CS 370
Introduction to Security

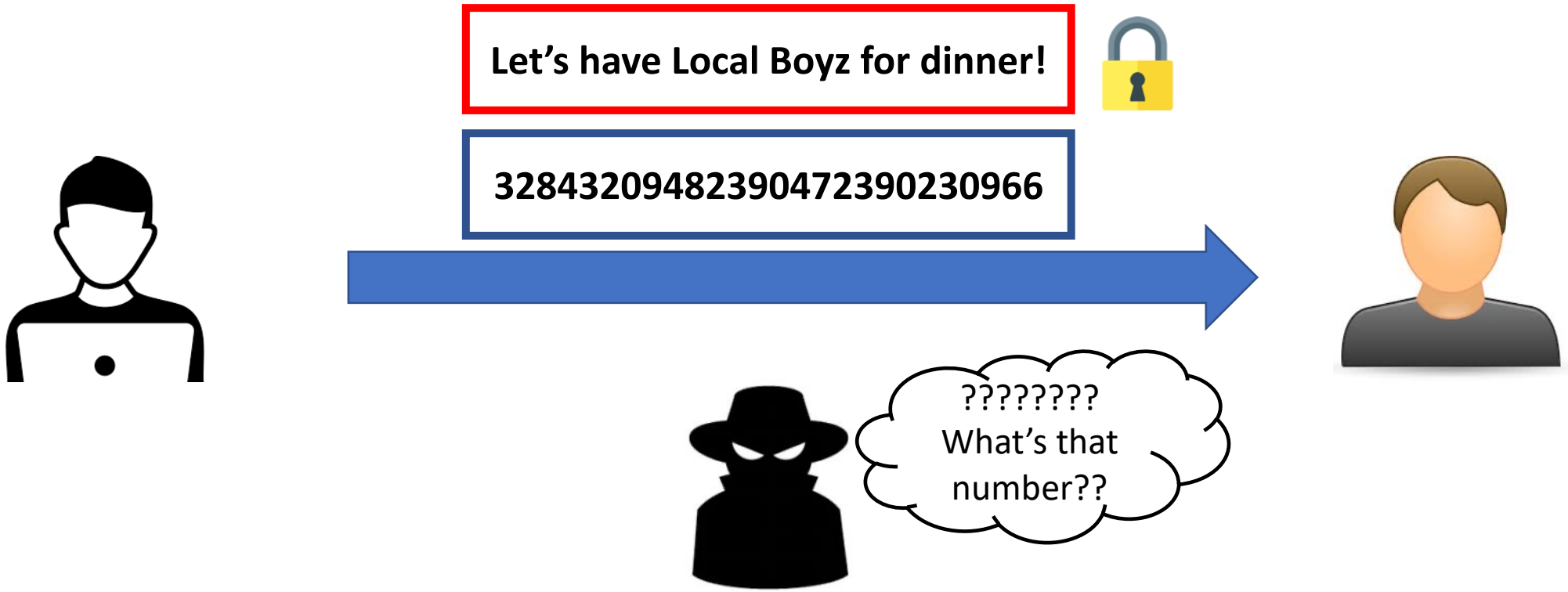
Symmetric Encryption (Block Cipher: DES/AES)

Yeongjin Jang

Oregon State University

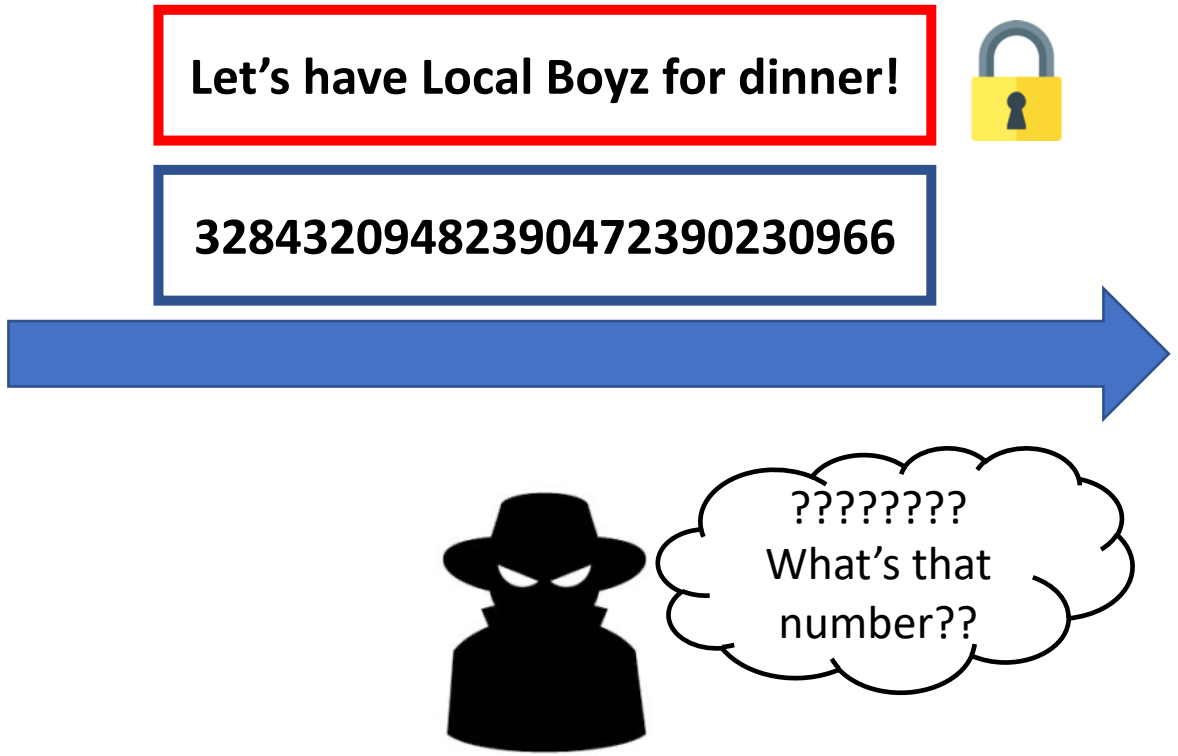
Recap: Cryptography

- Cryptography can make our communication secure



Recap: Perfect Secrecy

- Implication
  - If the distribution of the ciphertext is the same as the distribution of a random number, then
  - Adversaries may never tell if the message is a ciphertext of a plaintext or
  - A random number…

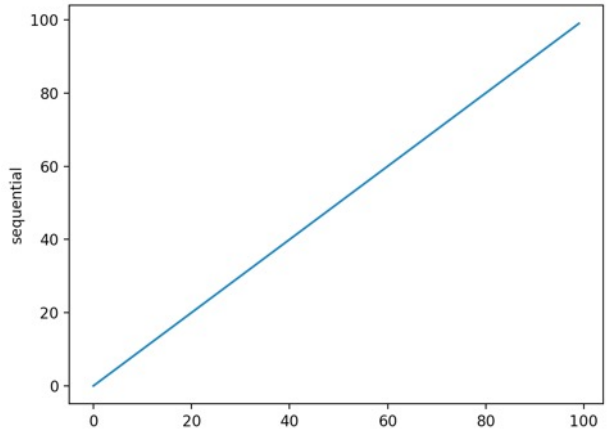


Recap: XOR Cipher

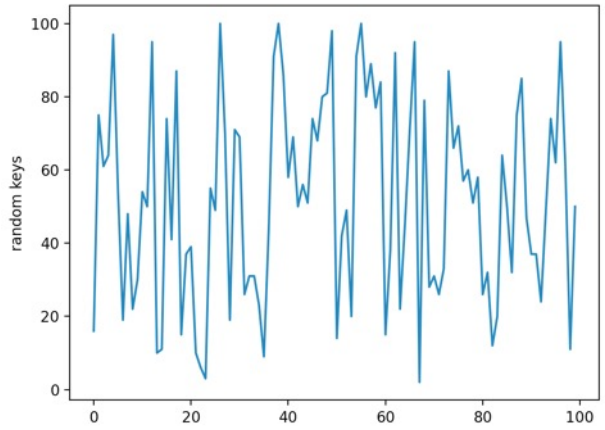
- Perfect Secrecy
  - Message M = 123456
  - Random Key K = 284603
  - Encrypted data E = M ^ K = 374267

  - Message M’ = 123457
  - Random Key K’ = 513773
  - Encrypted Data E = M’ ^ K’ = 406700

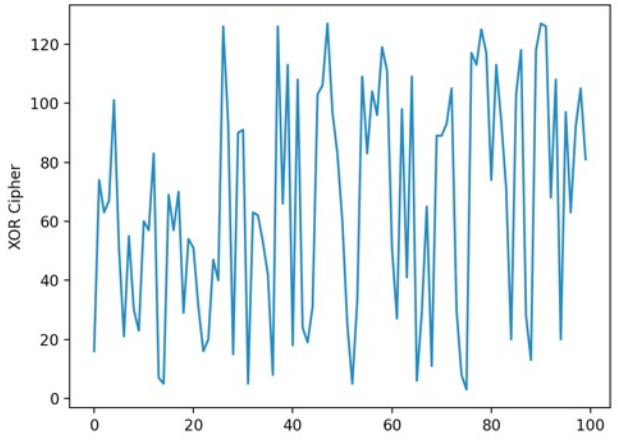
  - Message M’’ = 123458
  - Random Key K’’ = 627151
  - Encrypted Data E = M’’ ^ K’’ = 553869



Plaintext:
0—100
sequentially



One-time Pad:
0—100
Randomly
chosen



Ciphertext:
0—128
Randomly
distributed

Limitations in Using Perfect Secrecy Scheme in Reality

- Key size must be the same with the message size
  - 1GB video file, then we need an 1GB key
  - Key MUST NOT be re-used!!!
- We need to share the key beforehand to the other party
  - Wait, share an 1GB key before hand to send 1GB of data later?
  - Then, what about hand out the data at that time, not the key???
- There could be some of such use cases but
  - Impractical… we wish to have a short key to encrypt a big chunks of data..

What if We Could Have an Identical Random Generator Pot?

- Dream about the world that has

A random number generator

Physically meet and share the identical generator securely

What if We Could Have an Identical Random Generator Pot?

- Dream about the world that has

A random number generator

A random number generator

What if We Could Have an Identical Random Generator Pot?

- Dream about the world that has

1: 0x1b395a46
2: 0xf1737202
3: 0xccf0de05
4: 0x908b0feb
5: 0x9d4c9add
…

A random number generator

1: 0x1b395a46
2: 0xf1737202
3: 0xccf0de05
4: 0x908b0feb
5: 0x9d4c9add
…

A random number generator

What if We Could Have an Identical Random Generator Pot?

