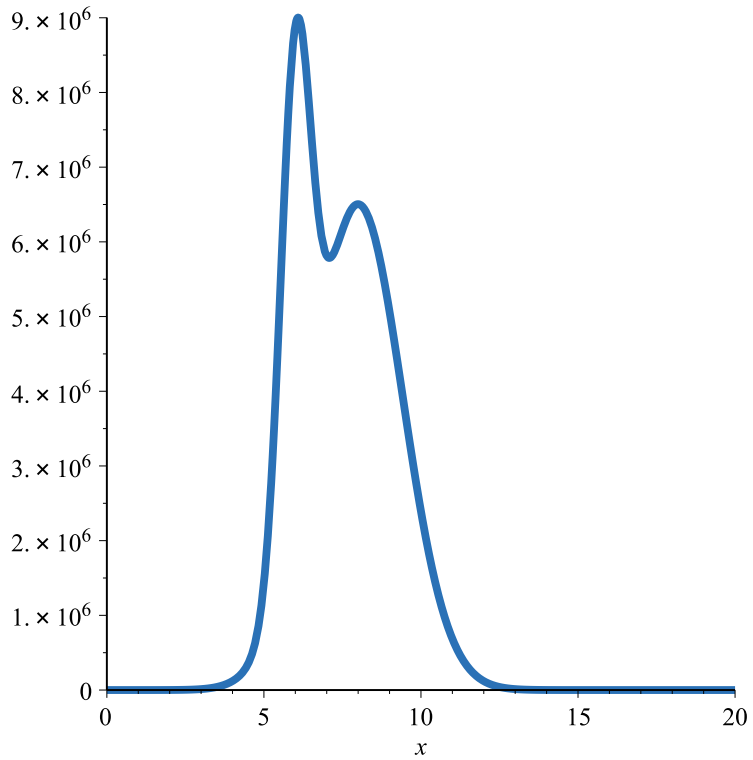


Figure: RockYou password character type distribution

Password length distribution

The password length distribution of RockYou Database:



Letter frequency distribution

Database	The first 10 most common letters
Google	$a, e, 1, i, n, r, o, s, 2, l$
Yahoo	$a, e, i, o, 1, r, n, s, l, t$
RockYou	$a, e, 1, i, o, n, r, l, s, 0$

Table: Letter frequencies in various databases.

Letter frequency distribution

It can be seen that the letter frequency distribution is similar to some inverse logarithmic distribution ($1 / \log N$).

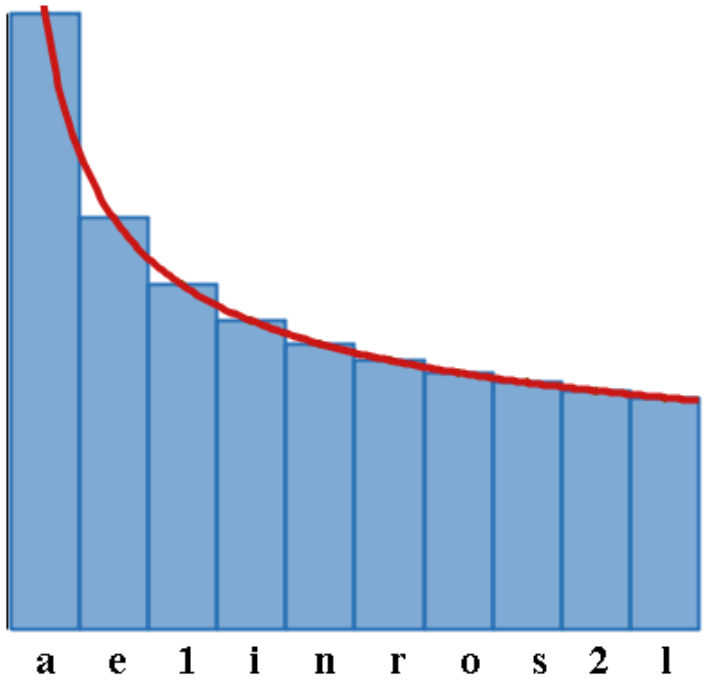


Figure: Letter frequency distribution

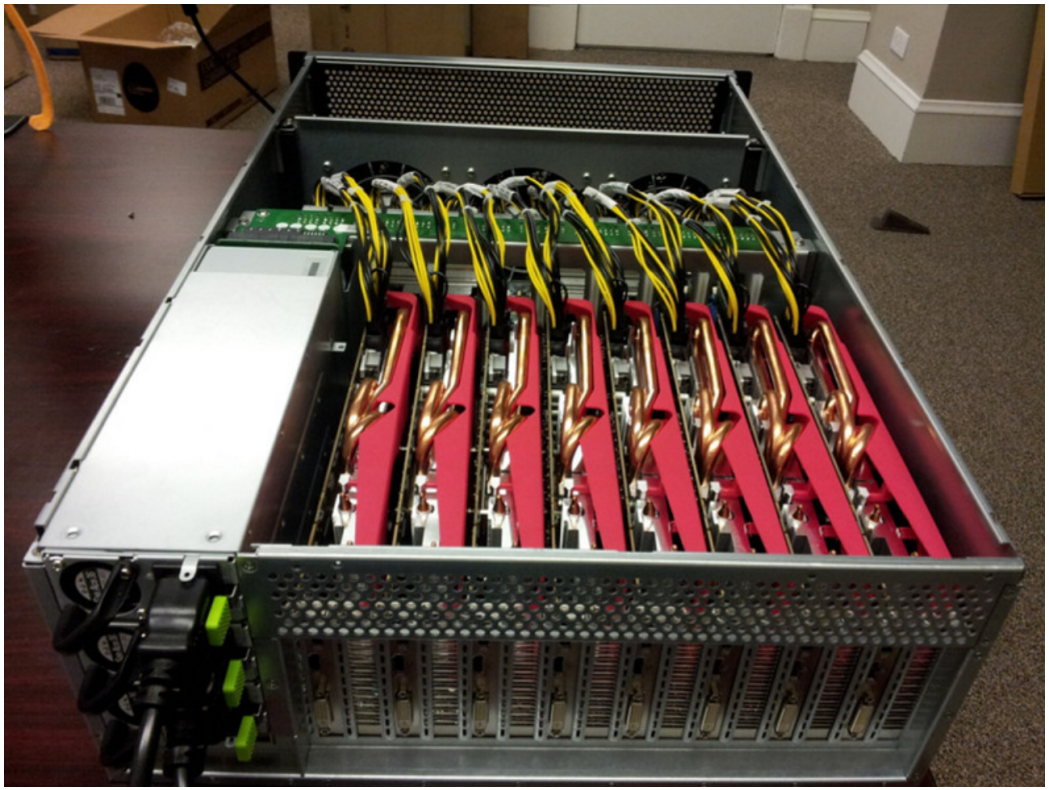
A sophisticated tool for Password cracking: hashcat

hashcat: World's fastest and most advanced GPGPU-based password recovery utility.

- Free and Open-source
- Multi-Hash (up to 100 million hashes)
- Multi-GPU (up to 128 gpus)
- Linux,Windows
- **rule-based** engine (Very fast Rule-engine)
- Able to work in an distributed environment
- **More than 300+ algorithms implemented**

Dedicated 25 GPU Monster

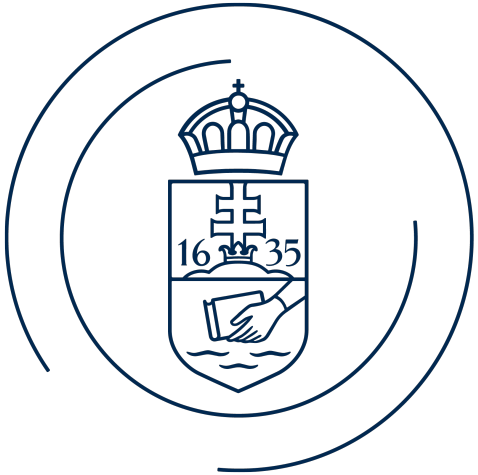
”25 GPUs brute-force 348 billion NTLM hashes / second”



Thank you for your attention!

ntihanyi@inf.elte.hu

@TihanyiNorbert



ELTE
EÖTVÖS LORÁND
UNIVERSITY

Cryptography and security -Practice (IPM-18sztKVSZKRBG)

Dr. Norbert Tihanyi

OSCP, OSCE, OSWP, CRTP, CEH, ECES , ISO27001 LA

ntihanyi@inf.elte.hu

@TihanyiNorbert

Module 0x05
Symmetric-key encryption : Block ciphers

Symmetric-key encryption

Definition

Symmetric-key encryption is a type of encryption where only one key (a secret key) is used to both encrypt and decrypt electronic information.

- Anyone who knows the secret key can decrypt the secret message.
- With symmetric-key encryption, the encryption key can be calculated from the decryption key, and vice versa.

Stream ciphers and Block ciphers

Symmetric-key encryption can use either **stream ciphers** or **block ciphers**.

Definition (Block ciphers)

Block ciphers take a number of bits (typically 64, 128 or 256 bits long) and encrypt them as a single unit, padding the plaintext so that it is a multiple of the block size.

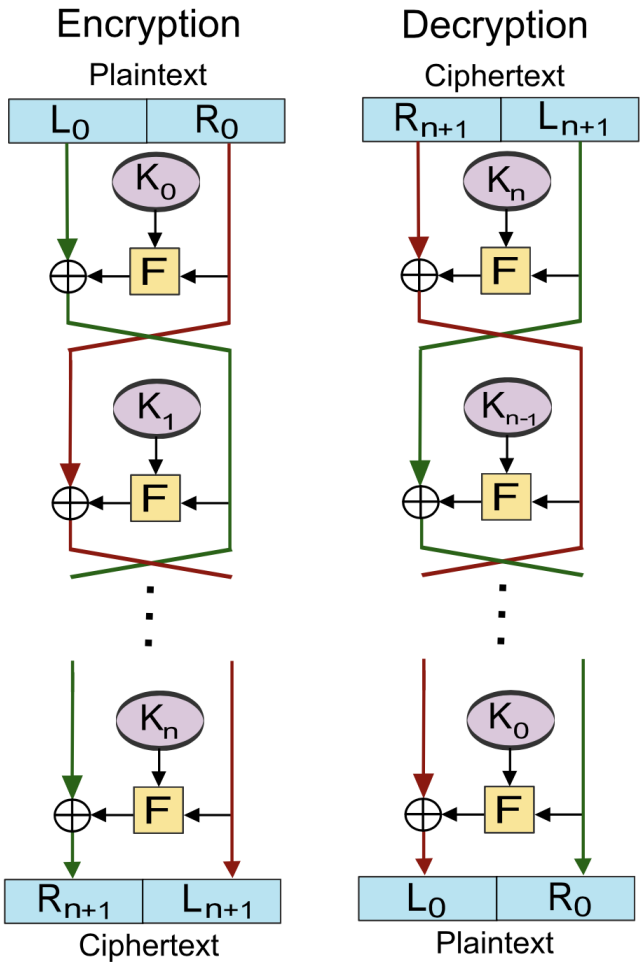
Definition (Stream ciphers)

Stream ciphers encrypt the digits (typically bytes), or letters (in substitution ciphers) of a message one at a time

Data Encryption Standard - DES

- 1976: DES is approved as a standard, first published in 1977
- based on an earlier design by Horst Feistel
- block size: 64 bits
- key size: 56 bits
- Today DES considered as an insecure algorithm
- SciEngines reported on July 15th 2008 to break DES in an average time of a single day using a setup of COPACOBANA.
- In 2016 the Open Source password cracking software **hashcat** added in DES brute force searching on general purpose GPUs.
- Systems have been built with eight GTX 1080 Ti GPUs which can recover a key in an average of under 2 days
- crack.sh : 48 Xilinx Virtex-6 LX240T FPGAs. - 26 hours

Feistel-network



The entire operation is guaranteed to be invertible (that is, encrypted data can be decrypted), **even if the round function is not itself invertible**.

Advanced Encryption Standard - AES

- AES was announced by National Institute of Standards and Technology (NIST) on November 26, 2001.
- Originally published as Rijndael (Joan Daemen and Vincent Rijmen)
- FIPS PUB 197: Advanced Encryption Standard (AES)
- block size: 128 bits
- three different key lengths: 128, 192 and 256 bits.
- Today AES considered as a **secure** algorithm
- For AES-128, the key can be recovered with a computational complexity of $2^{126.1}$ using the biclique attack. (this is not practical attack)
- Related-key attacks can break AES-256 and AES-192 with complexities $2^{99.5}$ and 2^{176} in both time and data, respectively.

Finite field- Galois field

Definition (Finite field)

A finite field or **Galois field** is a field that contains a finite number of elements.

The finite field with p^n elements is denoted $GF(p^n)$. The notation $GF(5)$ means we have a finite field with the integers $\{0, 1, 2, 3, 4\}$.
Example in $GF(5)$:

$$\texttt{Addition:}\ (3+4) \mod 5 = 2 \tag{1}$$

$$\texttt{Substraction:}\ (4-2) \mod 5 = 2 \tag{2}$$

$$\texttt{Multiplication:}\ (2*4) \mod 5 = 3 \tag{3}$$

Zero divisor: $a * b = 0$ and $a, b \neq 0$

$$\texttt{Multiplication:}\ (5*2) \mod 5 = 0 \tag{4}$$

Rijndael's (AES) finite field

AES uses the characteristic 2 finite field with 256 elements: $GF(2^8)$.
AES employs the folllowing reducing polynomial for multiplication:

$$x^8 + x^4 + x^3 + x + 1 \tag{5}$$

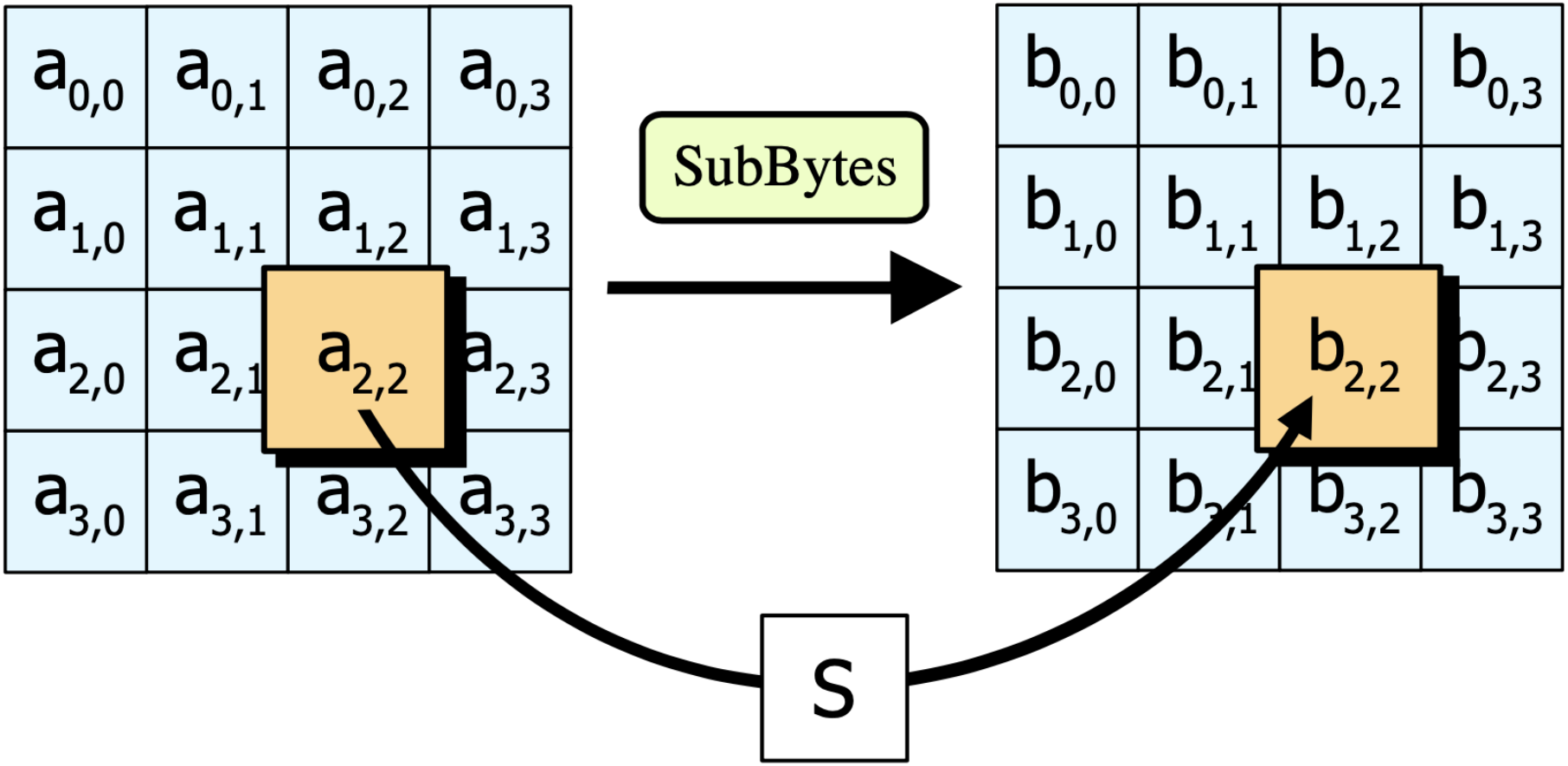
Multiplication in $GF(2^8)$ example: $0x53 \cdot 0xCA =?$

$$53_{16} = 83_{10} = (2^6 + 2^4 + 2 + 1) = (x^6 + x^4 + x + 1)$$
$$CA_{16} = 202_{10} = (2^7 + 2^6 + 2^3 + 2) = (x^7 + x^6 + x^3 + x)$$
$$53_{16} \cdot CA_{16} = (x^6 + x^4 + x + 1)(x^7 + x^6 + x^3 + x) =$$
$$x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x$$
$$\text{mod } x^8 + x^4 + x^3 + x + 1 =$$
$$3F7E_{16} \mod 11B_{16} = 1$$

So in Rijndael's field the multiplication $83 \cdot 202 = 1$.

