

Using Cryptography to Protect Confidentiality

Thierry Sans

Overview

- **Symmetric Encryption**
 - Stream Cipher
 - Block Cipher (and block cipher modes)
- **Asymmetric Encryption**
- **Key Exchange**

Symmetric Encryption

Design principles (reminder)

1. **Kerkoff Principle**

The security of a cryptosystem must not rely on keeping the algorithm secret

2. **Diffusion**

Mixing-up symbols

3. **Confusion**

Replacing a symbol with another

4. **Randomization**

Repeated encryptions of the same text are different

The attacker's model

- **Exhaustive Search**

Try all possible n keys (in average it takes $n/2$ tries)

- **Ciphertext only**

You know one or several random ciphertexts

- **Known plaintext**

You know one or several pairs of random plaintext and their corresponding ciphertexts

- **Chosen plaintext**

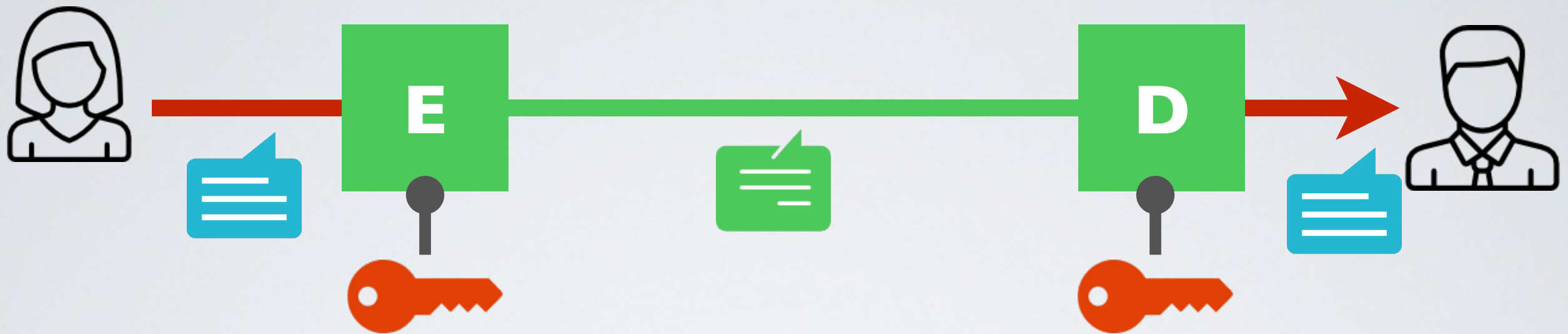
You know one or several pairs of chosen plaintext and their corresponding ciphertexts

- **Chosen ciphertext**

You know one or several pairs of plaintext and their corresponding chosen ciphertexts

➔ **A good crypto system resists all attacks**

Functional Requirements



➡ The same key k is used for encryption E and decryption D

1. $D_k(E_k(m))=m$ for every k , E_k is an injection with inverse D_k
2. $E_k(m)$ is easy to compute (either polynomial or linear)
3. $D_k(c)$ is easy to compute (either polynomial or linear)
4. $c = E_k(m)$ finding m is hard without k (exponential)

Two Families of Symmetric Encryption Schemes

Stream cipher

RC4 - Rivest Cipher 4 (now deprecated)

Salsa20 (and ChaCha20)

Block cipher

- Encryption standards

DES (and 3DES) - Data Encryption Standard (now deprecated)

AES - Advanced Encryption Standard

- Block cipher modes of operation

Symmetric Encryption

Stream Cipher

XOR Cipher (a.k.a Vernham Cipher)

a modern version of Vigenere

Use \oplus to combine the message and the key

$$E_k(m) = k \oplus m$$

$$D_k(c) = k \oplus c$$

$$D_k(E_k(m)) = k \oplus (k \oplus m) = m$$

Problem : known-plaintext attack

$$\text{so } k = (k \oplus m) \oplus m$$

$x \oplus x = 0$
$x \oplus 0 = x$

Mauborgne Cipher - a modern version of OTP

Use a random stream as encryption key

➔ Defeats the know-plaintext attack

Problem : Key-reused attack (a.k.a two-time pad)

$$C_1 = k \oplus m_1$$

$$C_2 = k \oplus m_2$$

$$\begin{aligned}\text{so } C_1 \oplus C_2 &= (k \oplus m_1) \oplus (k \oplus m_2) \\ &= (m_1 \oplus m_2) \oplus 0 \\ &= (m_1 \oplus m_2)\end{aligned}$$

$x \oplus x = 0$
$x \oplus 0 = x$

Random Number Generator

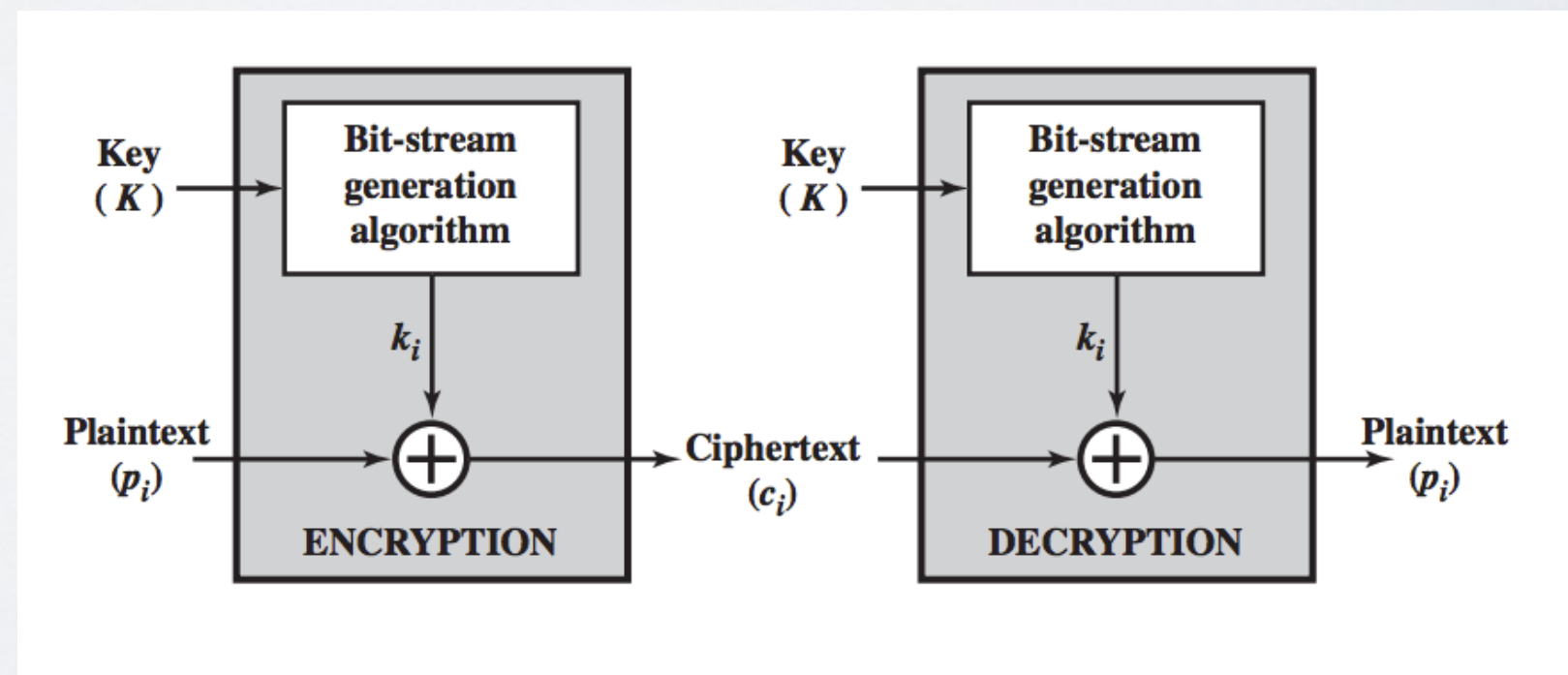
```
int getRandomNumber()  
{  
    return 4; // chosen by fair dice roll.  
             // guaranteed to be random.  
}
```

True Random Number Generator

➔ No, because we want to be able to encrypt and decrypt

Pseudo-Random Generator

➔ Stretch a fixed-size seed to obtain an unbounded random sequence



Stream cipher

Can we use k as a seed?

$$E_k(m) = m \oplus \text{RNG}(k)$$

➡ Be careful of key reused attack !