- Dream about the world that has

Encrypt message 1 with 0x1b395a46
Encrypt message 2 with 0xf1737202
Encrypt message 3 with 0xccf0de05…

1: 0x1b395a46
2: 0xf1737202
3: 0xccf0de05
4: 0x908b0feb
5: 0x9d4c9add
…

A random number generator

Decrypt message 1 with 0x1b395a46
Decrypt message 2 with 0xf1737202
Decrypt message 3 with 0xccf0de05…

1: 0x1b395a46
2: 0xf1737202
3: 0xccf0de05
4: 0x908b0feb
5: 0x9d4c9add
…

A random number generator

What if We Could Have an Identical Random Generator Pot?

- Dream about the world that has

Encrypt message 1 with 0x1b395a46
Encrypt message 2 with 0xf1737202
Encrypt message 3 with 0xccf0de05…

A random number generator

We can securely encrypt messages with One-time Pad (e.g., XOR Cipher)

1: 0x1b395a46
2: 0xf1737202
3: 0xccf0de05
4: 0x908b0feb
5: 0x9d4c9add
…

Decrypt message 1 with 0x1b395a46
Decrypt message 2 with 0xf1737202
Decrypt message 3 with 0xccf0de05…

1: 0x1b395a46
2: 0xf1737202
3: 0xccf0de05
4: 0x908b0feb
5: 0x9d4c9add
…

A random number generator

Stream Cipher

- If we have an identical random number generator, we can use that for the One-time Pad?

- Can we create one?

RC4/RC5

- A stream of random number generator based on the selected key
  - https://en.wikipedia.org/wiki/RC4
  - https://en.wikipedia.org/wiki/RC5

Physically meet and share
40~2048 bits of keys
(max 512 bytes, short!)

RC4 or
RC5
Algorithm

RC4 or
RC5
Algorithm

RC4/RC5

- A stream of random number generator based on the selected key

RC4 or
RC5
Algorithm

Key: 0xd7fe6798a7c6a9859b289ce

RC4 or
RC5
Algorithm

Key: 0xd7fe6798a7c6a9859b289ce

RC4/RC5

- A stream of random number generator based on the selected key

RC4 or RC5 Algorithm

1: 0x1b395a46
2: 0xf1737202
3: 0xccf0de05
4: 0x908b0feb
5: 0x9d4c9add
…

Key: 0xd7fe6798a7c6a9859b289ce

RC4 or RC5 Algorithm

1: 0x1b395a46
2: 0xf1737202
3: 0xccf0de05
4: 0x908b0feb
5: 0x9d4c9add
…

Key: 0xd7fe6798a7c6a9859b289ce

RC4/RC5

- A stream of random number generator based on the selected key

Encrypt message 1 with 0x1b395a46
Encrypt message 2 with 0xf1737202
Encrypt message 3 with 0xccf0de05…

RC4 or RC5 Algorithm

Key: 0xd7fe6798a7c6a9859b289ce

1: 0x1b395a46
2: 0xf1737202
3: 0xccf0de05
4: 0x908b0feb
5: 0x9d4c9add
…

We can securely encrypt messages with One-time Pad (e.g., XOR Cipher)

Decrypt message 1 with 0x1b395a46
Decrypt message 2 with 0xf1737202
Decrypt message 3 with 0xccf0de05…

1: 0x1b395a46
2: 0xf1737202
3: 0xccf0de05
4: 0x908b0feb
5: 0x9d4c9add
…

RC4 or RC5 Algorithm

Key: 0xd7fe6798a7c6a9859b289ce

Insecure RC4/RC5

- Read the Wikipedia articles for their weaknesses
  - https://en.wikipedia.org/wiki/RC4
  - https://en.wikipedia.org/wiki/RC5
- RC4 RC5

As of 2015, there is speculation that some state cryptologic agencies may possess the capability to break RC4 when used in the TLS protocol.[6] IETF has published RFC 7465 to prohibit the use of RC4 in TLS;[3] Mozilla and Microsoft have issued similar recommendations.[7][8]

Best public cryptanalysis

12-round RC5 (with 64-bit blocks) is susceptible to a differential attack using 2^{44} chosen plaintexts.[1]

- Have no mathematical proof or
- Have an attack against these…

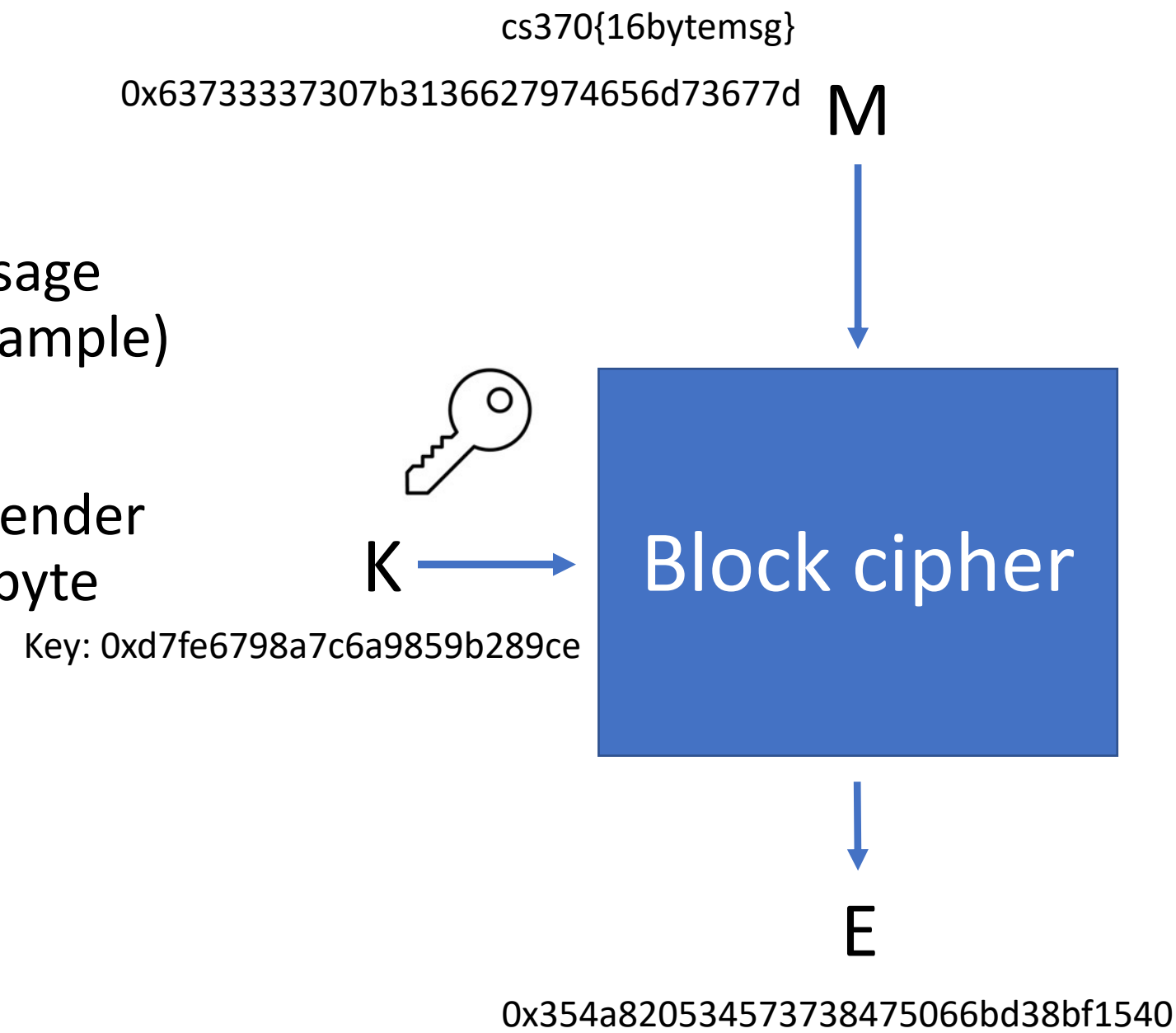
Block Cipher

- Cryptographers built a restricted encryption model to only encrypt
  - Fixed size message (a block, e.g., 16-byte (128-bit) or 32-byte (256-bit))

- Why having such a restriction?
  - Restricting the size simplifies modeling the encryption
  - They may provide a proof that attackers break this with a very low chance
    - E.g., breaking once in 2^{126} cases, requiring 85070591730234615865843651857942052864 trials
  - Breaking here means that distinguishing the part ciphertext is
  - not a random number at all

