CS 177 - Computer Security

Cryptography I

CRYPTO
MEANS CRYPTOGRAPHY.

Cryptography = Hidden Writing

Important component of secure systems: Ensures **confidentiality** and **integrity** of information.

First use: 1900 BC, ancient Egypt

Today: Still hard to get right, but used ubiquitously.

Images: https://en.wikipedia.org/wiki/Cryptography

Why Cryptography?

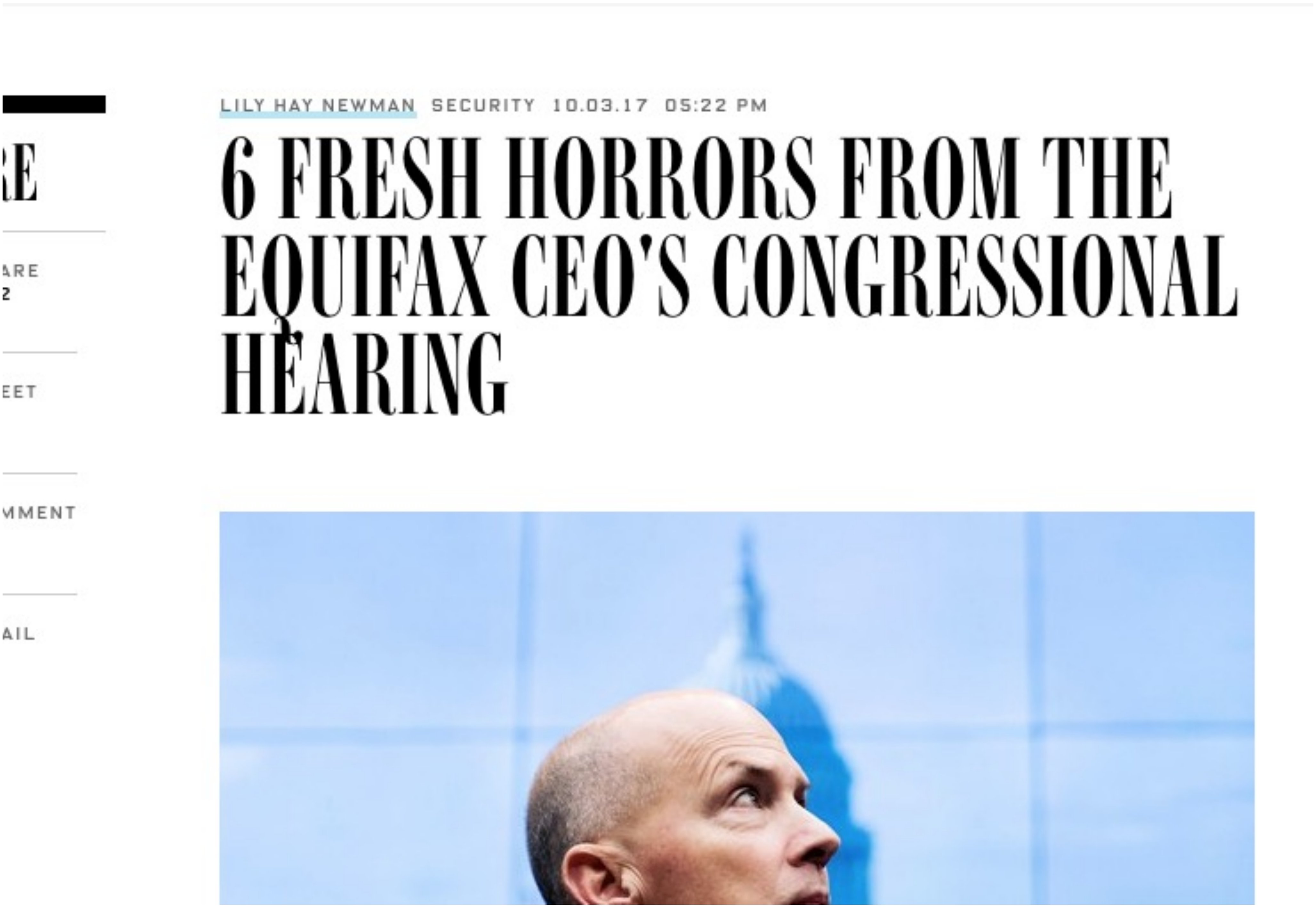
Cryptography is important component of every secure system!

Correct deployment of cryptography is very tricky

Even the best do it wrong!

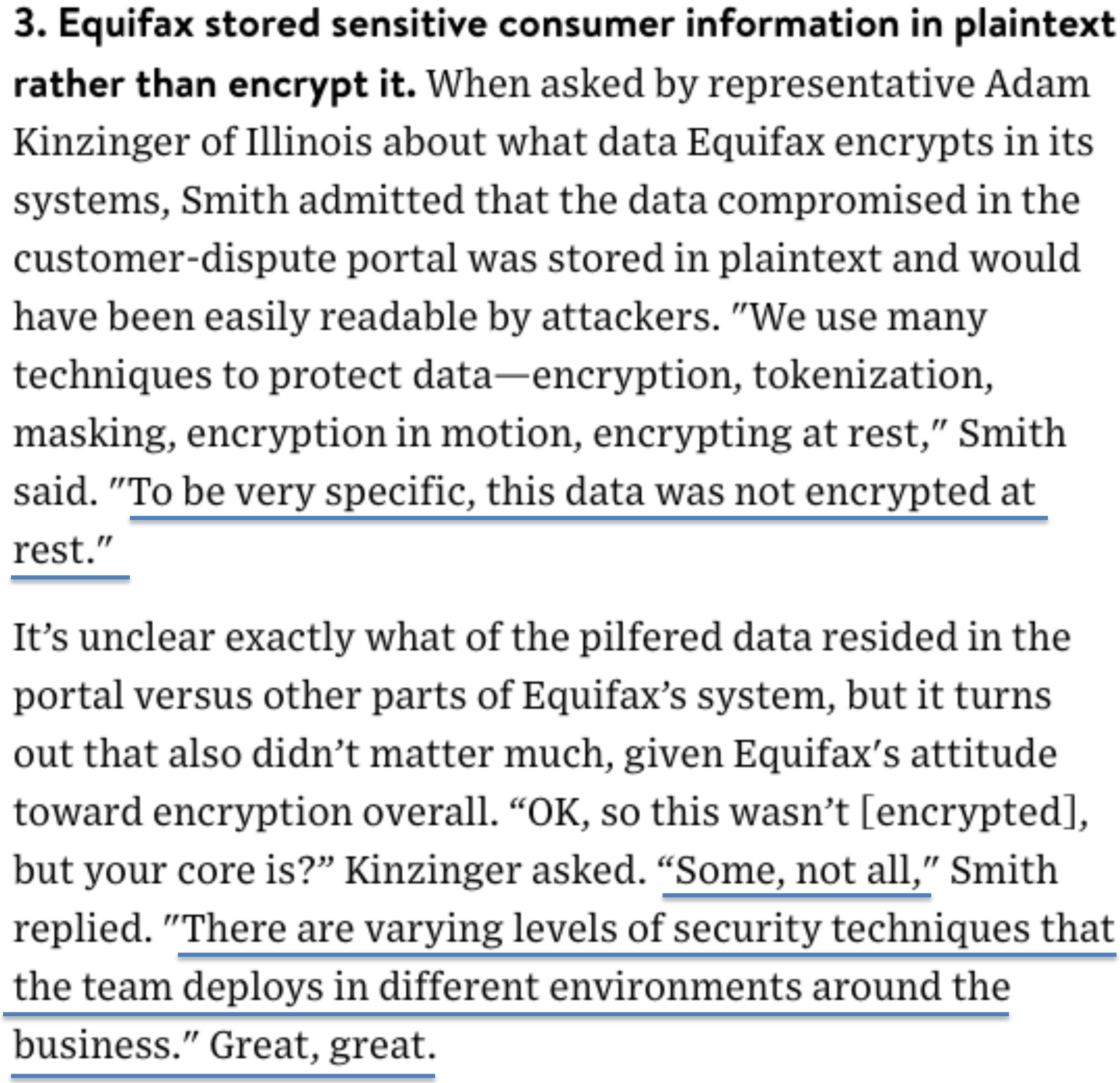
Cryptography - Tasks

- Guaranteeing confidentiality of data (encryption)
- Guaranteeing integrity of data (message authentication, digital signatures)
- Guaranteeing identity of data sender (digital signatures, digital certificates, public-key infrastructures, …)
- … and much more!



LILY HAY NEWMAN SECURITY 10.03.17 05:22 PM

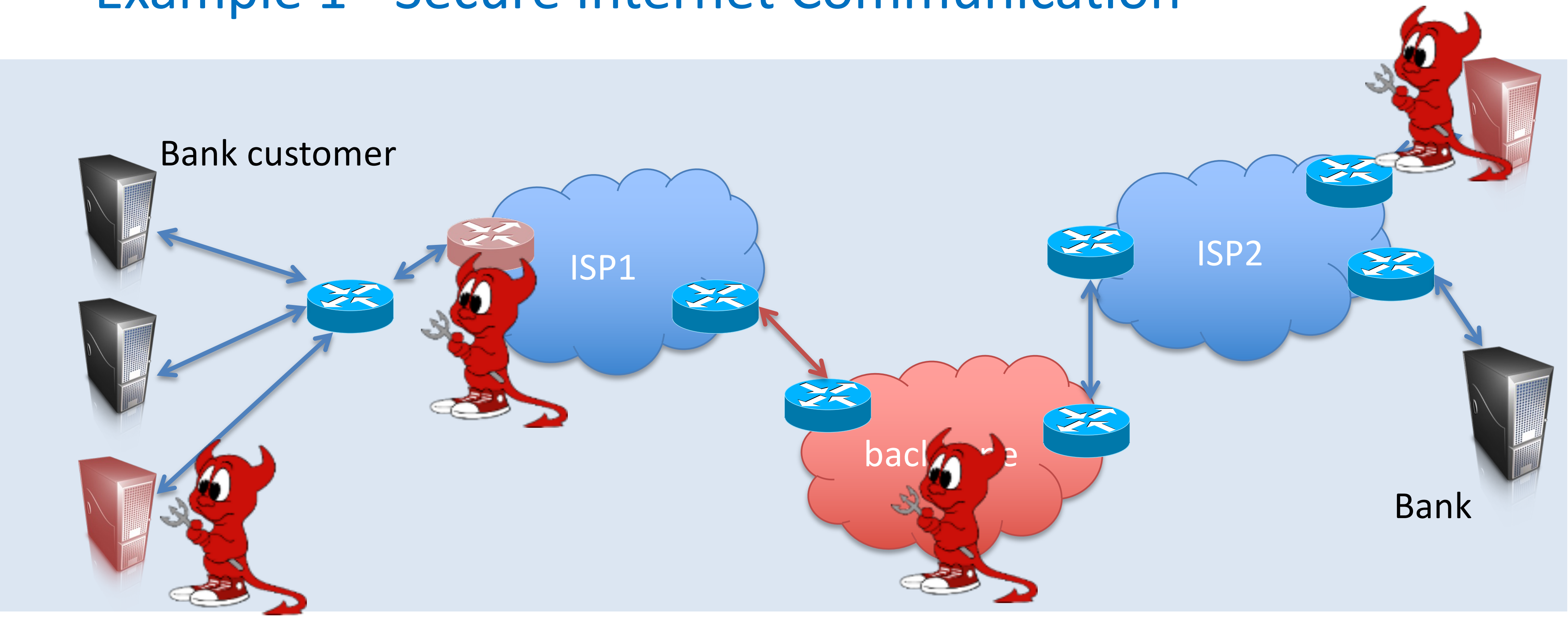
6 FRESH HORRORS FROM THE EQUIFAX CEO'S CONGRESSIONAL HEARING



3. Equifax stored sensitive consumer information in plaintext rather than encrypt it. When asked by representative Adam Kinzinger of Illinois about what data Equifax encrypts in its systems, Smith admitted that the data compromised in the customer-dispute portal was stored in plaintext and would have been easily readable by attackers. "We use many techniques to protect data—encryption, tokenization, masking, encryption in motion, encrypting at rest," Smith said. "To be very specific, this data was not encrypted at rest."