RC4 - Rivest Cipher 4

Key Size	40 - 2048 bits
Speed	~ 8 cycles / byte

Very simple implementation

Designed in 1987 ... but broken in 2015

[Home](#) / [Business Software](#)

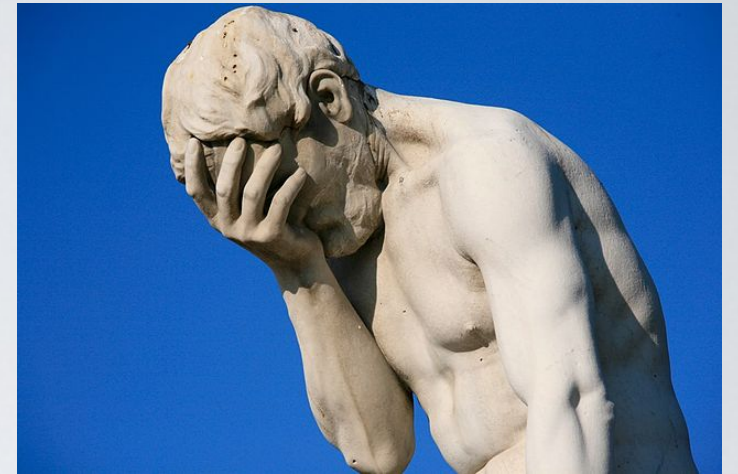
'Serious' Microsoft Office Encryption Flaw Uncovered

 [COMMENTS](#)

By [John E. Dunn](#), IDG News Service
Jan 27, 2005 4:00 PM

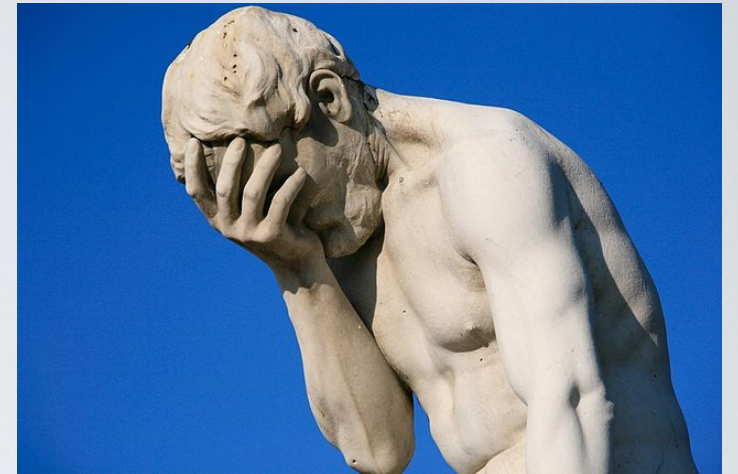
Cryptography expert Phil Zimmermann says he believes a flaw recently discovered in Microsoft Office's Word and Excel encryption is serious and warrants immediate attention.

"I think this is a serious flaw--it is highly exploitable. It is not a theoretical attack," says Zimmermann, referring to a flaw in Microsoft's use of RC4 document encryption unearthed recently by a researcher in Singapore.



MS Word and Excel 2003 used the same key to re-encrypt documents after editing changes

WEP - Wired Equivalent Privacy



- ➔ A random number IV (24 bits only) transmitted in clear between the clients and the base station

$$\text{RC4_key} = \text{IV} + \text{SSID_password}$$

- ⦿ 50% chance the same IV will be used again after 5000 packets

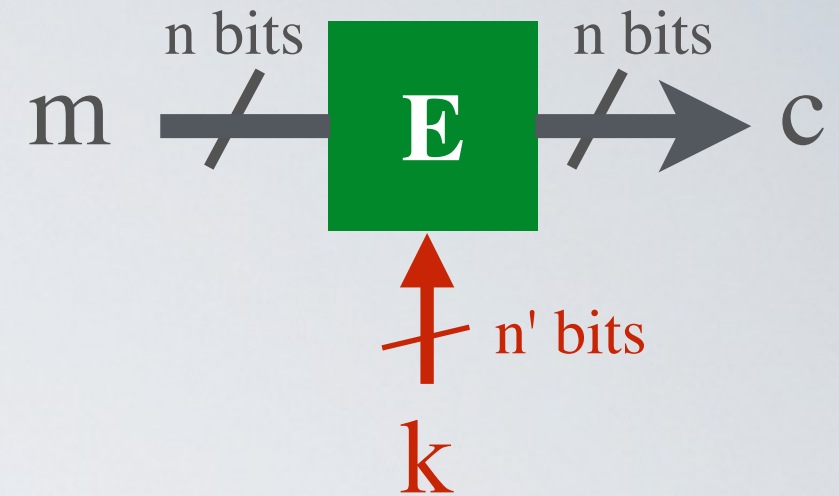
Salsa20 (and ChaCha20)

Key Size	128 or 256 bits
Speed	~ 4 cycles / byte

Symmetric Encryption

Block Cipher

Ideal block cipher



- Combines confusion (substitution) and diffusion (permutation)
 - Changing single bit in plaintext block or key results in changes to approximately half the ciphertext bits
- ➡ Completely obscure statistical properties of the original message
- ➡ A known-plaintext attack does not reveal the key

DES - Data Encryption Standard

Block size	64 bits
Key Size	56 bits
Speed	~ 50 cycles per byte
Algorithm	Feistel Network

Timeline

- **1972** NBS call for proposals
- **1974** IBM Lucifer proposal
analyzed by DOD and enhanced by NSA
- **1976** adopted as standard
- **2004** NIST withdraws the standard

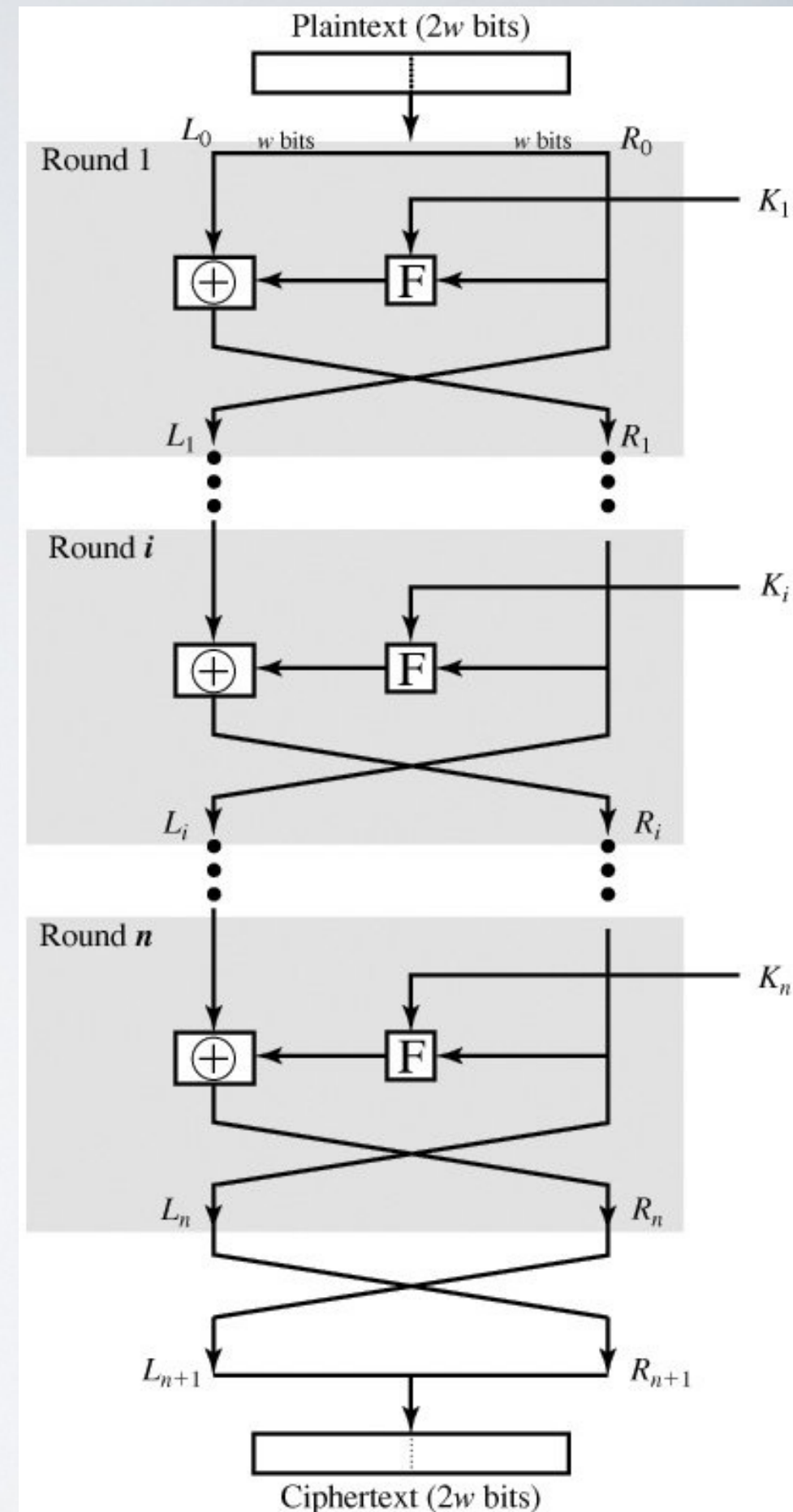
(FYI) Feistel Network

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F_i(R_{i-1}, k_i)$$

Properties:

- F is an arbitrary function that scrambles the input based on a key
 - F is not necessary invertible
 - A Feistel Network is invertible
- ➡ Achieves confusion and diffusion



Security of DES - DES Challenges (brute force contests)

1998 *Deep Crack*, the EFF's DES cracking machine used 1,856 custom chips

- Speed : matter of days
- Cost : \$250,000

2006 *COPACOBANA*, the Cost-optimized Parallel CodeBreaker used 120 FPGAs

- Speed : less than 24h
- Cost : \$10,000

How about 2DES ?

$$2DES_{k_1, k_2}(m) = E_{k_2}(E_{k_1}(m))$$

Meet-in-the-middle attack - known-plaintext attack

1. Brute force $E_{k_1}(m)$ and save results in a table called TE (2^{56} entries)
2. Brute force $D_{k_2}(c)$ and save results in a table called TD (2^{56} entries)
3. Match the two tables together to get the key candidates
 - ➡ The more plaintext you know, the lesser key candidates
 - ➡ Effective key-length (entropy) is **57 bits**
 - ➡ This attacks applies to every encryption algorithm used as such

3DES (Triple DES)

$$3DES_{k1,k2,k3}(m) = E_{k3}(D_{k2}(E_{k1}(m)))$$

- ➡ Effective key length (entropy) : 112 bits
- ✓ Very popular, used in PGP, TLS (SSL) ...
- ⦿ But terribly slow

AES - Advanced Encryption Standard

Timeline

- **1996** NIST issues public call for proposal
- **1998** 15 algorithms selected
- **2001** winner was announced