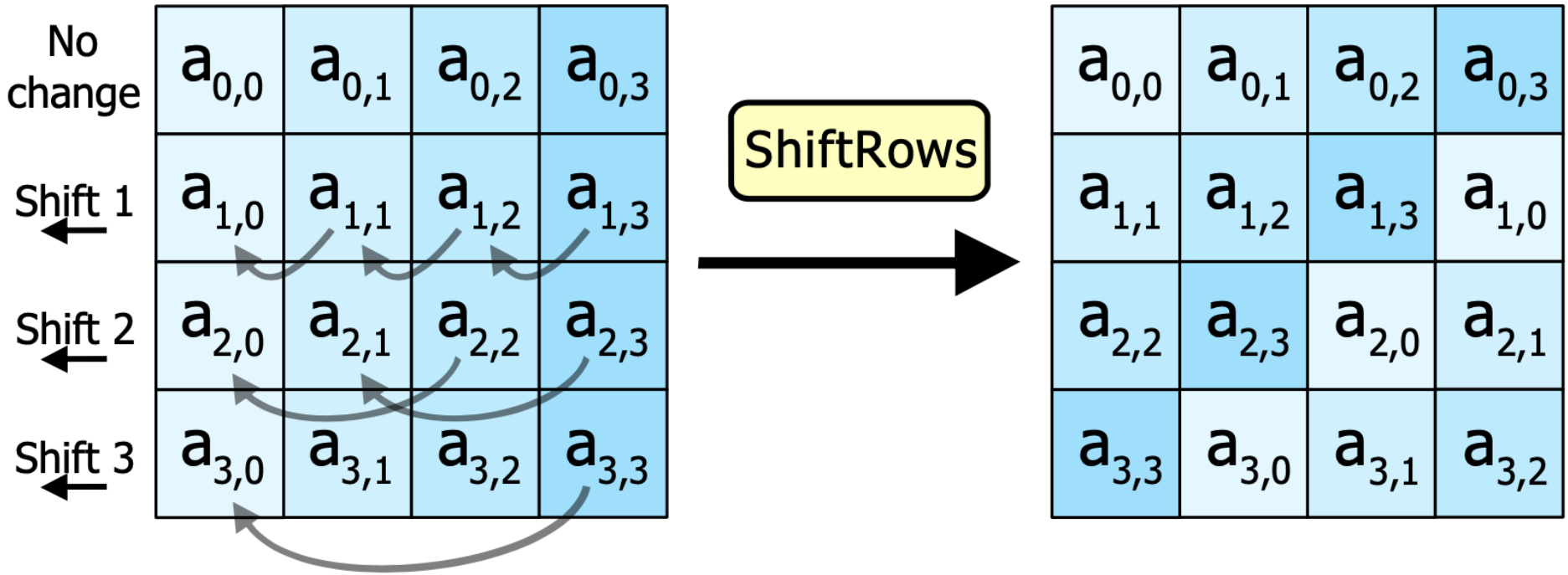
The AES SubBytes step

In the SubBytes step, each byte $a_{i,j}$ in the state array is replaced with a SubByte $S(a_{i,j})$ using an 8-bit substitution box. **This operation provides the non-linearity in the cipher.**



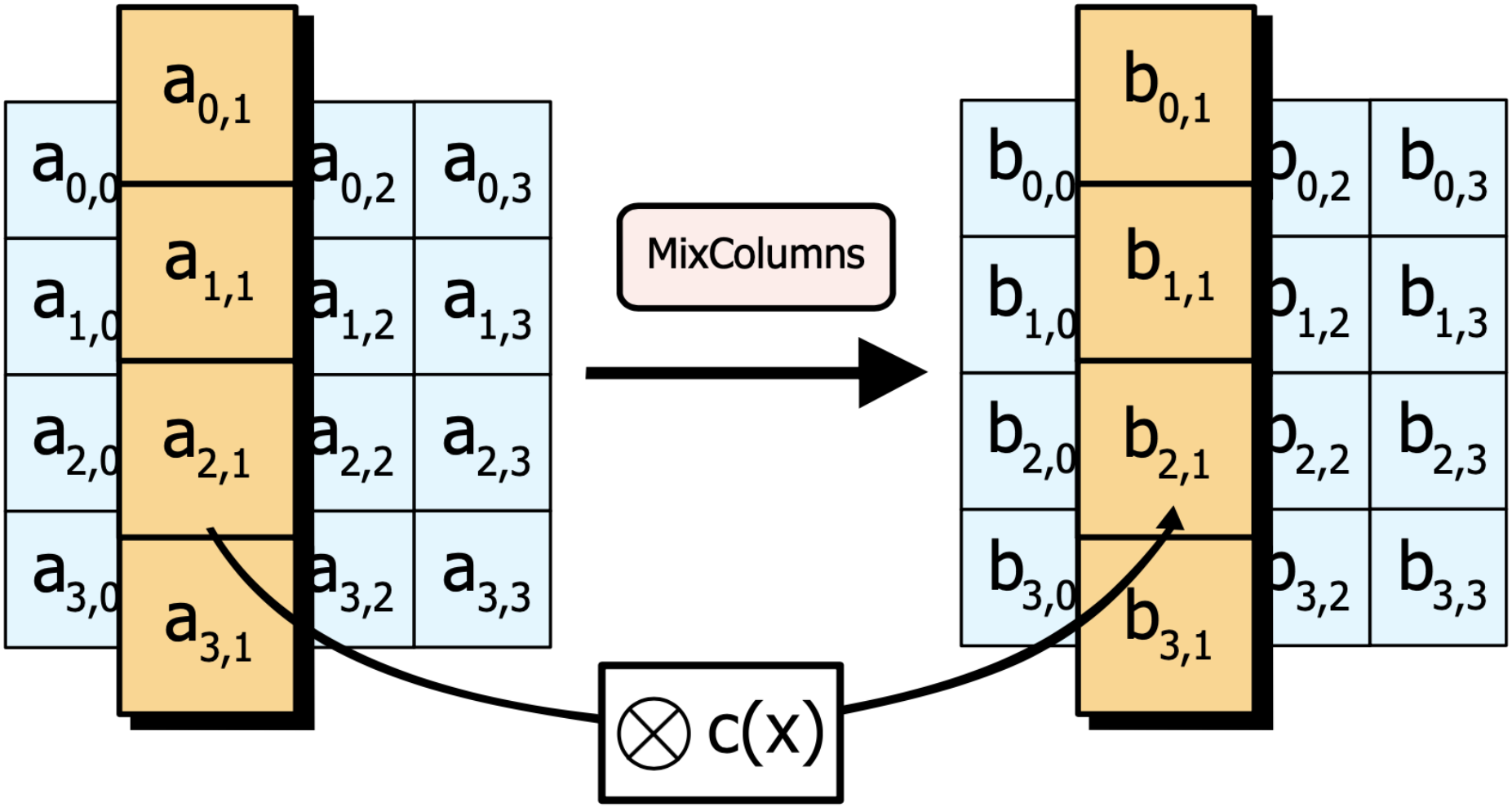
The AES ShiftRows step

The ShiftRows step operates on the rows of the state; it cyclically shifts the bytes in each row by a certain offset.



The AES MixColumns step

In the MixColumns step, the four bytes of each column of the state are combined using an invertible linear transformation. **Together with ShiftRows, MixColumns provides diffusion in the cipher.**



AES is unbreakable as of 2021

AES is considered a secure algorithm as of 2021. There is no known practical attack that would allow someone without knowledge of the key to read data encrypted by AES when **correctly implemented**.

Shannon's properties

Confusion and **diffusion** are two properties of the operation of a secure cipher described by **Claude E. Shannon** in 1945.

Definition (Confusion)

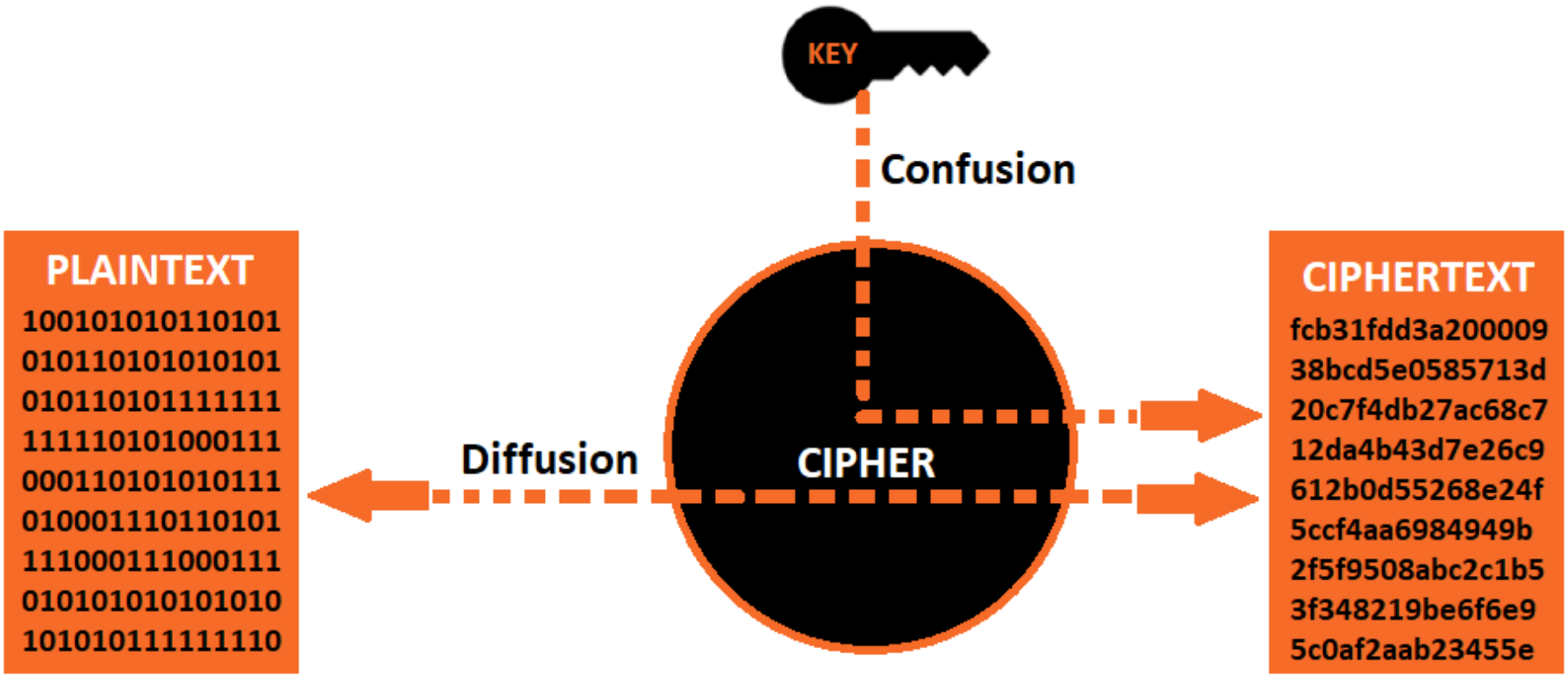
Confusion means that each binary digit (bit) of the ciphertext should depend on several parts of the key, obscuring the connections between the two.

Definition (Diffusion)

Diffusion means that if we change a single bit of the plaintext, then (statistically) half of the bits in the ciphertext should change, and similarly, if we change one bit of the ciphertext, then approximately one half of the plaintext bits should change.

Confusion & Diffusion

- The idea of diffusion is to **hide the relationship between the ciphertext and the plain text**.
- The property of confusion **hides the relationship between the ciphertext and the key**



Linear cryptanalysis since 1992

- Linear cryptanalysis described by **Mitsuru Matsui** who first applied the technique to the **FEAL cipher** in EUROCRYPT '92.
- Matsui also published an attack on **DES**, eventually leading to the first experimental cryptanalysis of the cipher in 1993.
- In 2021 linear cryptanalysis is still the **best known attack against DES** which requires 2^{43} known plaintexts.
- Linear cryptanalysis is a **known-plaintext** attack where the attacker has access to both the plaintext and its encrypted version.

Principle of Linear Cryptanalysis

From the original Matsumi Paper:

Definition (Linear Cryptanalysis)

The purpose of Linear Cryptanalysis is to find the following "effective" linear expression for a given cipher algorithm:

$$P[i_1, i_2, \cdots, i_a] \oplus C[j_1, j_2, \cdots, j_b] = K[k_1, k_2, \cdots, k_c] \tag{6}$$

where $i_1, i_2, \cdots, i_a, j_1, j_2, \cdots, j_b$ and $k_1, k_2, \cdots, k_c$ denote fixed bit locations, and equation (1) holds with probability $p \neq \frac{1}{2}$ for randomly given plaintext P and the corresponding ciphertext C.

$$\left(\bigoplus_{i \in \{1 \ldots a\}} \mathcal{P}^{(i)} \right) \oplus \left(\bigoplus_{j \in \{1 \ldots b\}} \mathcal{C}^{(j)} \right) = \bigoplus_{k \in \{1 \ldots c\}} \mathcal{K}^{(k)} \tag{7}$$

Linear probability bias

$$\left(\bigoplus_{i\in\{1\ldots128\}} \mathcal{P}^{(i)}\right) \oplus \left(\bigoplus_{j\in\{1\ldots128\}} \mathcal{C}^{(j)}\right) = \bigoplus_{k\in\{1\ldots256\}} \mathcal{K}^{(k)} \tag{8}$$

For a perfect 128-bit block cipher with a 256-bit key all combination of equation (3) will hold with $p = \frac{1}{2}$ for a randomly given plaintext P and the corresponding ciphertext C.
The magnitude

$$\epsilon = \left| p - \frac{1}{2} \right| \tag{9}$$

represents the effectiveness of the linear expression.
The main goal of linear cryptanalysis is to find the best linear expression which holds with the bigger bias ϵ.

Example for ϵ-bias

Definition ($\epsilon - bias$)

The higher the magnitude of the probability bias $\epsilon = \left|p - \frac{1}{2}\right|$, the better the applicability of linear cryptanalysis with fewer known plaintexts required in the attack.

Example for ϵ-bias

Consider the following linear expression:

$$P_2 \oplus P_4 \oplus P_{11} \oplus C_3 \oplus C_7 = K_{14} \tag{10}$$

Let's assume that the equation (5) will yield 1 with probability $p = \frac{1}{2} + \epsilon$ where $\epsilon = 1.23 \times 2^{-9}$.

NOTE: No obvious linearity such as above should hold for all input and output values or the cipher would be trivially weak.

If you do the XOR with many plaintexts and corresponding ciphertexts, you will get 1 a little more often that you will get 0. Collecting many (Plaintext,Ciphertext) pairs one can guess the 14^{th} bits of the key.

SPN-cipher

Definition (SPN-cipher)

an SP-network, or substitution–permutation network (SPN), is a series of linked mathematical operations used in block cipher algorithms.

Some well-known algorithms using SPN structure:

- AES
- PRESENT
- SAFER
- SHARK
- Squark
- 3-way

Matsui's Piling-up lemma

- The piling-up lemma is a principle used in linear cryptanalysis to construct linear approximation.
- It was introduced by Mitsuru Matsui (1993) as an analytical tool for linear cryptanalysis.
- The piling-up lemma allows the cryptanalyst to determine the probability that the equality:

$$X_1 \oplus X_2 \oplus X_3 \cdots X_n = 0 \tag{11}$$

holds where the X's are binary variables.

Example for encrypting

Let the plaintext $\mathcal{P}$ and the secret $\mathcal{K}$ key are the following ones :

$$\mathcal{P} = 1000_{10} = 0000001111101000_2 = 0x3E8_{16} \quad (12)$$

$$K = 0x0D1125D4C4816F1B7E9E_{16} \quad (13)$$

ROUND 1 steps:

1. 0x03E8 $\oplus$ 0x0D11 = 0xEF9
2. S-BOX(0xEF9) = 0xE07A
3. P-BOX(0xE07A) =0x9AB2

# ROUND	XOR	S-BOX	P-BOX
1	0x0EF9	0xE07A	0x9AB2
2	0xBF66	0xC7BB	0xBC77
3	0x78F6	0x837B	0x9277

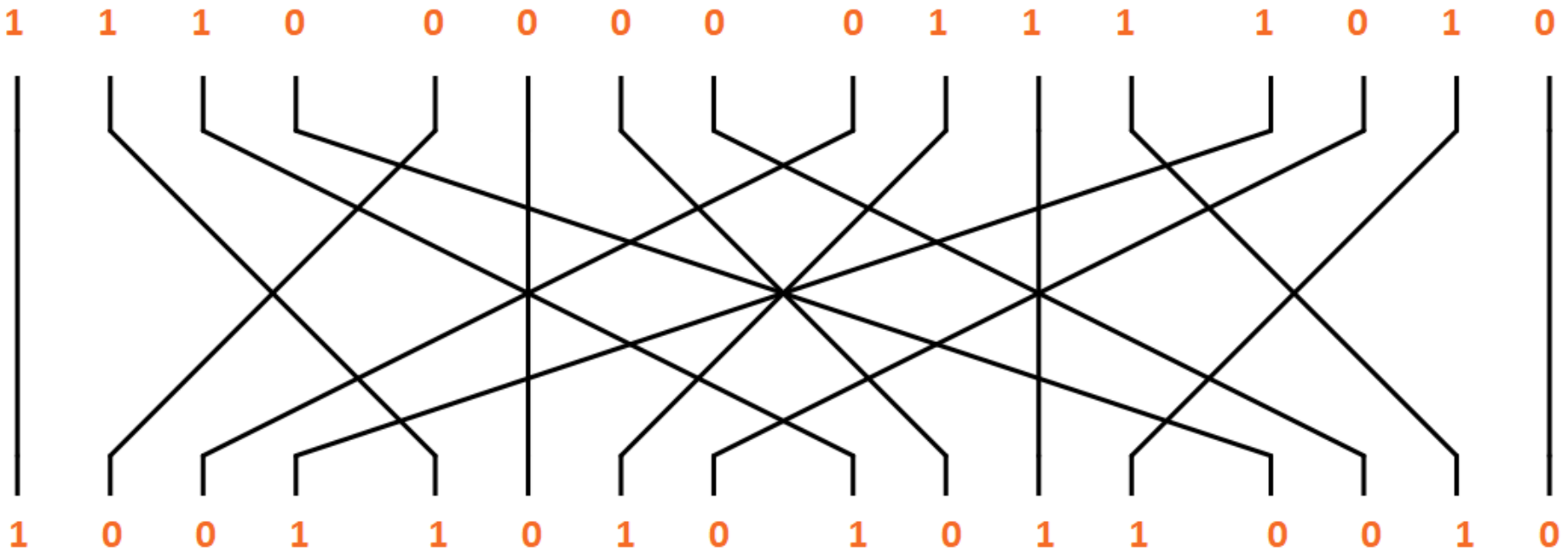
Permutation BOX

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15

Example: P-BOX(0xE07A) $\rightarrow$ 0x9AB2

$0xE07A_{16} = 1110000001111010_2 \rightarrow 1110000001111010_2 = 0x9AB2_{16}$

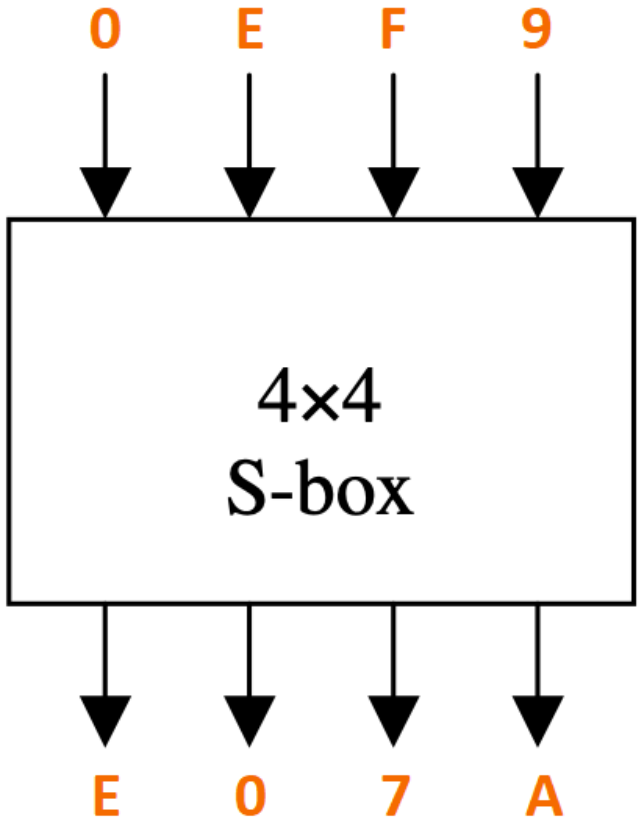
Visualization of the permutation box:



Substitution BOX

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7

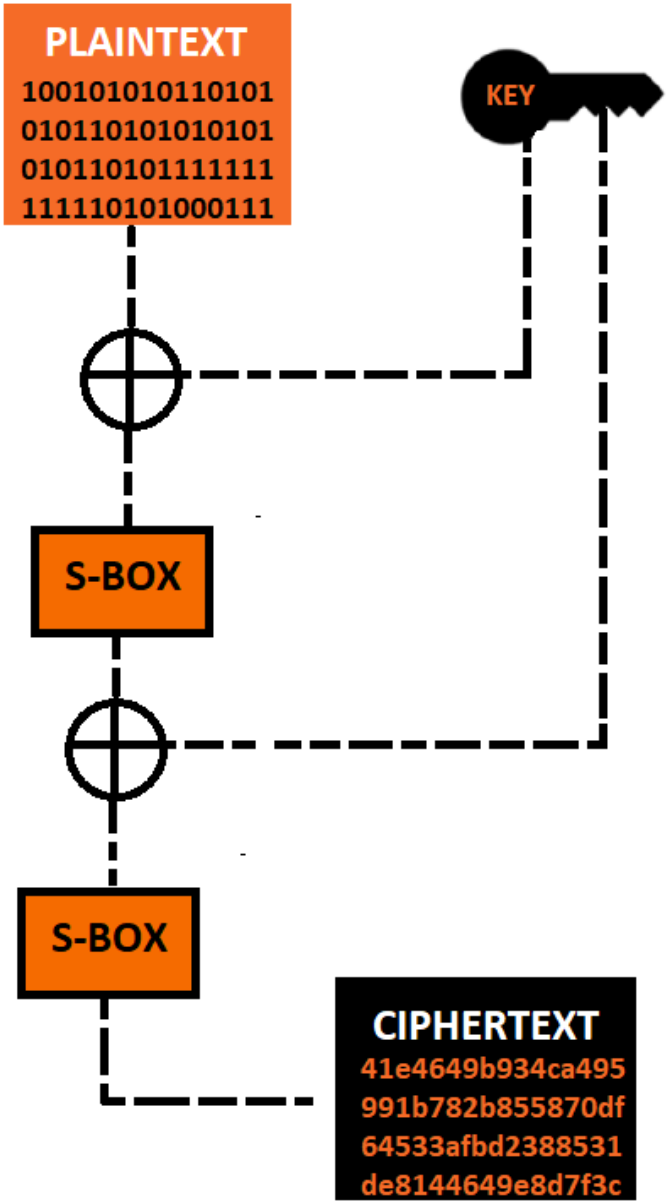
Example: S-BOX(0x0EF9) $\rightarrow$ 0xE07A
Visualization of the substitution box:



Substitution BOX

1. Calculate Linear Approxiumation Table (LAT)
2. Find the best linear approximation to the S-BOX

Substitution BOX



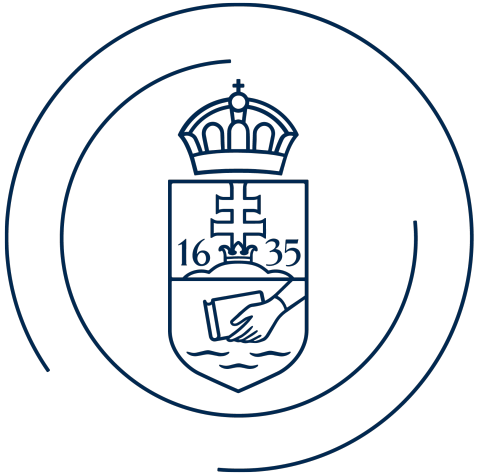
Round -4 Cryptanalysis

————————ROUND1————————
XOR: 0xEF9
S-BOX: 0xE07A
P-BOX: 0x9AB2
————————ROUND2————————
XOR: 0xEF9
S-BOX: 0xE07A
P-BOX: 0x9AB2
————————ROUND3————————
XOR: 0xEF9
S-BOX: 0xE07A
P-BOX: 0x9AB2
————————ROUND4————————
XOR: 0xEF9
S-BOX: 0xE07A
XOR: 0x9AB2

Thank you for your attention!

ntihanyi@inf.elte.hu

@TihanyiNorbert



ELTE
EÖTVÖS LORÁND
UNIVERSITY

Cryptography and security -Practice (IPM-18sztKVSZKRBG)

Dr. Norbert Tihanyi

OSCP, OSCE, OSWP, CRTP, CEH, ECES , ISO27001 LA

ntihanyi@inf.elte.hu

@TihanyiNorbert

Module 0x06
Block cipher mode of operation

Symmetric-key encryption

Definition

Symmetric-key encryption is a type of encryption where only one key (a secret key) is used to both encrypt and decrypt electronic information.