It's unclear exactly what of the pilfered data resided in the portal versus other parts of Equifax's system, but it turns out that also didn't matter much, given Equifax's attitude toward encryption overall. "OK, so this wasn't [encrypted], but your core is?" Kinzinger asked. "Some, not all," Smith replied. "There are varying levels of security techniques that the team deploys in different environments around the business." Great, great.

Example 1 - Secure Internet Communication

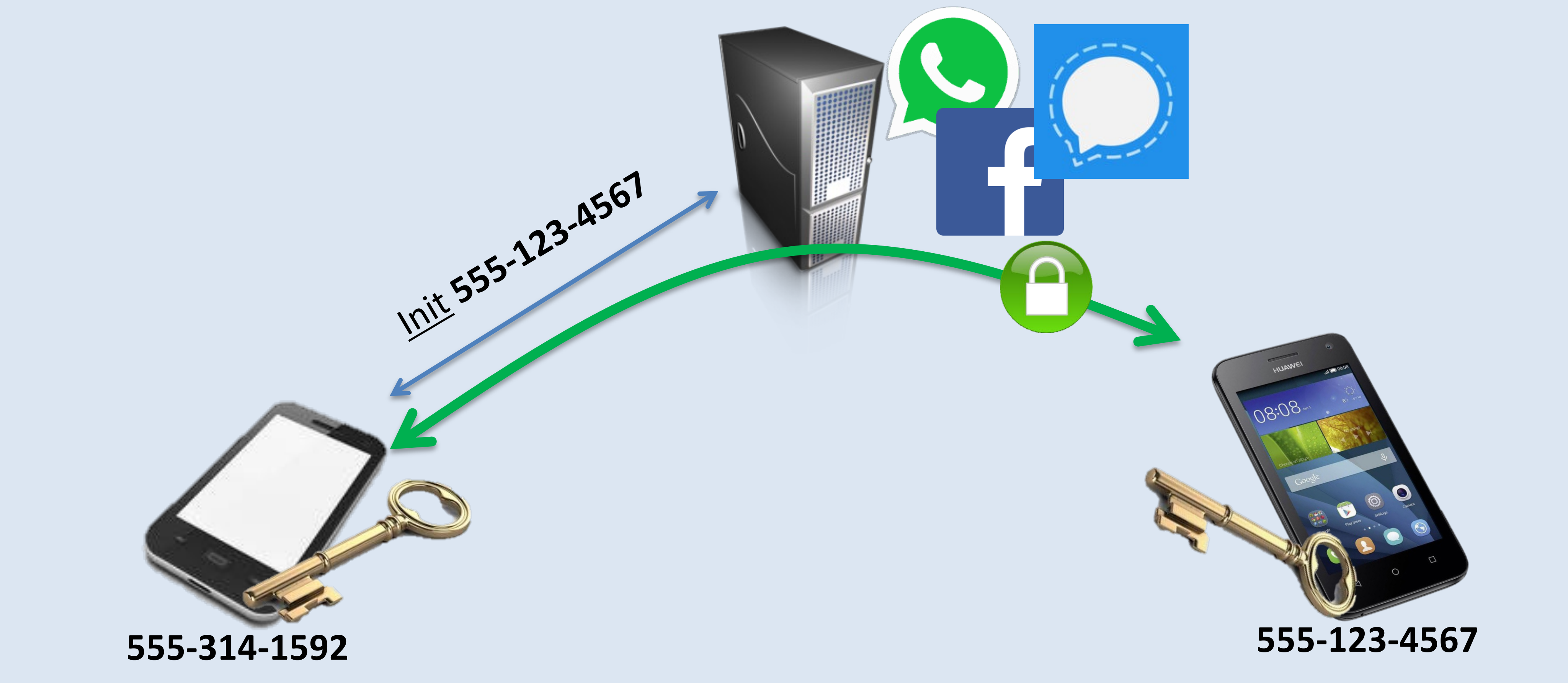


Customer and bank want to communicate securely:

- **Confidentiality** (messages remain private)
- **Integrity** (accepted messages are as sent)
- **Authenticity** (is it the bank? is this the customer?)

Example of protocols achieving this: TLS, SSH, IPsec, PGP

Example 2 – Secure Messaging



Example: Signal protocol (used by WhatsApp, Facebook, Google, …)

End-to-end encryption (provider does not learn the keys!)

Forward secrecy: Device compromise does not help decrypting previous communication

Data protection: In transit vs at rest

- The above examples are of data protection ”in transit”
- Other application scenario: Data protection at rest.

Example 3 – Storage Server

amazon cloud drive
Dropbox
Google Drive
OneDrive

Assumption: Cloud storage is not trusted

- Surveillance
- Account may be compromised

User knows **secret key**, provider does not

Symmetric Cryptography

Symmetric cryptography (aka. secret-key cryptography) considers the setting where the **sender** and the **receiver** share the same secret key, and want to communicate securely in presence of an **adversary**.

- Sender = **Alice**, Receiver = **Bob**, Adversary = **Eve**
- Secret key is usually a (randomly chosen) string, hard to guess – e.g., 128 bits or more
- Sender = receiver possible (data-at-rest scenarios, Eve is the server/memory)

Symmetric Cryptography – Tasks

1. Symmetric encryption
2. Symmetric message authentication

combination = authenticated encryption

Why important? A large portion of the cryptography you actually end up using is symmetric

- Super-fast, on-chip support, …

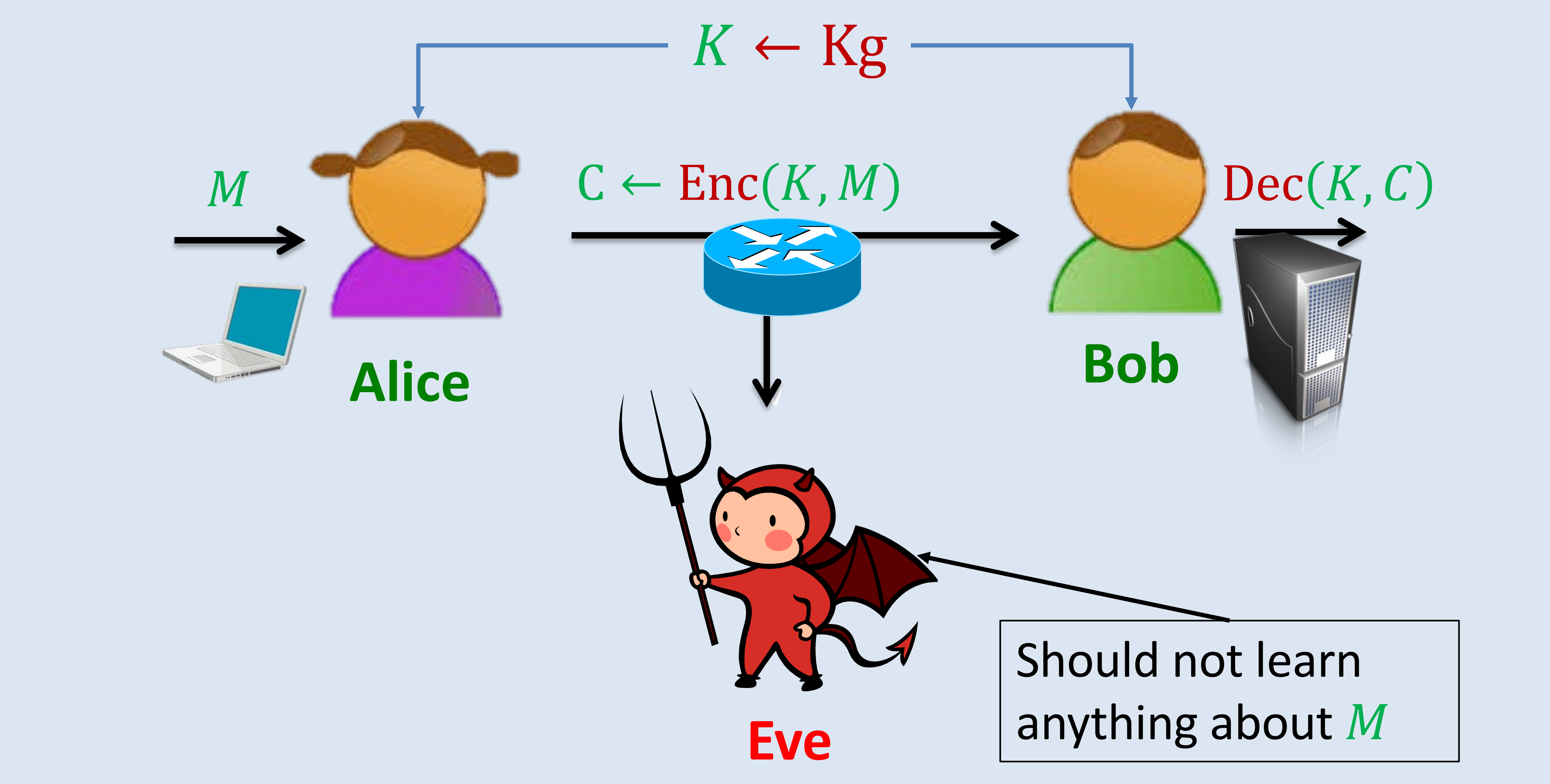
Symmetric Encryption Scheme

Definition. A **symmetric encryption scheme** consists of three algorithms Kg, Enc, and Dec

- **Key generation algorithm** Kg, takes no input and outputs a (random) *secret key* K
- **Encryption algorithm** Enc, takes input the key K and the *plaintext* M, outputs *ciphertext* $\mathrm{C} \leftarrow \mathrm{Enc}(K, M)$
- **Decryption algorithm** Dec, is such that

$$\mathrm{Dec}\big(K, \mathrm{Enc}(K, M)\big) = M$$

Symmetric Encryption – Confidentiality


$K \leftarrow \mathrm{Kg}$
M
$\mathrm{C} \leftarrow \mathrm{Enc}(K, M)$
$\mathrm{Dec}(K, C)$
Alice
Bob
Eve

Should not learn anything about M

How do Alice and Bob agree on a key: Later in this class!