Rijndael by *J. Daemen and V. Rijmen*

Block size	128 bits
Key Size	128, 192, 256 bits
Speed	~18-20 cycles / byte
Mathematical Foundation	Galois Fields
Implementation	<ul style="list-style-type: none">• Basic operations : \oplus, $+$, shift• Small code : 98k

Adopted by the NIST in December 2001

(pure) Encryption Modes

a.k.a. how to encrypt long messages

ECB - Electronic Code Book

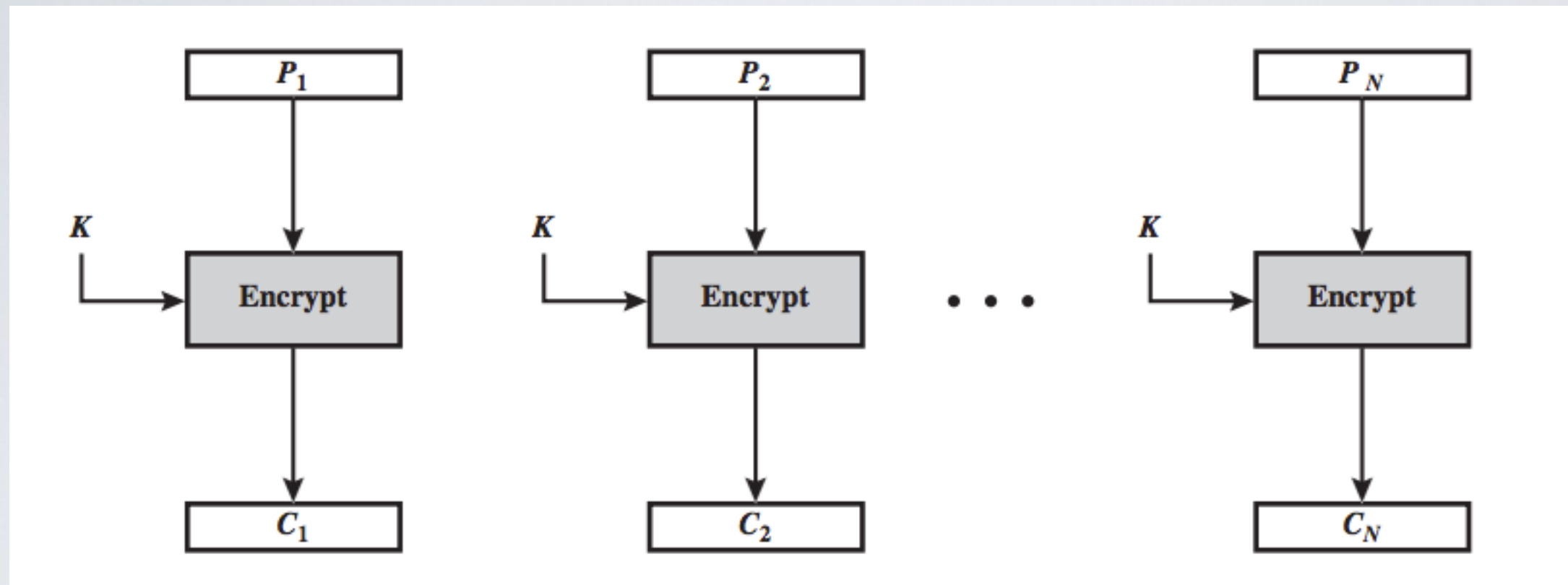
CBC - Cipher Block Chaining

CFB - Cipher Feedback

OFB - Output Feedback

CTR - Counter

ECB - Electronic Code Book (a.k.a Block Mode)



Each plaintext block is encrypted independently with the key

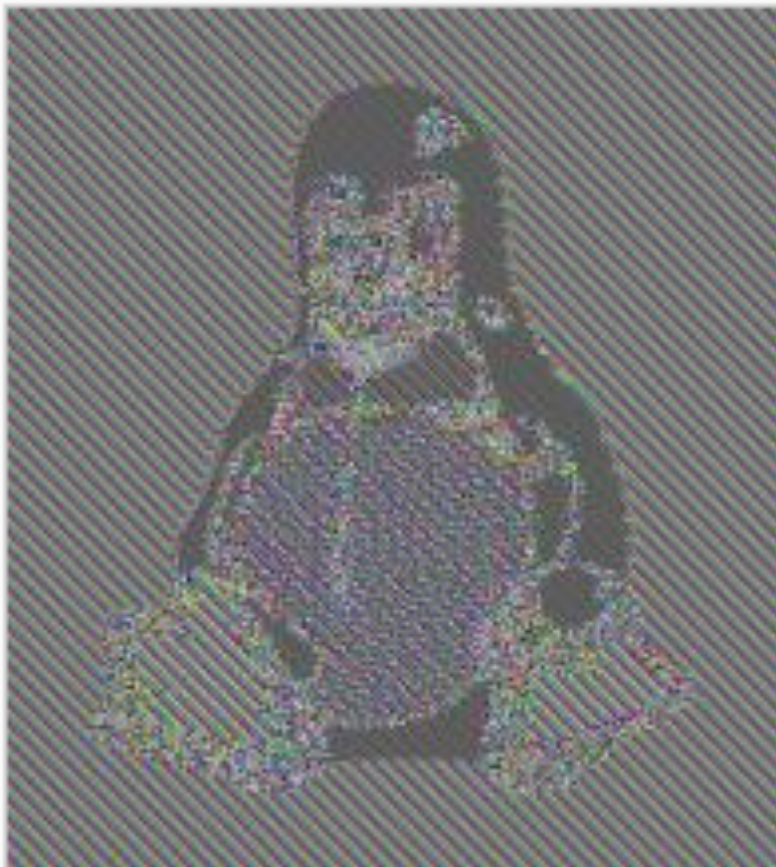
✓ Block can be encrypted in parallel

⦿ The same block is encrypted to the same ciphertext

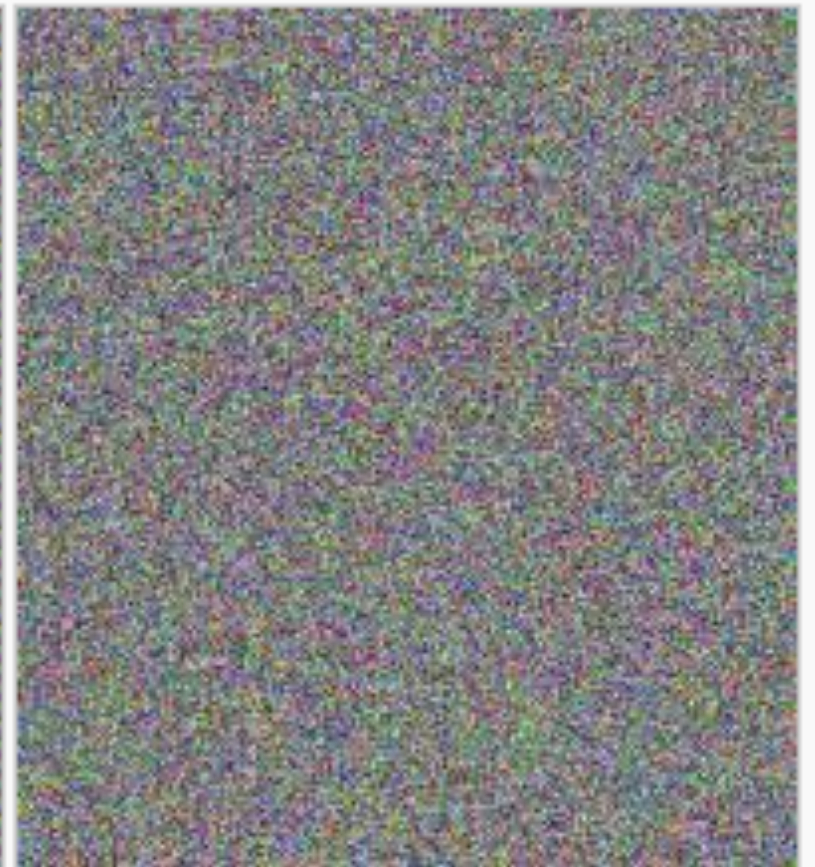
How bad is ECB mode with a large data?



Original image



Encrypted using ECB mode



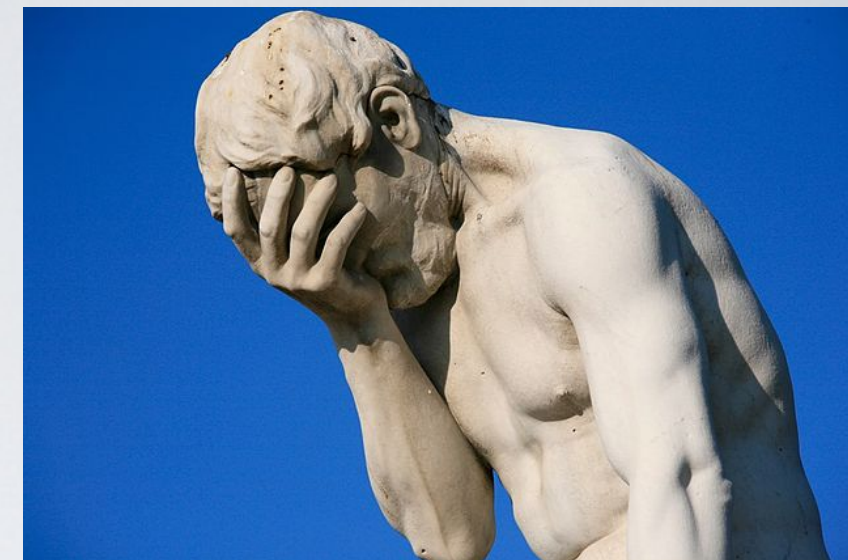
Modes other than ECB result in pseudo-randomness