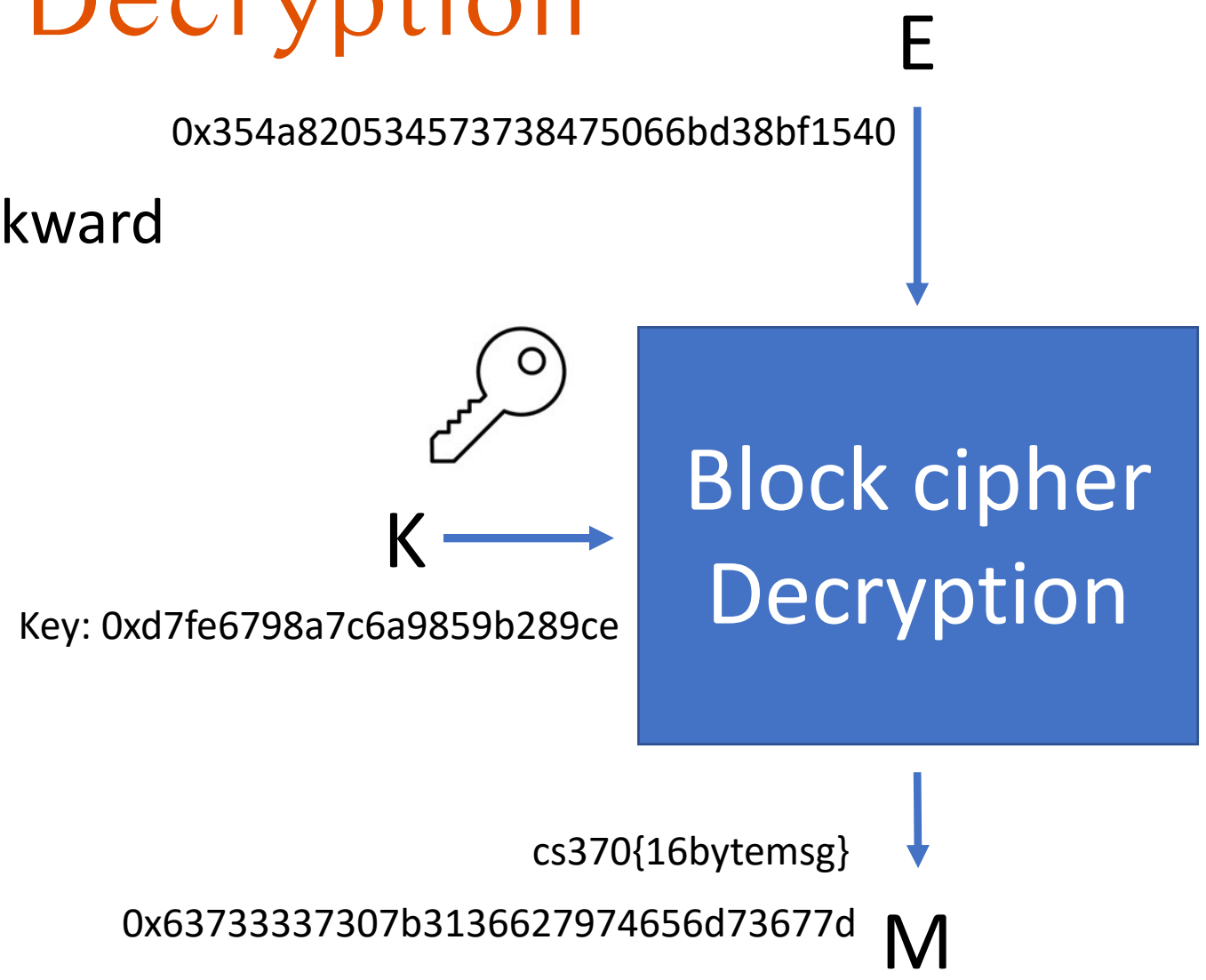
Block Cipher

- Block: a fixed sized message (16-byte here for the example)
- Key: A secret between sender and receiver (short, 16-byte here for the example)
- M: Plaintext message
- E : Encrypted message



Block Cipher Decryption

- Decryption works in backward
- The inverse function of encryption
- M: Plaintext message
- E : Encrypted message



What Does the Block Cipher Do?

- Generating a permutation of the numbers in
  - $\{0,1\}^n$ -> $\{0,1\}^n$
  - It's permutation, so it must be one-to-one mapping

- It's like shuffling the card

msg	Ciphertext
0	0xaf531b0e1
1	0x14a986e7a
2	0xad738009d
3	0x5ed6985c5
4	0xf3b8aa2e8
5	0xad04ec00e
…	0x59fd94c21

What Does the Block Cipher Do?

- The key (e.g., 16-byte or 32-byte) determines how to shuffle the numbers

K1

msg	Ciphertext
0	0xaf531b0e1
1	0x14a986e7a
2	0xad738009d
3	0x5ed6985c5
4	0xf3b8aa2e8
5	0xad04ec00e
…	0x59fd94c21

K2

msg	Ciphertext
0	0x89075eff9
1	0xdb441917
2	0xe18452e0
3	0x5d4566ecc
4	0x120a1ce0b
5	0x14e09f4c3
…	0x38057e94

K3

msg	Ciphertext
0	0x3ee82813
1	0xbc28e1c1
2	0x700dba64
3	0xfe9586d9
4	0xe3b0fa46
5	0x41c5f70f
…	0x845c6d67

K4

msg	Ciphertext
0	0x87372de1
1	0x19f1a578
2	0x9449fe68
3	0xa0e82ffe
4	0xba9f4d4b
5	0x5156b9ba
…	0x61e0bb0d