A toy cipher: Mono-alphabetic substitution

Key Generation: Random one-to-one mapping {A, B, …, Z} → {A, B, …, Z}

A	B	C	D	E	F	G	H	I	J	K	L	M
O	C	S	G	K	H	T	E	U	Y	P	A	X

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
R	I	N	D	V	Q	F	J	Z	B	M	W	L

How many possible keys are there?

$$26 \times 25 \times 24 \times \cdots = 26!$$

A toy cipher: Mono-alphabetic substitution

Key Generation: Random one-to-one mapping {A, B, …, Z} → {A, B, …, Z}

A	B	C	D	E	F	G	H	I	J	K	L	M
O	C	S	G	K	H	T	E	U	Y	P	A	X

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
R	I	N	D	V	Q	F	J	Z	B	M	W	L

Encryption: Replace every single letter with corresponding letter in secret key.

A toy cipher: Mono-alphabetic substitution

Key Generation: Random one-to-one mapping {A, B, …, Z} → {A, B, …, Z}

A	B	C	D	E	F	G	H	I	J	K	L	M
O	C	S	G	K	H	T	E	U	Y	P	A	X

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
R	I	N	D	V	Q	F	J	Z	B	M	W	L

Encryption: Replace every single letter with corresponding letter in secret key.

Plaintext =

S	O		L	O	N	G		A	N	D		T	H	A	N	K	S		F	O	R		A	L	L		T	H	E		F	I	S	H

Ciphertext =

Q	I		A	I	R	T		O	R	G		F	E	O	R	P	Q		H	I	V		O	A	A		F	E	K		H	U	Q	E

Decryption?

Can be cast in the format defined above!

Definition. A **symmetric encryption scheme** consists of three algorithms Kg, Enc, and Dec

- **Key generation algorithm** Kg, takes no input and outputs a (random) *secret key* K
- **Encryption algorithm** Enc, takes input the key K and the *plaintext* M, outputs *ciphertext* $\mathrm{C} \leftarrow \mathrm{Enc}(K, M)$
- **Decryption algorithm** Dec, is such that

$$\mathrm{Dec}\big(K, \mathrm{Enc}(K, M)\big) = M$$

Is monoalphabetic substitution secure?

How do we decide whether a scheme is secure?

Breaking encryption

We need to address two questions:

1. What does the adversary know to start with?
2. What do we want to protect?

What does the attacker know: Kerchoff's principle

A cryptosystem must be secure even if everything about the system, except the key, is public knowledge.

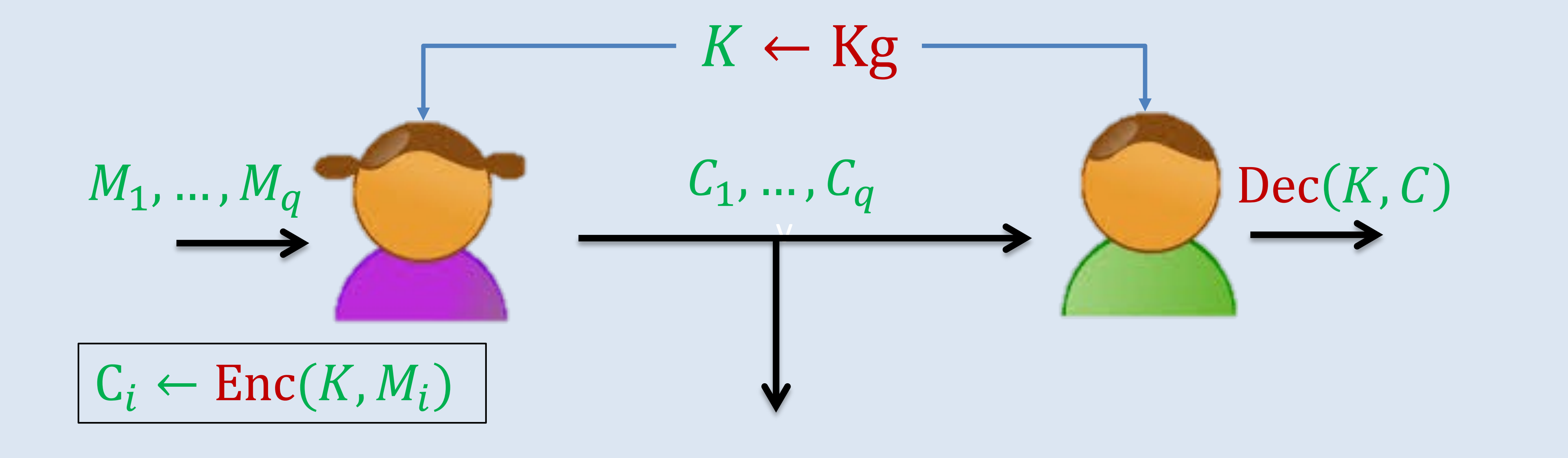
Why is this a good idea?

In other words: **No security by obscurity!**

- Hiding specifications of encryption schemes still common in industrial settings
- However, often fails. Example: GSM standard
  https://www.sans.org/reading-room/whitepapers/telephone/gsm-standard-an-overview-security-317

Breaking encryption – the basic game

- Adversary does not learn the secret key
- Adversary knows the specification of Kg, Enc, and Dec.



$K \leftarrow \mathrm{Kg}$

$M_1, \ldots, M_q$

$C_1, \ldots, C_q$

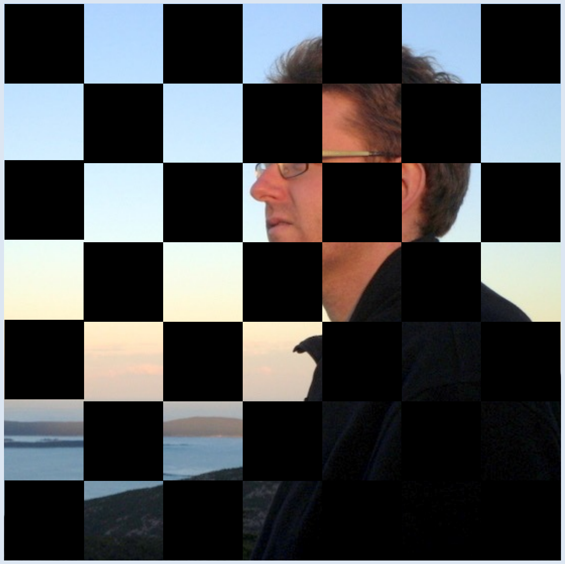
$\mathrm{Dec}(K, C)$

$\mathrm{C}_i \leftarrow \mathrm{Enc}(K, M_i)$

Ciphertext-only attack: The adversary sees ciphertexts $C_1, \ldots, C_q$, and should not be able to recover any "useful information" about the plaintexts $M_1, \ldots, M_q$

What is useful information?

- Recovering all of $M_1, \ldots, M_q$ is certain useful
- But recovering **partial information** is also useful (context-dependent)



Example of useful partial information:
Around 50% of the data has been erased from this picture, yet contents are still clear

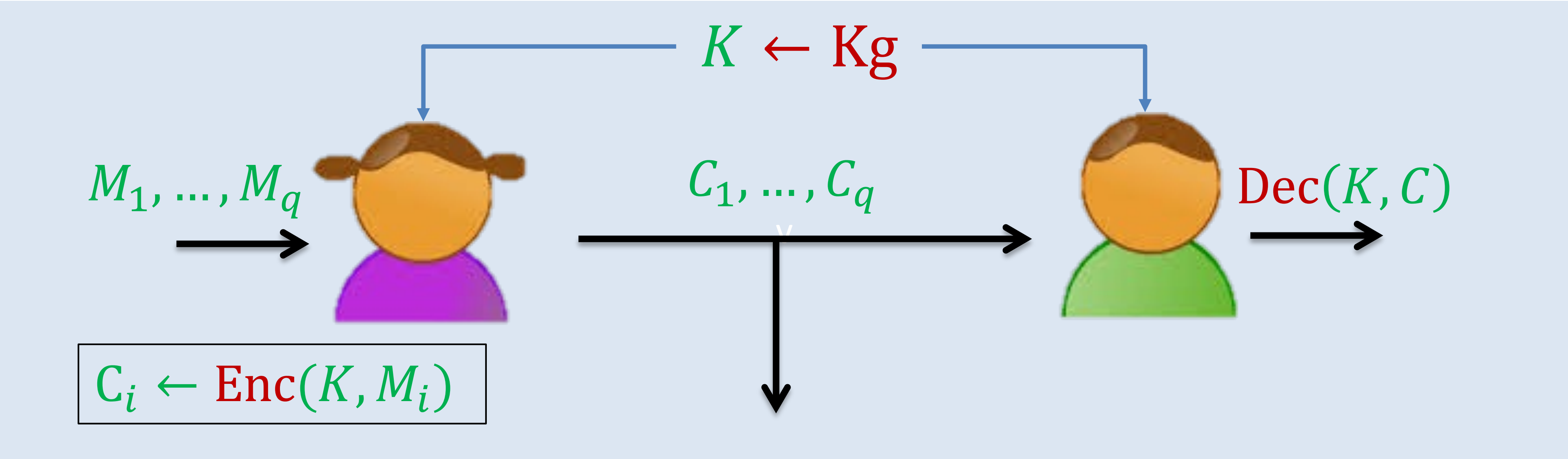
Warning: One may have some a-priori information about the plaintext(s) (e.g., data consists of election results, pictures, tax forms). The goal is to recover information that is not a-priori known!

An important distinction – Security vs attacks

An **attack** is successful as long as it recovers some useful information about the plaintext(s).

A **secure encryption scheme** should hide all possible partial information about the plaintext(s), since what is useful is usage-dependent.

Do you need to recover the key?