source: *Wikimedia*

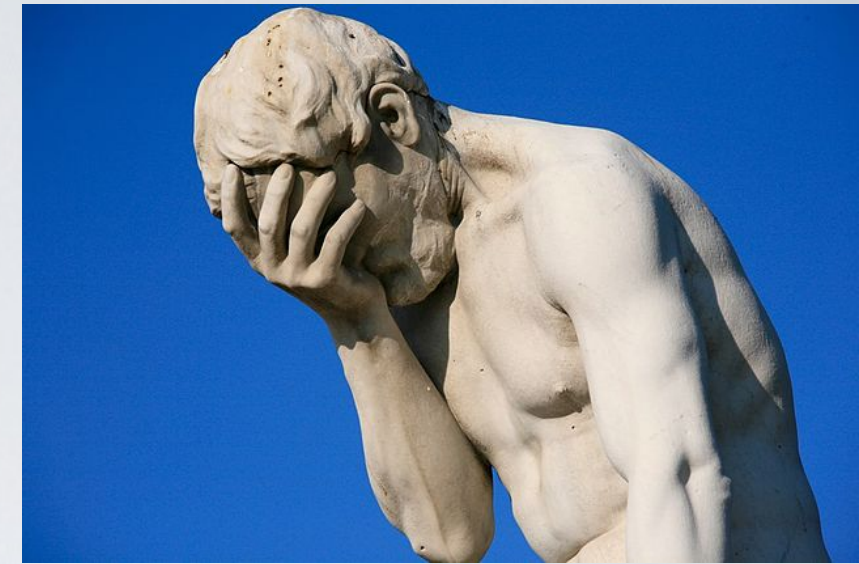
HACKERS RECENTLY LEAKED **153 MILLION** ADOBE USER EMAILS, ENCRYPTED PASSWORDS, AND PASSWORD HINTS. ADOBE ENCRYPTED THE PASSWORDS IMPROPERLY, MISUSING BLOCK-MODE 3DES. THE RESULT IS SOMETHING WONDERFUL:

USER	PASSWORD	HINT	
4e18acc1ab27a2d6		WEATHER VANE SWORD	<input type="text"/>
4e18acc1ab27a2d6			<input type="text"/>
4e18acc1ab27a2d6	a0a2876eb1ea1fca	NAME 1	<input type="text"/>
8babbb6299e06eb6d		DUH	
8babbb6299e06eb6d	a0a2876eb1ea1fca		<input type="text"/>
8babbb6299e06eb6d	85e9da81a8a78adc	57	
4e18acc1ab27a2d6		FAVORITE OF 12 APOSTLES	
1ab29ae86da6e5ca	7a2d6a0a2876eb1e	WITH YOUR OWN HAND YOU HAVE DONE ALL THIS	
a1f9b2b6299e7a2b	e0dec1e6ab797397	SEXY EARLOBES	<input type="text"/>
a1f9b2b6299e7a2b	617ab0277727ad85	BEST TOS EPISODE	<input type="text"/>
39738b7adb0b8af7	617ab0277727ad85	SUGARLAND	
1ab29ae86da6e5ca		NAME + JERSEY #	
877ab7889d3862b1		ALPHA	<input type="text"/>
877ab7889d3862b1			<input type="text"/>
877ab7889d3862b1			<input type="text"/>
877ab7889d3862b1		OBVIOUS	<input type="text"/>
877ab7889d3862b1		MICHAEL JACKSON	<input type="text"/>
38a7c9279codeb44	9dca1d79d4dec6d5		
38a7c9279codeb44	9dca1d79d4dec6d5	HE DID THE MASH, HE DID THE	<input type="text"/>
38a7c9279codeb44		PURLOINED	<input type="text"/>
08ae5745a7b7af7a	9dca1d79d4dec6d5	FAV. LATER-3 POKEMON	

THE GREATEST CROSSWORD PUZZLE
IN THE HISTORY OF THE WORLD



source: XKCD



Simple Illustration of Zoom Encryption Failure



by Davi Ottenheimer on April 10, 2020

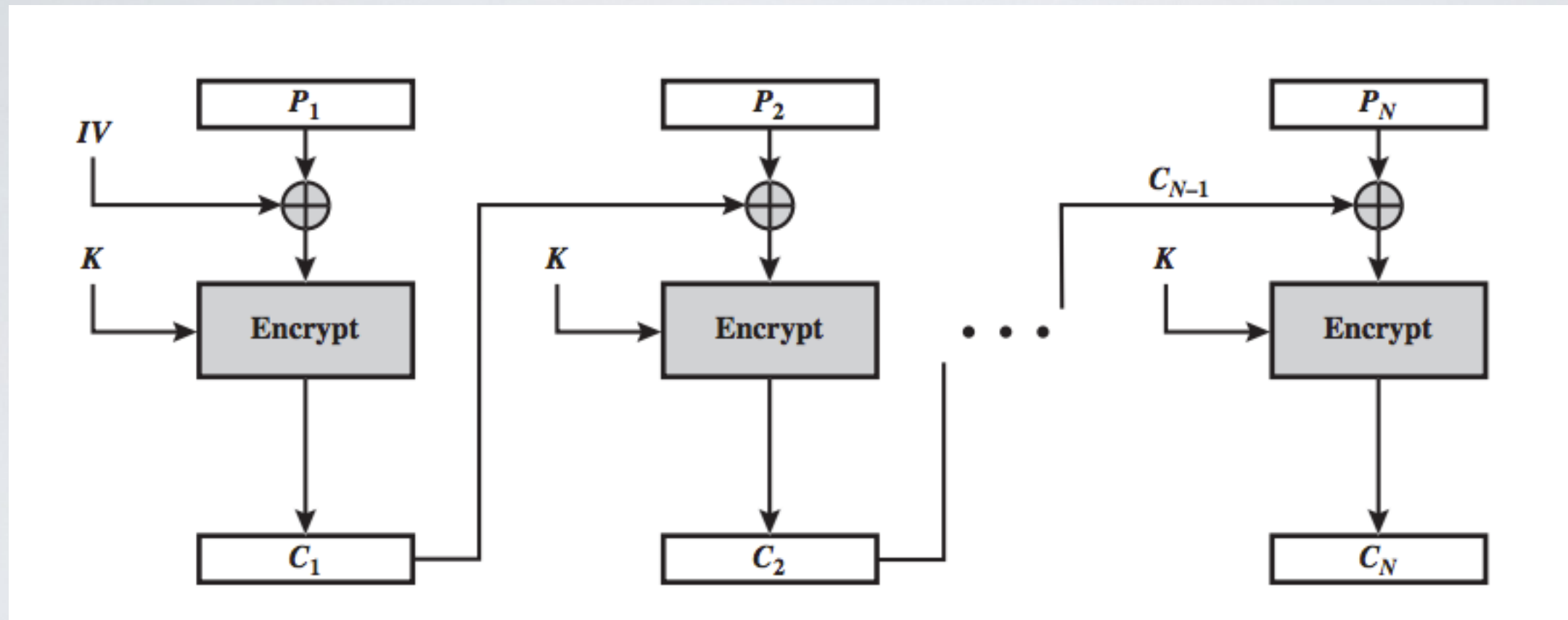
The Citizen Lab [April 3rd, 2020 report](#) broke the news on Zoom using weak encryption and gave this top-level finding:

“

Zoom [documentation](#) claims that the app uses “AES-256” encryption for meetings where possible. However, we find that in each Zoom meeting, a single AES-128 key is used in ECB mode by all participants to encrypt and decrypt audio and video. The use of ECB mode is not recommended because patterns present in the plaintext are preserved during encryption.

source: *Security Boulevard*

CBC - Cipher Block Chaining (a.k.a Chaining Mode)



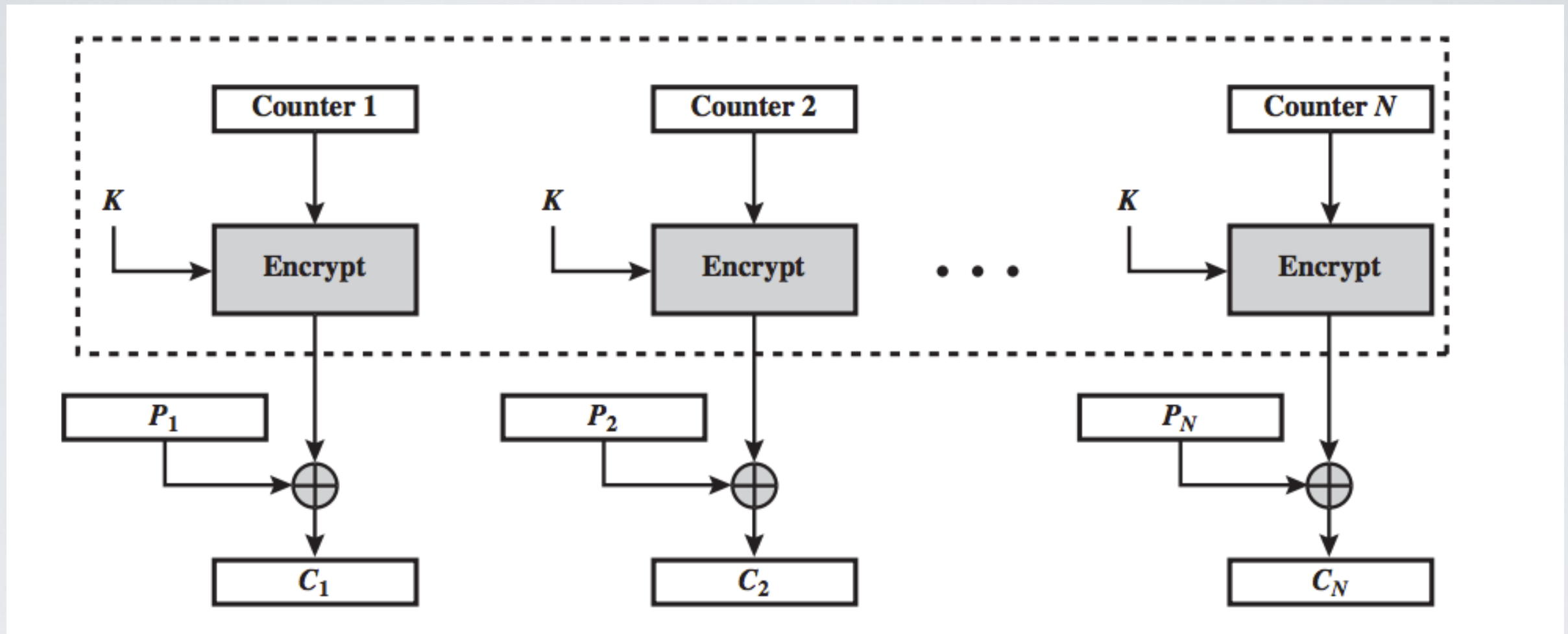
Introduce some randomness using the previous ciphertext block