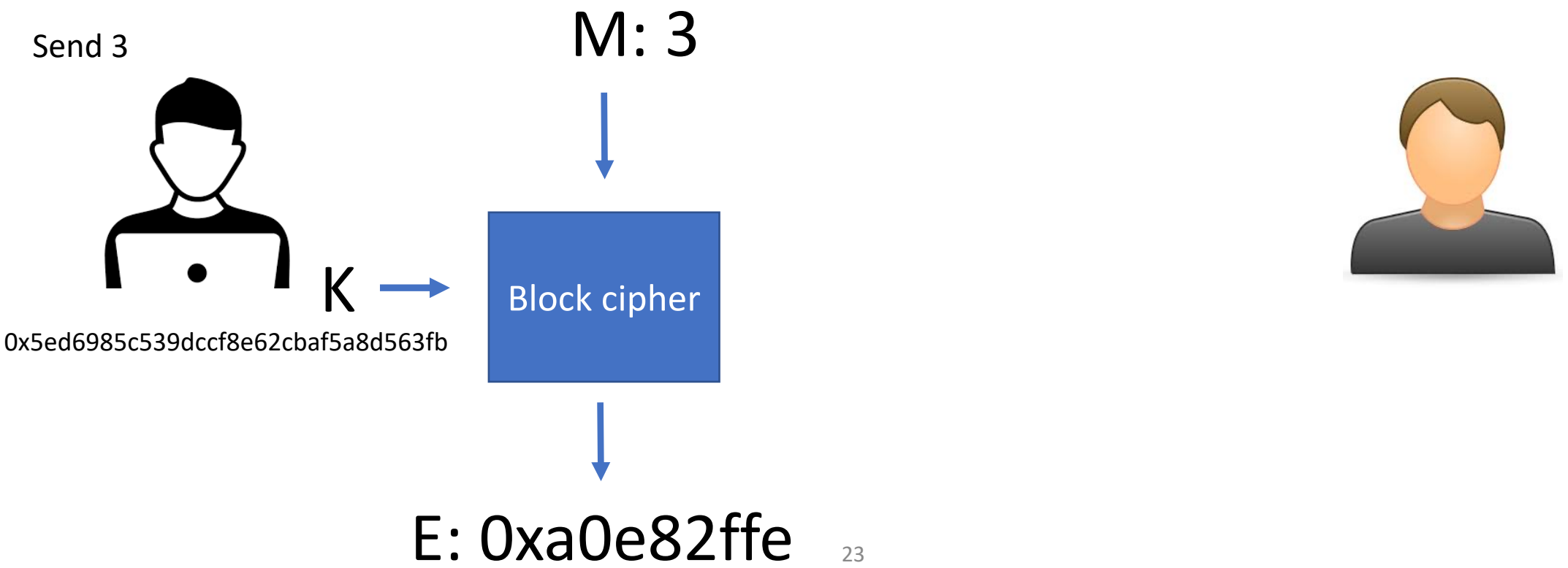
How Can We Use a Block Cipher

- Suppose you need to communicate with others securely/privately

Meet and exchange the 16-byte key
0x5ed6985c539dccf8e62cbaf5a8d563fb

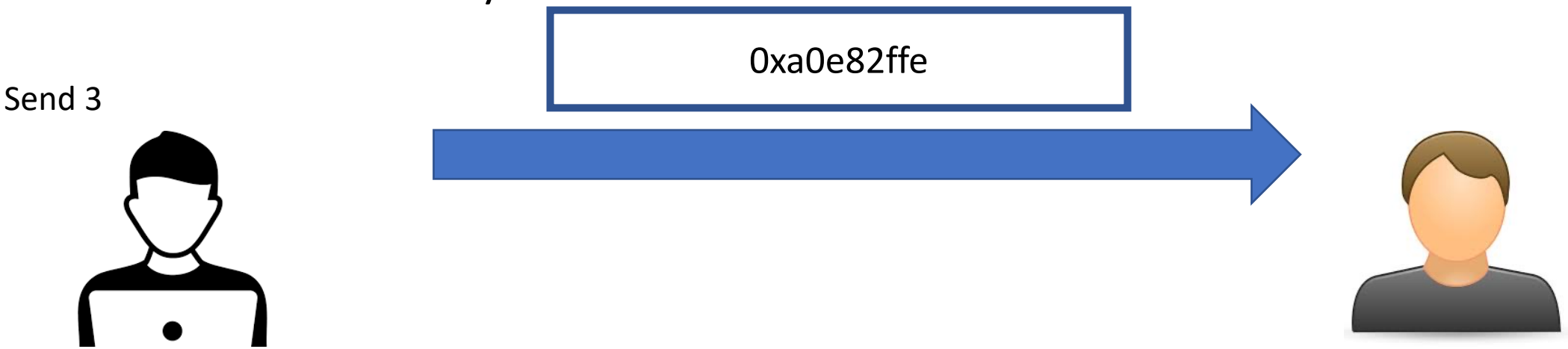
How Can We Use a Block Cipher

- Each will use the same block cipher algorithm
  - With the same key K



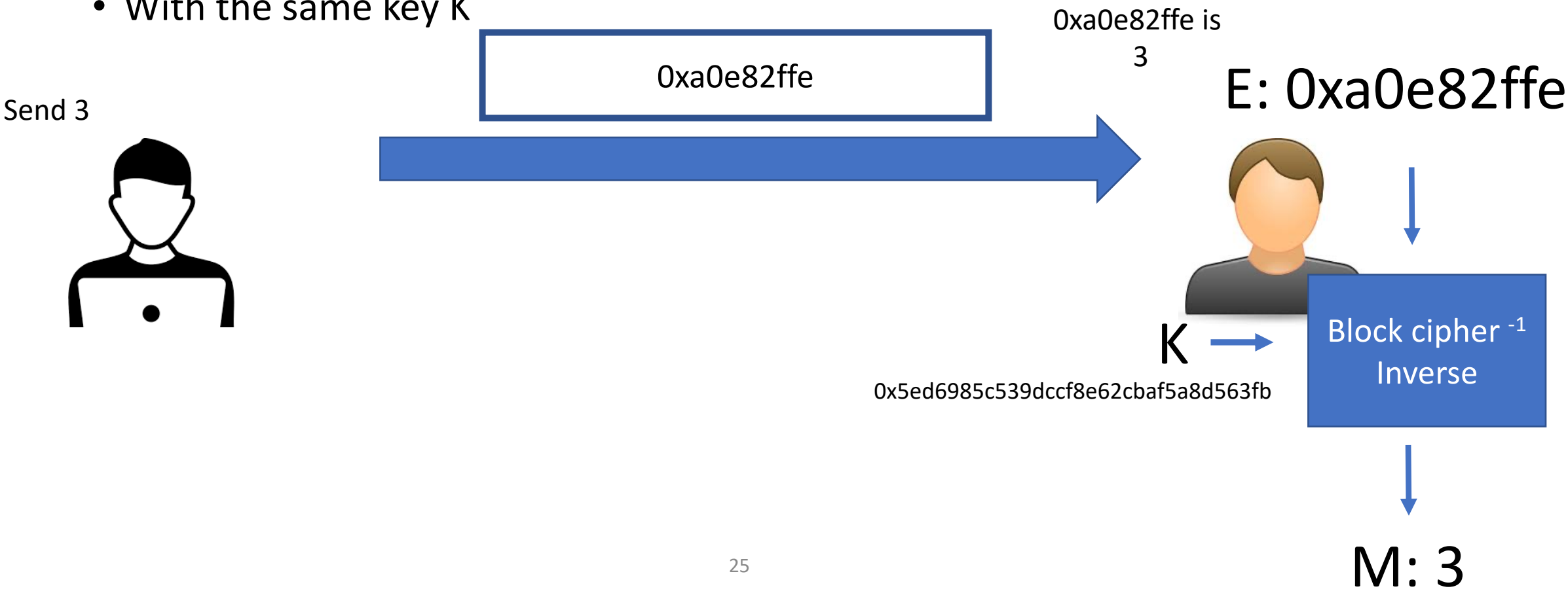
How Can We Use a Block Cipher

- Each will use the same block cipher algorithm
  - With the same key K



How Can We Use a Block Cipher

- Each will use the same block cipher algorithm
  - With the same key K

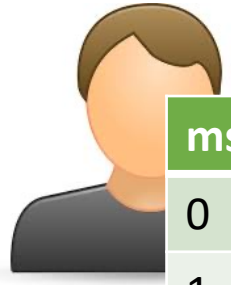


How Can We Use a Block Cipher

- Each will use the same block cipher algorithm
  - With the same key K

Send 3

msg	Ciphertext
0	0x87372de1
1	0x19f1a578
2	0x9449fe68
3	0xa0e82ffe
4	0xba9f4d4b
5	0x5156b9ba
…	0x61e0bb0d



msg	Ciphertext
0	0x87372de1
1	0x19f1a578
2	0x9449fe68
3	0xa0e82ffe
4	0xba9f4d4b
5	0x5156b9ba
…	0x61e0bb0d

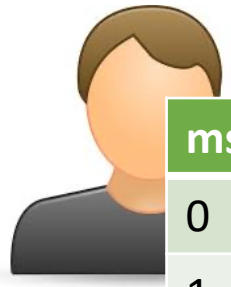
How Can We Use a Block Cipher

- Each will use the same block cipher algorithm
  - With the same key K

Send 3

0xa0e82ffe

0xa0e82ffe is 3

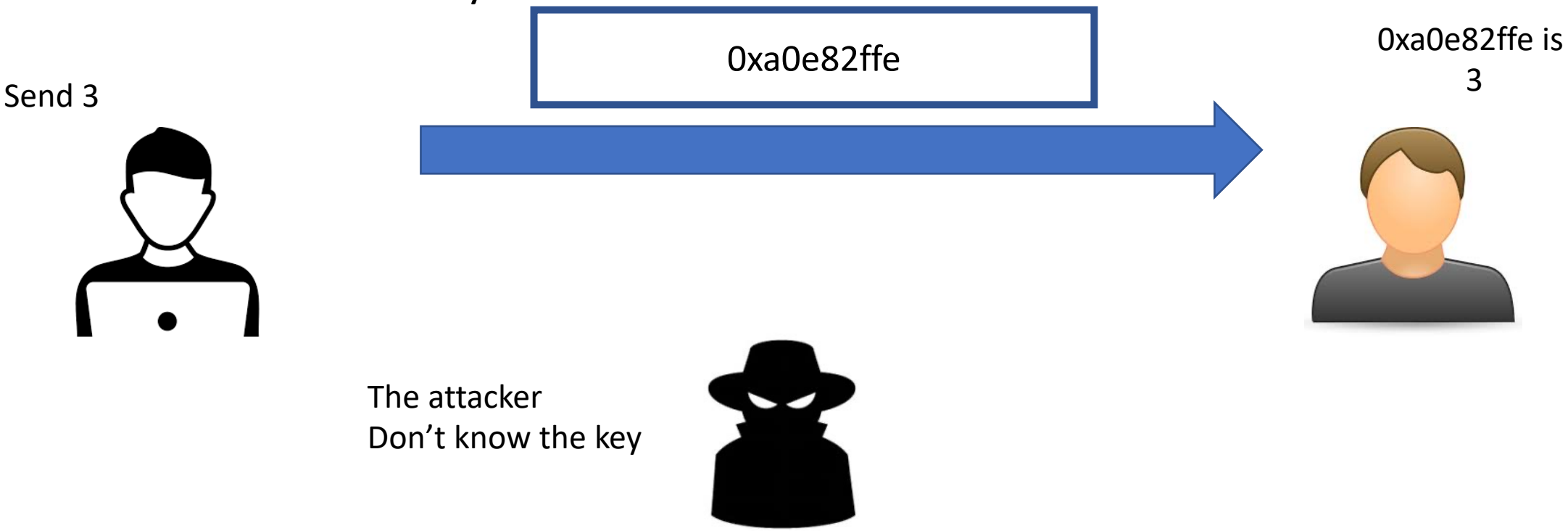


msg	Ciphertext
0	0x87372de1
1	0x19f1a578
2	0x9449fe68
3	0xa0e82ffe
4	0xba9f4d4b
5	0x5156b9ba
…	0x61e0bb0d

msg	Ciphertext
0	0x87372de1
1	0x19f1a578
2	0x9449fe68
3	0xa0e82ffe
4	0xba9f4d4b
5	0x5156b9ba
…	0x61e0bb0d

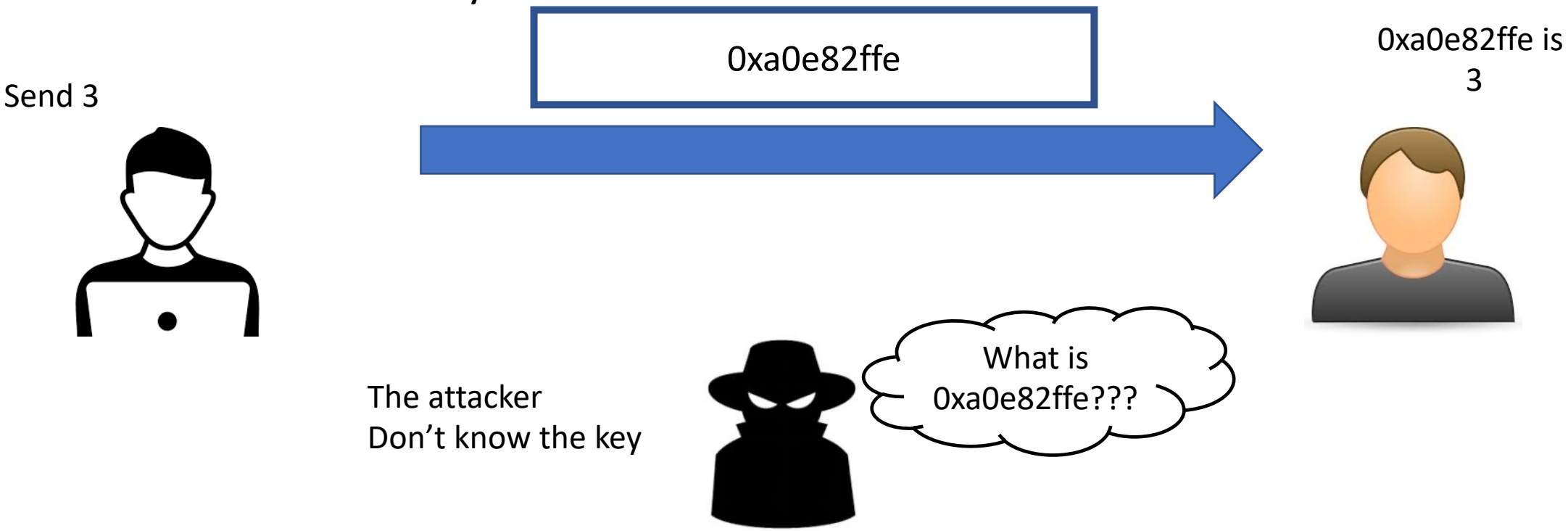
How Can We Use a Block Cipher

- Each will use the same block cipher algorithm
  - With the same key K



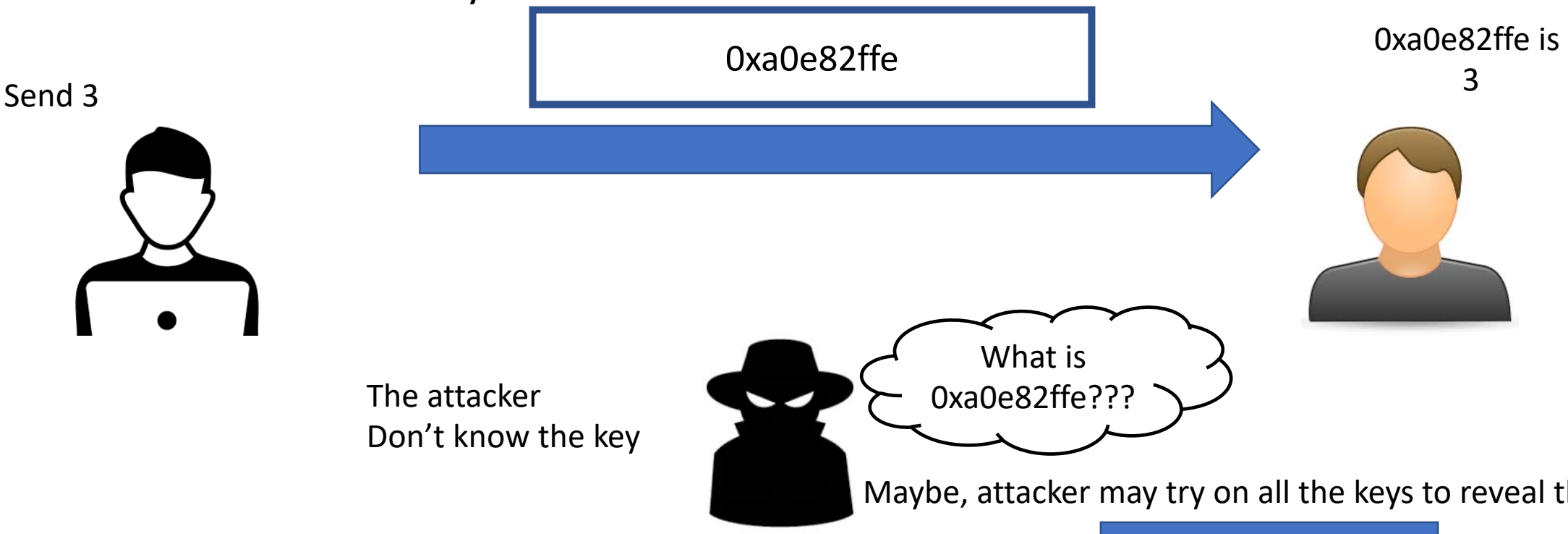
How Can We Use a Block Cipher

- Each will use the same block cipher algorithm
  - With the same key K



How Can We Use a Block Cipher

- Each will use the same block cipher algorithm
  - With the same key K



Pseudorandom Permutation (PRP)