$K \leftarrow \mathrm{Kg}$

$M_1, \ldots, M_q$

$C_1, \ldots, C_q$

$\mathrm{Dec}(K, C)$

$\mathrm{C}_i \leftarrow \mathrm{Enc}(K, M_i)$

Full break: Attacker recovers K from $C_1, \ldots C_q$

Easy to recover any plaintext given K: $M_i \leftarrow \mathrm{Dec}(K, C_i)$

However: attacker does not necessarily need to learn the key to compromise the encryption.

Mono-alphabetic Substitution Ciphers

Key Generation: Random one-to-one mapping {A, B, …, Z} → {A, B, …, Z}

	A	B	C	D	E	F	…	X	Y	Z	plaintext digit
K =	W	T	F	N	B	H	…	Y	O	I	ciphertext digit

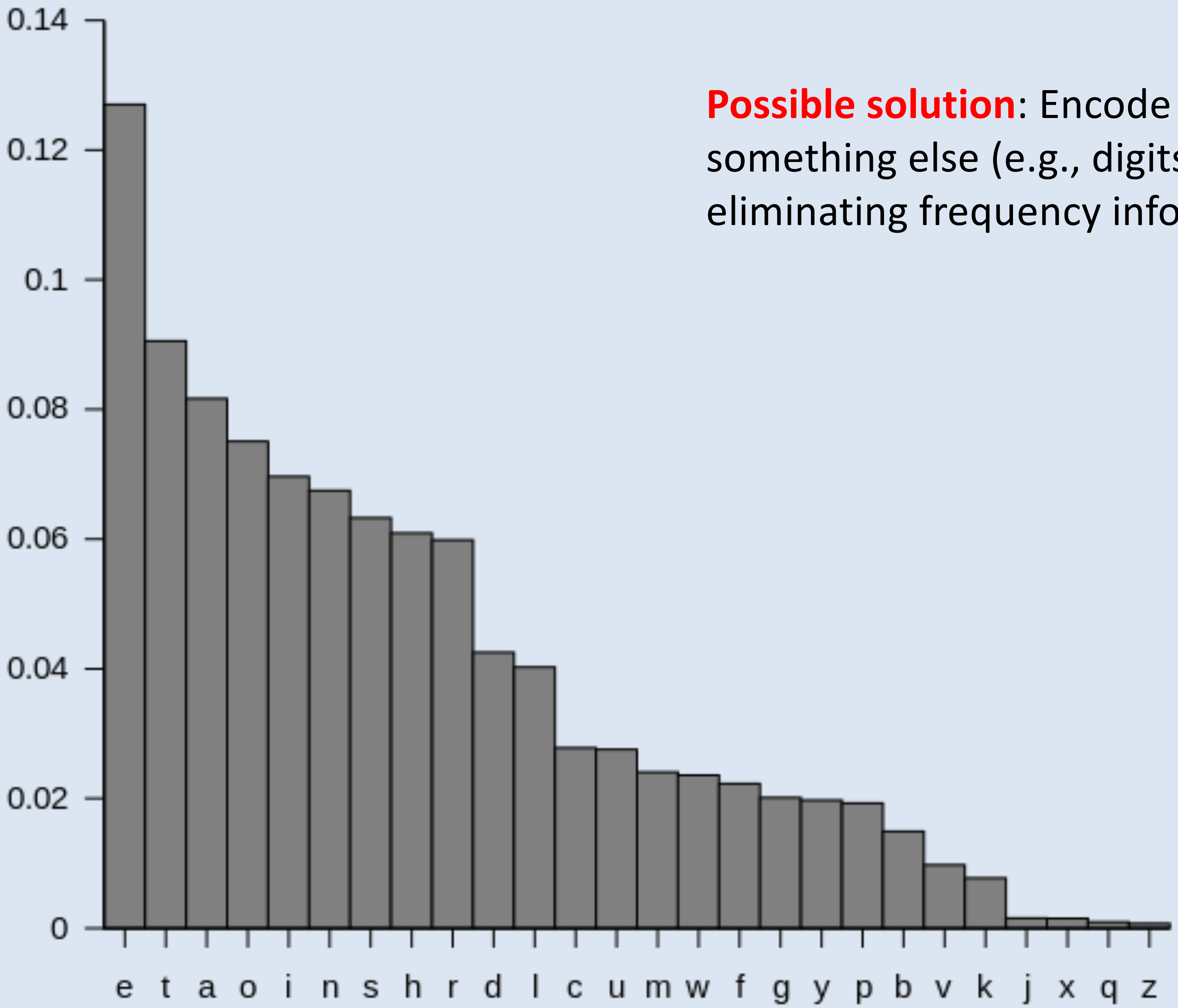
Encryption: Replace every single letter with corresponding letter in secret key.

so long and thanks
for all the fish

id udea wen mswevi hdk
wuu msb hris

Can we break it with an ciphertext-only attack?

Not all letters are equally likely!



Possible solution: Encode text as something else (e.g., digits), eliminating frequency information.

"English letter frequency (frequency)" by Nandhp - Own work

Is this the only way to break a system?

No! Often, attacker can exploit additional partial information about the behavior of the scheme under the given secret key

Known-plaintext attacks

”The Bletchley Park team would guess some of the plaintext based upon when the message was sent. For instance, a daily weather report was transmitted by the Germans, at the same time every day. Due to the regimented style of military reports, it would contain the word *Wetter* (German for "weather") at the same location in every message and knowing the local weather conditions helped Bletchley Park guess other parts of the plaintext as well.”

https://en.wikipedia.org/wiki/Known-plaintext_attack

Common attack settings

Ciphertext-only attack

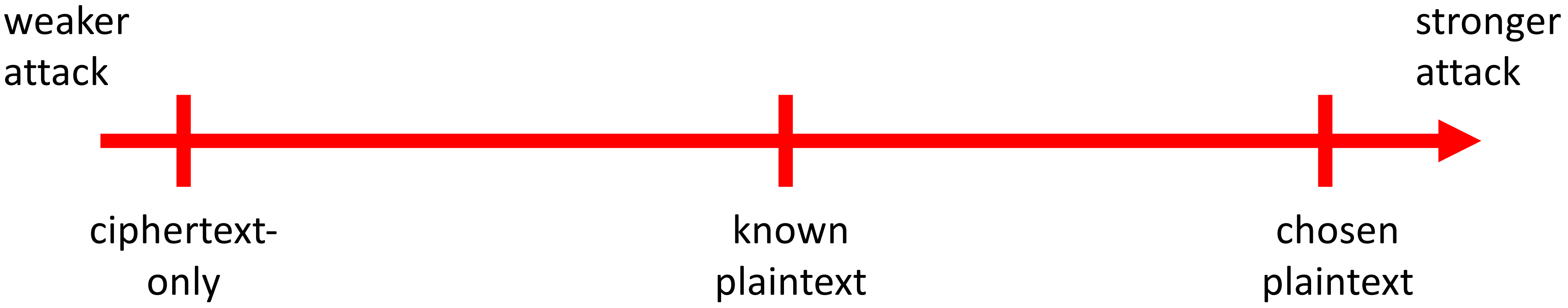
- attacker only sees ciphertexts $C_1, \ldots, C_q$

Known-plaintext attack

- attacker learn additionally some examples consisting of plaintext-ciphertext pairs $(M_1^*, C_1^*), \ldots, (M_p^*, C_p^*)$

Chosen-plaintext attack

- attacker can arbitrarily choose what the plaintexts $M_1^*, \ldots, M_p^*$ are in the examples



If encryption resists stronger class of attacks, it also offers stronger security!

Bottom line: Good encryption must resist chosen-plaintext attacks!

How does a known-plaintext attack affect monoalphabetic substitution?

Intercepted ciphertext:

$C =$	I	J	K	H	L	M	A	U	B	A	I	L	U	W	M	R

Short = very hard, no effective statistical analysis!

Additionally intercepted plaintext-ciphertext pair:

$M^* =$	N	O	T	H	I	N	G	P	R	E	A	C	H	E	S	B	E	T	T	E	R	T	H	A	N	T	H	E	A	C	T
$C^* =$	D	J	M	O	W	D	Z	H	U	A	S	I	O	A	B	E	A	M	M	A	U	M	O	S	D	M	O	A	S	I	M

Secret key?

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
?	?	?	?	?	?	?	?	?	?	?	?	?

Intercepted ciphertext:

$C =$	I	J	K	H	L	M	A	U	B	A	I	L	U	W	M	R
$M =$	C	O	?	P	?	T	E	R	S	E	C	?	R	I	T	?

Short = very hard, no effective statistical analysis!

Additionally intercepted plaintext-ciphertext pair:

$M^* =$	N	O	T	H	I	N	G	P	R	E	A	C	H	E	S	B	E	T	T	E	R	T	H	A	N	T	H	E	A	C	T
$C^* =$	D	J	M	O	W	D	Z	H	U	A	S	I	O	A	B	E	A	M	M	A	U	M	O	S	D	M	O	A	S	I	M

Secret key?

A	B	C	D	E	F	G	H	I	J	K	L	M
S	E	I	?	A	?	Z	O	W	?	?	?	?

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	J	H	?	U	B	M	?	?	?	?	?	?

Fixing mono-alphabetic substitution

How can we prevent the above attacks?

Possible solution: Encrypt multiple characters at a time (call these blocks)!

AAAAAAAAAA	AAAAAAAAAB	AAAAAAAAAC	…	ZZZZZZZZZZ
ZIMDQQLPCV	QVMWIAZFAA	CUHDJQNXAZ		MNIOWEWMBA

Why better?

Statistical analysis harder, even known-plaintext attack above not very useful

What is the problem?

Key length becomes enormous! Here, 26^{10} != 1.4 ×10^{14}! keys

Solution: Block cipher

Informal definition. A **block cipher** is a substitution cipher where the plaintext is made of blocks from a very large alphabet, but with a very compact key.