✓ Repeating plaintext blocks are not exposed in the ciphertext

⦿ No parallelism

➡ The Initialization Vector should be known by the recipient

CTR - Counter Mode



Introduce some randomness using a counter

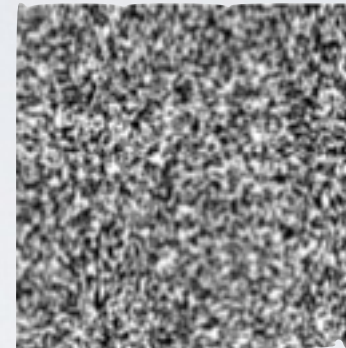
✓ High entropy and parallelism

⦿ Behaves as a stream cipher - sensitive to key-reused attack

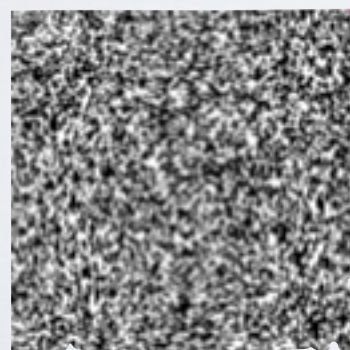
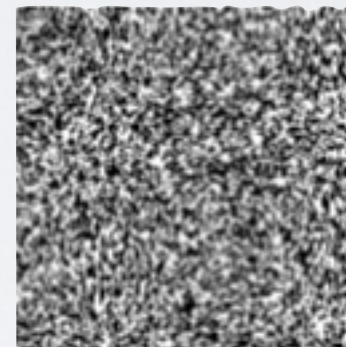
Key-reused attack on CTR



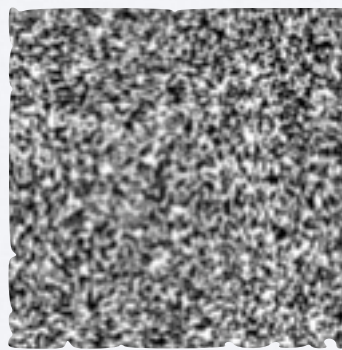
$$\oplus K =$$



$$\oplus K =$$



$$\oplus$$



$$=$$



Symmetric Encryption

Stream Cipher vs Block Cipher

	Stream Cipher	Block Cipher
Approach	Encrypt one symbol of plaintext directly into a symbol of ciphertext	Encrypt a group of plaintext symbols as one block
Pro	Fast	High diffusion
Cons	Low diffusion	Slow

Stream cipher and block cipher are often used together

- Stream cipher for encrypting large volume of data
- Block cipher for encrypting fresh pseudo-random seeds

Latest trends

AES is now hardware accelerated (AES-NI native instruction)

- ➡ AES is fast enough (~ 1.3 cycles per byte)
to be used as the go-to cipher for any application

<https://security.stackexchange.com/questions/22905/how-long-would-it-take-a-single-processor-with-the-aes-ni-instruction-set-to-bru>

Are we secured?

Security goals vs attacker's model

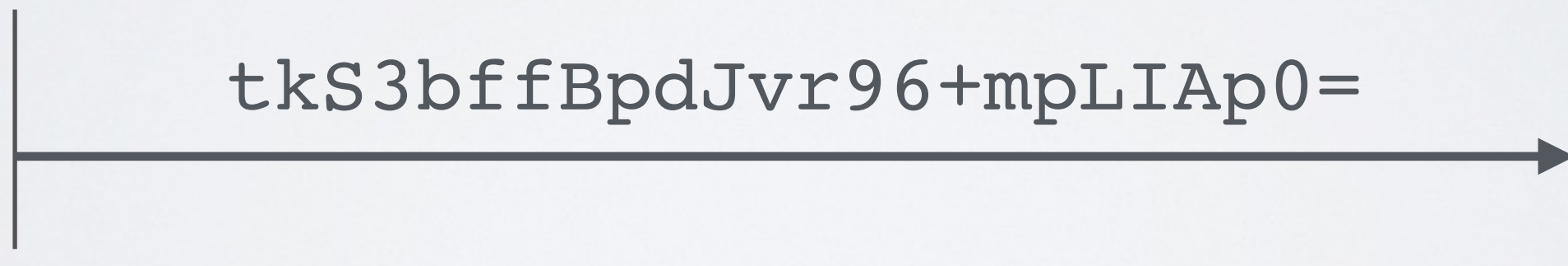


Let us consider **confidentiality, integrity and availability**

(pure) encryption ensures confidentiality ...



$E_k(m) = \text{tkS3bffBp} \dots$

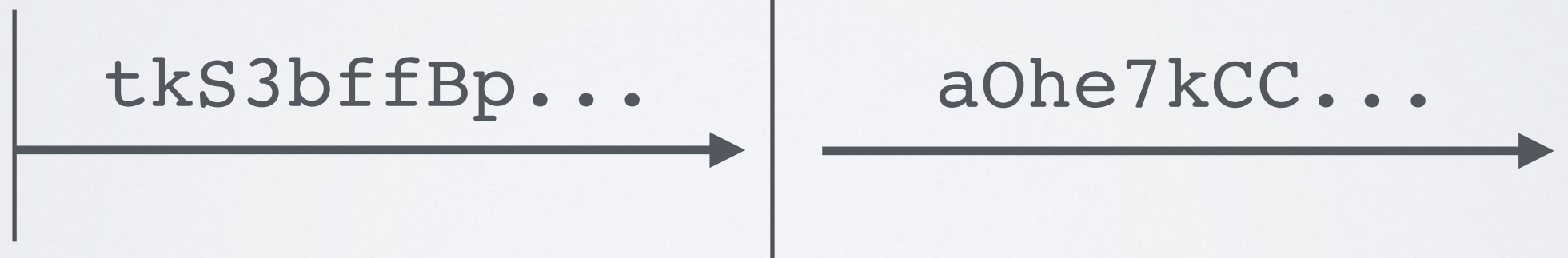


$D_k(\text{"tkS3bffBp} \dots \text{"}) = m$

... but does not ensure integrity !



$E_K(m) = \text{tkS3bffBp} \dots$



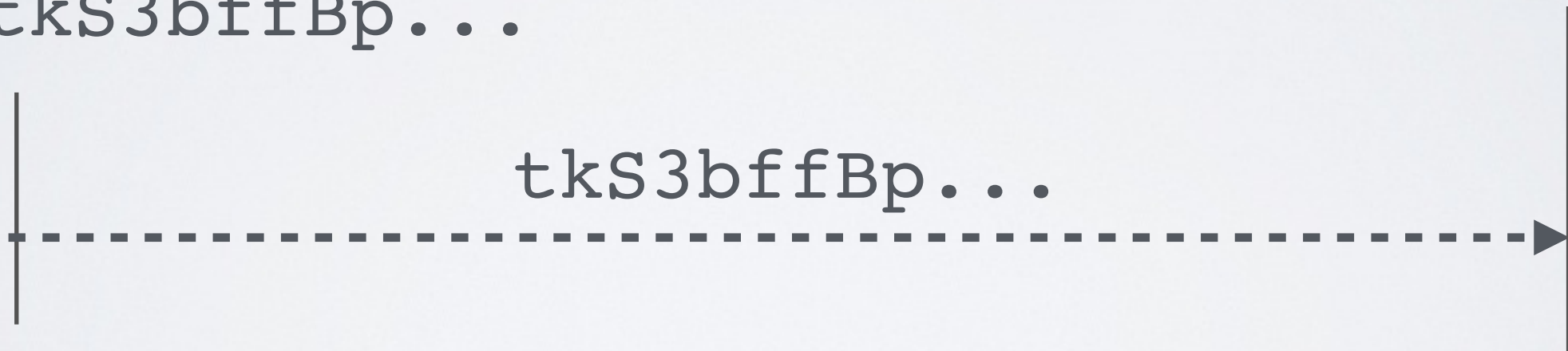
$D_K(\text{"a0he7kCC} \dots \text{"}) = m'$

⦿ Encrypting a message does not authenticate it

One more issue ...