- A function generates a permutation (based on a key K)
  - Whose sequence cannot be distinguished from a random permutation
- https://en.wikipedia.org/wiki/Pseudorandom_permutation

- Ideally Block Cipher must be a random permutation
  - We cannot have a true random permutation

- Pseudorandom permutation is not a true random permutation
  - Not perfectly secure

Pseudorandom Permutation (PRP)

- A well-designed PRP should have
  - Attacker's chance to distinguish the permutation (shuffling) from random is
  - EXTREMELY LOW
- Not ideal, but works in reality
  - Example (AES): https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
  - After computing $2^{126.1}$, it may be broken

Best public cryptanalysis

Attacks have been published that are computationally faster than a full brute-force attack, though none as of 2013 are computationally feasible.[1] For AES-128, the key can be recovered with a computational complexity of $2^{126.1}$ using the biclique attack.

How Difficult $2^{126.1}$ is?

- $2^{126.1}$
  - $9.1176402658 * 10^{37}$

- Suppose you can compute AES in 1 cycle (needs 20~30 cycles though)
- We have 6GHz CPU
- $9.1176402658 * 10^{37} / 6*10^{9}$
- Would take $1.5196 * 10^{28}$ seconds
- More than 481,860,000,000,000,000,000 years
  - With 1 billion computers? A supercomputer?
  - Would took 481,860,000,000 years

Summary

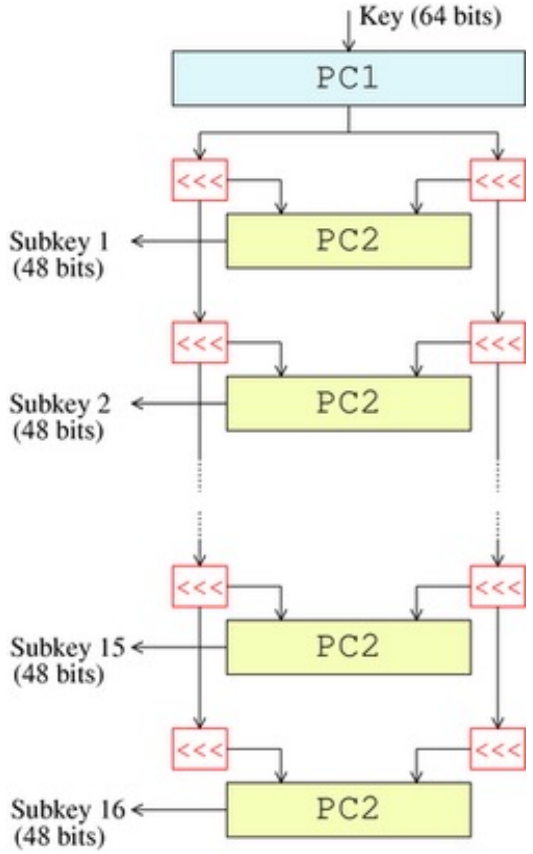
- We can share a short key and communicate if
  - We can generate many random numbers (or numbers looking like random)
  - It must be shared between two party (identical random number generator)

- Stream Cipher uses random number generator but
  - It's hard to provide mathematical proof -> attack can be possible

- Block Cipher has a restricted model but
  - Cryptographers can provide a proof that
  - A Block Cipher is PRP, and it is hard to break (e.g., requiring $2^{126.1}$ trials)

A History of the Block Cipher

- DES (Data Encryption Standard)
  - https://en.wikipedia.org/wiki/Data_Encryption_Standard



M (64 bits)

K
(56 bits)

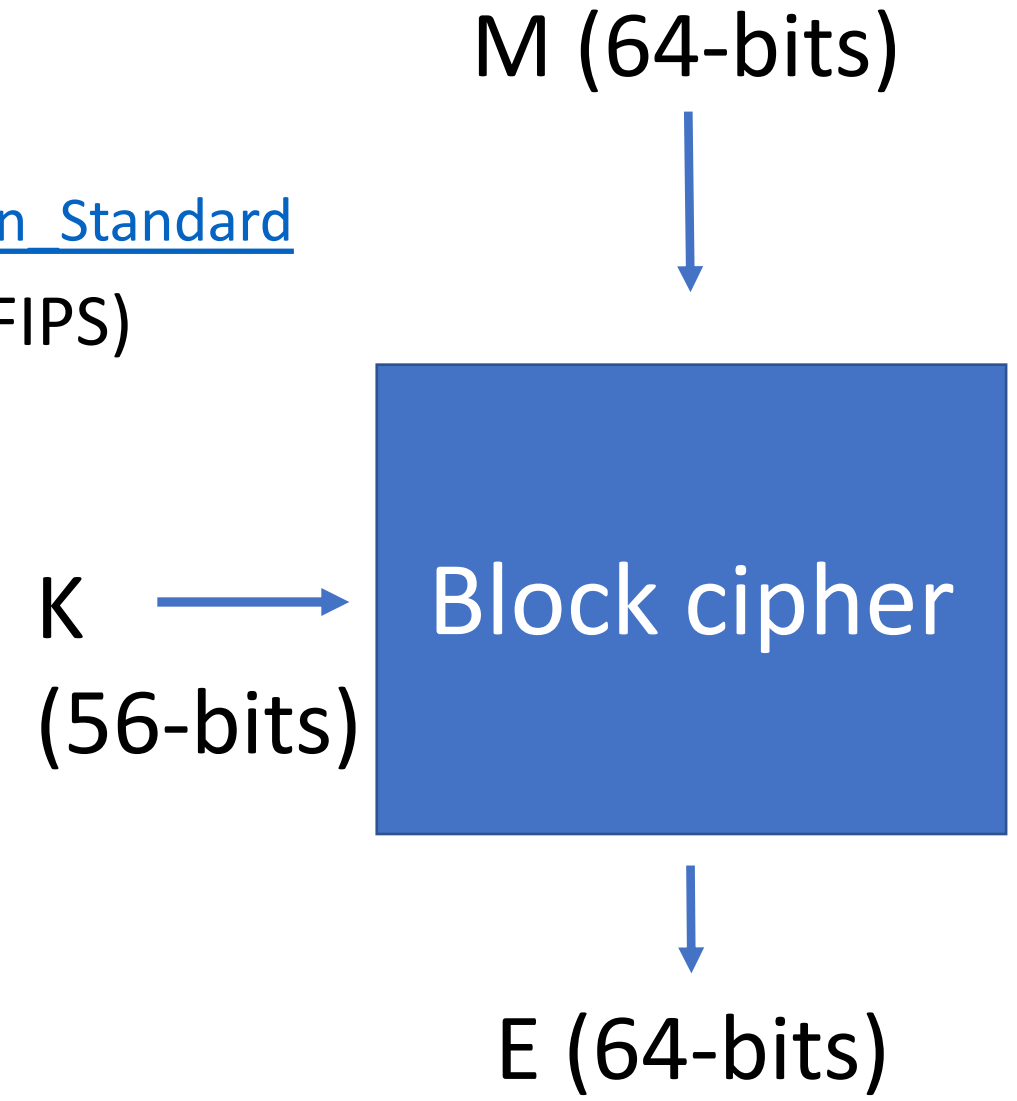
Block cipher

E (64 bits)

A History of the Block Cipher

- DES (Data Encryption Standard)
  - https://en.wikipedia.org/wiki/Data_Encryption_Standard
- Federal Information Processing Standard (FIPS)
  - Developed by IBM

- Keysize: 56-bits, 7-bytes (short)
- Block size: 64-bits (8-bytes)
- Ciphertext size: 64-bits (8-bytes)



Why the Key Size is 56-bits?

- 64-bits brute-forcing (trying all the keys to reveal the permutation)
- Was difficult in 1977 (the year the DES was standardized)
- Then why shorter key?

> DES is insecure due to the relatively short 56-bit key size. In January 1999, distributed.net and the Electronic Frontier Foundation collaborated to publicly break a DES key in 22 hours and 15 minutes (see chronology). There are also some analytical results which

- Broken in 1999 by EFF?!?!?!?!

