Module 3

Internet Security and Privacy

Some communication mediums are unsafe

What can be eavesdropped upon?

- Air (for broadcast messages such as wireless)
- Copper wires (vampire tap)
- Optical fiber
- Devices (phones, computers, etc.)

Our goals:

- **Confidentiality** – Safeguard packets from eavesdropping
- **Integrity** – Prevent packet modification in transmission
- **Authenticity** – Prove the identity of the sender

Cryptography

A cryptosystem consists of:

- Key(s)
- Encryption mechanism
- Decryption mechanism

Kerckhoffs' Principle states that:

The key(s) of a cryptosystem should be hidden, but the mechanisms should be public.

(Why?)

The XOR function $\oplus$

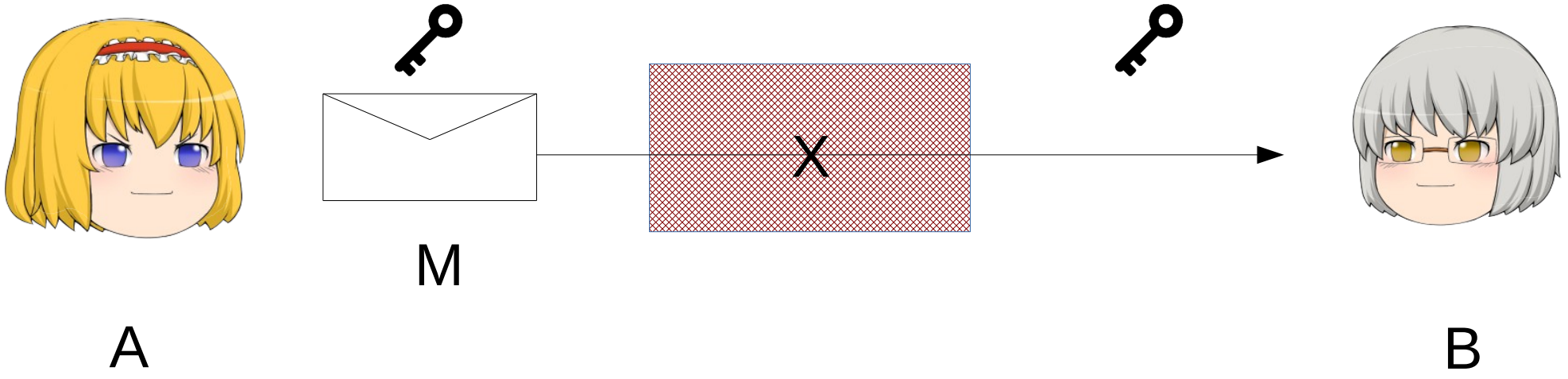
Value table of XOR:

$\oplus$	**0**	**1**
0	0	1
1	1	0

XOR is the same as “Addition modulo 2”.
Bit-by-bit XOR of two bit strings:

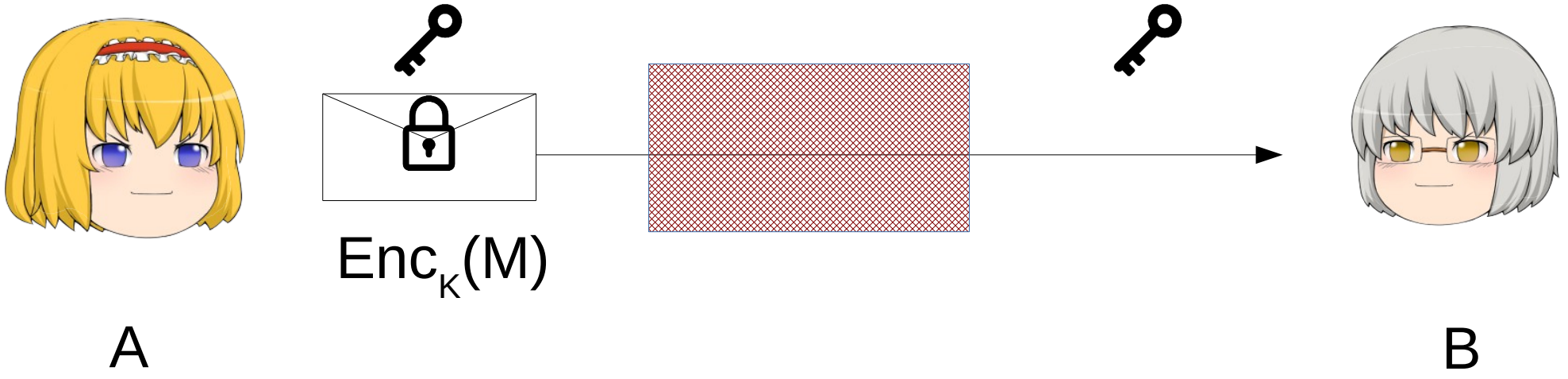
$$(0110) \oplus (1011) = (1101)$$

Encryption and decryption



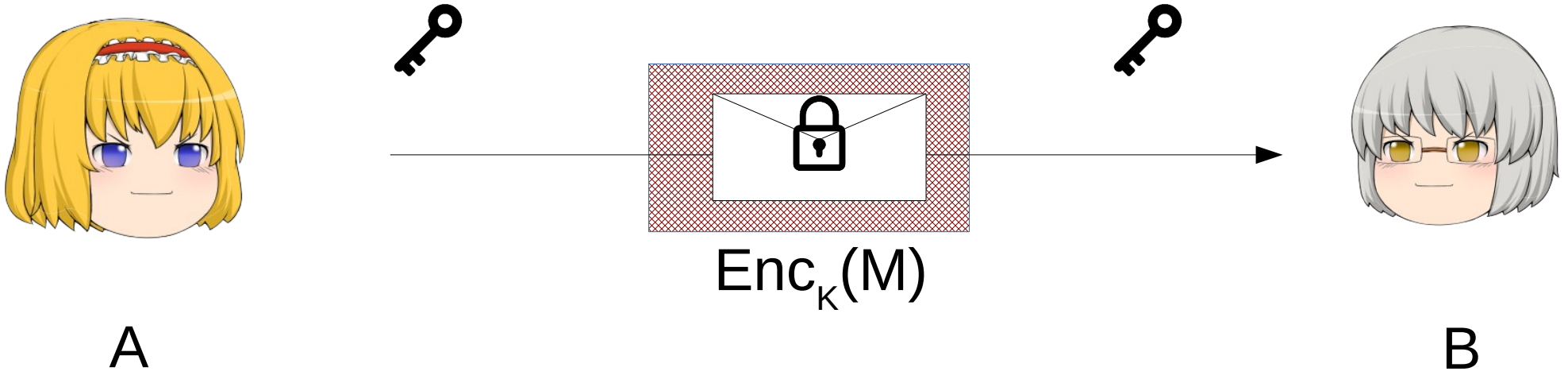
Scenario: A wants to send **plaintext** M to B, but doesn't want the attacker to see M when it passes through the unsafe medium (red).
A and B both already know some key K.