$$E_k(m) = \text{tkS3bffBp} \dots$$

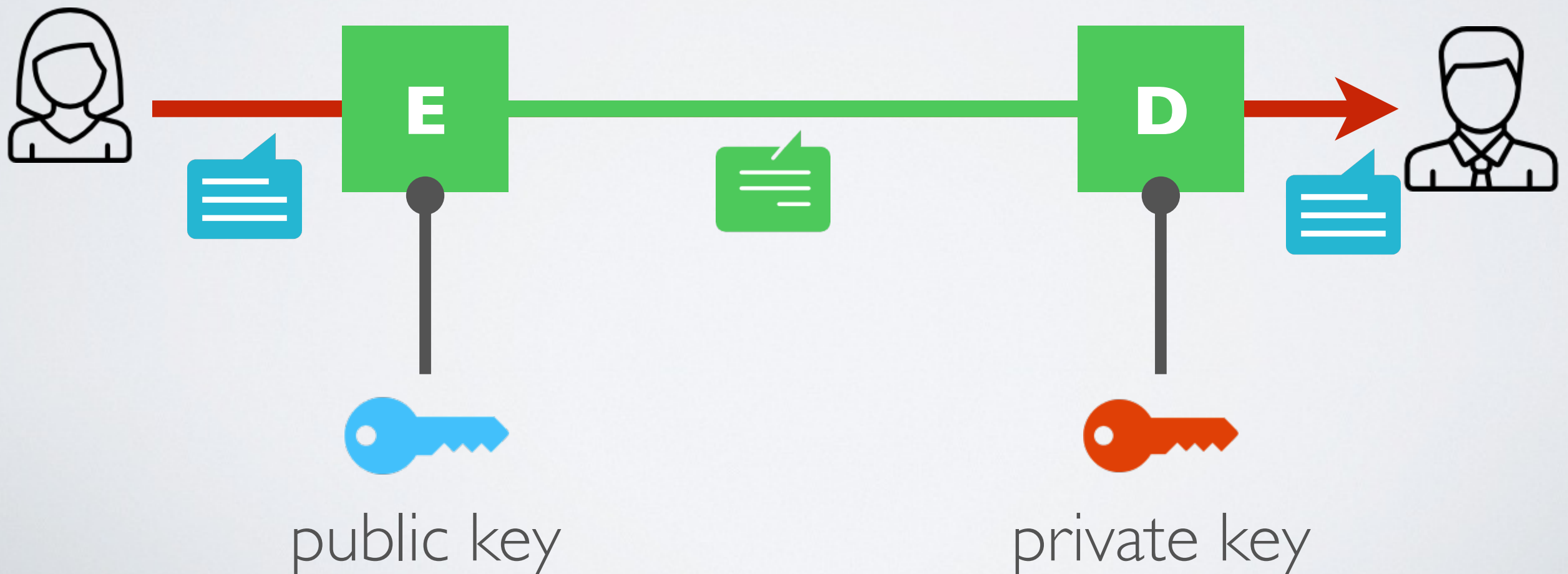


● How does Alice and Bob agree on a symmetric key?

Asymmetric Encryption

Asymmetric encryption a.k.a Public Key Cryptography

- ➡ The public key for encryption
- ➡ The private key for decryption



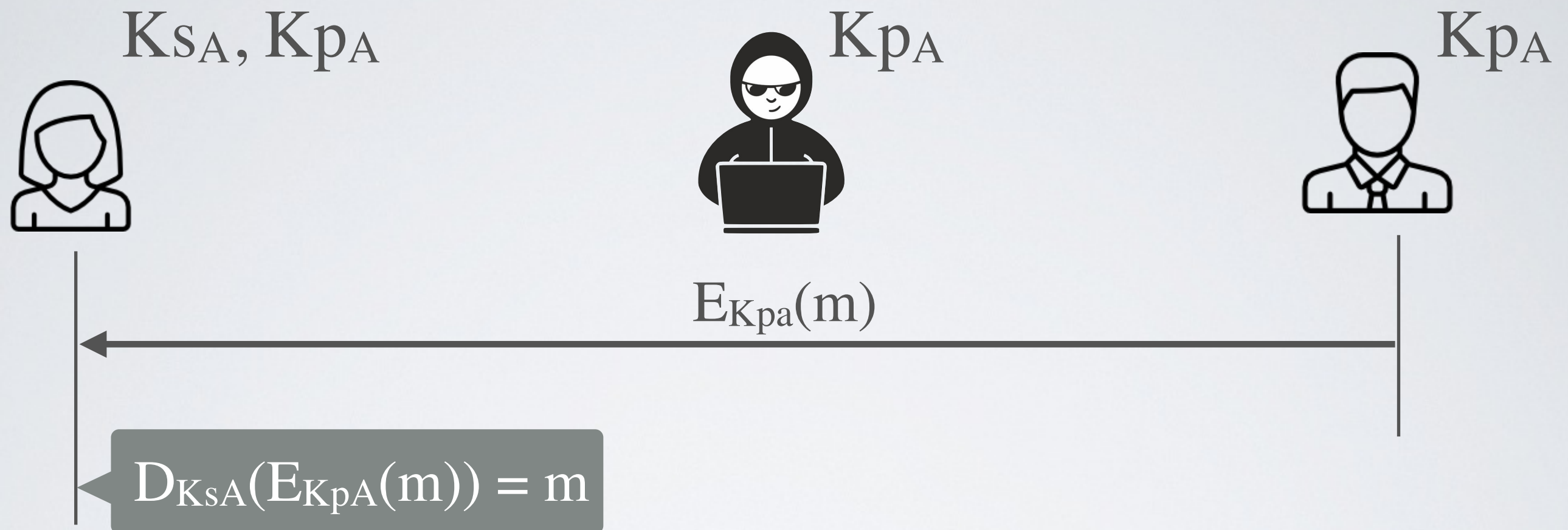
Asymmetric keys



Alice generates a pair of asymmetric keys

- K_{s_A} is the secret key that Alice keeps for herself
 - K_{p_A} is the public key that Alice gives to everyone (even Mallory)
- ➔ These two keys K_{s_A} and K_{p_A} work together

Asymmetric encryption for **confidentiality**



Bob encrypts a message m with Alice's public key K_{pA}

➔ Nobody can decrypt m , except Alice with her private key K_{sA}

✓ Confidentiality without the need to exchange a secret key

Functional Requirements

$D_{K_s}(E_{K_p}(m)) = m$ and $D_{K_p}(E_{K_s}(m)) = m$ for every pair (K_p, K_s)

- ✓ Generating a pair (K_p, K_s) is easy to compute (polynomial)
- ✓ Encryption is easy to compute (either polynomial or linear)
- ✓ Decryption is easy to compute (either polynomial or linear)
- Finding a matching key K_s for a given K_p is hard (exponential)
- Decryption without knowing the corresponding key is hard (exponential)

RSA - Rivest, Shamir and Alderman

Key Size	1024 - 4096
Speed	<p>~ factor of 10^6 cycles / byte</p> <ul style="list-style-type: none">• Key generation: 10 - 100 ms• Encryption: 0.2 - 2 ms• Decryption: 5 - 10 ms
Mathematical Foundation	Prime number theory

Number Theory - Prime numbers

Prime Numbers

- p is prime if 1 and p are its only divisors e.g 3, 5, 7, 11 ...
 - p and q are relatively prime (a.k.a. coprime) if $\gcd(p,q) = 1$
e.g $\gcd(4,5) = 1$
- ➔ There are infinitely many primes

Euler-Fermat Theorem

If $n = p \cdot q$ and $z = (p-1) \cdot (q-1)$

and a such that a and n are relative primes

Then $a^z \equiv 1 \pmod{n}$

Computational Complexity

Easy problems with prime numbers

- Generating a prime number p
- Addition, multiplication, exponentiation
- Inversion, solving linear equations

Hard problem with prime numbers

- Factoring primes
e.g. given n find p and q such that $n = p \cdot q$

RSA - generating the key pair

1. Pick p and q two large prime numbers and calculate $n = p \cdot q$
(see primality tests)
2. Compute $z = (p-1) \cdot (q-1)$
3. Pick a prime number $e < z$ such that e and z are relative primes
➔ (e, n) is the **public key**
4. Solve the linear equation $e * d = 1 \pmod{z}$ to find d
➔ (d, n) is the **private key**
however p and q must be kept secret too

RSA - encryption and decryption

Given $K_p = (e, n)$ and $K_s = (d, n)$

➡ Encryption : $E_{kp}(m) = m^e \bmod n = c$

➡ Decryption : $D_{ks}(c) = c^d \bmod n = m$

➡ **$(m^e)^d \bmod n = (m^d)^e \bmod n = m$**

The security of RSA

RSA Labs Challenge : factoring primes set

Key length	Year	Time
140	1999	1 month
155	1999	4 months
160	2003	20 days
200	2005	18 months
768	2009	3 years

Challenges are no longer active

ECC - Elliptic-Curve Cryptography

Key Size	256 or 448 bits
Speed	<p>~ factor of 10^6 cycles / operation</p> <ul style="list-style-type: none">• Key generation: 1 - 5 ms (way faster than RSA)• Encryption: 1 - 5 ms• Decryption: 1 - 5 ms
Mathematical Foundation	Elliptic curves over finite fields