Why the Key Size is 56-bits?

different."[8] The United States Senate Select Committee on Intelligence reviewed the NSA's actions to determine whether there had been any improper involvement. In the unclassified summary of their findings, published in 1978, the Committee wrote:

> In the development of DES, NSA convinced IBM that a reduced key size was sufficient; indirectly assisted in the development of the S-box structures; and certified that the final DES algorithm was, to the best of their knowledge, free from any statistical or mathematical weakness.[9]

However, it also found that

> NSA did not tamper with the design of the algorithm in any way. IBM invented and designed the algorithm, made all pertinent decisions regarding it, and concurred that the agreed upon key size was more than adequate for all commercial applications for which the DES was intended.[10]

Another member of the DES team, Walter Tuchman, stated "We developed the DES algorithm entirely within IBM using IBMers. The NSA did not dictate a single wire!"[11] In contrast, a declassified NSA book on cryptologic history states:

> In 1973 NBS solicited private industry for a data encryption standard (DES). The first offerings were disappointing, so NSA began working on its own algorithm. Then Howard Rosenblum, deputy director for research and engineering, discovered that Walter Tuchman of IBM was working on a modification to Lucifer for general use. NSA gave Tuchman a clearance and brought him in to work jointly with the Agency on his Lucifer modification."[12]

and

> NSA worked closely with IBM to strengthen the algorithm against all except brute-force attacks and to strengthen substitution tables, called S-boxes. Conversely, NSA tried to convince IBM to reduce the length of the key from 64 to 48 bits. Ultimately they compromised on a 56-bit key.[13][14]

Implications

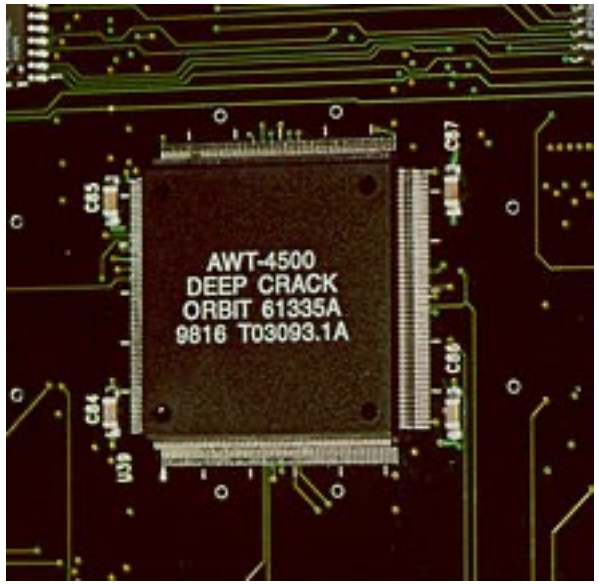
- The key size determines a block cipher's strength
  - If the algorithm is designed well and proved as a
  - Pseudorandom Permutation (PRP)

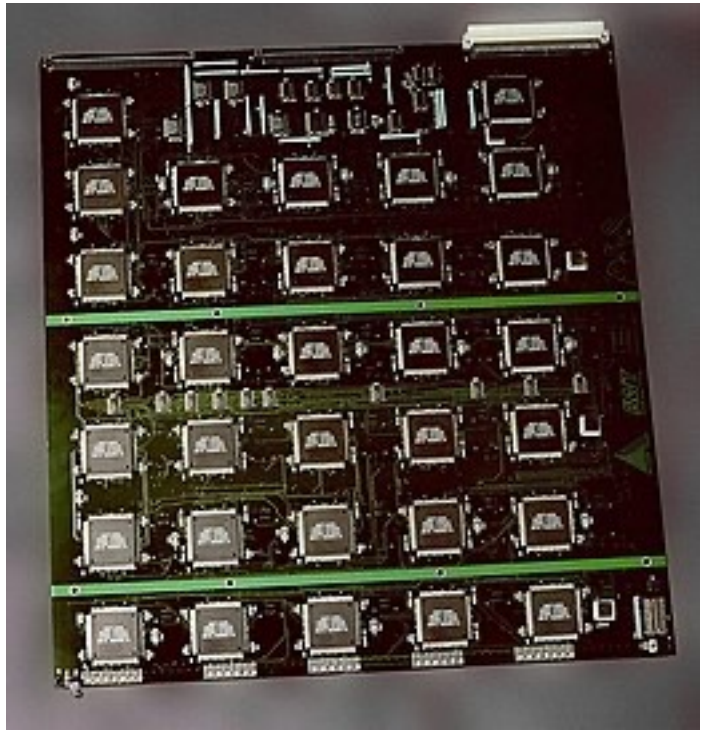
- Using a smaller number of bits in key would make the cipher
  - Be more easily broken by brute-force attacker

- What is a brute-force attack against the Block Cipher?
  - Trying all the keys to decrypt the message
  - (Trying all possible permutations…)

Advanced Encryption Standards (AES)

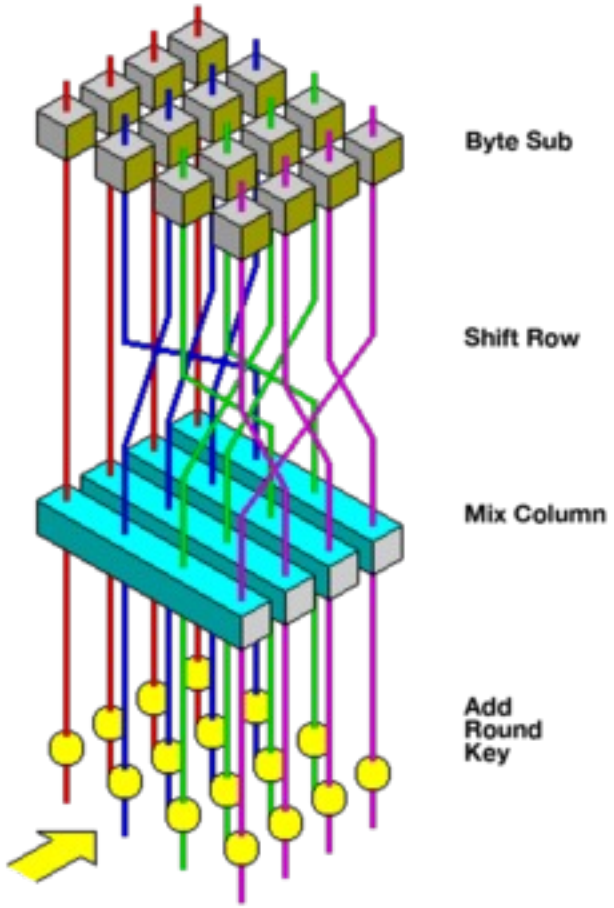
- Developed to replace weak DES
  - EFF break DES in 1999
  - https://en.wikipedia.org/wiki/EFF_DES_cracker
  - Deep Crack chip





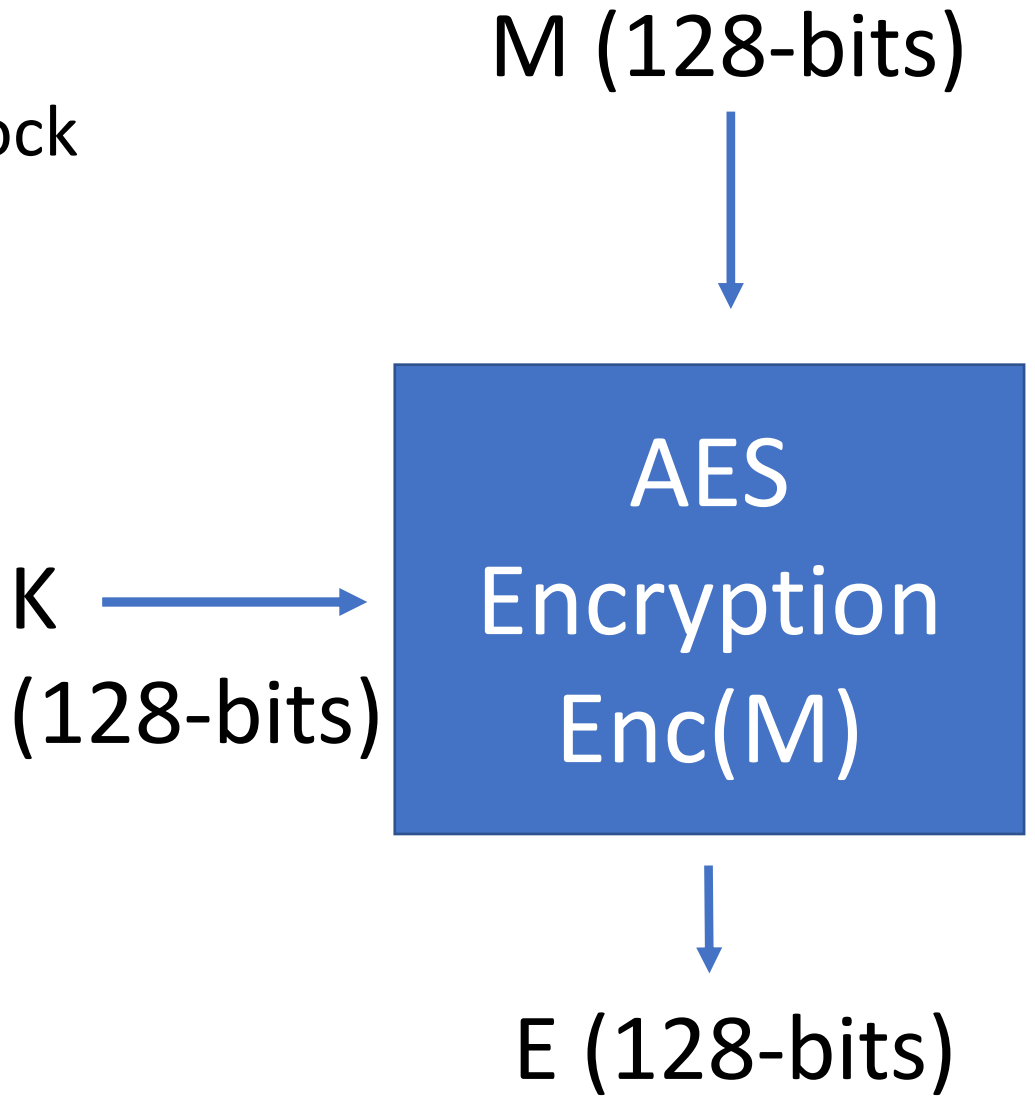
AES

- Key size
  - 128-bits, 192-bits, 256-bits
  - We can choose any of bit-length of the key among those three
- Block size
  - 16-byte
  - Will have more rounds for longer keys
- Ciphertext size
  - The same as the block size