- Anyone who knows the secret key can decrypt the secret message.
- With symmetric-key encryption, the encryption key can be calculated from the decryption key, and vice versa.

ECB mode

ECB = Electronic Code Book

- Simplest encryption mode
- The message is divided into blocks, and each block is encrypted separately.
- The main disadvantage of this method is a lack of **diffusion**.
- ECB encrypts identical plaintext blocks into identical ciphertext blocks.
- **It does not hide data patterns**
- It is not recommended for use in cryptographic protocols (if it is more than one block is encerypted)

Shannon's properties

Confusion and **diffusion** are two properties of the operation of a secure cipher described by **Claude E. Shannon** in 1945.

Definition (Confusion)

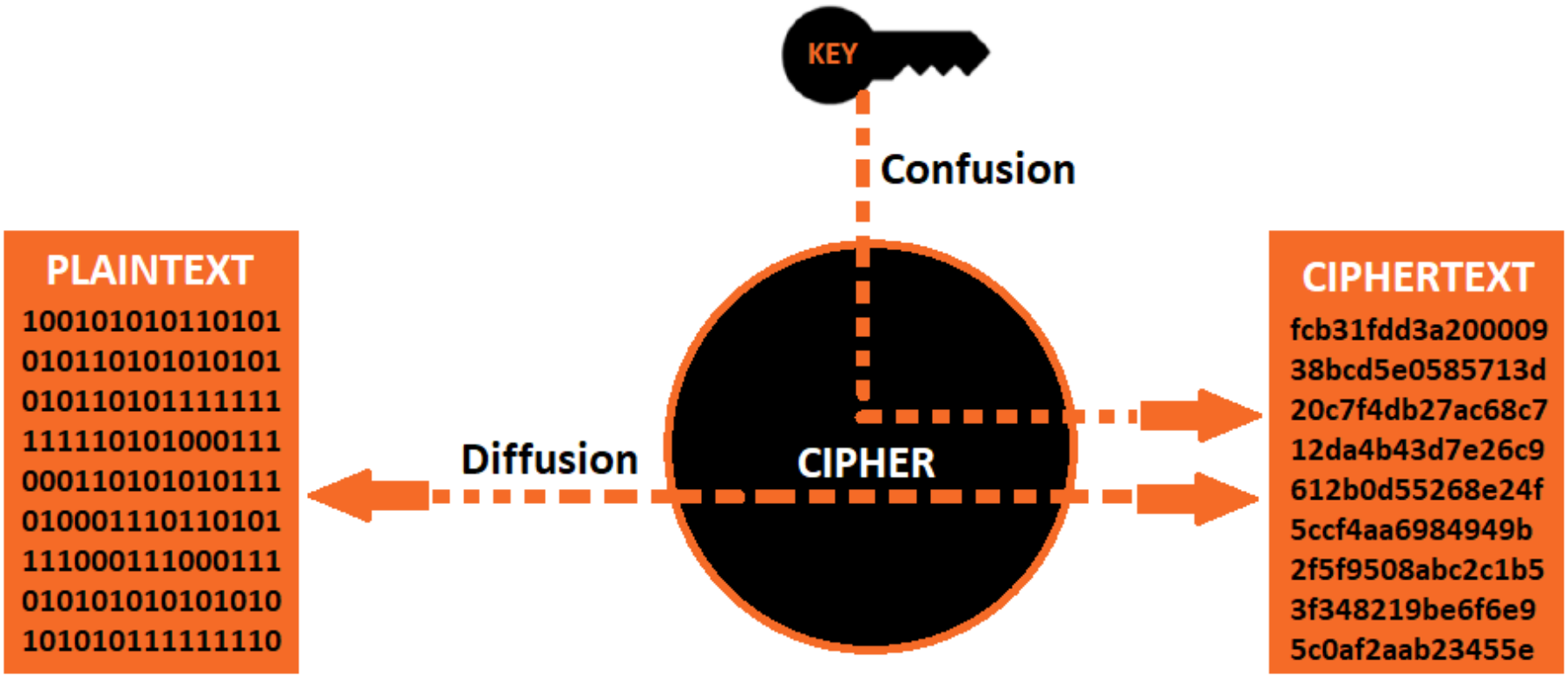
Confusion means that each binary digit (bit) of the ciphertext should depend on several parts of the key, obscuring the connections between the two.

Definition (Diffusion)

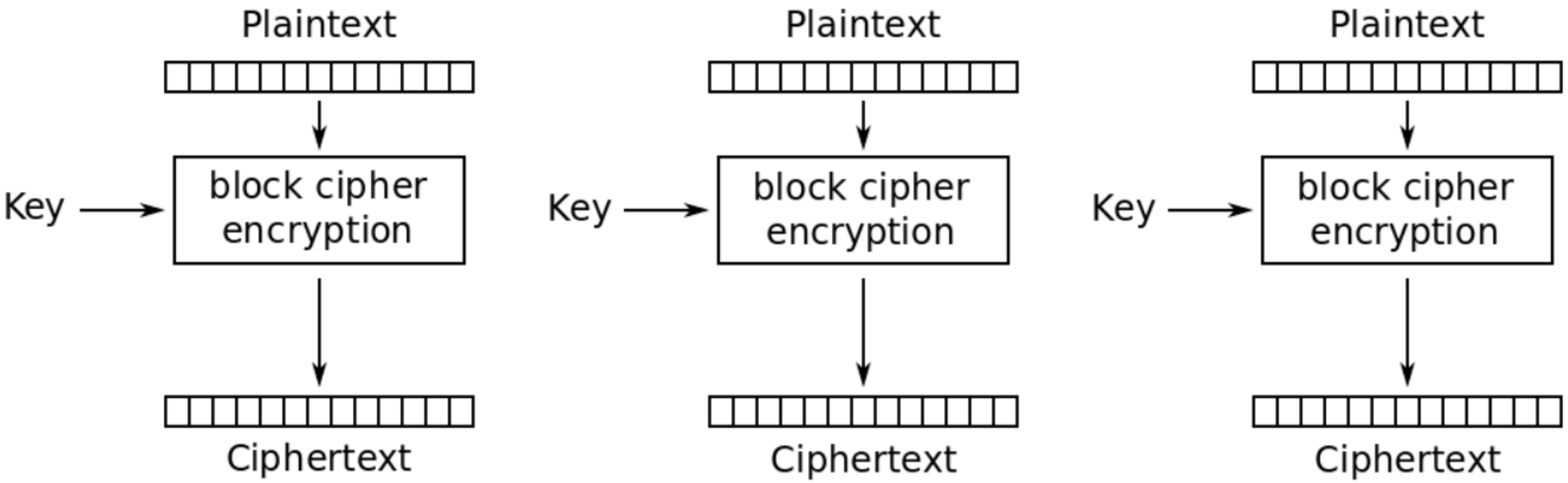
Diffusion means that if we change a single bit of the plaintext, then (statistically) half of the bits in the ciphertext should change, and similarly, if we change one bit of the ciphertext, then approximately one half of the plaintext bits should change.

Confusion & Diffusion

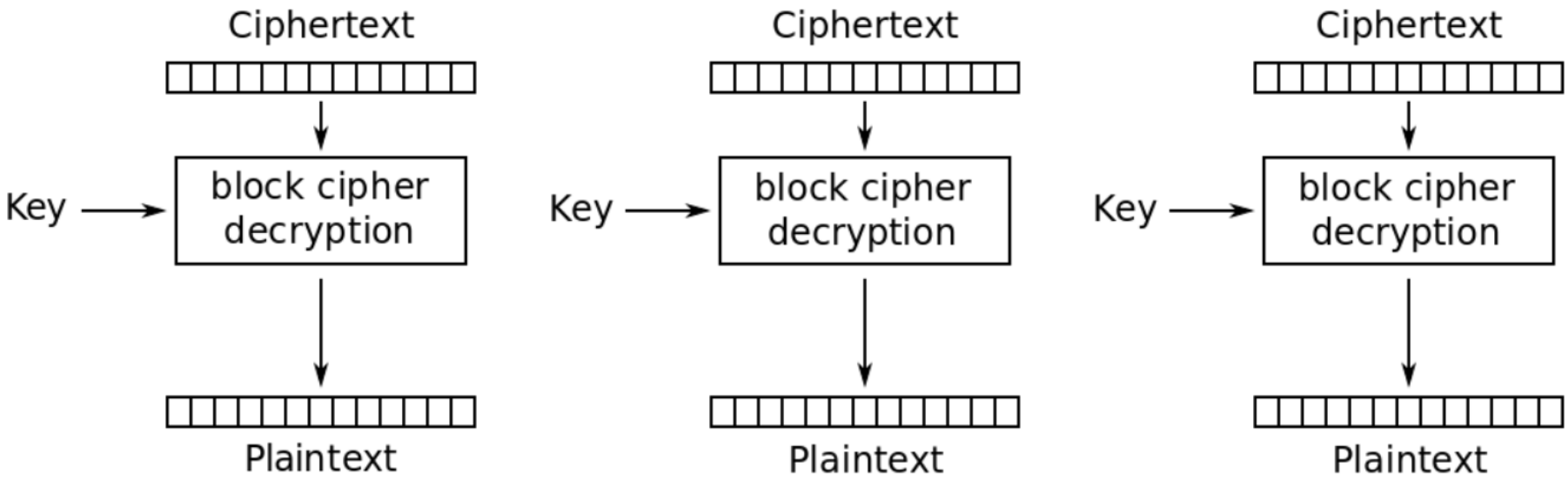
- The idea of diffusion is to **hide the relationship between the ciphertext and the plain text**.
- The property of confusion **hides the relationship between the ciphertext and the key**



ECB mode - encryption

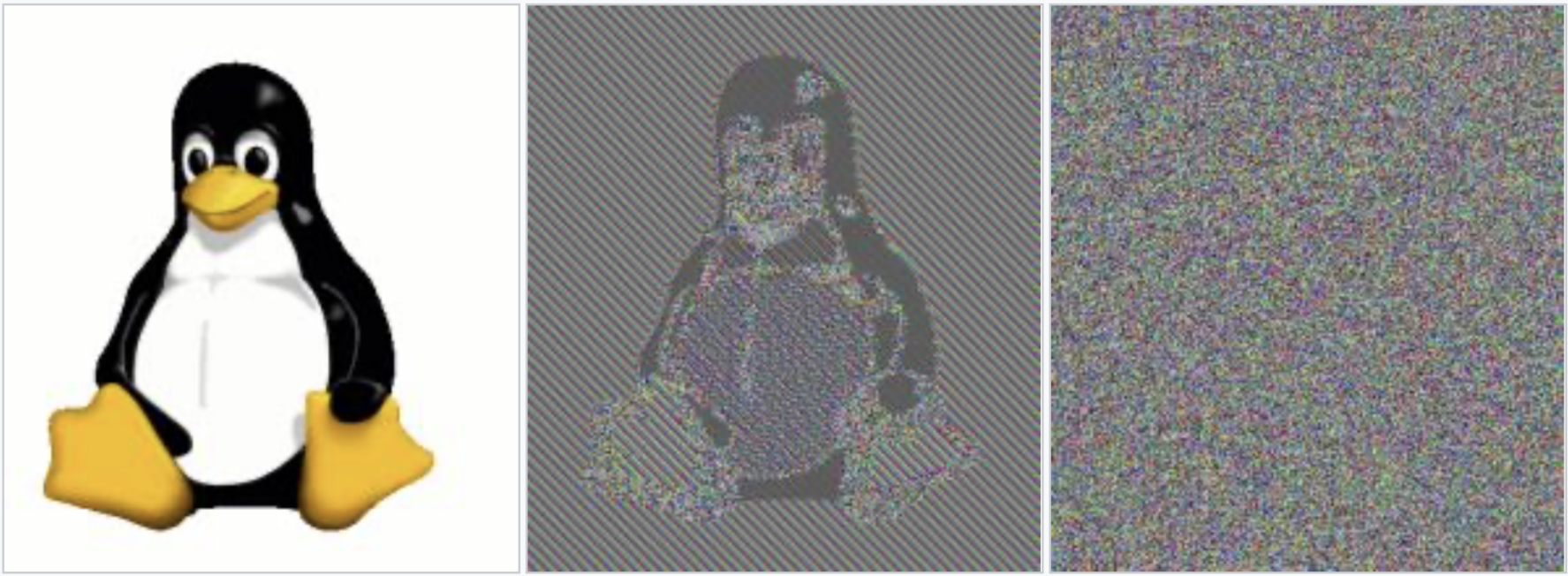


ECB mode - decryption



The ECB Penguin - ”Tux”

Lunkwill created the very famous ECB Penguin picture in 2004. The second picture is encrypted by **AES** in **ECB** mode.



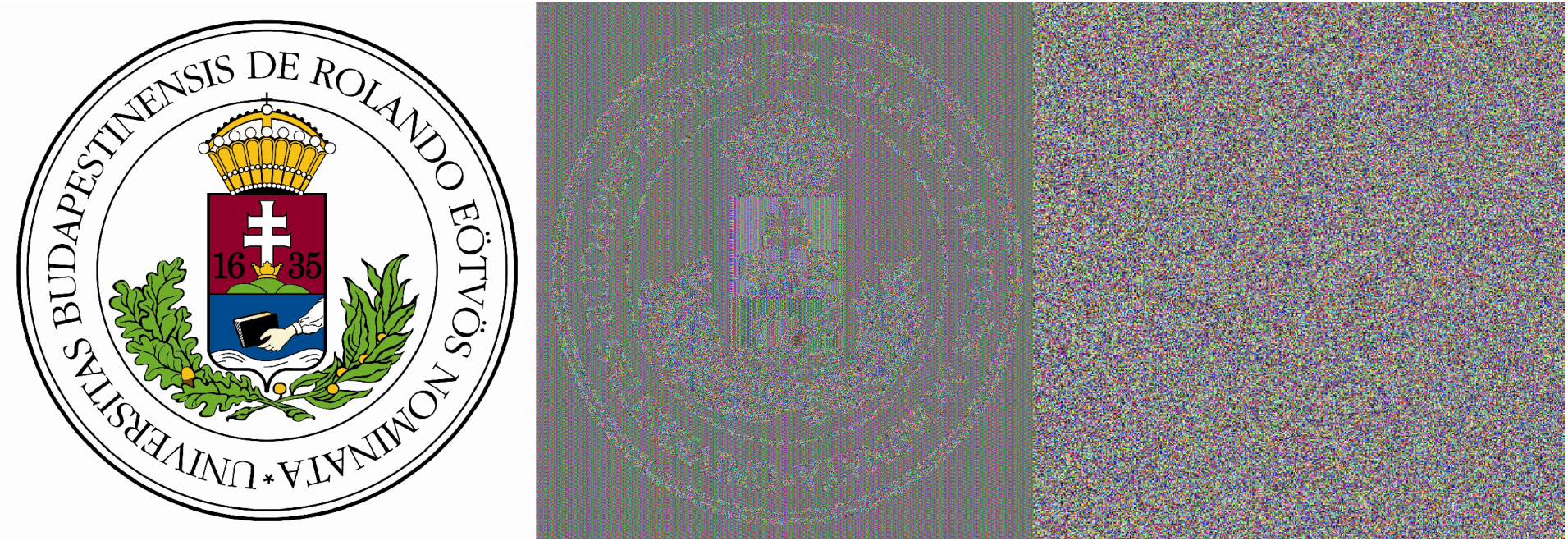
Create your own ECB encryption example

Steps to reproduce the ECB Penguin example:

1. Convert the picture to **PPM binary raw** format
2. Save the header: `head -n 4 example.ppm > header.txt`
3. Save the body :`tail -n +5 example.ppm > body.bin`
4. Encrypt the body: `openssl enc -aes-128-ecb -nosalt -pass pass:"VerylongPassword" -in body.bin -out body.ecb.bin`
5. Create the final picture: `cat header.txt body.ecb.bin > encrypted.ppm`

ECB mode example

Recreated example:



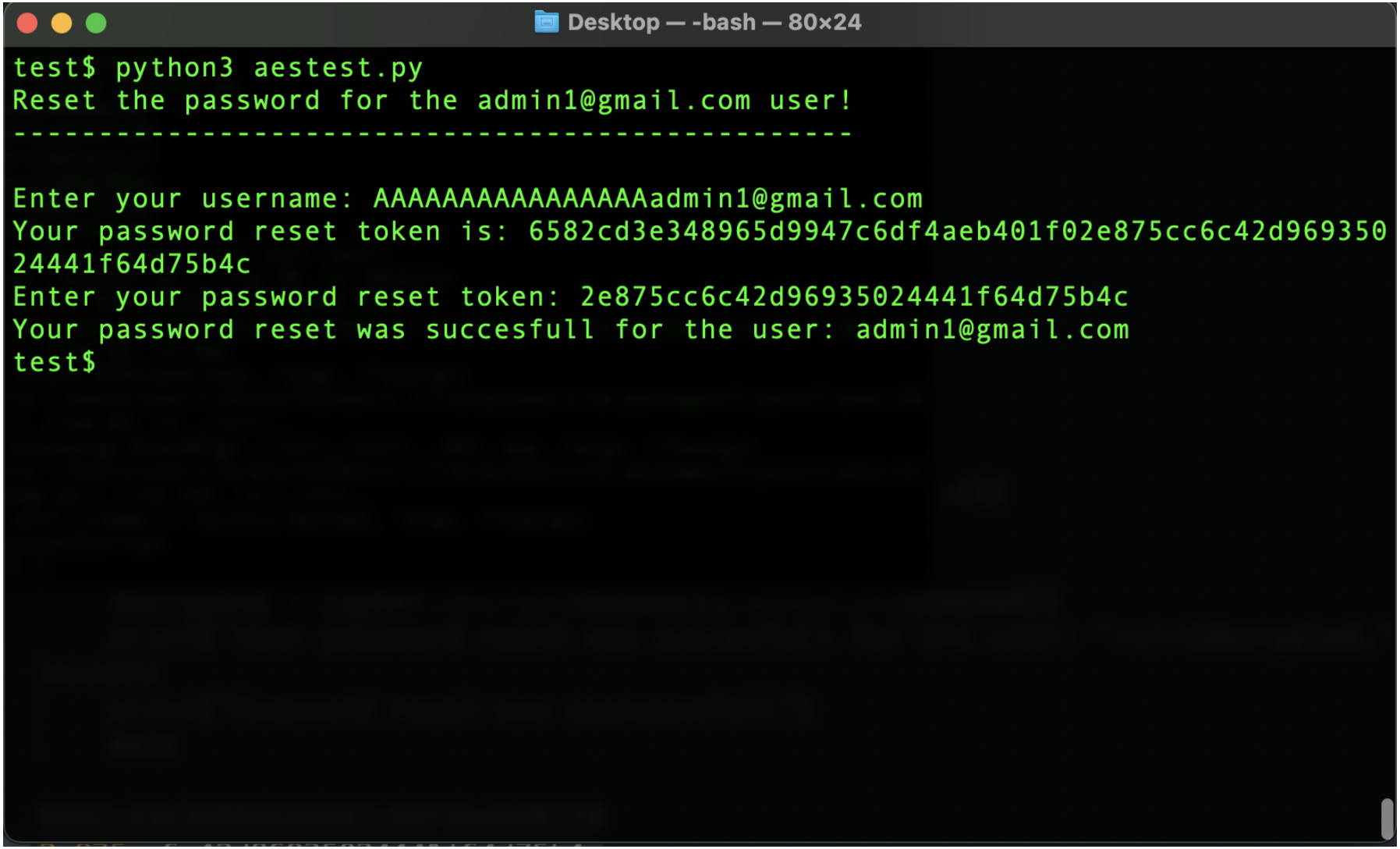
AES challenge

Try to reset the **admin1@gmail.com** acount

```
print("Reset the password for the admin1@gmail.com user!")
print("-----------------------------------------\n")
user = input("Enter your username: ")
user=user.strip()
if user[-10:]!="@gmail.com":
    print("Only @gmail.com account can be reseted")
    exit()
#restricted user
if user=="admin1@gmail.com":
    print("Not authorized to reset this user")
    exit()

padded_text=encoder.encode(user)
key=os.urandom(16)
cipher = AES.new(key, AES.MODE_ECB)
encrypted = cipher.encrypt(padded_text)
print("Your password reset token is: "+encrypted.hex())
passwd = input("Enter your password reset token: ")
try:
    decrypted = cipher.decrypt(binascii.unhexlify(passwd))
    print("Your password reset was succesfull for the user: "+str(decrypted,"utf-8").strip())
except:
    print("Password reset was unsuccesfull")
    exit
```

AES challenge - solution

ECB encrypts identical plaintext block into identical ciphertext blocks!