Formally, a special case of encryption algorithm:

- Kg outputs a random key of length k bits
  Typically $k = 128$ or $k = 256$
- Enc is such that for all keys K, $\mathrm{Enc}(K, .)$ is a one-to-one function $\{0,1\}^n \rightarrow \{0,1\}^n$ [The substitution table]
  n = **block length**; usually $n = 128$

That is, the blocks are n-bit strings

The magic of block ciphers

- There are $2^{128}!$ permutations over 128-bit strings. *If the key described the table for randomly chosen such permutation, the key would be roughly* 128×2^{128} *bits long*
- A block cipher is "as good as" choosing such a randomly chosen permutation, but only needs a short key, e.g., 128 bits
- How do we build them?
- Two examples next: **DES** and **AES**

Data encryption standard (DES)

Originally called Lucifer

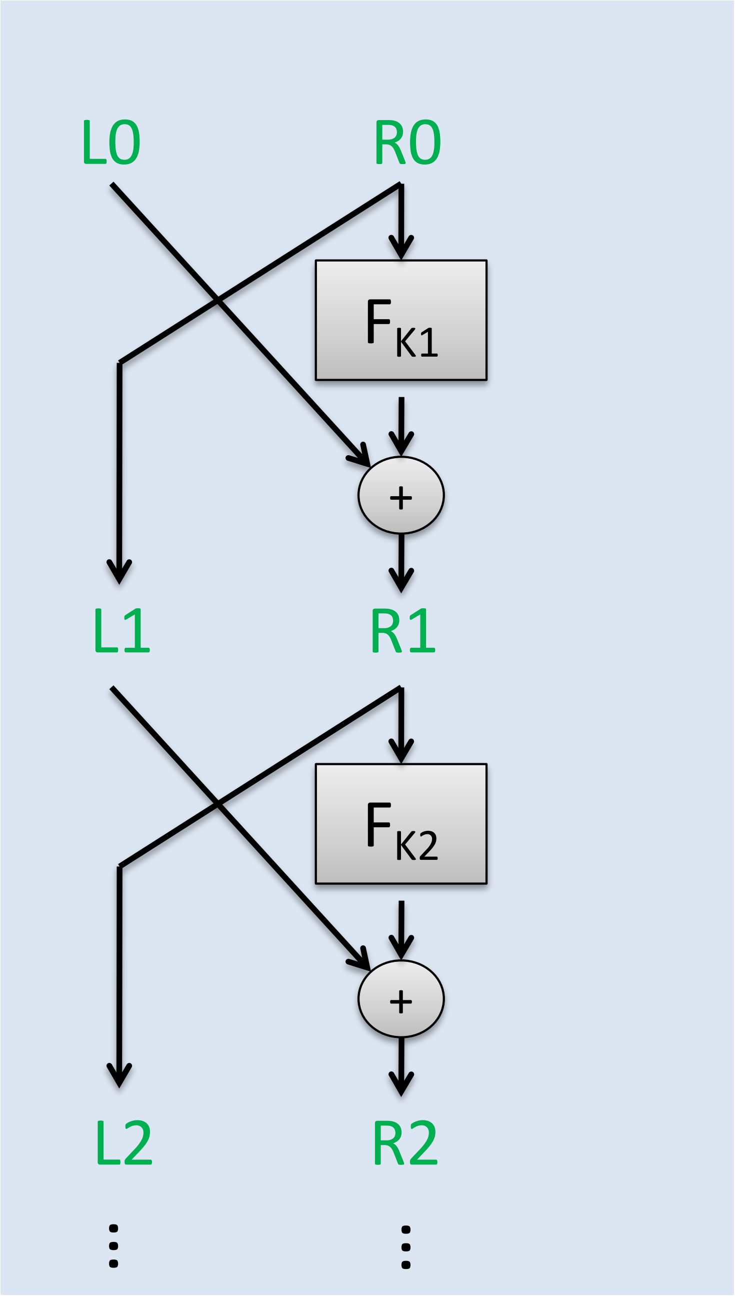
- Team at IBM, led by Horst Feistel
- Input from NSA
- Standardized by NIST in 1976

$$n = 64$$
$$k = 56$$

Split 64-bit input into L0,R0 of 32 bits each

Repeat round 16 times

Each round applies function F using separate round keys K1 … K16 derived from main key.



DES is essentially broken

Attack	Attack type	Complexity	Year
Biham, Shamir	Chosen plaintexts, recovers key	2^{47} plaintext, ciphertext pairs	1992
DESCHALL	Unknown plaintext, recovers key	$2^{56/4}$ DES computations 41 days	1997
EFF Deepcrack	Unknown plaintext, recovers key	~4.5 days	1998
Deepcrack + DESCHALL	Unknown plaintext, recovers key	22 hours	1999

3DES (use DES 3 times in a row with more keys) expands keyspace to 168 bits and still found in practice
(E.g., PIN-based card transactions)

Bottom line: While 3DES may still be fine, just stay away from it if you can!

Advanced Encryption Standard (AES)

Response to 1990s attacks:

- NIST has design competition for new block cipher standard
- 5 year design competition
- 15 designs, Rijndael design chosen
- AES is very fast (AES-NI native instruction on modern chips)

Advanced Encryption Standard (AES)

Rijndael (Rijmen and Daemen)

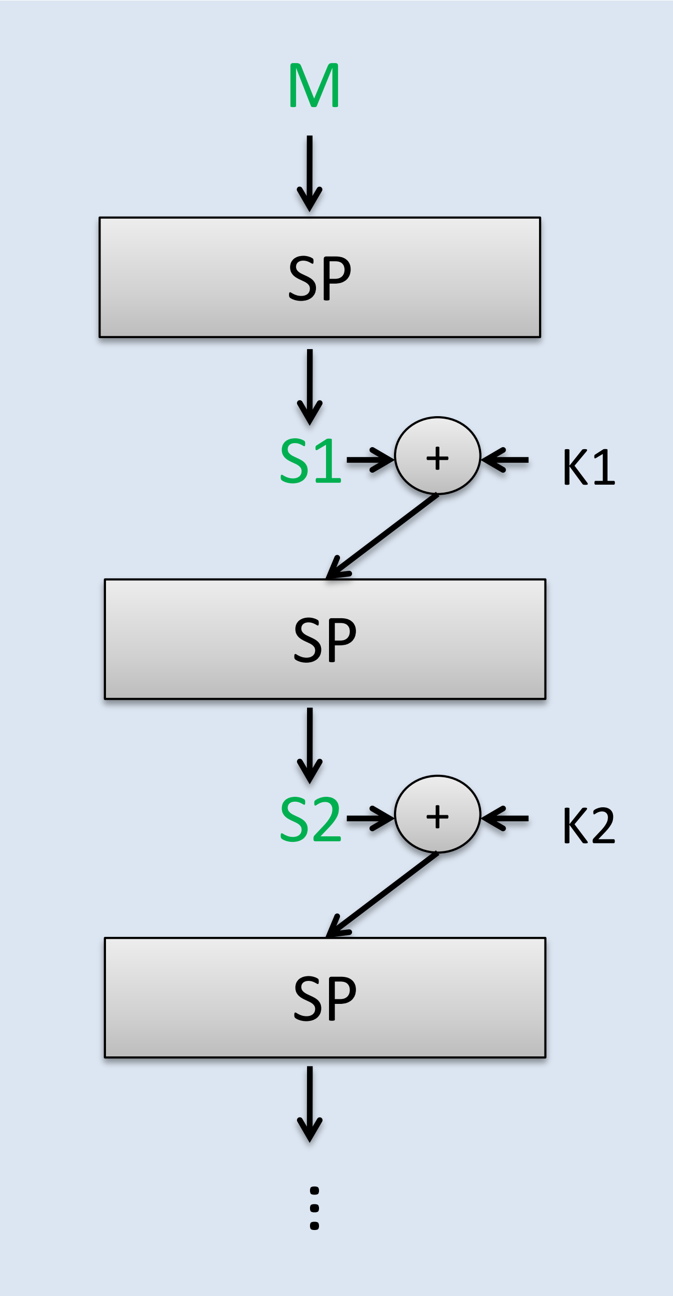
$$n = 128$$
$$k = 128, 192, 256$$

For $k = 128$ uses 10 rounds of:

1) Substitution–Permutation (SP) step :

SubBytes (non-linear S-boxes)
ShiftRows + MixCols (invertible linear transform)

2) XOR in a round key derived from K



Best attacks against AES

Attack	Attack type	Complexity	Year
Bogdanov, Khovratovich, Rechberger	key recovery	$2^{126.1}$ time + some data overheads	2011

- Brute force requires time 2^{128}
- Approximately factor 4 speedup

Bottom line: AES is very secure, and there has been surprisingly little progress on breaking it!
Pro-tip: If someone wants to use any different than AES, insist to know why and ask a cryptographer you trust. (Never trust home-brewed crypto.)

Exception: Lightweight applications (IoT, memory encryption) often require simpler ciphers, but no clear standard

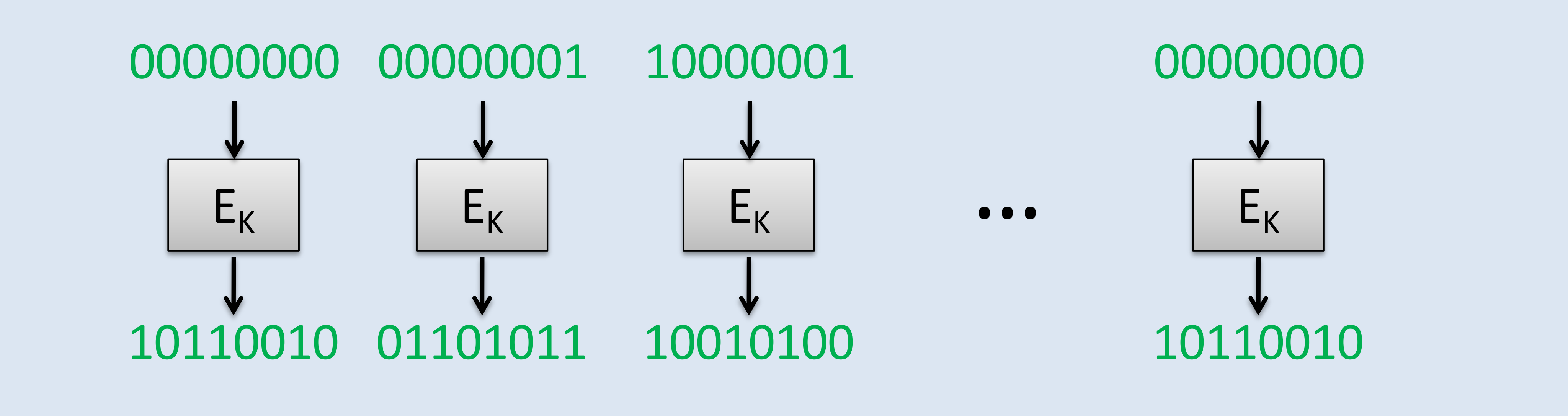
Block Cipher Security Goal

Pseudorandom permutation (PRP)
Informally: A block cipher (e.g., AES) with block length n under a random secret key behaves as an ideal mono-alphabetic substitution cipher with n-bit alphabet

- This can hold only for adversaries which have limited (yet large) adversarial resources, e.g., they are not able to recover the key (otherwise they will be able to tell)
- But as a system designer, you have to think of block ciphers as providing this functionality

Pseudorandom Permutations

Key K: Secret and random!