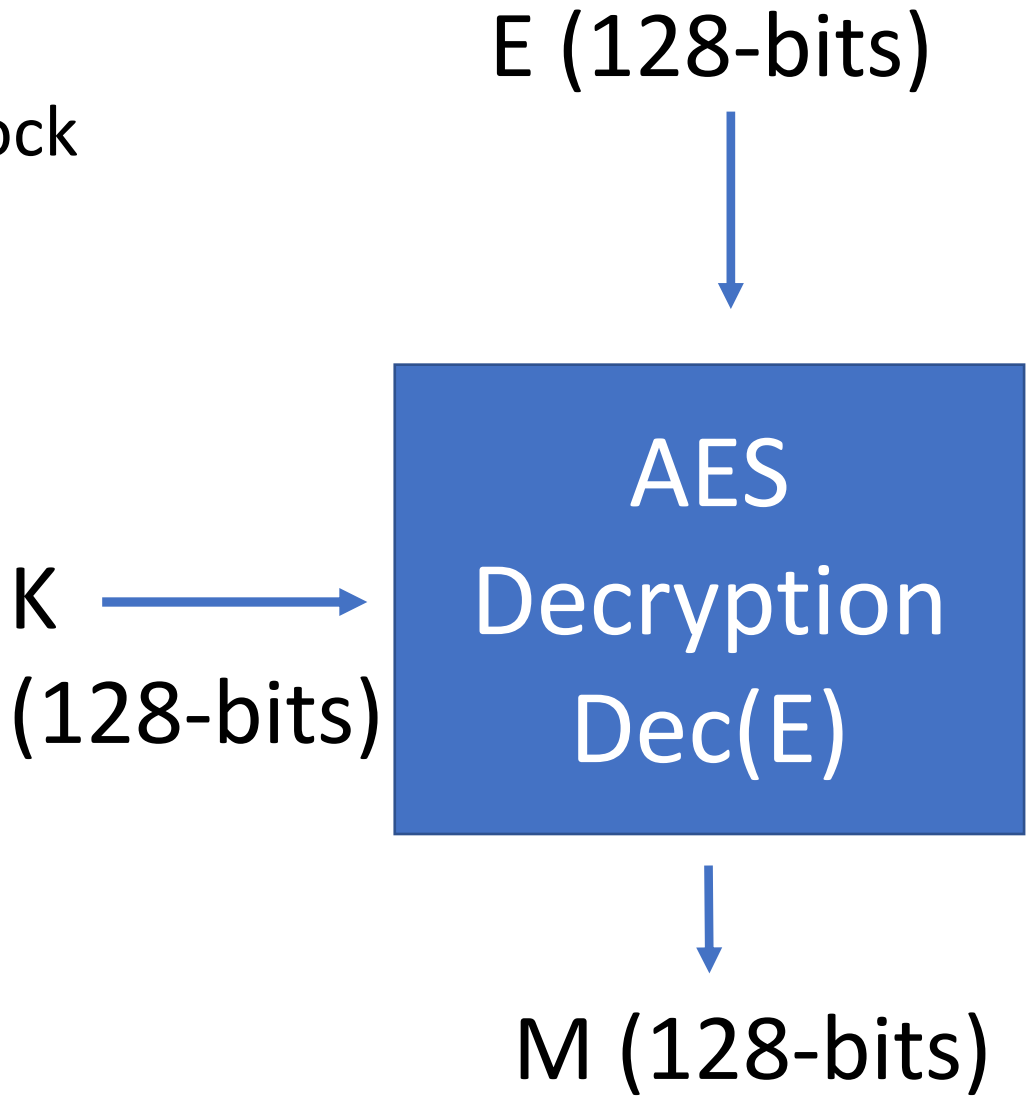
How Can We Use AES (Encryption)?

- Suppose we need to encrypt a 16-byte block
- Then we can get Enc(M) = E
  - E is also in 16-byte



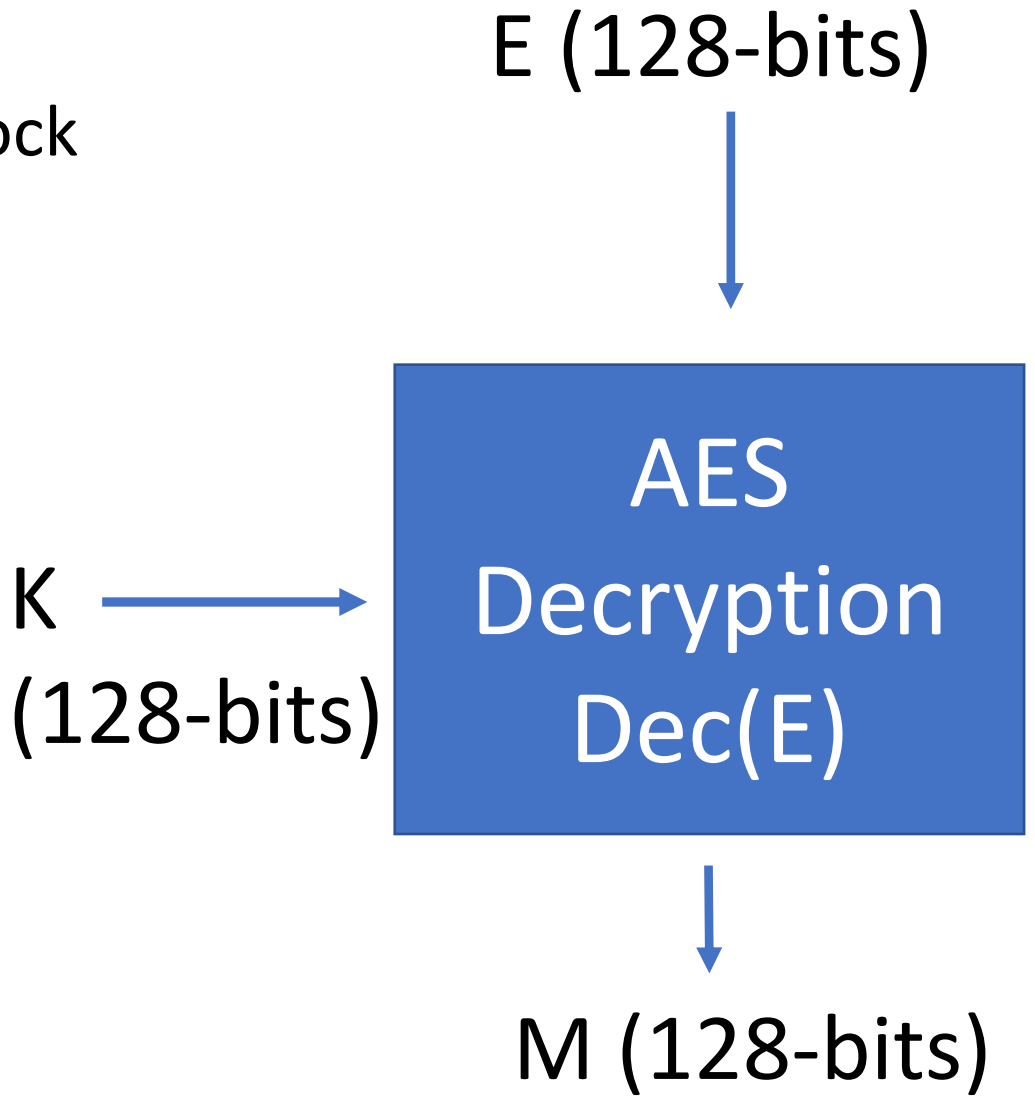
How Can We Use AES (Decryption)?

- Suppose we need to decrypt a 16-byte block
- Then we can get Dec(E) = M
  - M is also in 16-byte



How Can We Use AES (Decryption)?

- Suppose we need to decrypt a 16-byte block
- Then we can get Dec(E) = M
  - M is also in 16-byte



We sometimes write Dec() as $Enc^{-1}()$ because The decryption operation is actually an inverse operation of the encryption

Can We Encrypt Blocks That Its Size is Less Than 16 byte?

- YES
  - We can ignore the rest of bits
- How?
  - Via padding (add some meaningless but identifiable data)

- ECB (Electronic Code Book)
  - A padding scheme to indicate the length of the message
  - Pad the length of the padding as byte for the length of the padding…

How ECB Works?

- Suppose you want to encrypt 15 byte of data
  - “0123456789ABCDE” ← a 15-byte string

- We need 1 byte padding to make it to be 16-byte block

pos	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	

How ECB Works?

- Suppose you want to encrypt 15 byte of data
  - “0123456789ABCDE” ← a 15-byte string
  - ‘0’ ~ ‘9’ are 0x31 ~ 0x39, ’A’ ~ ‘E’ are 0x41 ~ 0x45

- We need 1 byte padding to make it to be 16-byte block

pos	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
val	0x30	0x31	0x32	0x33	0x34	0x35	0x36	0x37	0x38	0x39	0x41	0x42	0x43	0x44	0x45	0x01

- Make the padding value as the length of the padding
  - 0x1

How ECB Works?

- Suppose you want to encrypt 14 byte of data
  - “0123456789ABCD” ← a 14-byte string
  - ‘0’ ~ ‘9’ are 0x31 ~ 0x39, ’A’ ~ ‘E’ are 0x41 ~ 0x45

- We need 2 bytes padding to make it to be 16-byte block

pos	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
val	0x30	0x31	0x32	0x33	0x34	0x35	0x36	0x37	0x38	0x39	0x41	0x42	0x43	0x44	0x02	0x02

- Make the padding value as the length of the padding
  - 0x2 * 2

How ECB Works?

- Suppose you want to encrypt 13 byte of data
  - “0123456789ABC” ← a 13-byte string
  - ‘0’ ~ ‘9’ are 0x31 ~ 0x39, ’A’ ~ ‘E’ are 0x41 ~ 0x45

- We need 3 bytes padding to make it to be 16-byte block

pos	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
val	0x30	0x31	0x32	0x33	0x34	0x35	0x36	0x37	0x38	0x39	0x41	0x42	0x43	0x03	0x03	0x03

- Make the padding value as the length of the padding
  - 0x3 * 3

How ECB Works?

- Suppose you want to encrypt 1 byte of data
  - “0” ← a 1-byte string
  - ‘0’ ~ ‘9’ are 0x31 ~ 0x39, ’A’ ~ ‘E’ are 0x41 ~ 0x45

- We need 15 bytes padding to make it to be 16-byte block

pos	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
val	0x30	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f	0x0f

- Make the padding value as the length of the padding
  - 0xf * 15

How Can We Encrypt 16-byte?

- ”0123456789ABCDE” ← a 15-byte string

pos	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
val	0x30	0x31	0x32	0x33	0x34	0x35	0x36	0x37	0x38	0x39	0x41	0x42	0x43	0x44	0x45	0x01

- What if we have a string that contains
  - “0123456789ABCDE\x01”?
  - Can we distinguish this from a 15-byte string encryption???

How Can We Encrypt 16-byte?