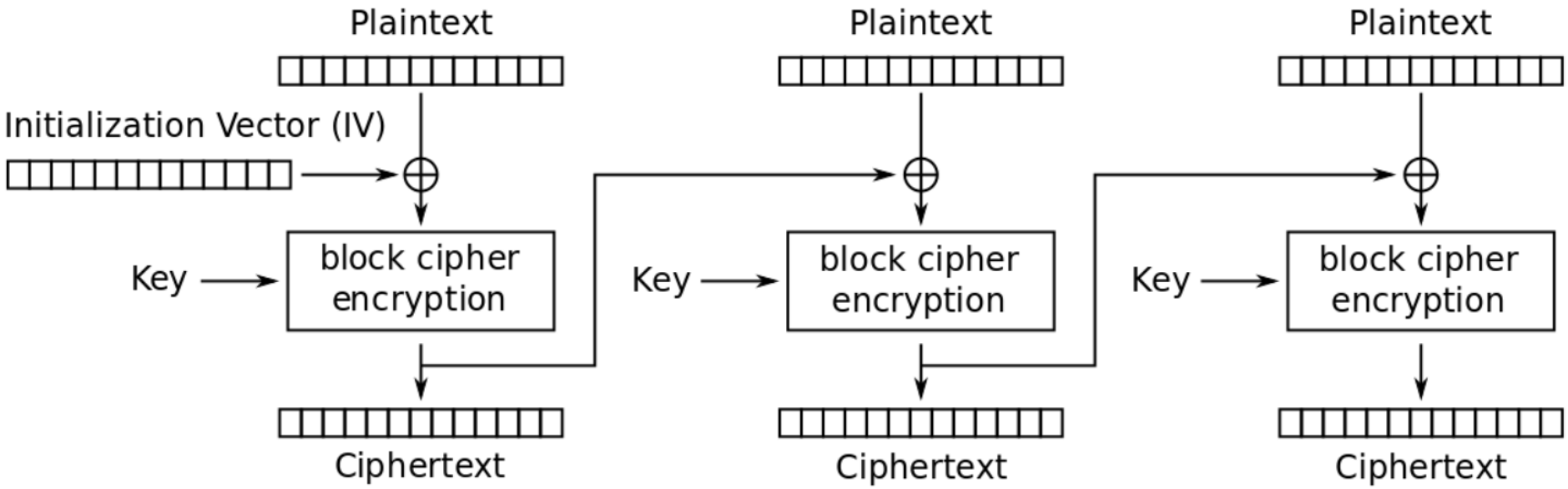


CBC mode

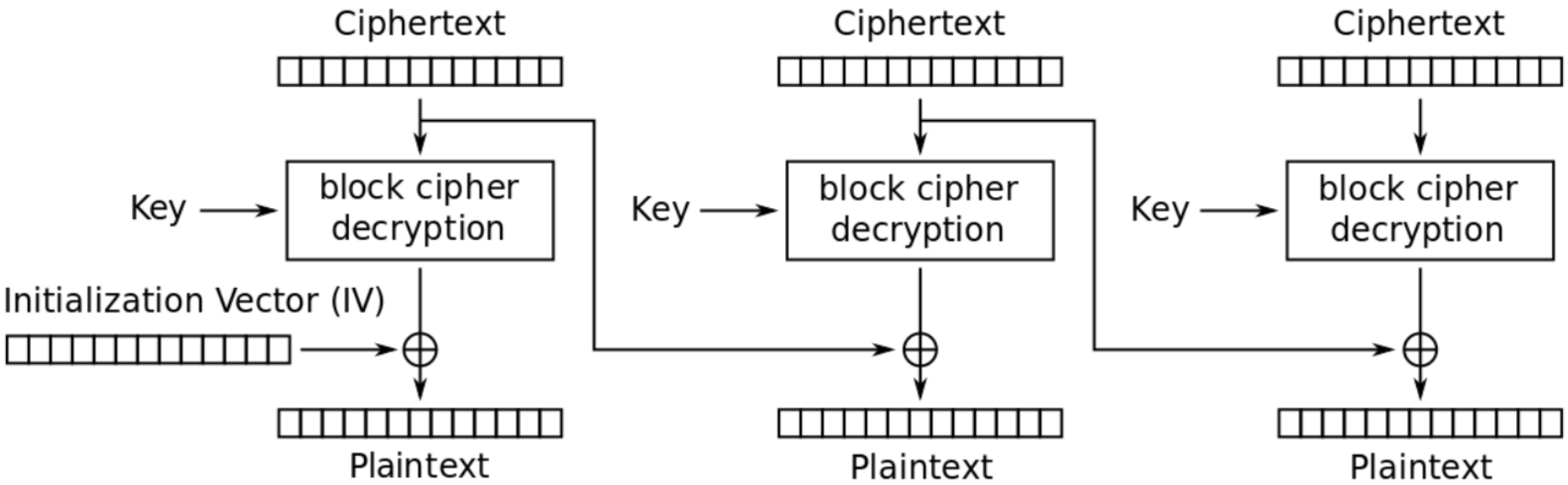
CBC = Cipher Block Chaining

- Ehrsam, Meyer, Smith and Tuchman invented CBC in 1976
- Each block of plaintext is XORed with the previous ciphertext block before being encrypted
- Each ciphertext block **depends on all plaintext blocks processed up to that point**.
- An initialization vector must be used in the first block.
- The main disadvantage of this method that **encryption cannot be parallelized**
- A plaintext block can be recovered from two adjacent blocks of ciphertext. As a consequence, **decryption** can be **parallelized**.
- Formula for encryption: $C_0 = IV, C_i = E_K(P_i \oplus C_{i-1})$
- Formula for decryption: $C_0 = IV, P_i = D_K(C_i) \oplus C_{i-1}$

CBC mode - encryption



CBC mode - decryption



Padding oracle attack

Definition

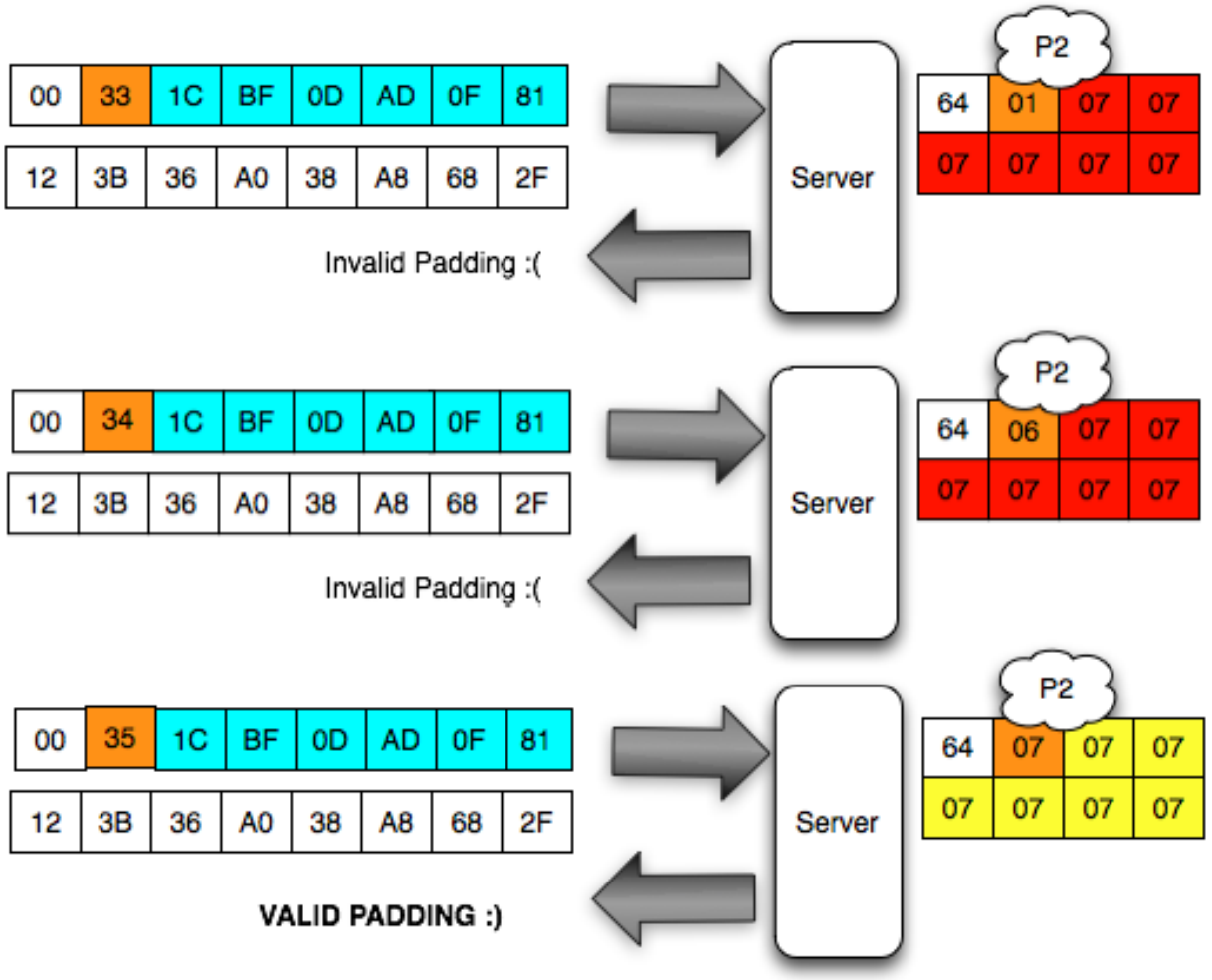
A **padding oracle attack** is an attack which uses the padding validation of a cryptographic message to decrypt the ciphertex

The original attack was published in 2002 by Serge Vaudenay.

Definition

The padding oracle attack can be applied to the **CBC mode of operation**, where the ”oracle” (usually a server) leaks data about whether the padding of an encrypted message is correct or not. Such data can allow attackers to decrypt (and sometimes encrypt) messages through the oracle using the oracle’s key, without knowing the encryption key.

Padding oracle attack

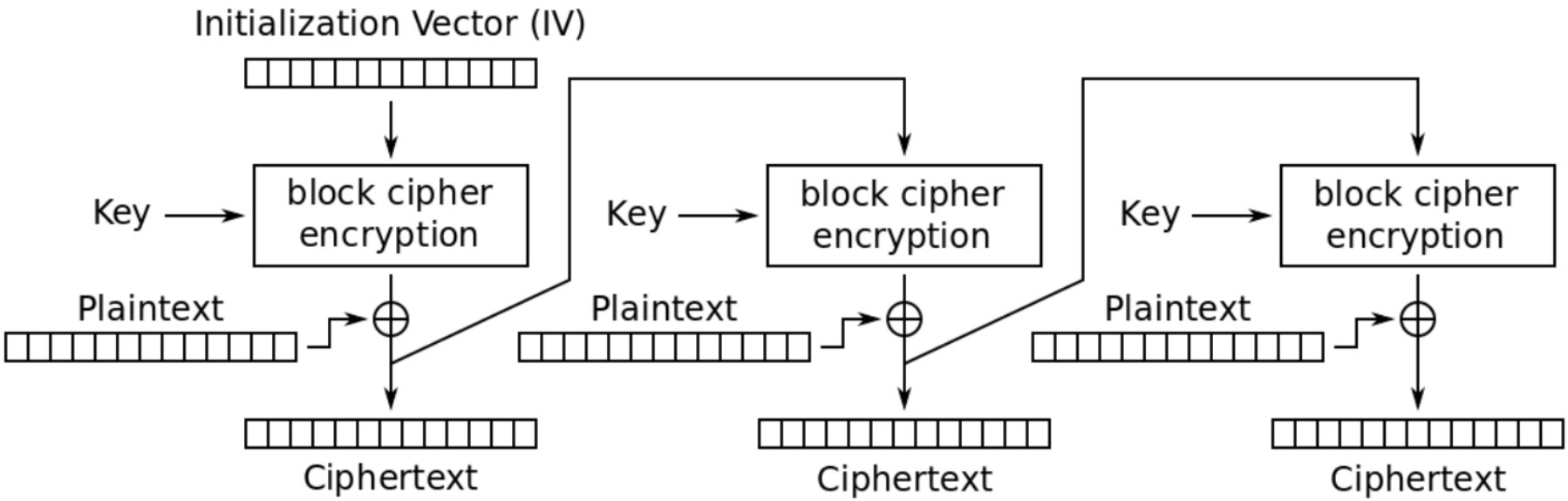


CFB mode

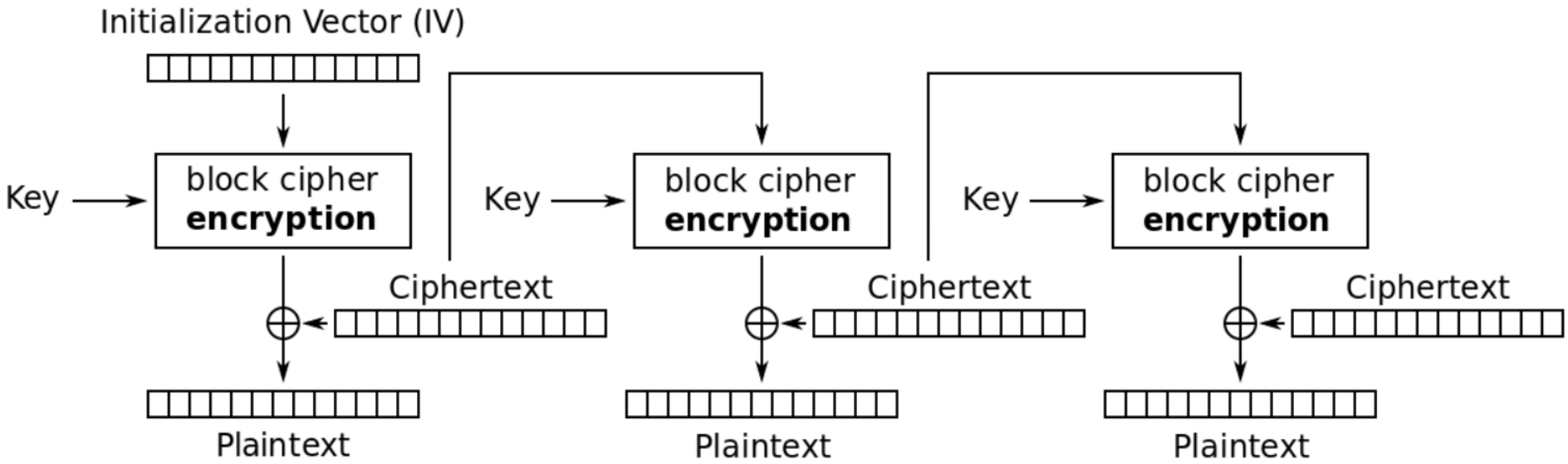
CFB = Cipher Feedback

- It is using the entire output of the block cipher.
- It is very similar to CBC, makes a block cipher into a self-synchronizing stream cipher
- The IV need not be secret, but it must be unpredictable.
- CFB decryption is almost identical to CBC encryption performed in reverse
- Formula for encryption: $C_0 = IV, C_i = E_K(C_{i-1}) \oplus P_i$
- Formula for decryption: $C_0 = IV, P_i = E_K(C_{i-1}) \oplus C_i$ (Only encryption is used !)

CFB mode - encryption



CFB mode - decryption

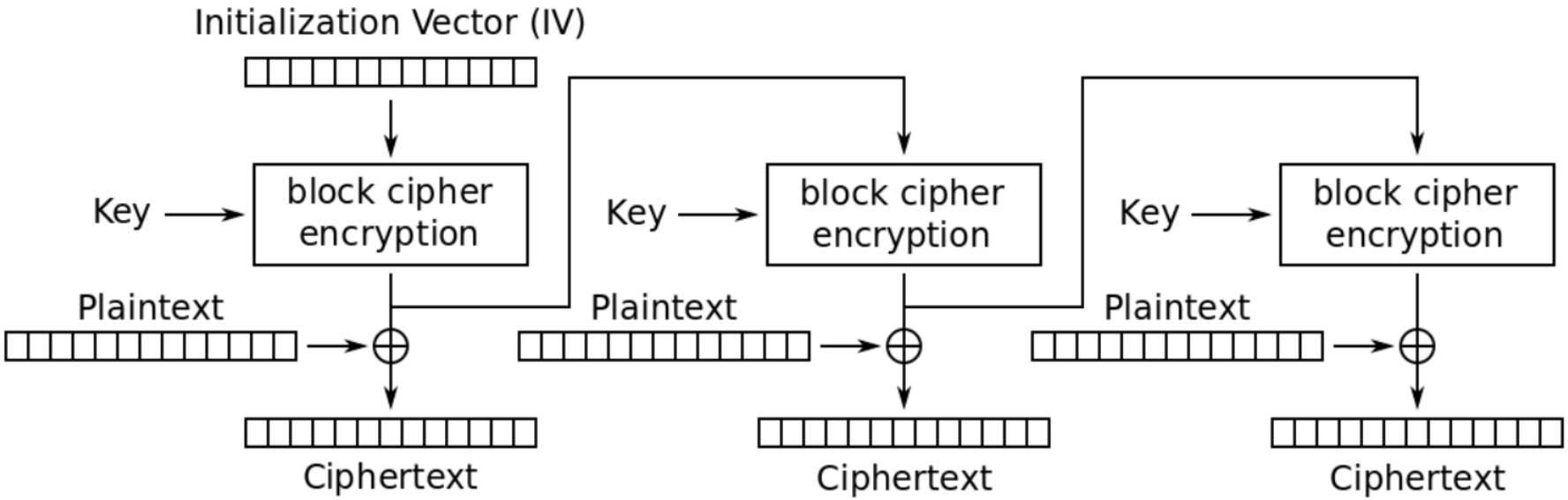


OFB mode

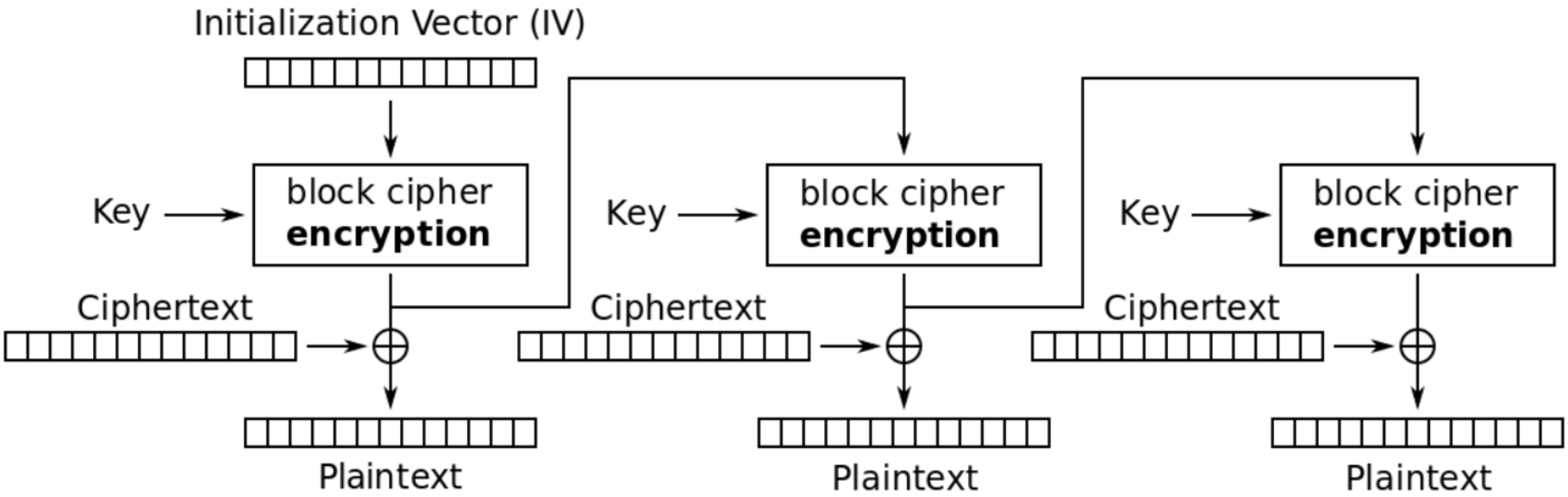
OFB = Output Feedback

- OFB makes a block cipher into a synchronous **stream cipher**.
- Flipping a bit in the ciphertext produces a flipped bit in the plaintext at the same location
- Each output feedback block cipher operation depends on all previous ones
- For the OFB mode, the IV **need not be unpredictable,** but it must be a nonce that is unique to each execution of the mode under the given key.
- Because of the symmetry of the XOR operation, encryption and decryption are exactly the same:
  $I_0 = IV, I_j = O_{j-1}, O_j = E_K(I_j), P_j = C_j \oplus O_j, C_j = P_j \oplus O_j$