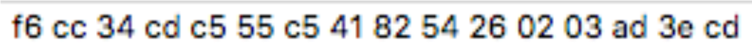


Important property: as long as the key is unknown and randomly chosen, outputs on different inputs look like **random and independent strings**

Example – Play with AES

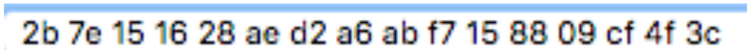
128 bits = 16 bytes

secret key 1 =



f6 cc 34 cd c5 55 c5 41 82 54 26 02 03 ad 3e cd

secret key 2 =



2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

Plaintext	Ciphertext (key 1)	Ciphertext (key 2)
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	5b c0 68 39 7b 4f 93 c4 ce d1 6a 79 94 1a 48 30	7d f7 6b 0c 1a b8 99 b3 3e 42 f0 47 b9 1b 54 6f
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01	33 93 ed 94 85 c8 90 a7 d0 33 9a 78 c0 63 33 d2	57 12 7d 40 34 b1 be bf ae f4 66 b9 c7 72 6f c6
33 93 ed 94 85 c8 90 a7 d0 33 9a 78 c0 63 33 d2	56 dc a7 27 4d eb d5 cd 71 9c 7e a5 7a 54 67 4d	8c f0 60 ca 67 67 8b 0a c6 64 9d 8f ae 76 d1 f8

How do we encrypt > 16 bytes?

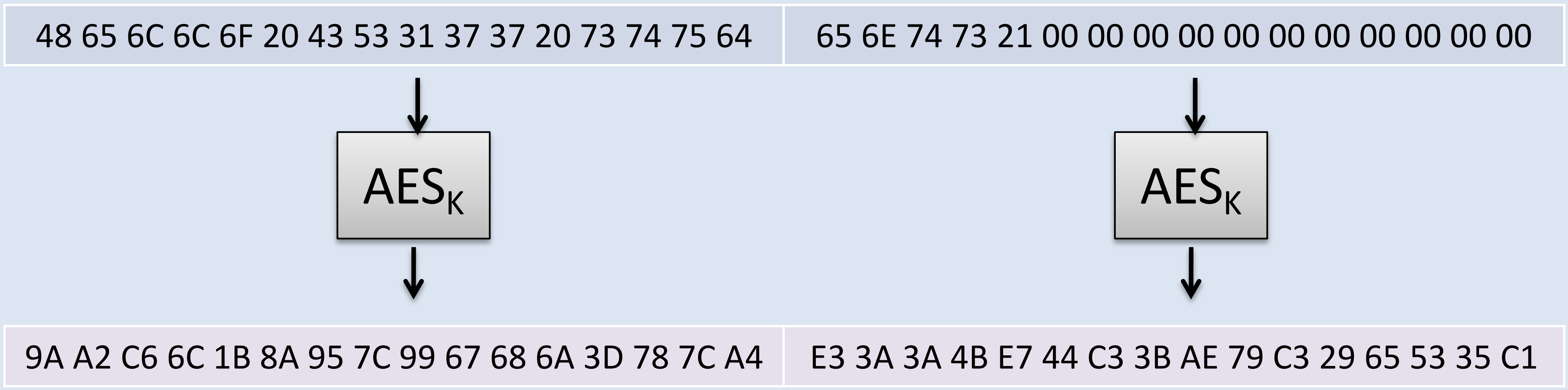
Long messages – Electronic Codebook (ECB)

Message M = “Hello CS177 students!”

We use AES, $n = 128 = 16$ bytes

	$M[1]$	$M[2]$
$M =$	48 65 6C 6C 6F 20 43 53 31 37 37 20 73 74 75 64	65 6E 74 73 21 00
$K =$	0F 15 71 C9 47 D9 E8 59 0C B7 AD D6 AF 7F 67 98	

To encrypt M using the key K:

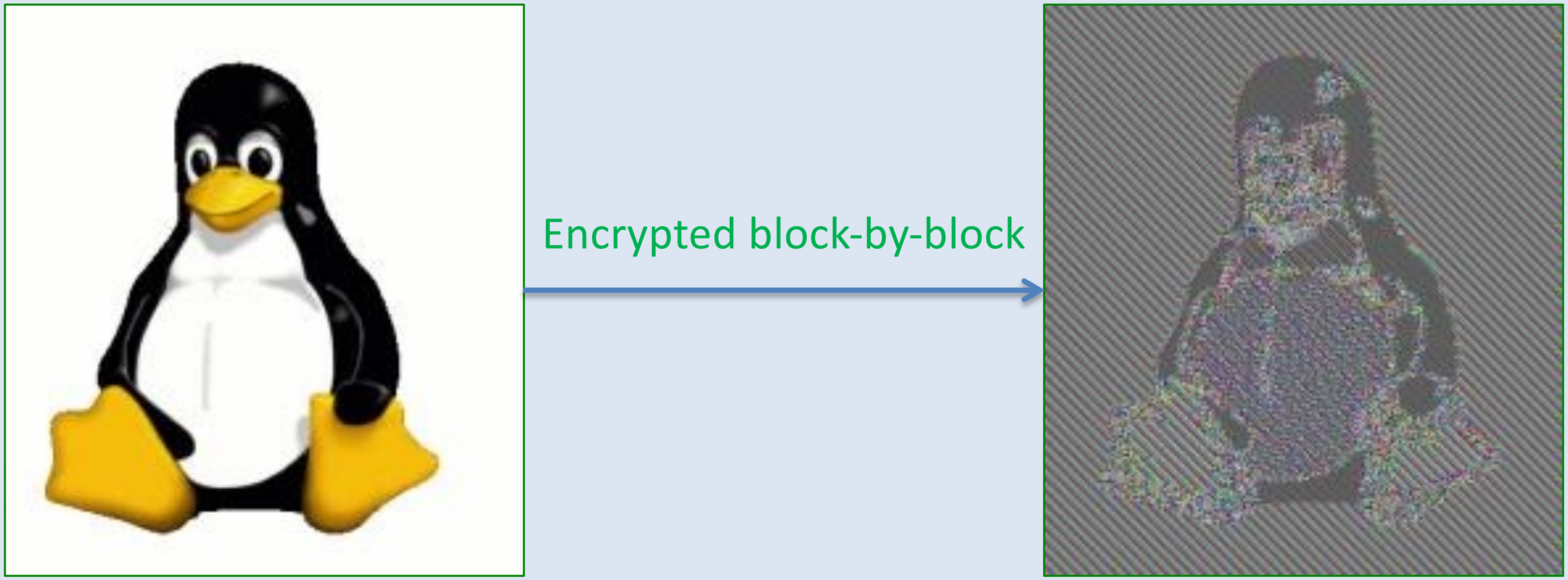


48 65 6C 6C 6F 20 43 53 31 37 37 20 73 74 75 64 | 65 6E 74 73 21 00 00 00 00 00 00 00 00 00 00 00

9A A2 C6 6C 1B 8A 95 7C 99 67 68 6A 3D 78 7C A4 | E3 3A 3A 4B E7 44 C3 3B AE 79 C3 29 65 53 35 C1

Drawbacks of block ciphers

The same 16-byte sequence is always encrypted in the same way. **Is this ever a problem?**



Encrypted block-by-block

http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

What does it mean to be secure? What should have we seen instead?

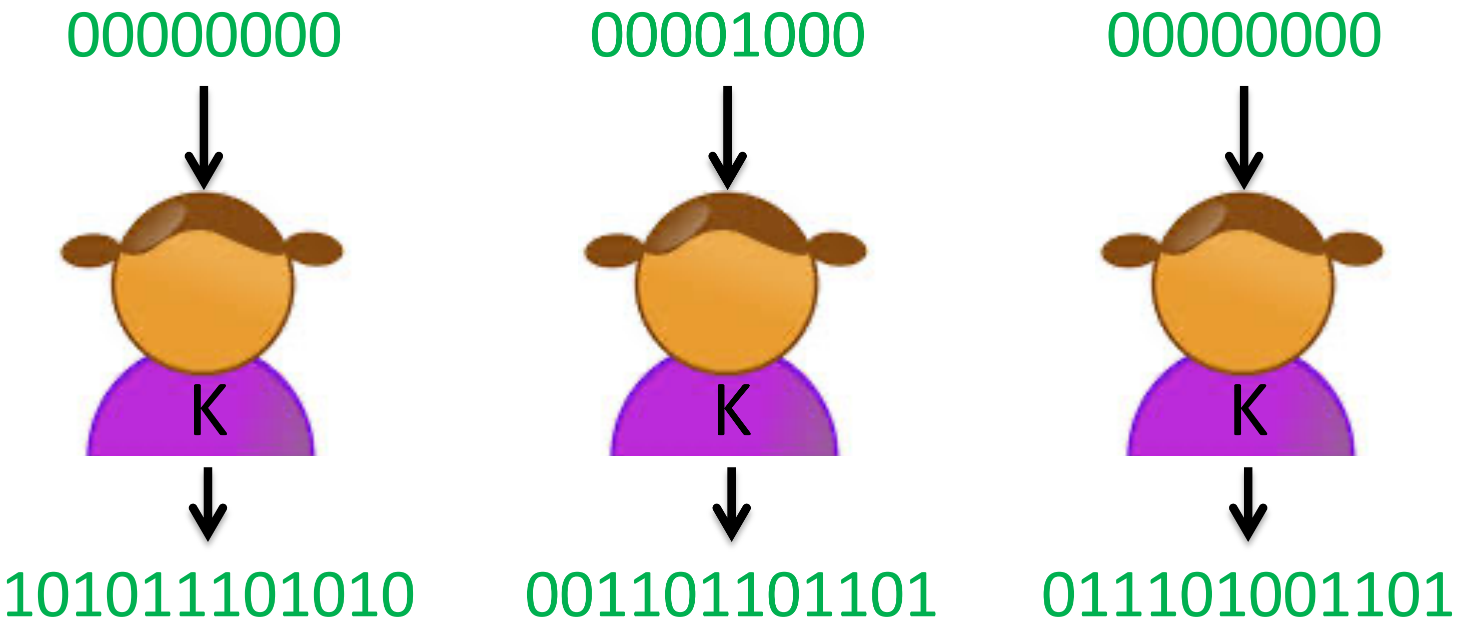
Right definition: Semantic Security [Goldwasser-Micali, 1982]

Turing Award (2013)

“Right” definition of message confidentiality

Semantic Security
[Goldwasser-Micali, 1982]

Encryption reveals **nothing about the message**, not even whether the same message was encrypted earlier!