Encryption and decryption



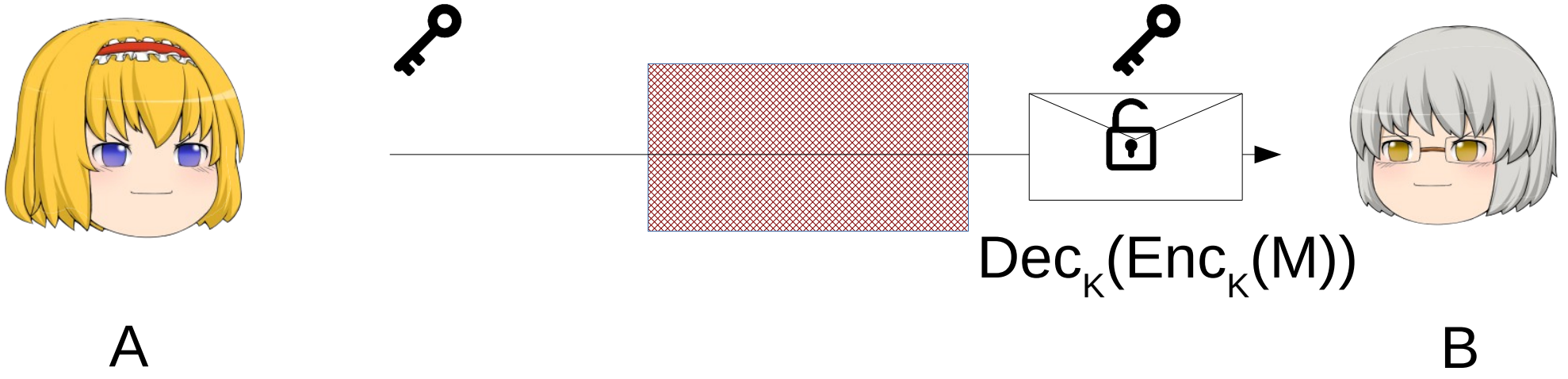
1. Using the encryption mechanism Enc() and key K, A encrypts M to a **ciphertext**, $Enc_K(M)$.

Encryption and decryption



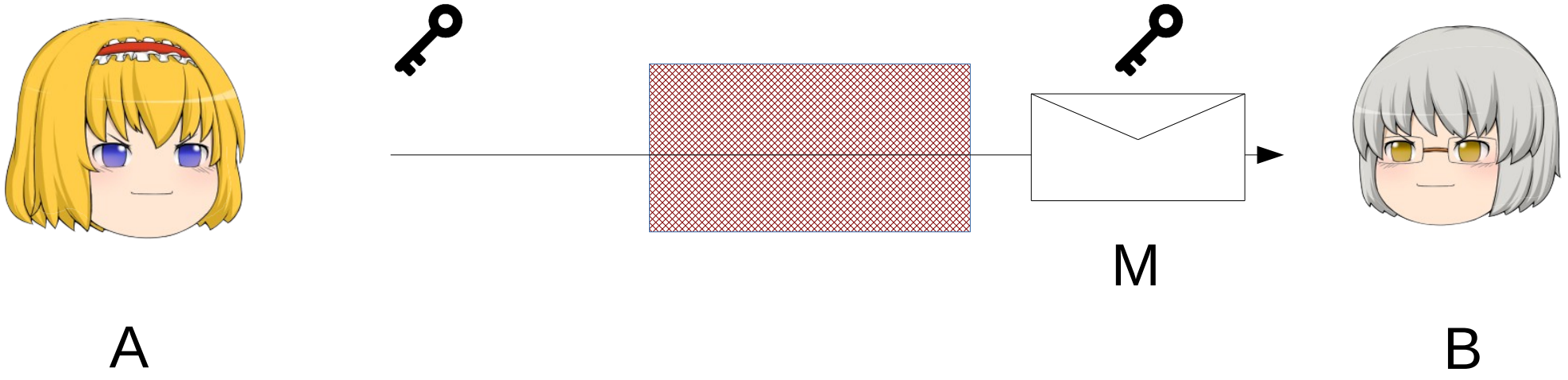
2. A sends $Enc_K(M)$ across the channel.

Encryption and decryption



3. B receives $Enc_K(M)$, and decrypts it using the decryption procedure Dec() and key K.

Encryption and decryption



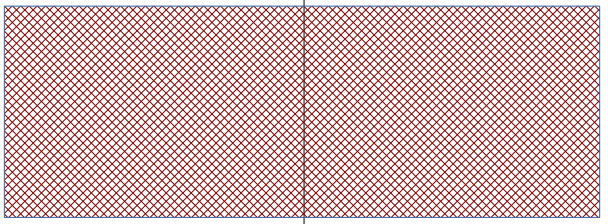
4. Dec(Enc(M)) = M; B receives the plaintext message M.