OFB mode - encryption



OFB mode - decryption

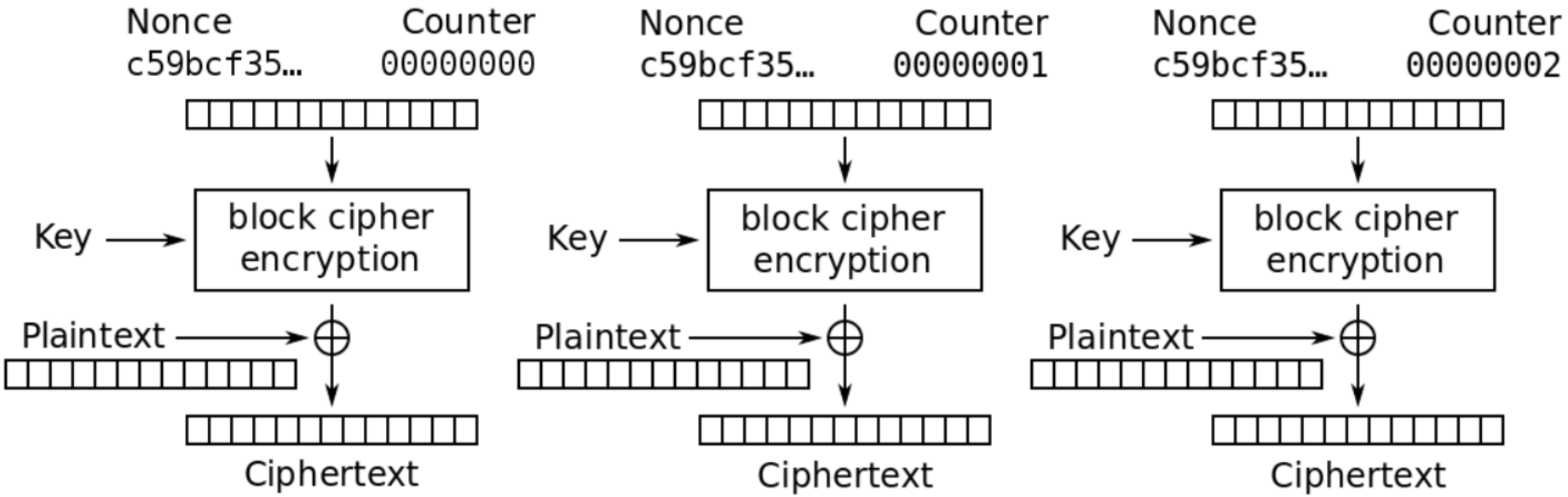


CTR mode

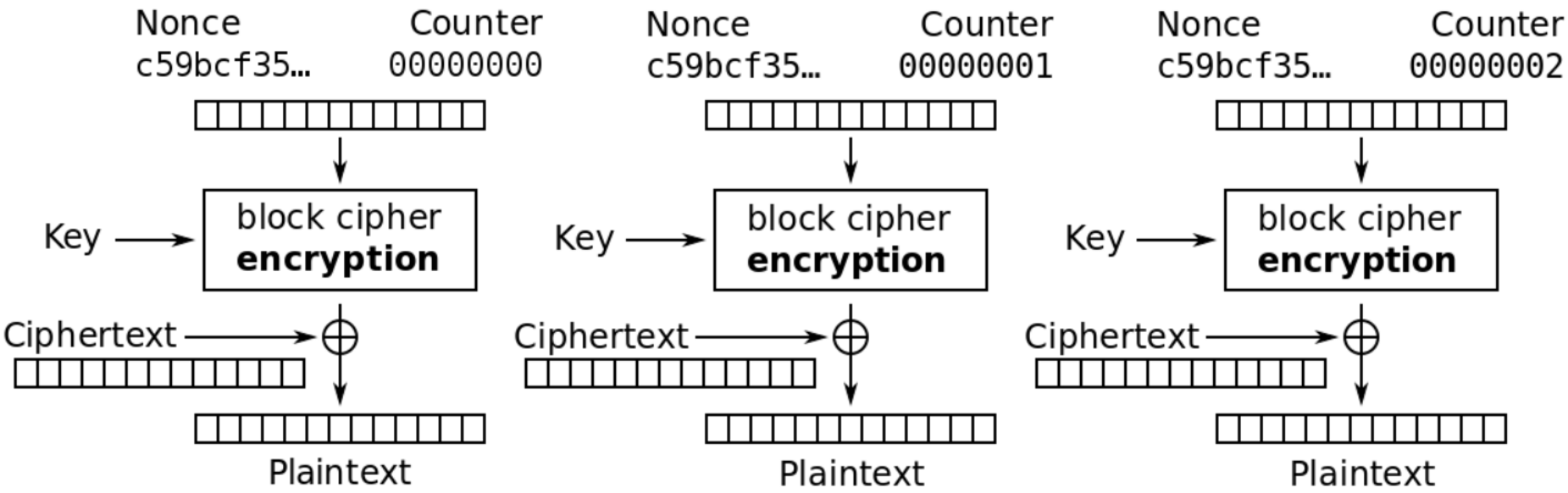
CTR = Counter

- CTR mode was introduced by Whitfield Diffie and Martin Hellman in 1979.
- Like OFB, counter mode turns a block cipher into a stream cipher.
- CTR allows a random access property during decryption
- Reusing an IV with the same key in CTR results in XORing the same keystream with two or more plaintexts, a clear misuse of a stream, with a catastrophic loss of security.

CTR mode - encryption



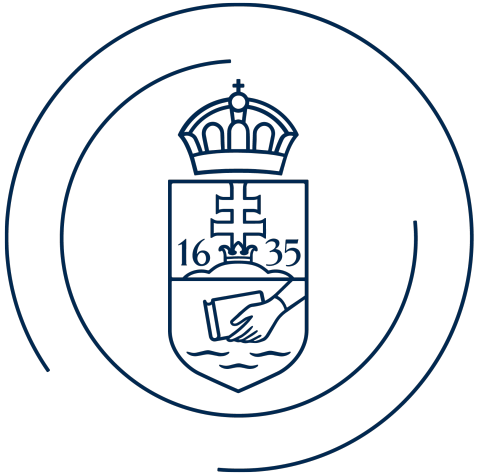
CTR mode - decryption



Thank you for your attention!

ntihanyi@inf.elte.hu

@TihanyiNorbert



ELTE
EÖTVÖS LORÁND
UNIVERSITY

Cryptography and security -Practice (IPM-18sztKVSZKRBG)

Dr. Norbert Tihanyi

OSCP, OSCE, OSWP, CRTP, CEH, ECES , ISO27001 LA

ntihanyi@inf.elte.hu

@TihanyiNorbert

Module 0x07
Side-channel attacks in Cryptography

Side-channel attack

Definition

Side-channel attack is any attack based on information gained from the implementation of a computer system, rather than weaknesses in the implemented algorithm itself (e.g. cryptanalysis and software bugs).

Extra source of information which can be exploited :

- **Timing information**
- Power consumption
- Electromagnetic leaks
- Sounds
- Any deviation from normnal behaviour

Timing attack

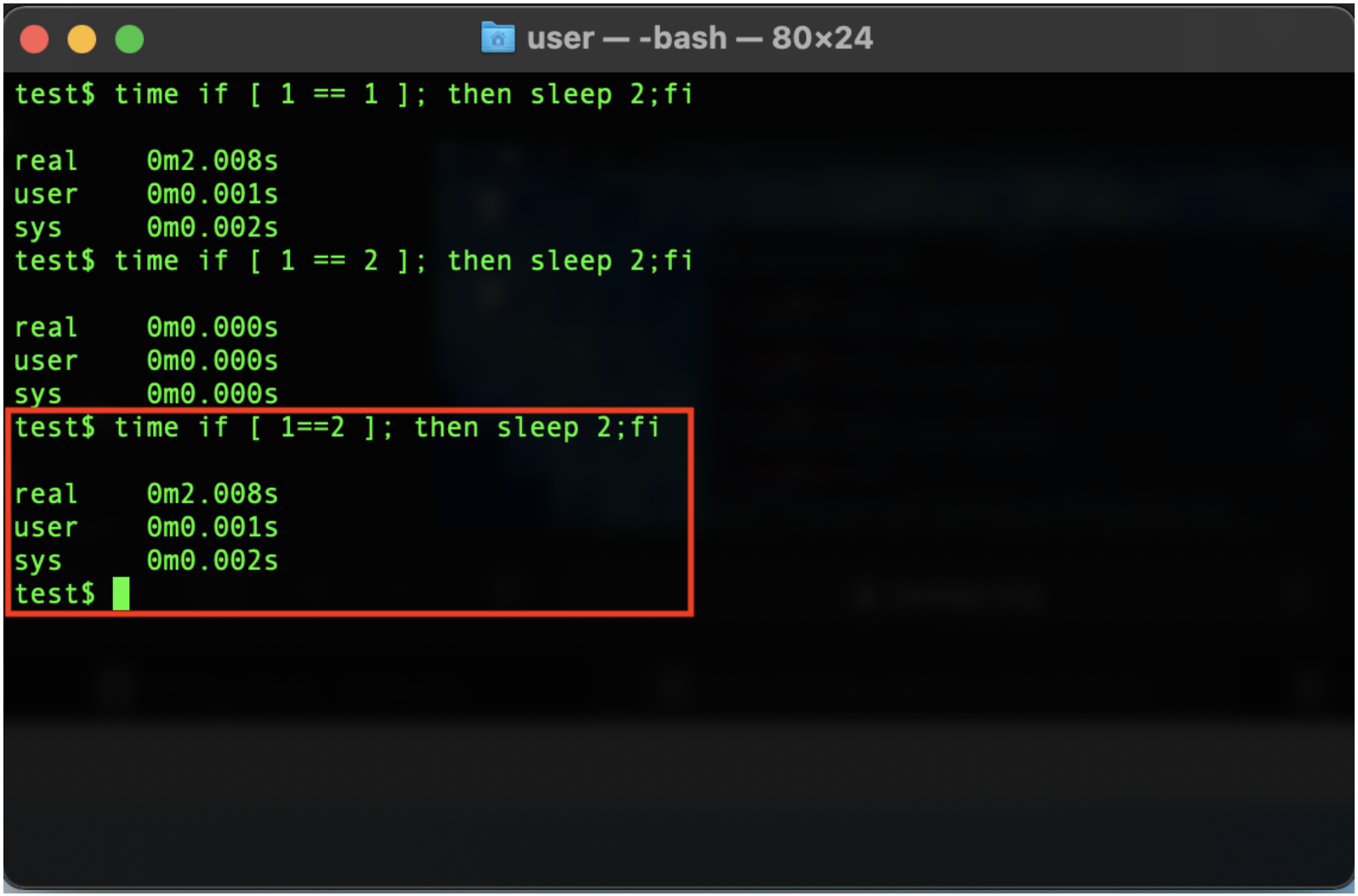
Definition

Timing attack is a side-channel attack in which the attacker attempts to compromise a cryptosystem by analyzing the time taken to execute cryptographic algorithms.

Timning attacks often overlooked in the design phase because they are so dependent on the implementation.

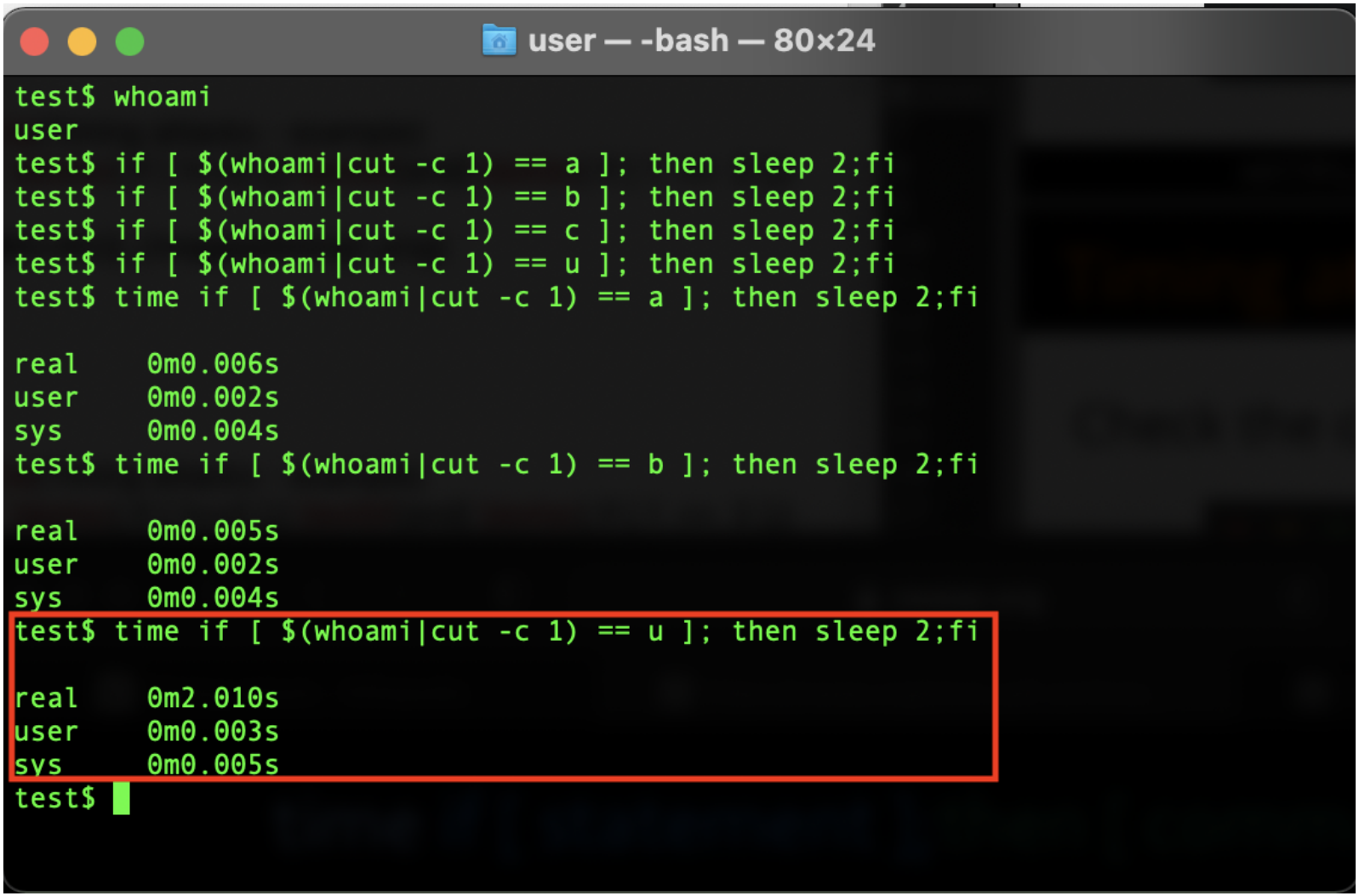
Timing attacks - example

Check the difference: IF [1==2]; **and** IF [1 == 2];



Timing attacks - example

No differnece in the standard output. The only difference is the execution time.



Timing attacks - example

Vulnerable string comparison:

```
if($correct_hash == $hash)
{
        //do authentication
};
```

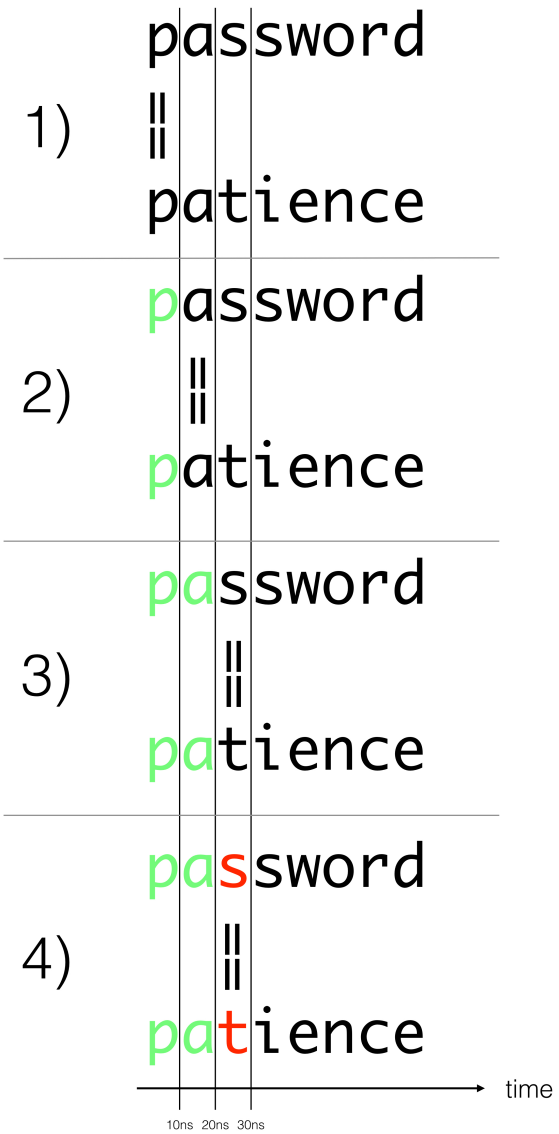
It iterates over each character of the two strings and returns False as soon as two characters differ.