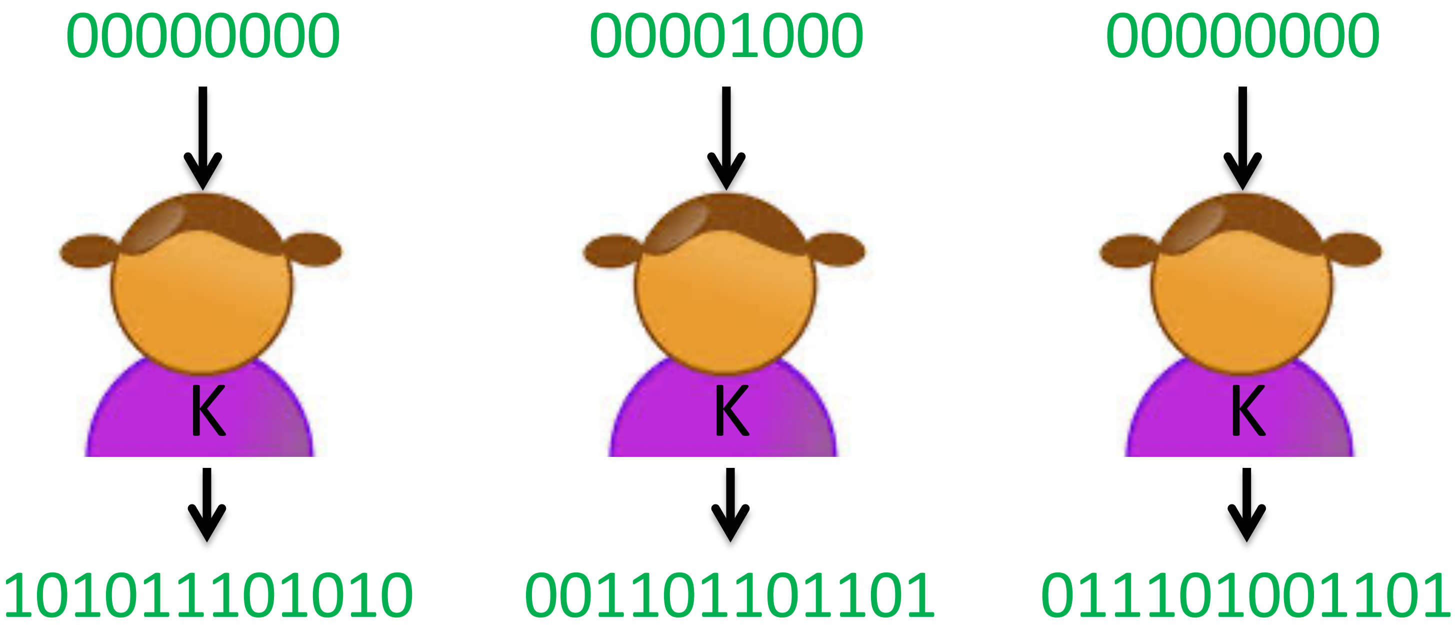
00000000 00001000 00000000

101011101010 001101101101 011101001101

Every time we encrypt (regardless of new or old message), ciphertext “looks random” to a computationally bounded adversary.

Semantic Security

[Goldwasser-Micali, 1982]



00000000 00001000 00000000

101011101010 001101101101 011101001101

To ensure this, encryption must use **randomization**
(many possible ciphertexts for same message)

Ciphertext must be longer than plaintext!

Semantically Secure Encryption

How do we achieve it from block ciphers?

A basic fact – Bitwise XOR

Bitwise XOR $\oplus$ of two strings $X, Y \in \{0,1\}^n$

Example, $n = 8$

X	0	1	0	0	1	0	1	1
Y	1	0	1	0	1	0	0	1
$X \oplus Y$	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**

XOR Magic - Masking

Fix a string $X \in \{0,1\}^n$

Imagine we pick $K \in \{0,1\}^n$ uniformly at random

That is each of the 2^n possible strings is equally likely

Question: How does $X \oplus K$ look like?

Answer: $X \oplus K$ can become every string Y with probability $1/2^n$

XOR Magic - Masking

Example. $X = 01001011$

What is the probability that $X \oplus K = 11110000$?

X	0	1	0	0	1	0	1	1
K	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
$X \oplus K$	**1**	**1**	**1**	**1**	**0**	**0**	**0**	**0**

This is exactly the probability that $K = 10111011$, which is exactly $1/2^n$

But the same is true for every X and every choice of $X \oplus K$ …

The learnt lesson!

Masking: “If we bitwise-xor a random string K to any string X, the outcome $C = X \oplus K$ is random and independent of the original X, and thus hides everything about X.”

A little bit of notation

For string $X \in \{0,1\}^n$ and natural number $a \in \mathbb{N}$, define $X + a$ as the n-bit string obtained by:

1. Interpreting X as the binary encoding of an integer b_X in $\{0,1,\dots,2^n - 1\}$
2. Compute the binary encoding Y of $b_X + a$
3. Let $X + a$ be the n least significant bits of Y

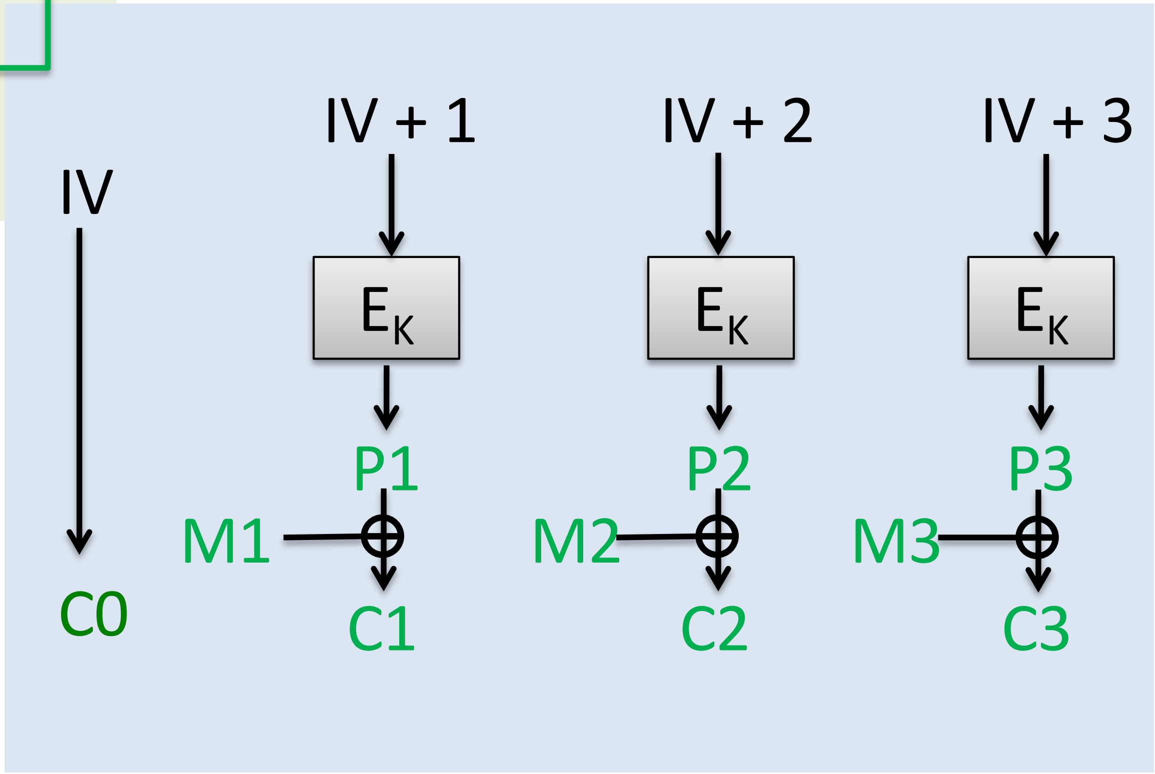
Examples: $n = 4$

$0000 + 1 = 0001$	$b_X = 0$	$b_X + a = 1$	$Y = 0001$
$0010 + 5 = 0111$	$b_X = 2$	$b_X + a = 7$	$Y = 0111$
$1111 + 2 = 0001$	$b_X = 15$	$b_X + a = 17$	$Y = 10001$

Counter Mode Encryption (CTR)

```
Algorithm $\mathrm{Enc}(K, M)$:
Split $M$ into blocks $M[1], \ldots, M[r]$
// all blocks except possibly $M[r]$ are $n$-bits
Pick random $\mathrm{IV} \in \{0,1\}^n$
$C[0] \leftarrow \mathrm{IV}$
for $i = 1, \ldots, r$ do
    $P[i] \leftarrow E_K(\mathrm{IV} + i)$
    $C[i] \leftarrow M[i] \oplus P[i]$
return $C[0], C[1], \ldots, C[r]$
```

Note: If $M[r]$ shorter than n bits, then also shorten $P[r]$ as necessary



How do we decrypt?

CTR – Example

Message M = "Hello CS177 students!"
We use CTR mode with AES, $n = 128 = 16$ bytes

	$M[1]$	$M[2]$
$M =$	48 65 6C 6C 6F 20 43 53 31 37 37 20 73 74 75 64	65 6E 74 73 21 00

$K =$ 0F 15 71 C9 47 D9 E8 59 0C B7 AD D6 AF 7F 67 98

To encrypt M using the key K:

We have chosen at random $\mathrm{IV} =$ CC 32 FA B3 E9 12 47 81 FF 1B 3C D6 AA 98 42 03

$\mathrm{IV} + 1 =$ CC 32 FA B3 E9 12 47 81 FF 1B 3C D6 AA 98 42 04

$\mathrm{IV} + 2 =$ CC 32 FA B3 E9 12 47 81 FF 1B 3C D6 AA 98 42 05

$P[1] = AES_K(\mathrm{IV} + 1) =$ CD 3E 82 98 67 6C BF 69 BA C2 67 E2 4A ED 06 6A

$P[2] = AES_K(\mathrm{IV} + 2) =$ 33 22 04 53 B0 3A 1D DA 84 A9 40 A8 52 75 0B F7

CTR – Example (cont'd)

Need to compute final ciphertext

$M[1]$	$M[2]$
48 65 6C 6C 6F 20 43 53 31 37 37 20 73 74 75 64	65 6E 74 73 21 00

$\oplus$

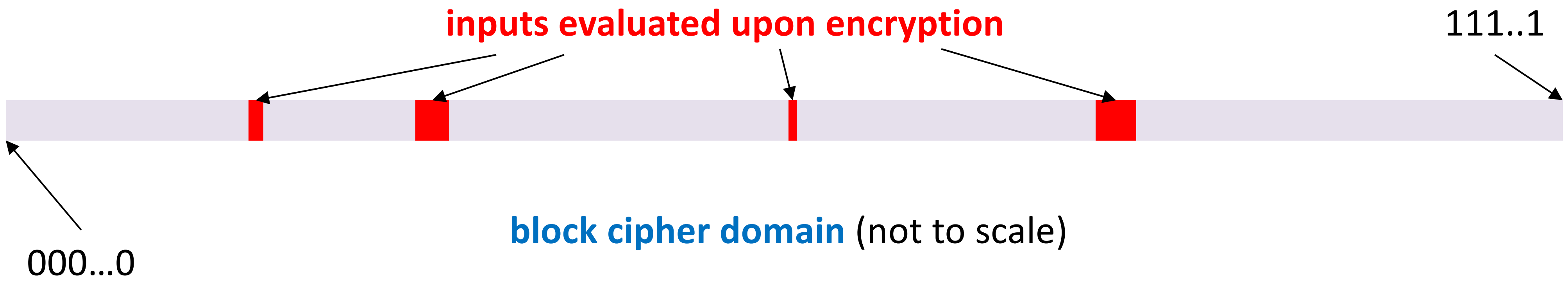
$P[1]$	$P[2]$
CD 3E 82 98 67 6C BF 69 BA C2 67 E2 4A ED 06 6A	33 22 04 53 B0 3A

CC 32 FA B3 E9 12 47 81 FF 1B 3C D6 AA 98 42 03	85 5B EE F4 08 4C FC 3A 8B F5 50 C2 39 99 73 0E	56 4C 70 20 91 3A
$C[0]$	$C[1]$	$C[2]$

Final ciphertext!