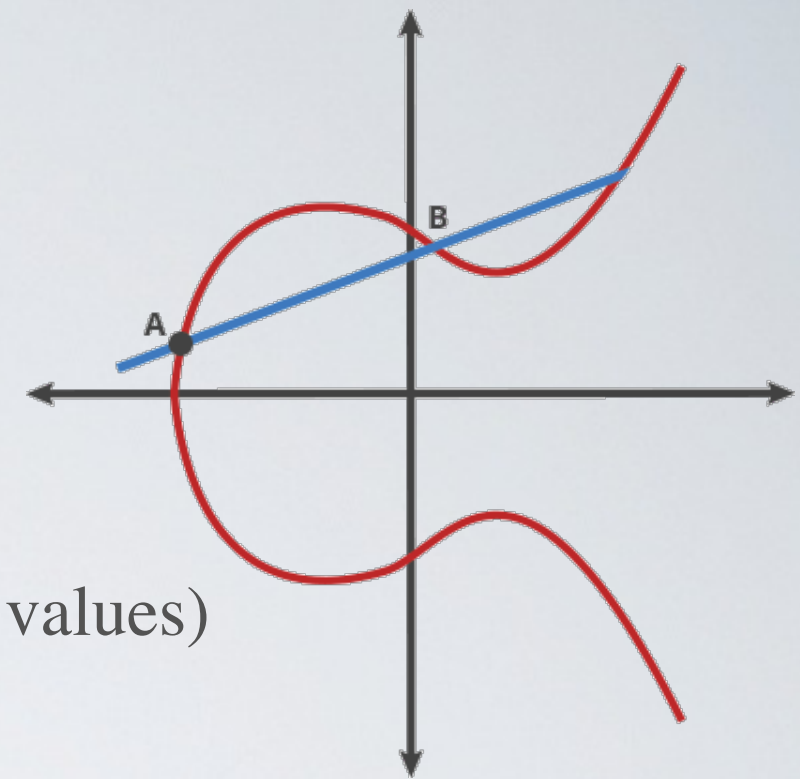
Main ECC Standards

	secp256k1	curve25519	curve448
Year	2000	2005	2014
Inventor	Standards for Efficient Cryptography Group (SECG)	Daniel J. Bernstein	Mike Hamburg
Key Size	256	256	448
Applications	Bitcoin Ethereum	TLS,TOR Signal Protocol Monero, Zcash	TLS
Performances	+	++	+++

Elliptic Curve Cryptography

Use Elliptic-curve for generating a cryptographic public-key pair
The algorithm is based on two public pieces:

- The curve equation $y^2 = x^3 + ax + b$ (a and b are fixed values)
- The generator point (fixed value)



When generating a key pair

1. the user "choose a random number" (within a given range) as private key
2. then derived the public key from the curve

- ✓ Smaller key sizes: 256 bits EC keys has the same entropy as RSA 3072 bits
- ✓ Can be used for digital signature (ECDSA algorithm)
- ✓ Can be used for key agreement (ECDH algorithm)

<https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/>

Symmetric vs Asymmetric

Key length and Key n-bit security

- RSA has very long keys, 1024, 2048 and 4096 are common
- ECC has shorter keys, 256 and 448 are common
- Is it more secure than symmetric crypto with key lengths of 56, 128, 192, 256 ?

➔ Key lengths **do not compare !**

RSA	ECC	Effective key length
1,024		80
2,048		112
3,072	256	128
4096		140
15,360	448	224 ~ 256

Asymmetric vs Symmetric

	Symmetric	Asymmetric
pro	Fast	No key agreement
cons	Key agreement	Very slow

The best of both worlds

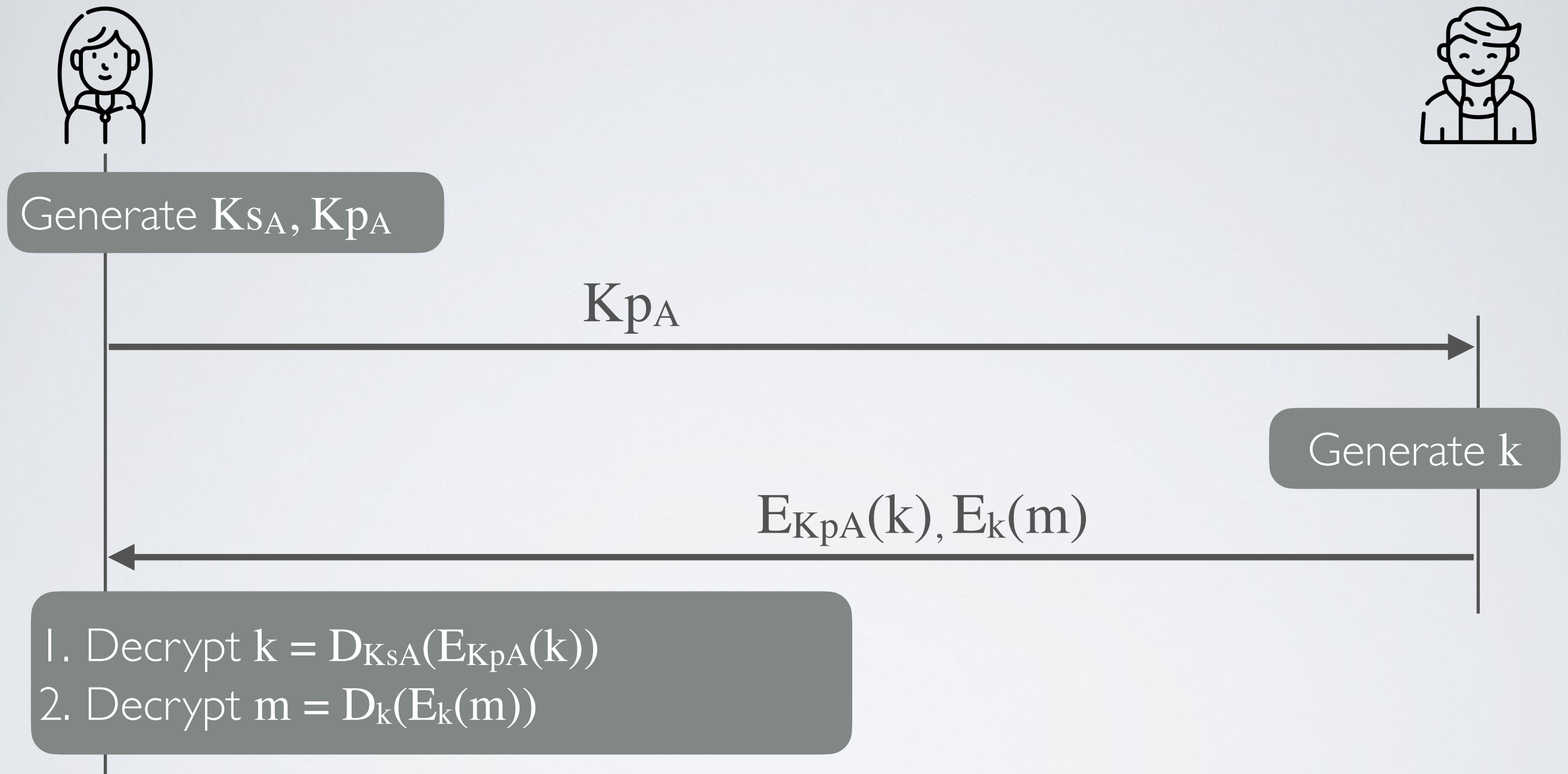
- ➡ Use asymmetric encryption to encrypt a shared key (or hash)
- ➡ Use symmetric encryption to encrypt message

$$E_{K_p}(m) = \text{RSA}_{K_p}(k), \text{AES}_k(m)$$

Naive
approach

Key Exchange Protocols

Naive key exchange using asymmetric encryption



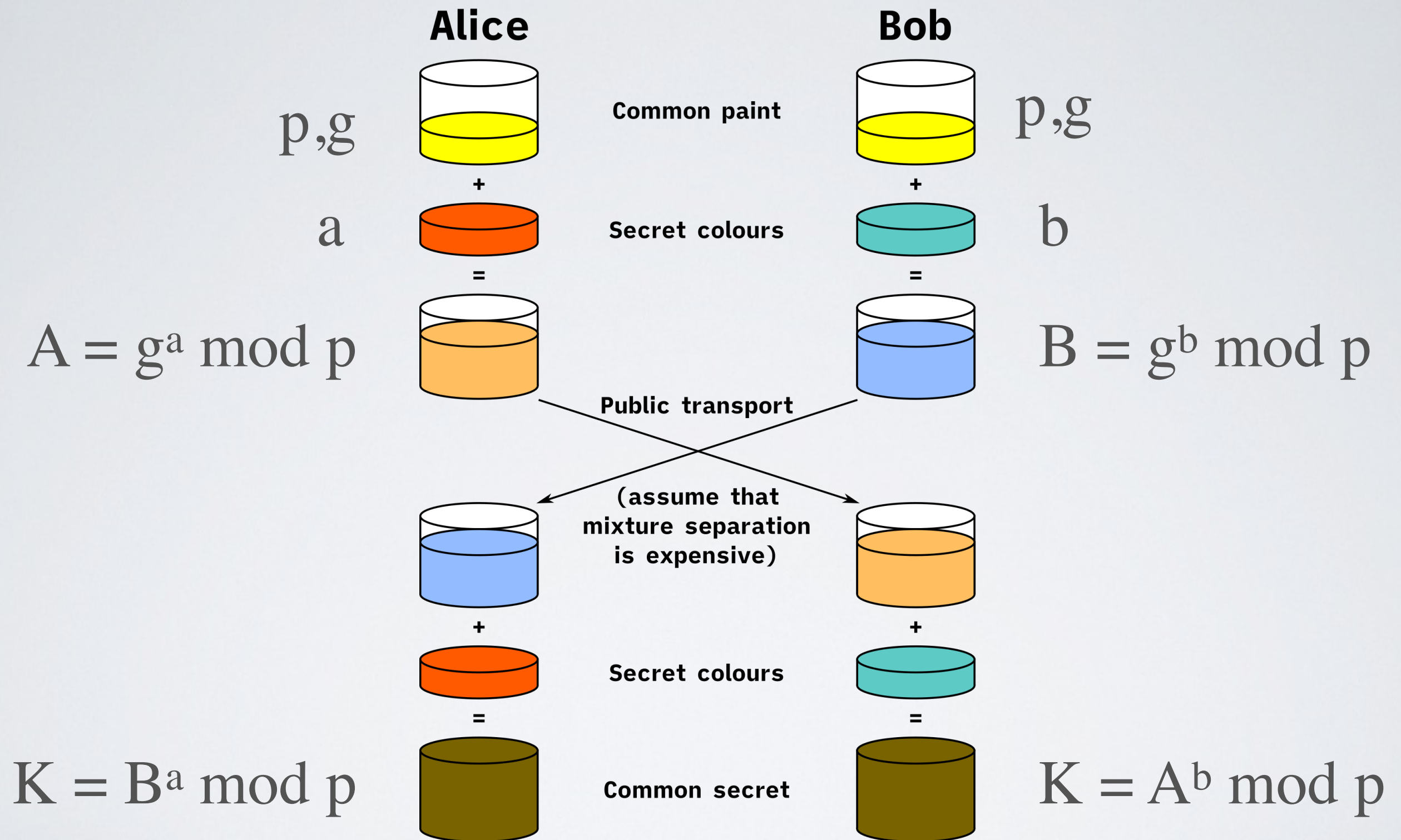
- ◉ Protecting the shared key is the **responsibility of Alice only**
- ◉ Generating the shared key is the **responsibility of Bob only**

What is the solution?