Safe string comparison:

```
if(hash_equals($correct_hash, $hash))
{
        //do authentication
};
```

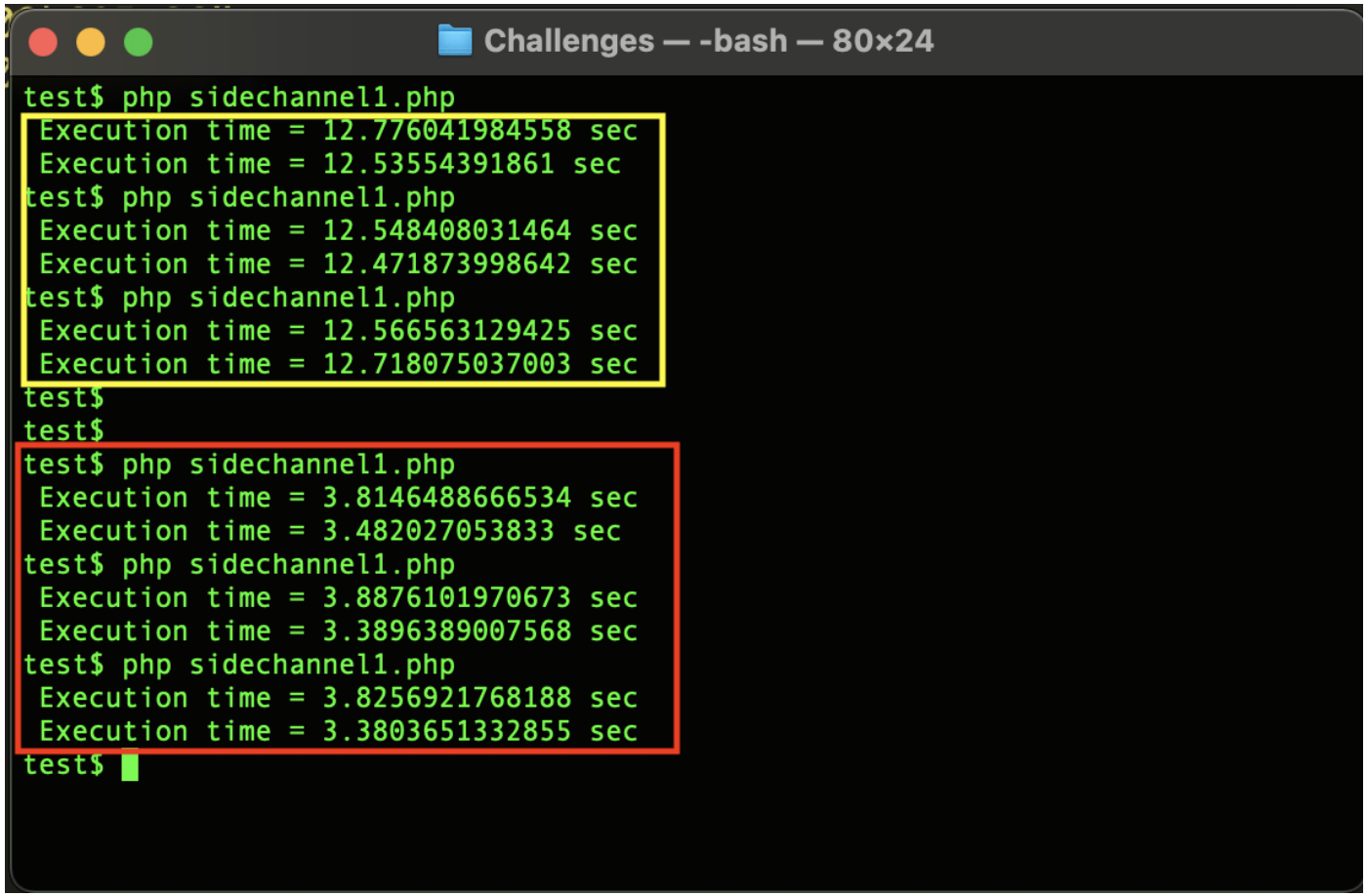
Compares two strings using the same time whether they're equal or not.

Timing attacks - Vulnerable string comparison



Timing attacks - example

Observable difference in execution time. The yellow one using `hash_equals` string comparison. Red one is using the vulnerable string comparison.



Cryptography is not a panacea

Bruce Schneier

*"**Strong cryptography** is very powerful when it is done right, but it is not a **panacea**. Focusing on cryptographic algorithms while ignoring other aspects of security is like defending your house not by building a fence a round it, but by putting an immense stake in the ground and hoping that your adversary runs right into it." - Bruce Schneier*

AES - PBKDF-HMAC-SHA512

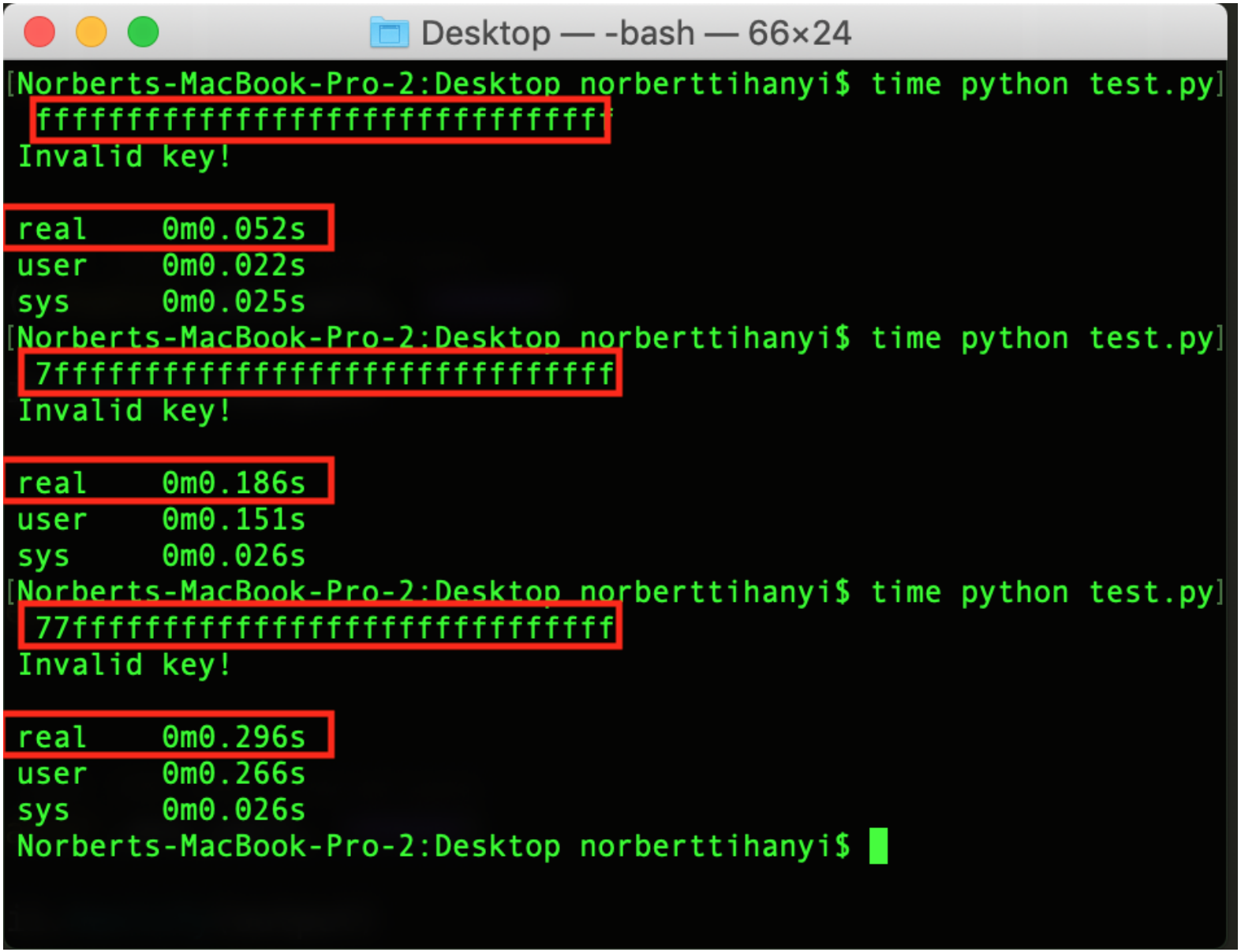
Using approved cryptographic protocols are not necessary secure

```python
import sys,base64,binascii,hashlib
from Crypto.Cipher import AES
from Crypto import Random

secret_key="7781c66d36dc3a22447a6e5bf8c89090"
salt="3ea58cc7a1176b368f0cf07fa5112964"
message="Message_to_sign!"
key = sys.argv[1]

def SEC_validate(key,message):
    C= AES.new(key, AES.MODE_ECB)
    enc=C.encrypt(message).encode('hex')
    for i in range(0,len(key)):
        if key[i]!=secret_key[i]:
            return "Invalid key!"
        # NIST approved 128-bit salt and 100,000 iterations
        output = hashlib.pbkdf2_hmac('sha512',enc,salt, 100000)

    return "SIGNED MESSAGE:", binascii.hexlify(output)

print SEC_validate(key,message)
```

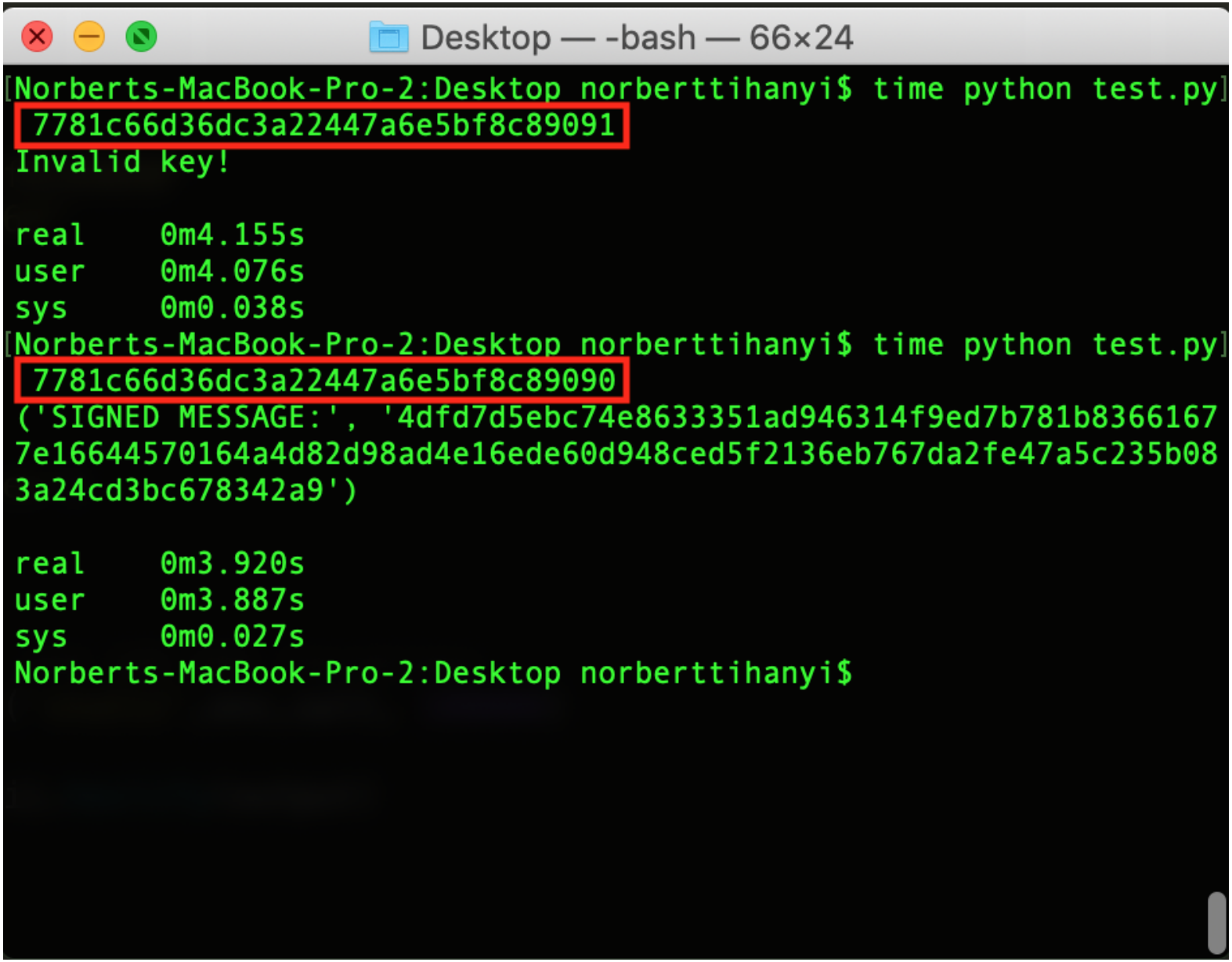
Right and Wrong implementation

```
def SEC_validate_WRONG(key,message):
    C= AES.new(key, AES.MODE_ECB)
    enc=C.encrypt(message).encode('hex')
    for i in range(0,len(key)):
        if key[i]!=secret_key[i]:
            return "Invalid key!"
    # NIST approved 128-bit salt and 100,000 iterations
    output = hashlib.pbkdf2_hmac('sha512',enc,salt, 100000)

    return "SIGNED MESSAGE:", binascii.hexlify(output)


def SEC_validate_GOOD(key,message):
    C= AES.new(key, AES.MODE_ECB)
    enc=C.encrypt(message).encode('hex')
    for i in range(0,len(key)):
        if key[i]!=secret_key[i]:
            return "Invalid key!"
    # NIST approved 128-bit salt and 100,000 iterations
    output = hashlib.pbkdf2_hmac('sha512',enc,salt, 100000)

    return "SIGNED MESSAGE:", binascii.hexlify(output)
```

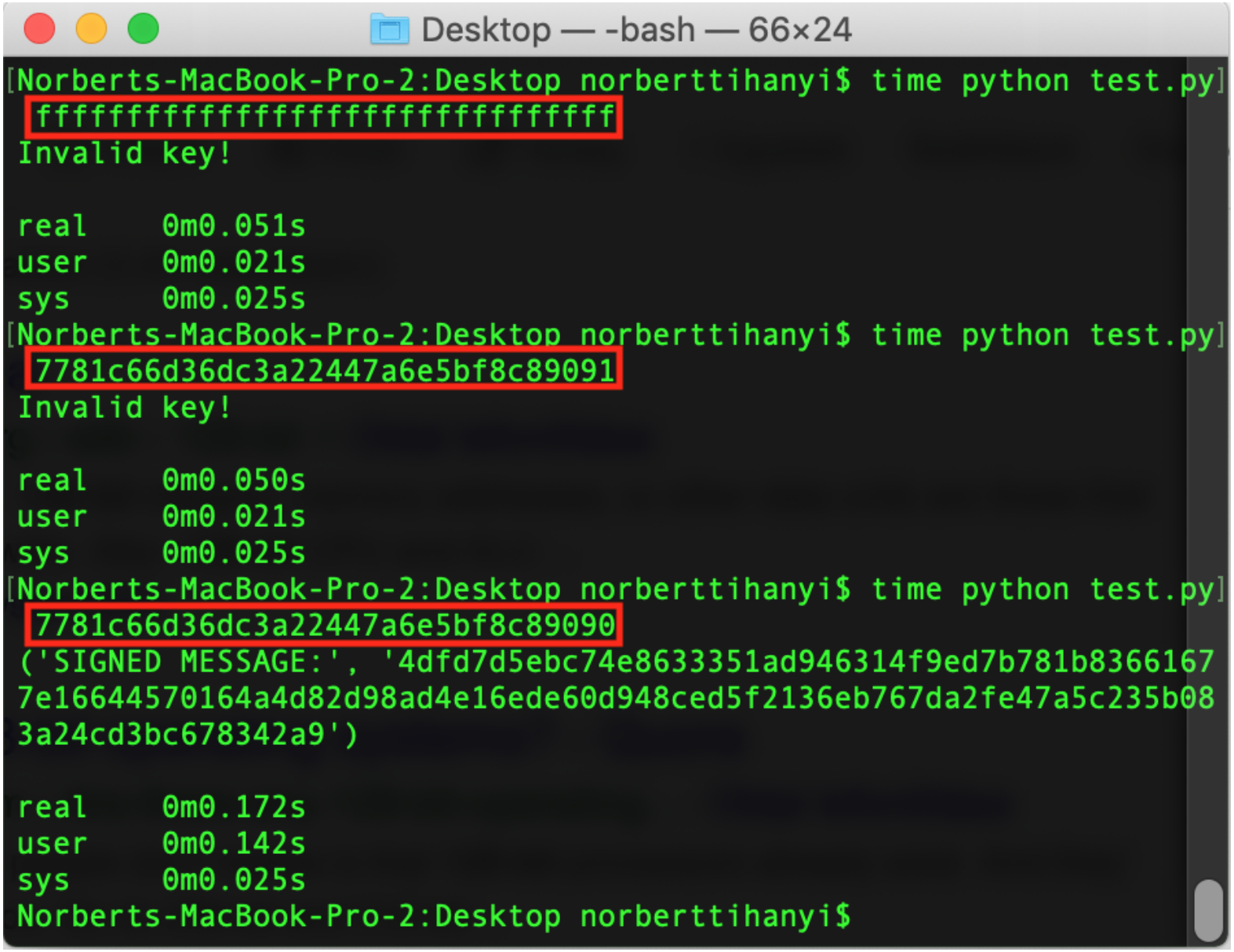
BAD implementation- TIMING attack



BAD implementation- TIMING attack



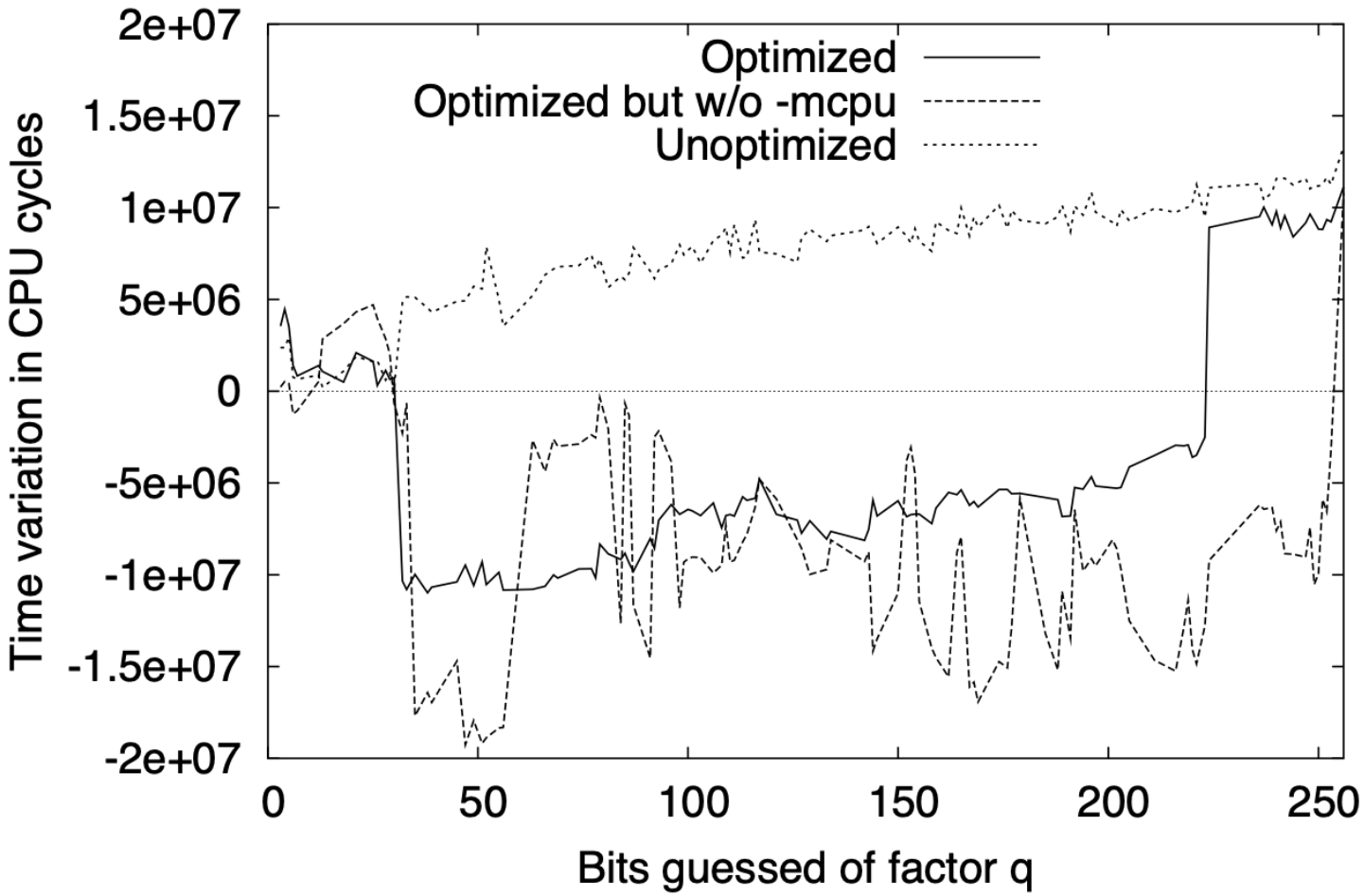
Good implementation - NO TIMING attack



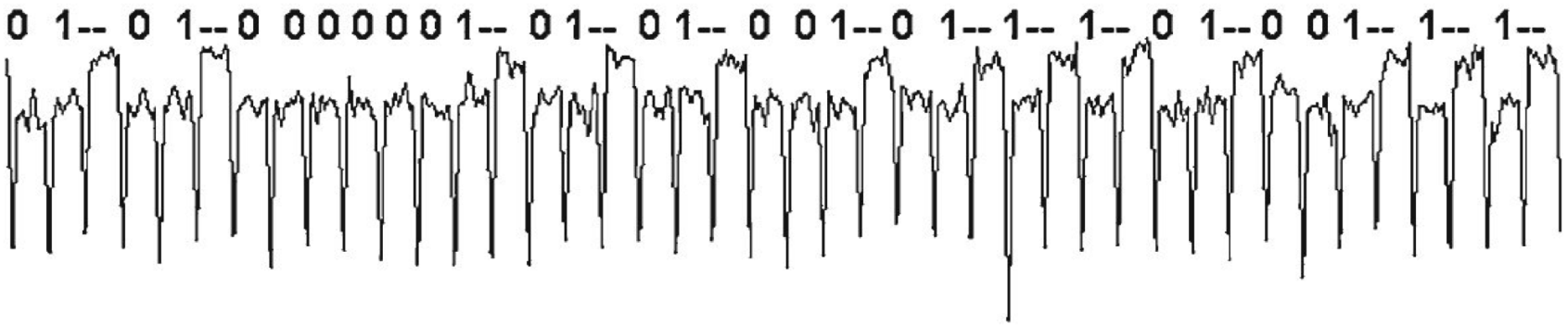
Timing attack on OpenSSL

Remote Timing Attacks are Practical by David Brumley and Dan Boneh : ”We show that using about a million queries we can remotely extract a 1024-bit RSA private key from an OpenSSL 0.9.7 server. The attack takes about two hours.”
Proceedings of the 12th conference on USENIX Security Symposium - Volume 12 August **2003**