Simple System: The Caesar Cipher

Plaintext: ATTACK “LONG A”

Encryption: Add K letters (K=1 here)

Ciphertext: BUUBDL “MPOH B”



Unsafe medium

Ciphertext: BUUBDL “MPOH B”

Decryption: Subtract K letters (K=1 here)

Plaintext: ATTACK “LONG A”

Simple System: The Caesar Cipher

Problems of this cryptosystem:

- **Ciphertext Repetition**: What if you see BUUBDL “MPOH B” and then EFGFOE “MPOH B”?
- **Key Update:** For security, we should update the key frequently. How can we do so?
- **Short Key Length:** How many possibilities are there for the encryption/decryption mechanism?
- **Frequency analysis:** If the letter “F” appears most frequently in ciphertexts, what does it mean?

Solving the Ciphertext Repetition Problem

- The problem is that since encryption is a deterministic function,

$$Enc_K(M) = Enc_K(M)$$

- Idea: Introduce a Initialization Vector (IV) into the encryption to modify the function

$$Enc_{K,\,IV1}(M) \neq Enc_{K,\,IV2}(M)$$

- Each message under the same key must have a different IV
- The IV is sent **publicly** alongside the message – it does not matter if the attacker sees it

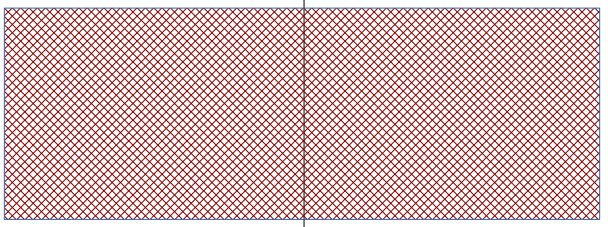
Simple System: The Caesar Cipher

Plaintext: ATTACK “LONG A”

Encryption: Add K’=6 letters

Ciphertext: GZZGIQ “RUTM G”, IV=3

Both parties calculate:
K’ = K*IV
(K=2, IV=3)



Unsafe medium

Ciphertext: GZZGIQ “RUTM G”, IV=3

Decryption: Subtract K’=6 letters

Plaintext: ATTACK “LONG A”

Solving the Key Update Problem

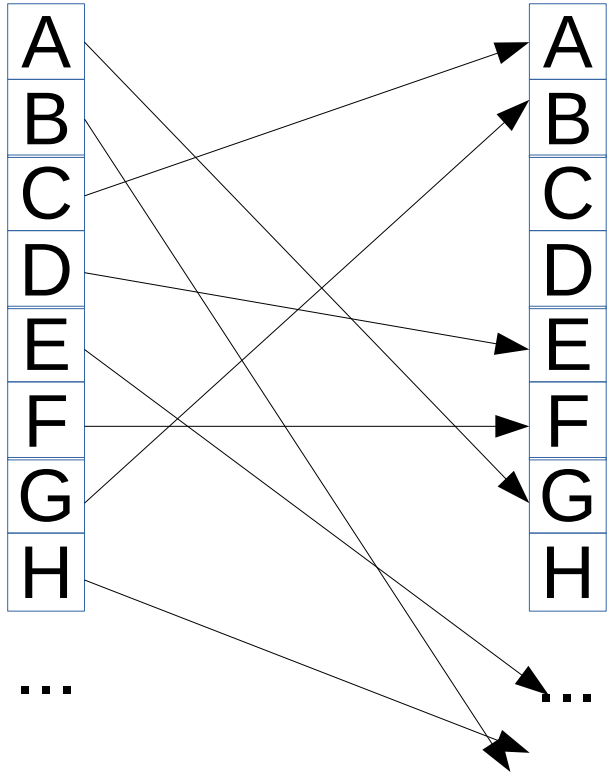
Find a safe channel to deliver the key instead

- Hand-delivered documents, cards
- Not practical for computer systems

Public Key Encryption

- In PKE, the encryption and decryption keys are different
- This can be used to create a safe channel on an unsafe one
- Only send the encryption key across the channel
- More later