- ”0123456789ABCDE\x01” ← a 16-byte string
- A 16-byte string will have a 16-byte of padding
  - There must be a padding block, for bytes % 16 == 0, we have 16 bytes padding

pos	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
val	0x30	0x31	0x32	0x33	0x34	0x35	0x36	0x37	0x38	0x39	0x41	0x42	0x43	0x44	0x45	0x01

pos	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
val	0x10	0x10	0x10	0x10	0x10	0x10	0x10	0x10	0x10	0x10	0x10	0x10	0x10	0x10	0x10	0x10

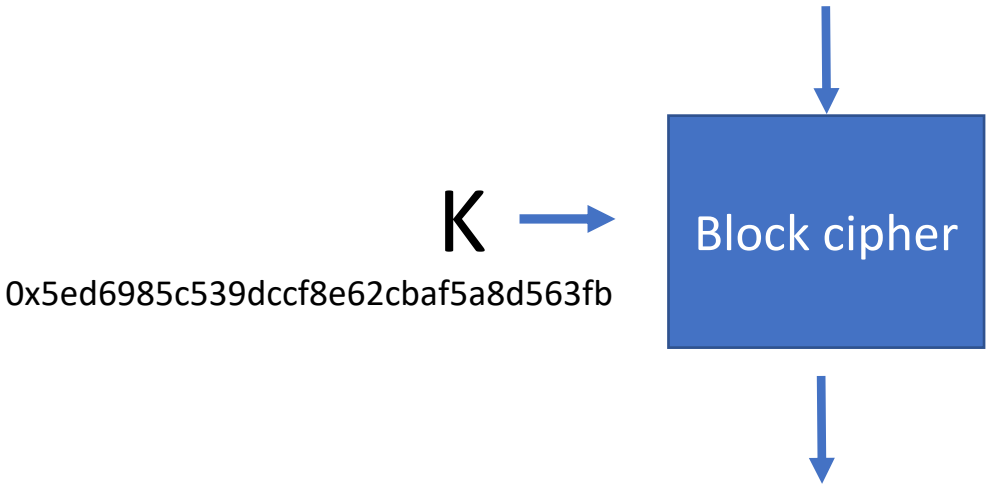
- We need to encrypt 2 blocks for a 16-byte

How to Encrypt 2 Blocks on Block Cipher?

- “0123456789ABCDE0x01” (16-byte) and “\x10\x10\x10…” (16-byte)

M: ”0123456789ABCDE\x01”

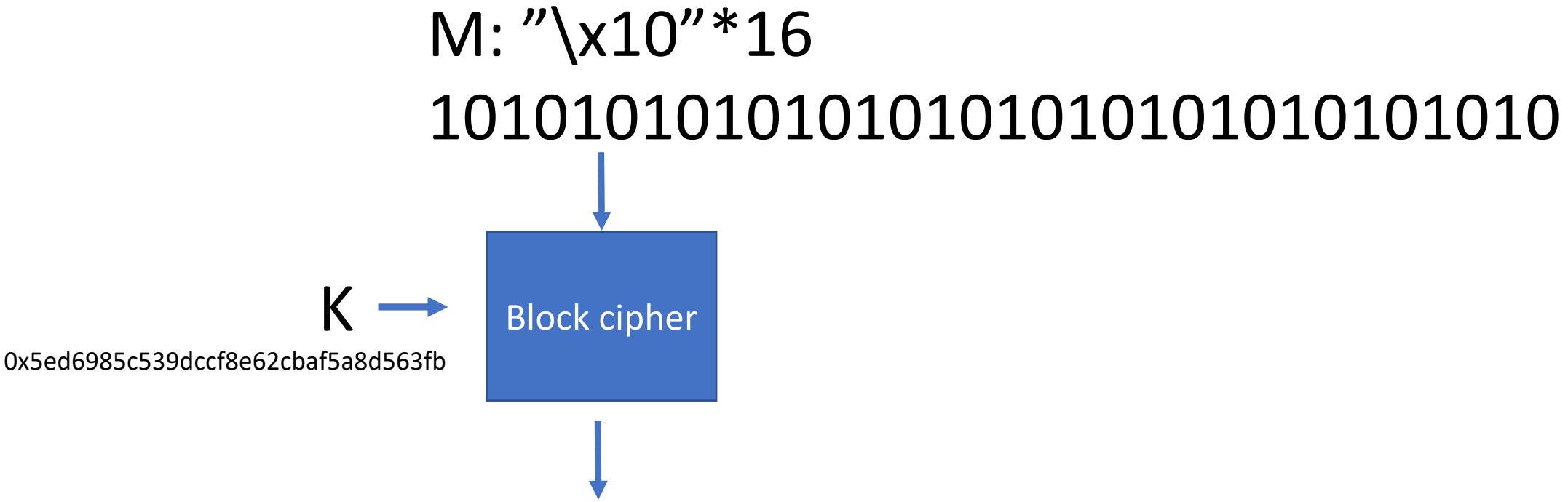
30313233343536373839414243444501



E: 23c934b65280ff27f7c2b7e131defcb6

How to Encrypt 2 Blocks on Block Cipher?

- “0123456789ABCDE0x01” (16-byte) and “\x10\x10\x10…” (16-byte)



How ECB Works?

- Can we encrypt more than 16 bytes?
- YES
  - “0123456789ABCDEFFEDCBA9876543210” ← this is a 32-byte block

- Break blocks into 16-byte granularity
  - “0123456789ABCDEF”
  - “FEDCBA9876543210”
  - Padding: “\x10” * 16
  - Encrypt those 3 blocks

How ECB Works?

- Can we encrypt more than 16 bytes?
- YES
  - “0123456789ABCDEFFEDCBA9876543210Q” ← this is a 33-byte block

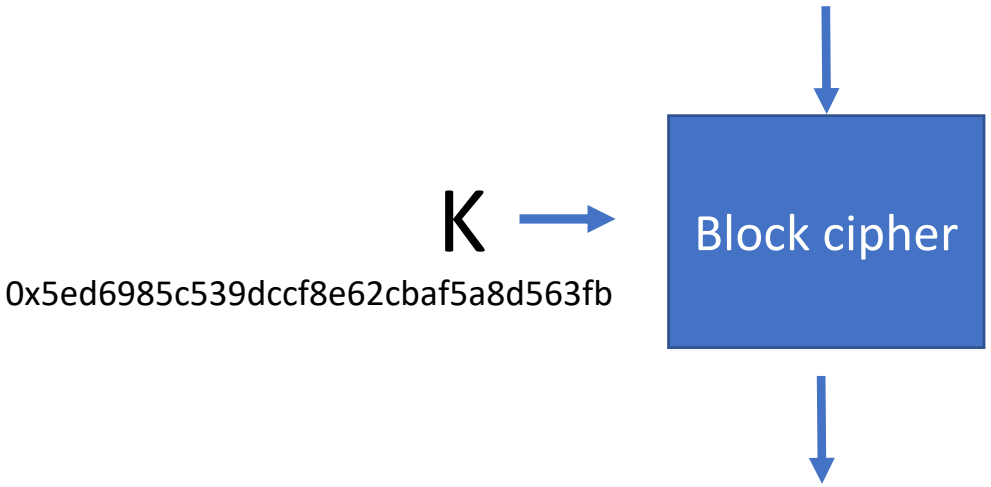
- Break blocks into 16-byte granularity
  - “0123456789ABCDEF”
  - “FEDCBA9876543210”
  - Last block and the padding: “Q” + “\x0f” * 15
  - Encrypt those 3 blocks

Are There Any Weaknesses in ECB?

- For the same key, the same plaintext block will result in the same ciphertext

M: ”\x10”*16

10101010101010101010101010101010



E: d303fe9c04a4876930e4a5728f1eda4c

Are There Any Weaknesses in ECB?

- For the same key, the same plaintext block will result in the same ciphertext

M: 10101010101010101010101010101010

Whenever an attacker observe d303fe9c04a4876930e4a5728f1eda4c
They know that it is the encryption of “\x10” * 16
(maybe the end of the message for 16-byte granularity)

E: d303fe9c04a4876930e4a5728f1eda4c

ECB Weakness

- Suppose a message 0 is encrypted to
  - 0x39827332…
- Suppose a message 1 is encrypted to
  - 0x5a83f874…
- Suppose the attacker knows this via
  - Pattern analysis or something

ECB Weakness

- Suppose a message 0 is encrypted to
  - 0x39827332…
- Suppose a message 1 is encrypted to
  - 0x5a83f874…
- Suppose the attacker knows this via
  - Pattern analysis or something

ECB Weakness

- Suppose a message 0 is encrypted to
  - 0x39827332…
- Suppose a message 1 is encrypted to
  - 0x5a83f874…
- Suppose the attacker knows this via
  - Pattern analysis or something

ECB Summary

- Electronic Code Book can be used for encrypting data less/more than the block size
  - We can add padding
- Padding Scheme
  - Padding size: 16 - Size % 16
  - ”Padding_size byte” * padding_size is the padding
  - E.g., if padding size is 1, then add \x01 at the end
  - 2, then \x02\x02
  - 3, then \x03\x03\x03
  - 15, then \x0f\x0f\x0f

ECB Summary

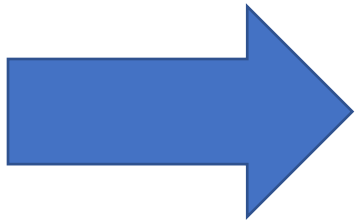
- For the size % 16 == 0, we need to have 16-byte padding
  - We definitely need to have a longer ciphertext than the plaintext size
  - At most 16-byte longer per each message

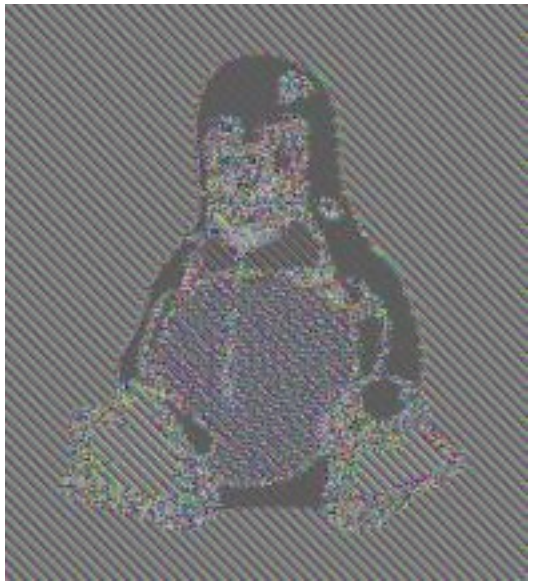
ECB Summary

- Weakness
  - Using the same key will leak the information about the plaintext data in ciphertext
  - This is because the permutation is not changed, so for the same plaintext, the block cipher will result in the same encrypted data

ECB Summary

- Weakness
  - In case we encrypt an image





ECB Exercise

- Please take a look at the ECB encrypted Image
- https://cs370.unexploitable.systems/_static/encrypted.bmp

ECB is Insecure

20

I have securely encrypted the BMP (bitmap) image file that contains the flag to this challenge.

I used AES-ECB-128 with a super secure random key, so I bet you cannot get the flag.

The image is at here:
https://cs370.unexploitable.systems/_static/encrypted.bmp

It starts with CS370{ but I do not want to let you know the rest.