Timing attack on OpenSSL



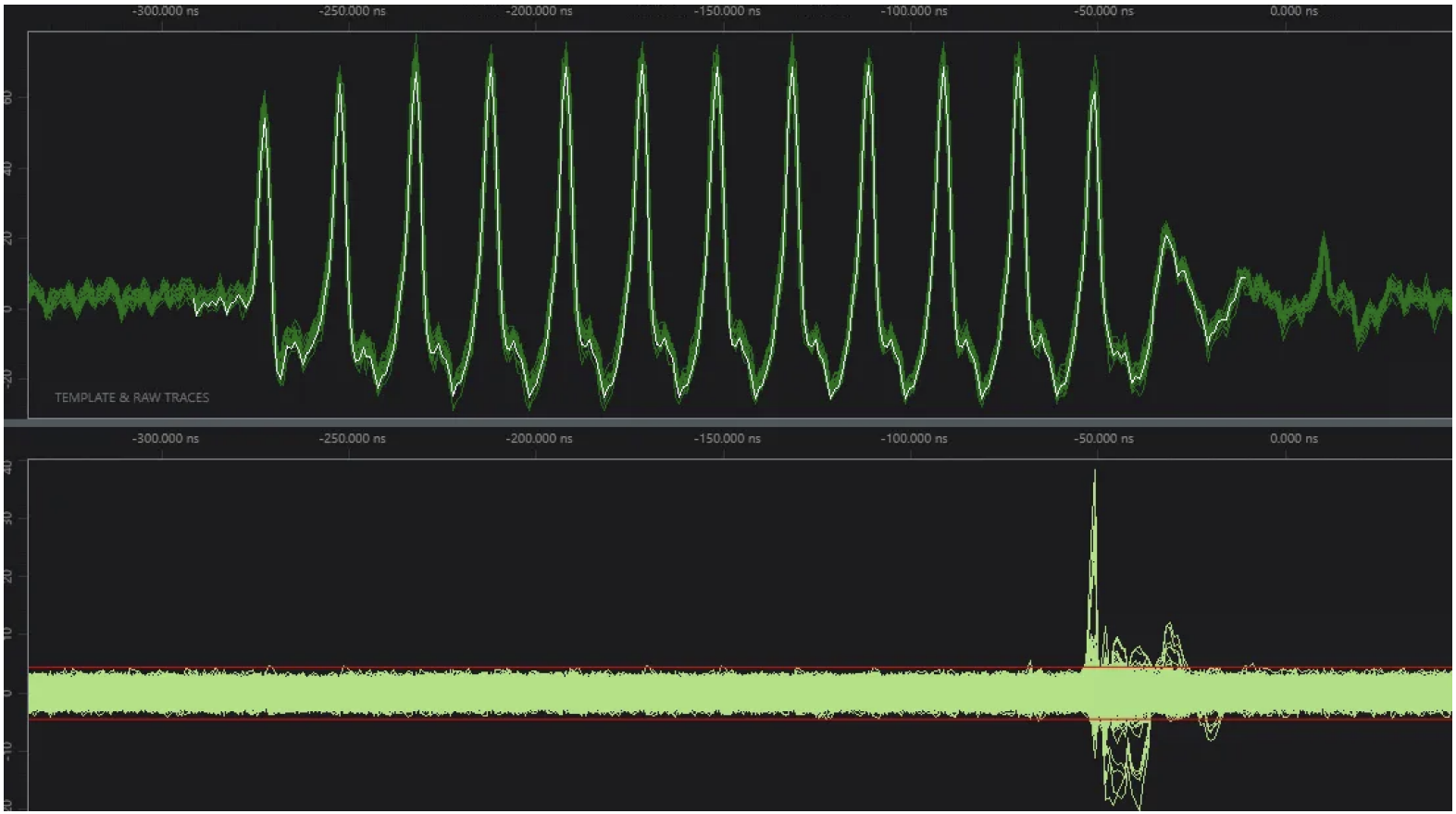
Recovering secret key



Power analysis

Definition

Power analysis is a side channel attack in which the attacker studies the power consumption of a cryptographic hardware device



Padding oracle attack

Definition

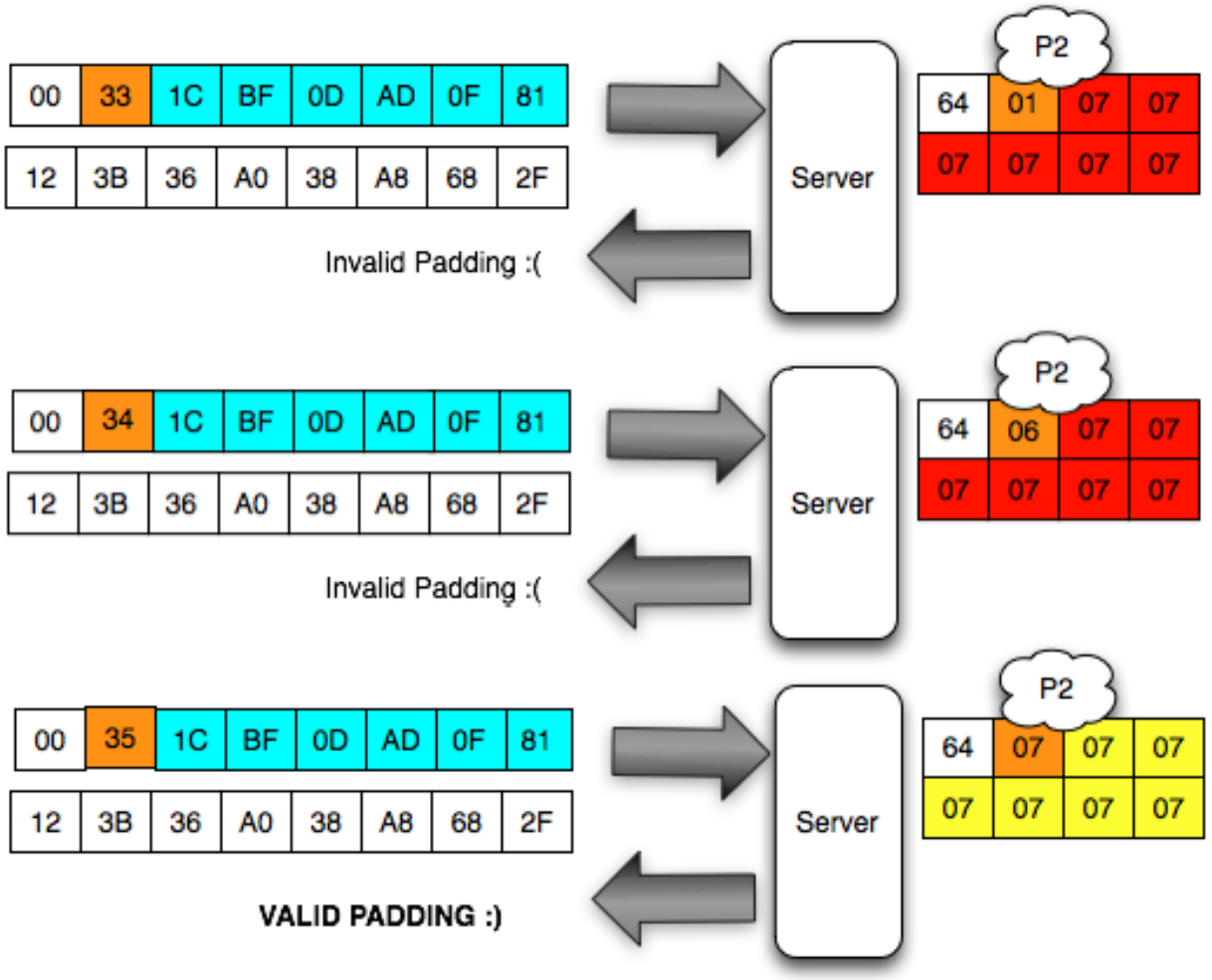
A **padding oracle attack** is an attack which uses the padding validation of a cryptographic message to decrypt the ciphertex

The original attack was published in 2002 by Serge Vaudenay.

Definition

The padding oracle attack can be applied to the **CBC mode of operation**, where the ”oracle” (usually a server) leaks data about whether the padding of an encrypted message is correct or not. Such data can allow attackers to decrypt (and sometimes encrypt) messages through the oracle using the oracle’s key, without knowing the encryption key.

Padding oracle attack



Lucky Thirteen attack

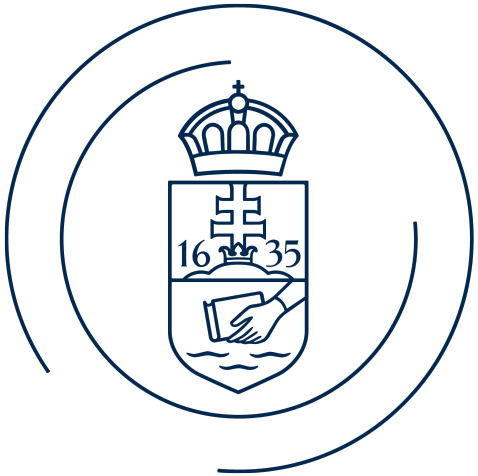
Definition

A **Lucky Thirteen** attack is a cryptographic timing attack against implementations of the Transport Layer Security (TLS) protocol that use the CBC mode of operation, first reported in February 2013 by its developers Nadhem J. AlFardan and Kenny Paterson of the Information Security Group at Royal Holloway, University of London.

Thank you for your attention!

ntihanyi@inf.elte.hu

@TihanyiNorbert



ELTE
EÖTVÖS LORÁND
UNIVERSITY

Cryptography and security -Practice (IPM-18sztKVSZKRBG)

Dr. Norbert Tihanyi

OSCP, OSCE, OSWP, CRTP, CEH, ECES , ISO27001 LA

ntihanyi@inf.elte.hu

@TihanyiNorbert

Module 0x08
Advanced password cracking techniques

Massive data breaches - I.

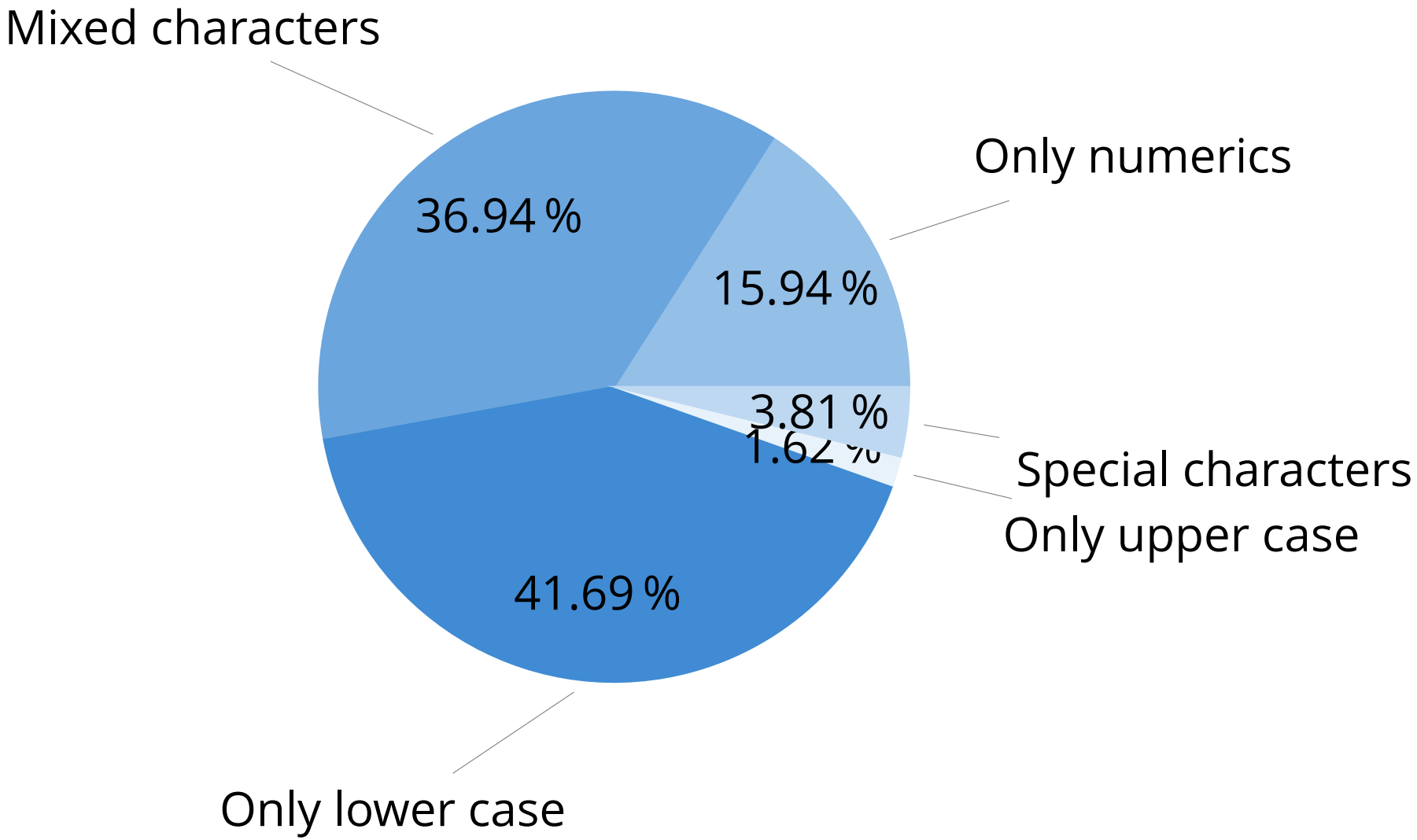
- **Yahoo:** Yahoo announced in September 2016 that in 2013-2014 it had been the victim of what would be **the biggest data breach in history**. Yahoo reported that approximately **3 billion** user accounts were compromised.
- **Mariott:** In 2018 November Marriott International announced that attackers had stolen data on approximately **500 million** customers.
- **Adult Friend Finder:** In October 2016 approximately **412.2 million** accounts compromised.
- **Zynga:** In Septemper 2019 approximately **218 million** Zynga email addresses, salted SHA-1 hashed passwords, phone numbers, and user IDs for Facebook and Zynga accounts were stolen.

Massive data breaches - II.

1. **Dubsmash:** In December 2018, New York-based video messaging service Dubsmash had **162 million** email addresses, usernames, PBKDF2 password hashes were compromised.
2. **Linkedin:** In October 2013 over **150 million** username and hashed password pairs taken from Adobe.
3. **Adobe:** In October 2013 over **150 million** username and hashed password pairs taken from Adobe.
4. **Ebay:** In 2014 march, 145 million eBay accounts were compromised in massive hack including email addresses and encrypted passwords.

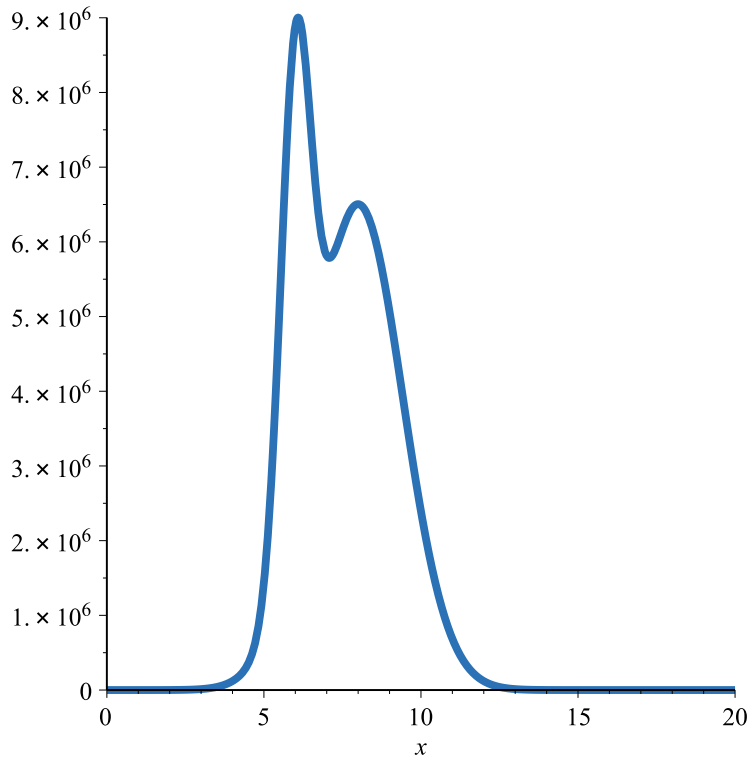
Analyzing cleartext passwords

Sometimes cleartext passwords are also compromised.
Analysis of 32 million cleartext password leaked from RockYou:



Password length distribution

The password length distribution of RockYou Database:



Password length distribution

By analyzing more than 100 different databases it has turned out that the password length distribution (PLD) can be approximated by a mixture of two modified Gaussian distribution function:

Theorem (Tihanyi & Kovaćs - 2015)

$$PWD(x, D_n, \sigma, \mu_1, \mu_2) = D_n \cdot \frac{1}{2\sqrt{2\pi}}(e^{\frac{-(x-\mu_1)^2}{\sigma^2}} + e^{\frac{-(x-\mu_2)^2}{\log(\sigma)/1.5}}) \quad (1)$$

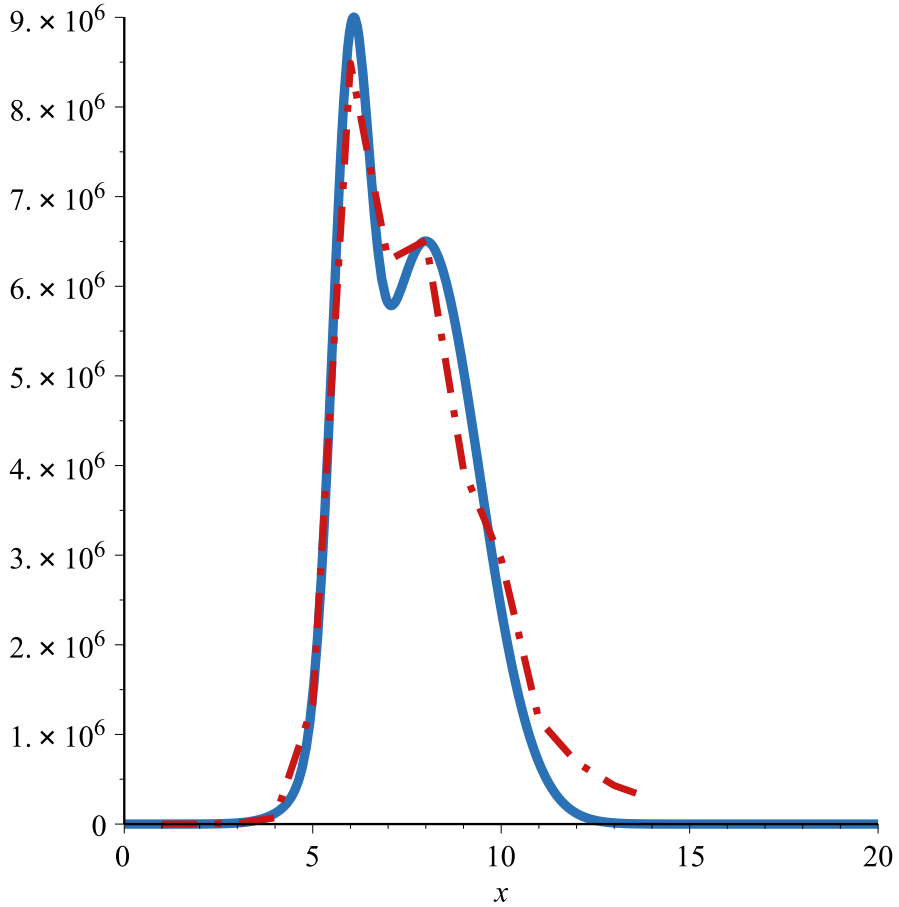


Figure: Rockyou real PLD (red) and approximated PWD (blue).