Solving the Key Length Problem (Substitution Cipher)



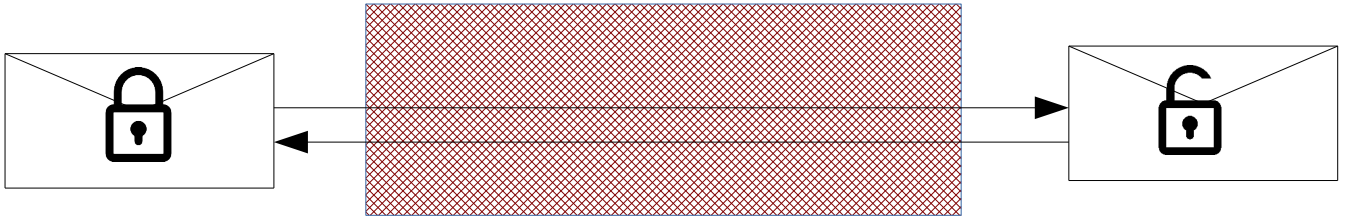
How many variations are there?

26! ~= 2^{88} => Key length is “88 bits”

Solving the Key Length Problem (Substitution Cipher)

We will use variation number 309273 to converse.

Sent in safe channel



$Enc_{309273}(M)$ $Dec_{309273}(M)$

The “variation number” is the cryptosystem's **key**

Solving the Frequency Analysis Problem

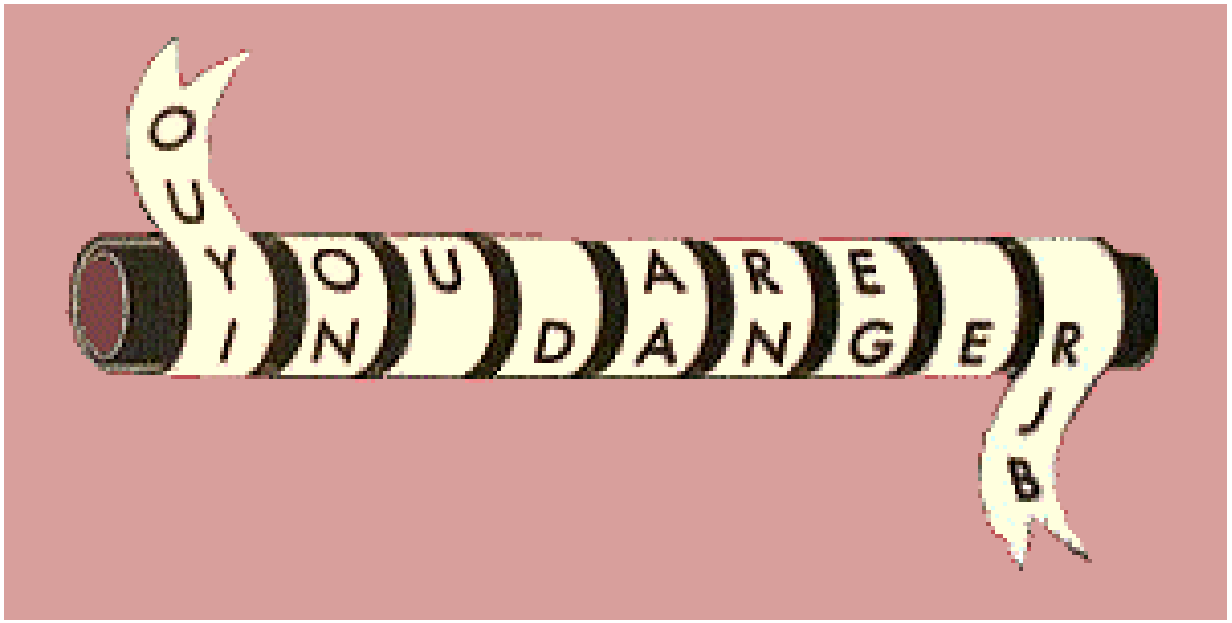
- We cannot do this easily – all substitution ciphers are weak to frequency analysis (cryptograms!)
- One suggested solution (Vignere ciphers): shift different letters based on their position using a key
  - e.g. key = DOG (4 15 7), then shift 1st letter by 4, 2nd by 15, 3rd by 7, 4th by 4, 5th by 15, ...
  - Easily defeated! (How?)
- Broader category of cryptanalysis can defeat almost all “homemade” cryptography