Could Alice and Bob could magically come up with a key without exchanging it over the network?

➔ The magic is called **Diffie-Hellman-Merkle Protocol**

The Diffie-Hellman-Merkel key exchange protocol



$$K = g^{ab} \text{ mod } p = (g^a \text{ mod } p)^b \text{ mod } p = (g^b \text{ mod } p)^a \text{ mod } p$$

The Diffie-Hellman-Merkel key exchange protocol



1. Generates public numbers p and g such that g is co-prime to $p-1$
2. Generates a secret number a
3. Sends $A = g^a \bmod p$ to Bob

A, p, g

1. Generates a secret number b
2. Sends $B = g^b \bmod p$ back to Alice
3. Calculates the key $K = A^b \bmod p$

B

4. Calculates the key $K = B^a \bmod p$

Diffie-Hellman-Merkle in practice

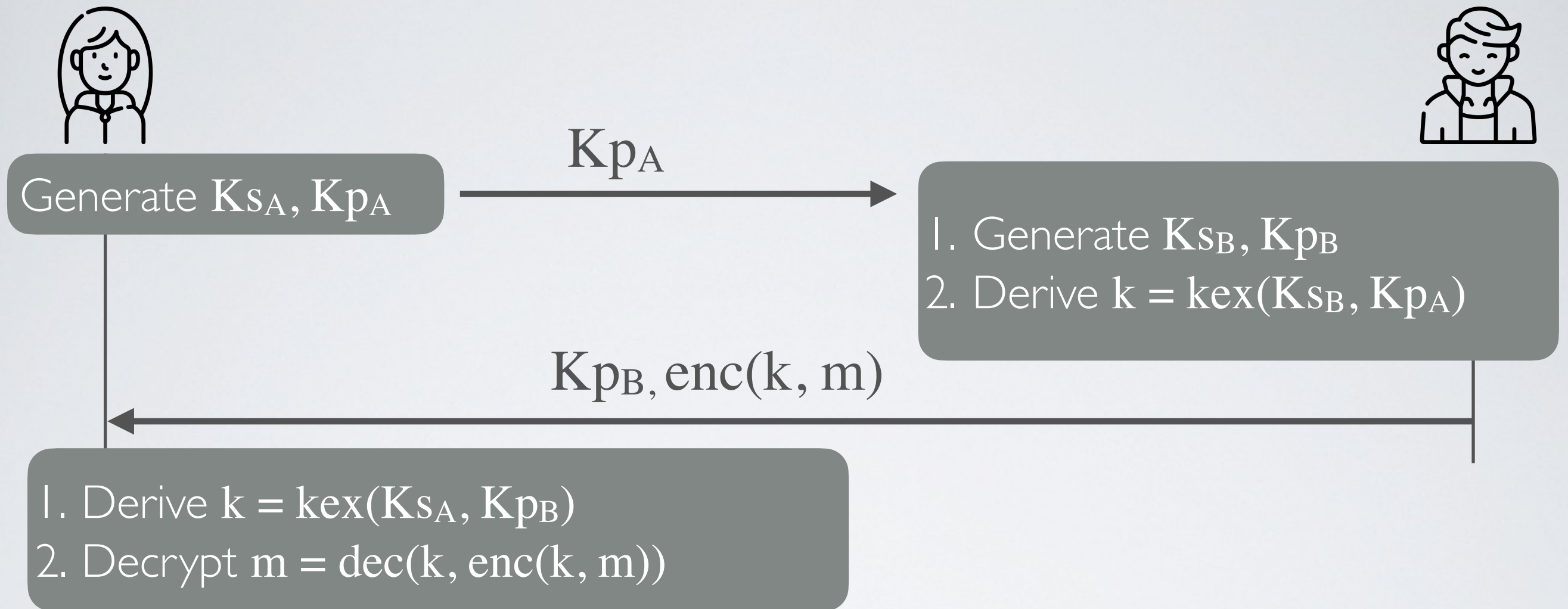
- g is small (either 3, 5 or 7 and fixed in practice)
 - p is at least 2048 bits (and fixed in practice)
 - private keys a and b are 2048 bits as well
- ➔ So the public values A and B
and the master key k are 2048 bits
- ➔ Use k to derive an AES key using a Key Derivation Function
(usually HKDF - the HMAC-based Extract-and-Expand key derivation function)

Elliptic Curve Diffie-Hellman-Merkle (ECDH)

- ➔ Generate a symmetric key k from two distinct asymmetric key pairs: K_{pA}, K_{sA} and K_{pB}, K_{sB}

$$k = \text{ECDH}(K_{sA}, K_{pB}) = \text{ECDH}(K_{sB}, K_{pA})$$

ECDH Key exchange



Diffie-Hellman-Merkle provides a way to generate a shared key from two asymmetric key pairs

$$\text{kex}(K_{sA}, K_{pB}) = \text{kex}(K_{sB}, K_{pA}) = k$$

- ✓ Mutual contribution to the key generation
- ✓ No need to send the encrypted shared key

A widely used key exchange protocol

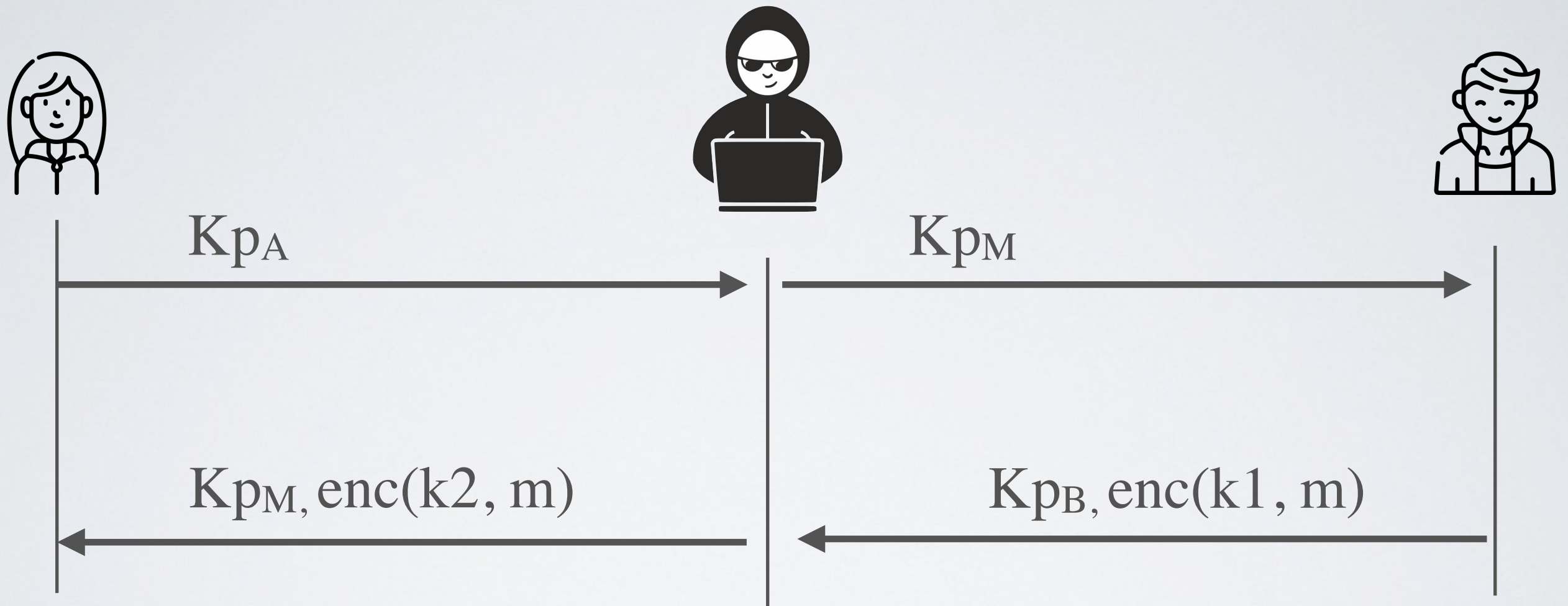
ECDH is in many protocols

- SSH
- TLS (used by HTTPS)
- Signal (used by most messaging apps like Whatsapp)
- and so on ...

✓ It is fast and requires two exchanges only

- ⦿ But how to make sure Alice is talking to Bob and vice-versa?
Diffie-Hellman-Merkle alone **does not ensure authentication**

Are we done yet?



✓ Encryption and key exchange protects against confidentiality ...

⊙ ... but not **does not protect integrity**