CTR – Why is it secure?

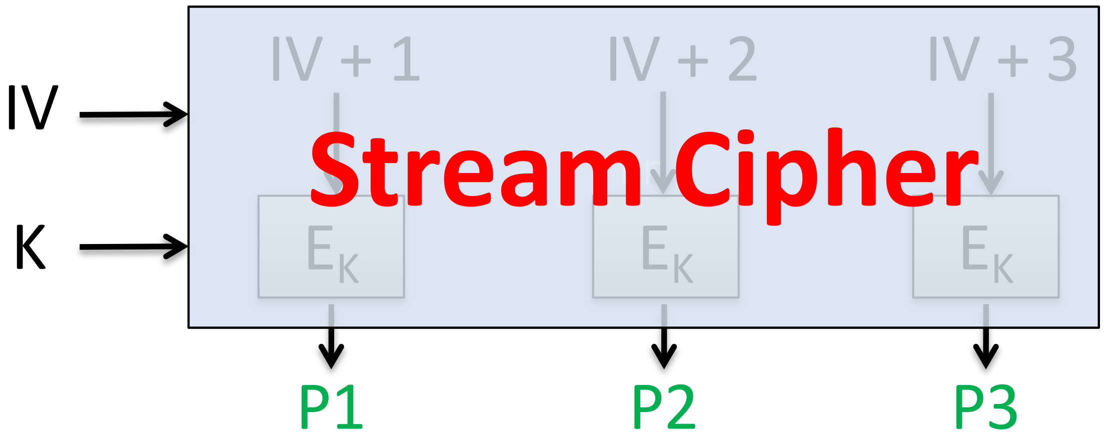
Masking strings $P[1], \ldots, P[r]$ generated upon each encryption will come from disjoint parts of the block cipher domain (with high probability), and thus look random and independent


inputs evaluated upon encryption
111..1
000…0
block cipher domain (not to scale)

Therefore: Every encryption adds a new, independent random mask to the plaintext, and thus (by our previously established fact about masking), every ciphertext looks like a fresh random string!

Stream Ciphers

Masking-generation part can be abstracted through the concept of a **stream cipher**



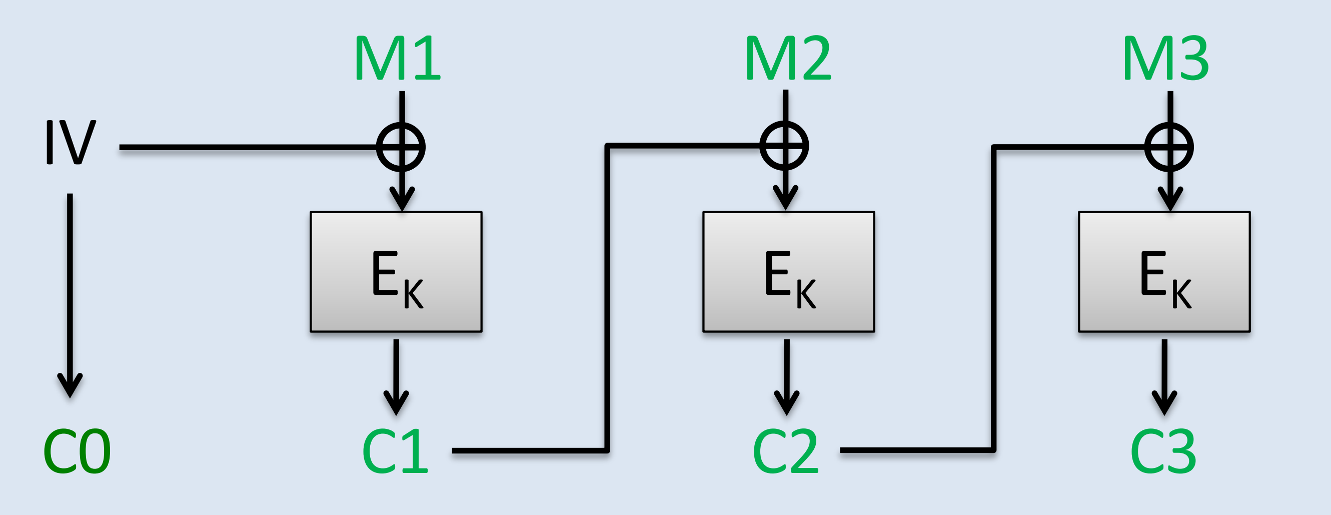
Efficient stream ciphers (not using block ciphers) exist:

- **RC4** -- completely broken, do not use, but has had a hard time dying out
  http://www.securityweek.com/new-attack-rc4-based-ssltls-leverages-13-year-old-vulnerability
- **Better: Salsa20/ChaCha**
  Especially fast on architectures without AES-NI hardware support
  Google likes it: https://security.googleblog.com/2014/04/speeding-up-and-strengthening-https.html

Alternative: Ciphertext Block Chaining (CBC)

```
**Algorithm** $\mathrm{Enc}(K, M)$:
Split $M$ into blocks $M[1], \ldots, M[r]$
// all blocks are $n$-bits
Pick random $\mathrm{IV} \in \{0,1\}^n$
$C[0] \leftarrow \mathrm{IV}$
for $i = 1, \ldots, r$ do
    $P[i] \leftarrow E_K(M[i] \oplus C[i-1])$
return $C[0], C[1], \ldots, C[r]$
```

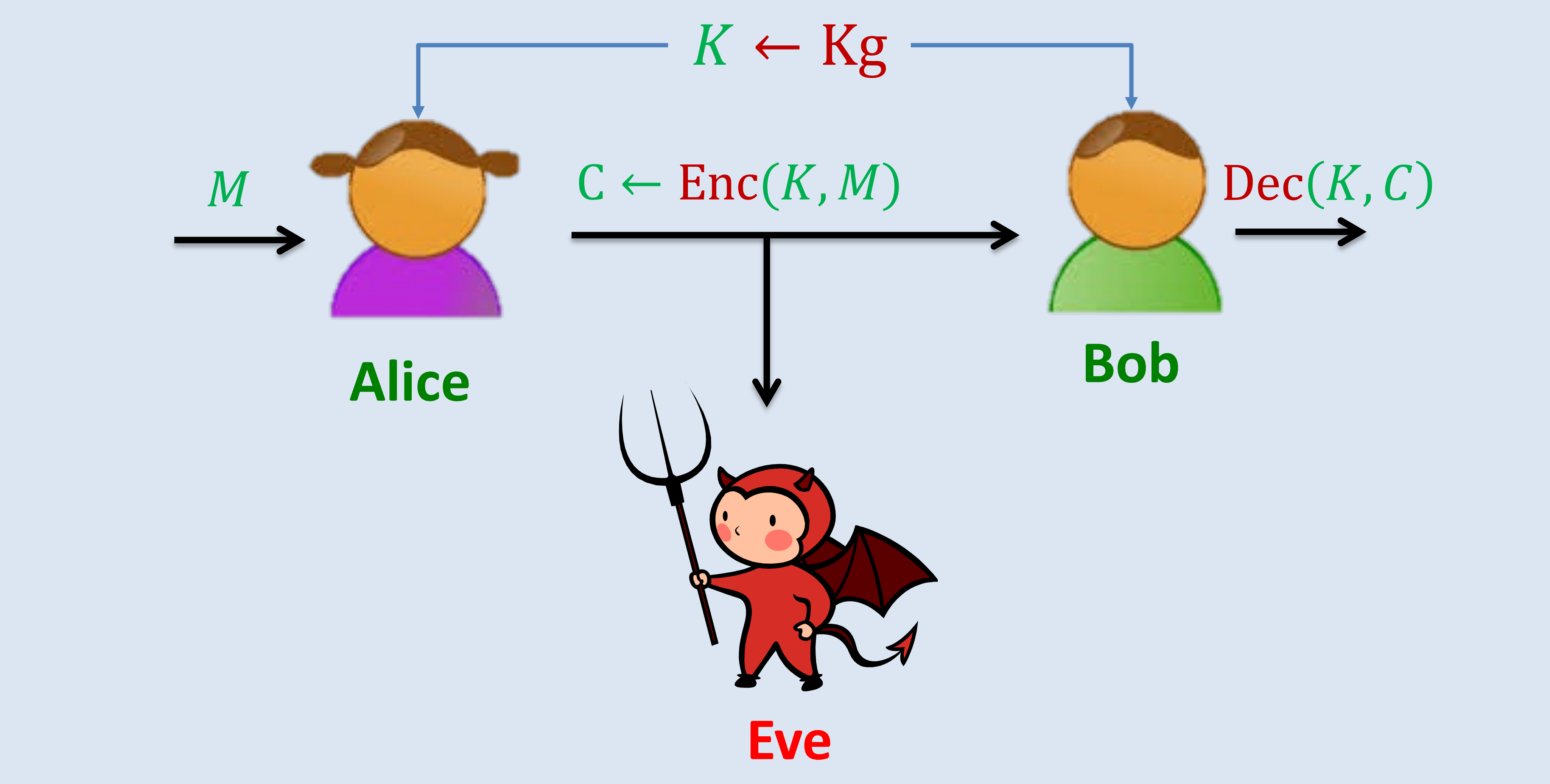
Note: One needs to make the message length a multiple of n bits.



CBC vs CTR

- CBC offers roughly same security as CTR
  - Security argument is less clear than in CTR
- For historical reasons, CBC more popular than CTR
- CTR is potentially faster than CBC, since the latter is inherently sequential, whereas CTR can be parallelized

Symmetric Encryption – Confidentiality


$K \leftarrow \mathrm{Kg}$
M
$\mathrm{C} \leftarrow \mathrm{Enc}(K, M)$
$\mathrm{Dec}(K, C)$
Alice
Bob
Eve