Letter frequency distribution

Database	The first 10 most common letters
Google	$a, e, 1, i, n, r, o, s, 2, l$
Yahoo	$a, e, i, o, 1, r, n, s, l, t$
RockYou	$a, e, 1, i, o, n, r, l, s, 0$

Table: Letter frequencies in various databases.

Letter frequency distribution

It can be seen that the letter frequency distribution is similar to some inverse logarithmic distribution ($1 / \log N$).

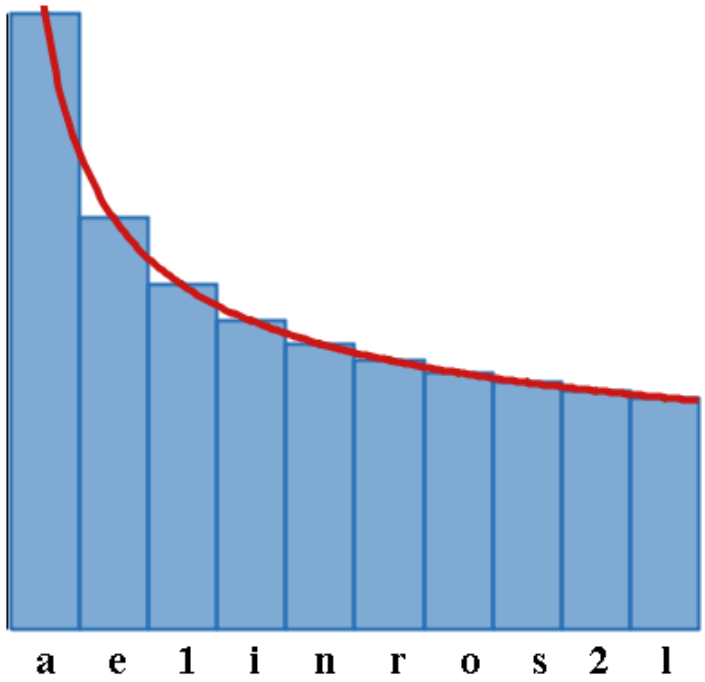


Figure: Letter frequency distribution

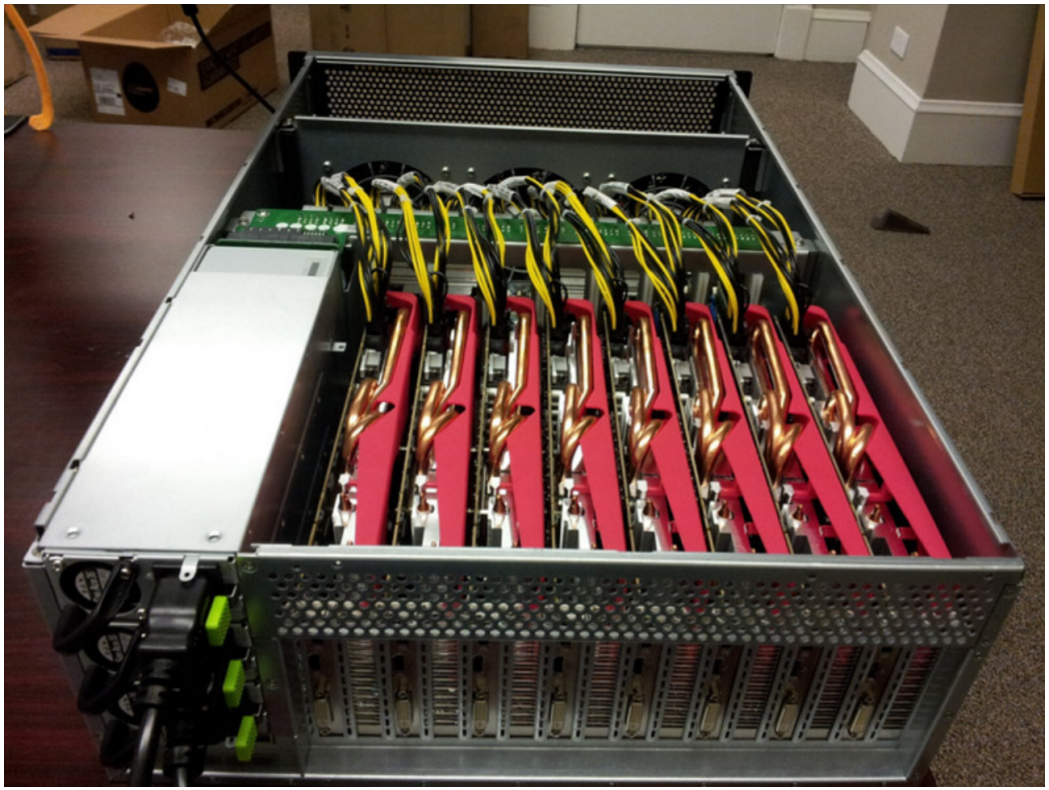
A sophisticated tool for Password cracking: hashcat

hashcat: World's fastest and most advanced GPGPU-based password recovery utility.

- Free and Open-source
- Multi-Hash (up to 100 million hashes)
- Multi-GPU (up to 128 gpus)
- Linux,Windows
- **rule-based** engine (Very fast Rule-engine)
- Able to work in an distributed environment
- **More than 300+ algorithms implemented**

Dedicated 25 GPU Monster - 2012

Five 4U servers equipped with 25 AMD Radeon-powered GPUs linked together using an Infiniband switched fabric link can crunch through up to 348 billion password hashes per second. **Complex Windows password can be cracked within a few hours.**



GPU Cracking in 2021 - RTX 3090

- GeForce RTX 3090 24GB RAM
- Current price: 15.000-17.000 AED
- up to 10496 CUDA cores
- Green super-computing at 1280 Watt per machine
- **121.2 billion NTLM passwords /sec**
- To achieve $\approx$ 0.5 trillion password /sec you need only 4 RTX 3090 GPU cards.
- Traditional DES can be cracked with **67.5 billion keys /sec**

Data Encryption Standard - DES cracking

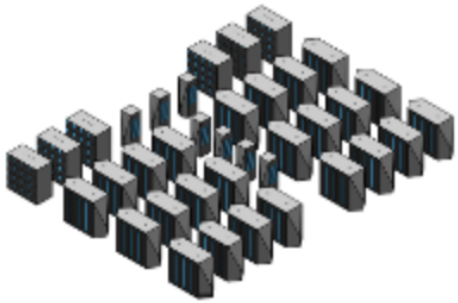
- 1976: DES is approved as a standard, first published in 1977.
- Today DES considered as an insecure algorithm.
- **SciEngines** reported on July 15th 2008 to break DES in an average time of a single day using a setup of COPACOBANA.

- Systems have been built with eight **GTX 1080 Ti** GPUs which can recover a key in an average of under 2 days

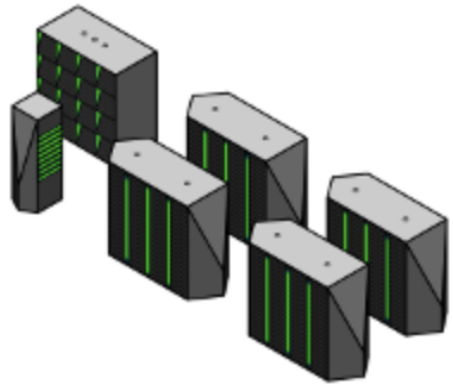
DES cracking - crack.sh

Online DES cracking : crack.sh

- Behind crack.sh is a system with **48 Xilinx Virtex-6 LX240T FPGAs**.
- Each FPGA contains a design with 40 fully pipelined DES cores running at 400MHz
- **768,000,000,000 keys/sec** for the whole system.
- Exhaustively search the entire 56-bit DES keyspace in ≈ **26 hours**

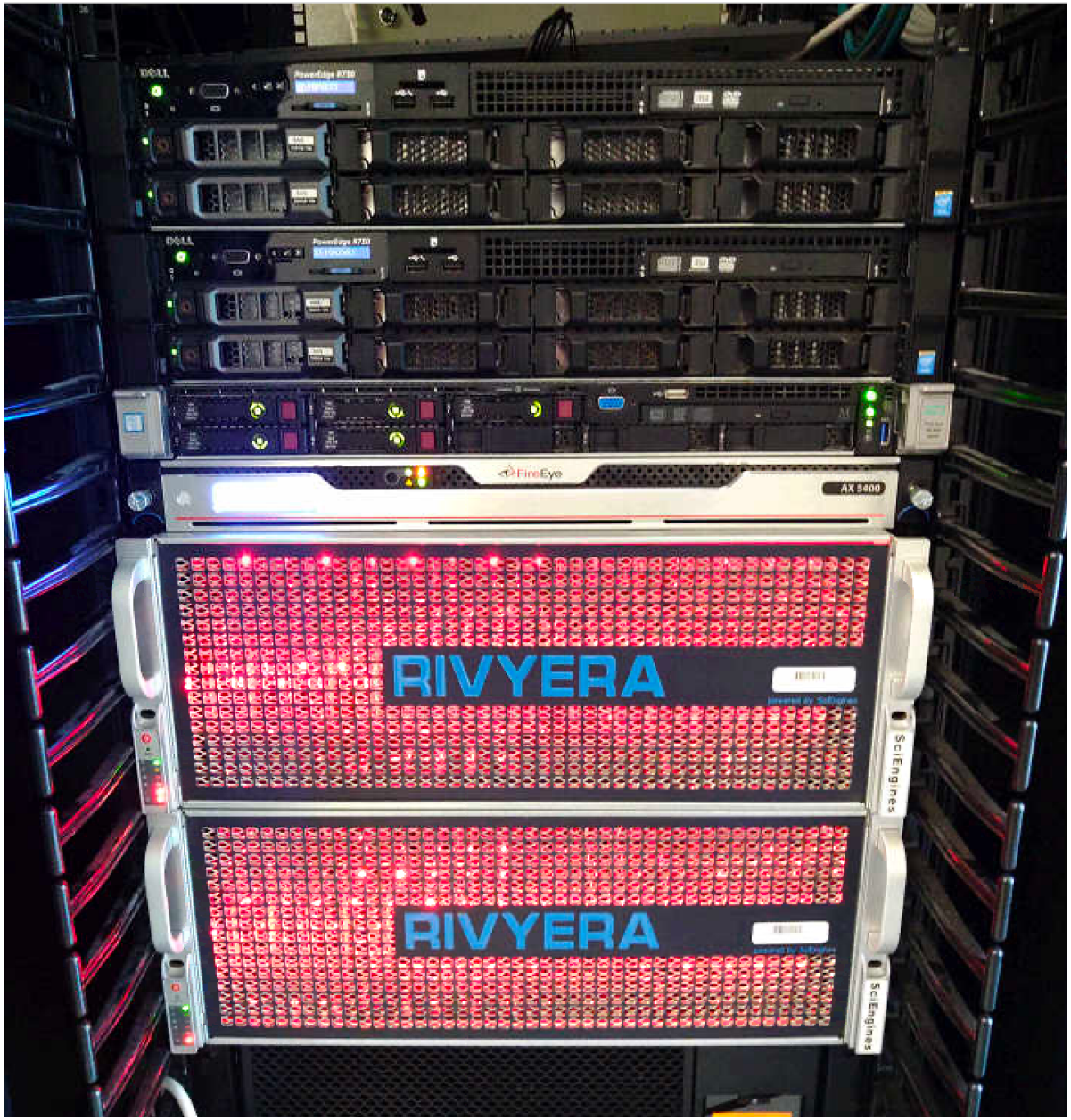


80,000 CPU cores
~$125,000 per key
(at $0.12 per CPU on EC2)



1,800 GPUs
~$20,000 per key
(at $2.10 per GPU on EC2)

RIVYERA


DELL
FireEye
AX 5400
RIVYERA
SciEngines
RIVYERA
SciEngines

RIVYERA

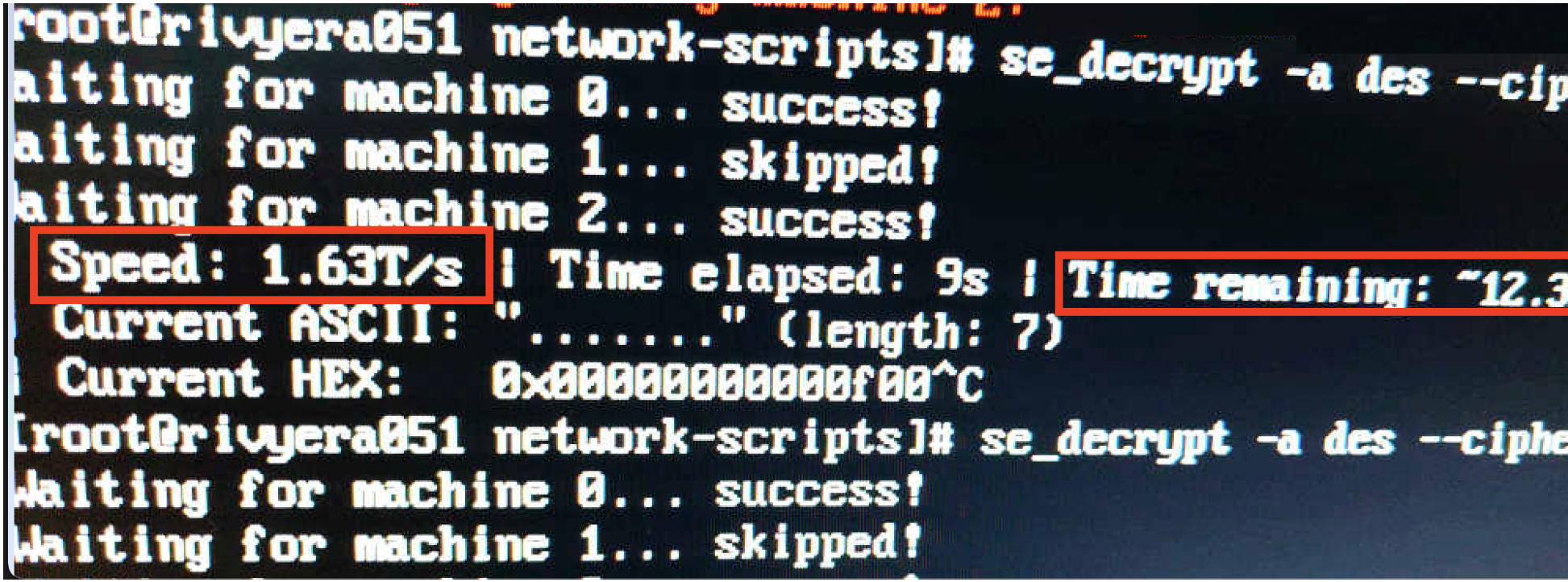
- 2x 4U chassis RIVYERA S6-LX150 cracking server
- Ecah server contains 128x Xilinx Spartan-6 LX150 (XC6SLX150) FPGAs
- up to 30000 CPU cores performance per machine
- Green super-computing at 1280 Watt per machine

DES cracking in my 2017 MacBook Pro

```
OpenCL API (OpenCL 1.2 (May  8 2021 03:14:28)) - Platform #1 [Apple]
=====================================================================
* Device #1: Intel(R) Core(TM) i7-7820HQ CPU @ 2.90GHz, skipped
* Device #2: Intel(R) HD Graphics 630, skipped
* Device #3: AMD Radeon Pro 560 Compute Engine, 4032/4096 MB (1024 MB allocatable), 16MCU

Benchmark relevant options:
===========================
* --backend-devices=3
* --optimized-kernel-enable

Hashmode: 14100 - 3DES (PT = $salt, key = $pass)

Speed.#3.........:   601.6 MH/s (54.24ms) @ Accel:32 Loops:1024 Thr:64 Vec:1

Started: Wed Jun  2 08:58:13 2021
Stopped: Wed Jun  2 08:58:19 2021
test$
```

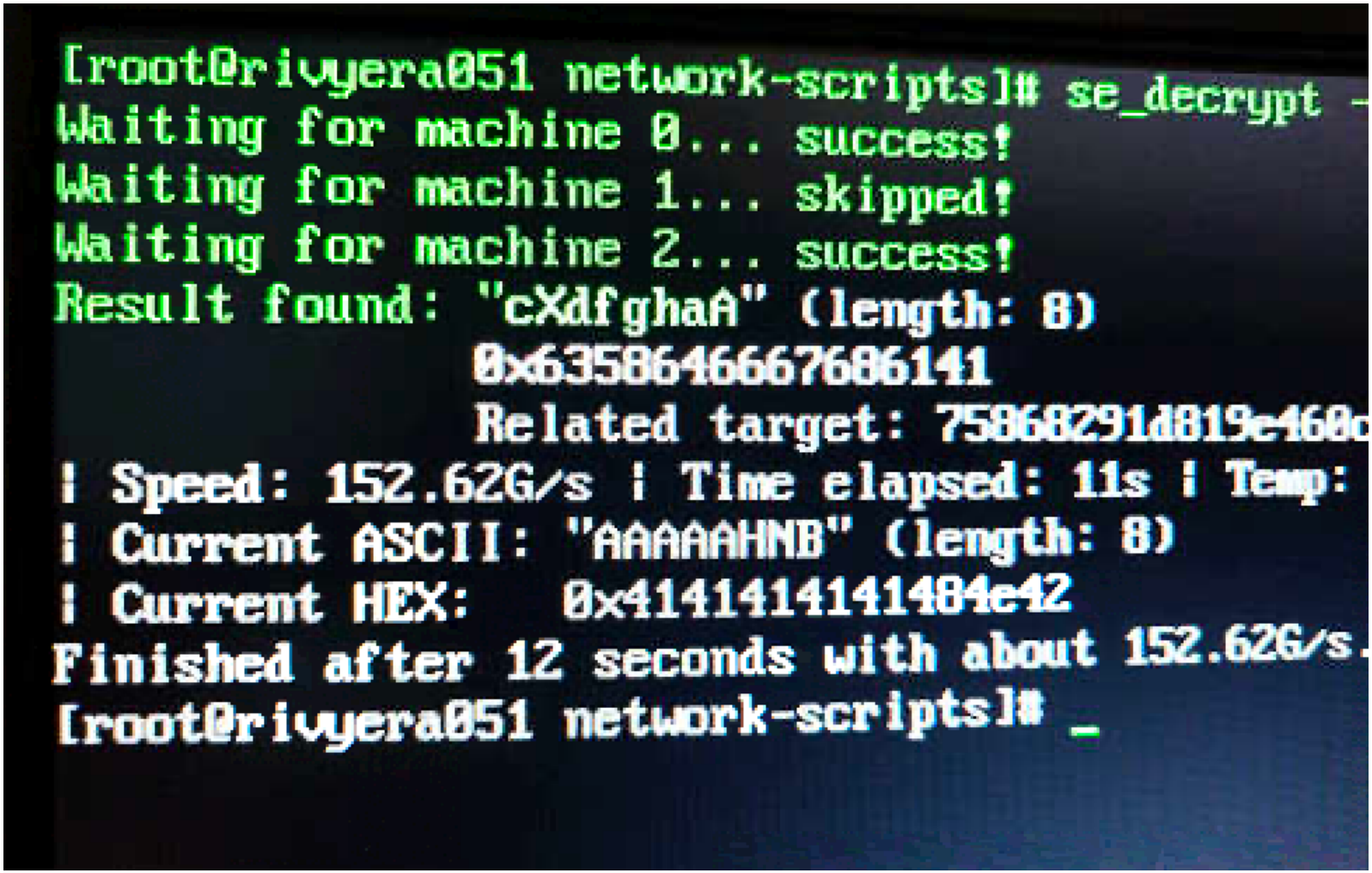
- **600 million key /sec**
- It can exhaustively search the entire 56-bit DES keyspace in $\approx 2^{56}/(600000000 * 3600)$= **3.808** years.

Outstanding performance in DES cracking



- **1.63 trillion key /sec**
- It can exhaustively search the entire 56-bit DES keyspace in $\approx 2^{56}/(1630000000000 * 3600)$= **12. 3** hours.

MD5 cracking example



MD5 example: Random password contains uppercase and lowercase letters cracked in 11 sec. The password is: **cXdfgghaA**.

Thank you for your attention!

ntihanyi@inf.elte.hu

@TihanyiNorbert