Symmetric Key Encryption (SKE)

- A type of cryptosystem where *the two parties both know a secret key.*
- If the key is K, then the encryption and decryption algorithms are $Enc_K()$ and $Dec_K()$.
- $Enc_K()$ and $Dec_K()$ are public, but K must be secret.
- $Enc_K(M)$ should not reveal either K or M.
- Both parties can encrypt and decrypt.

We will discuss three types: OTP, Stream Ciphers and Block Ciphers

Scytale



What is the key in this cryptosystem?

Enigma machine

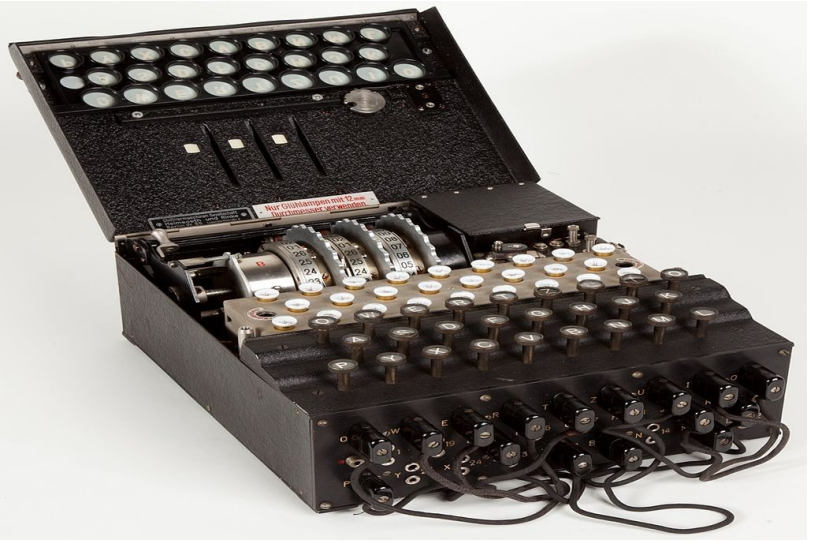
- The key is the rotor position
- Codebook contains an initial position

1)Set to initial position

2)Type a new position

3)Set machine to new position

4)Type message



One-Time Pad

Plaintext: Write in bit form (e.g. “ABC”)

`01000001 01000010 01000011`

Key: Uniformly random bit sequence

`10110100 01010101 10001111`

Encrypt: Bit-by-bit XOR key with plaintext

Ciphertext: `11110101 00010111 11001100`

Decrypt: Bit-by-bit XOR key with ciphertext

`01000001 01000010 01000011`

One-Time Pad

"Perfectly" information secure if:

- Key is truly uniformly random
- Key is only used once, ever

(Why is it perfectly secure?)

VENONA project code-breakers (1943)

One-Time Pad

Breaking a Two-Time Pad:

Suppose the attacker intercepts two ciphertexts:

$$C = M \oplus K \text{ and } C' = M' \oplus K$$

The attacker applies XOR to the ciphertexts to obtain:

$$C \oplus C' = M \oplus K \oplus M' \oplus K$$
$$= M \oplus M'$$

The result is the XOR of the plaintexts.

- If the attacker correctly guesses M, he can obtain M' by $M \oplus C \oplus C'$.
- If the attacker correctly guesses only one word of M (and its position), he can still obtain some letters in M' (at the same position) – he can drag this guess around and observe the result, known as crib-dragging.

Stream cipher

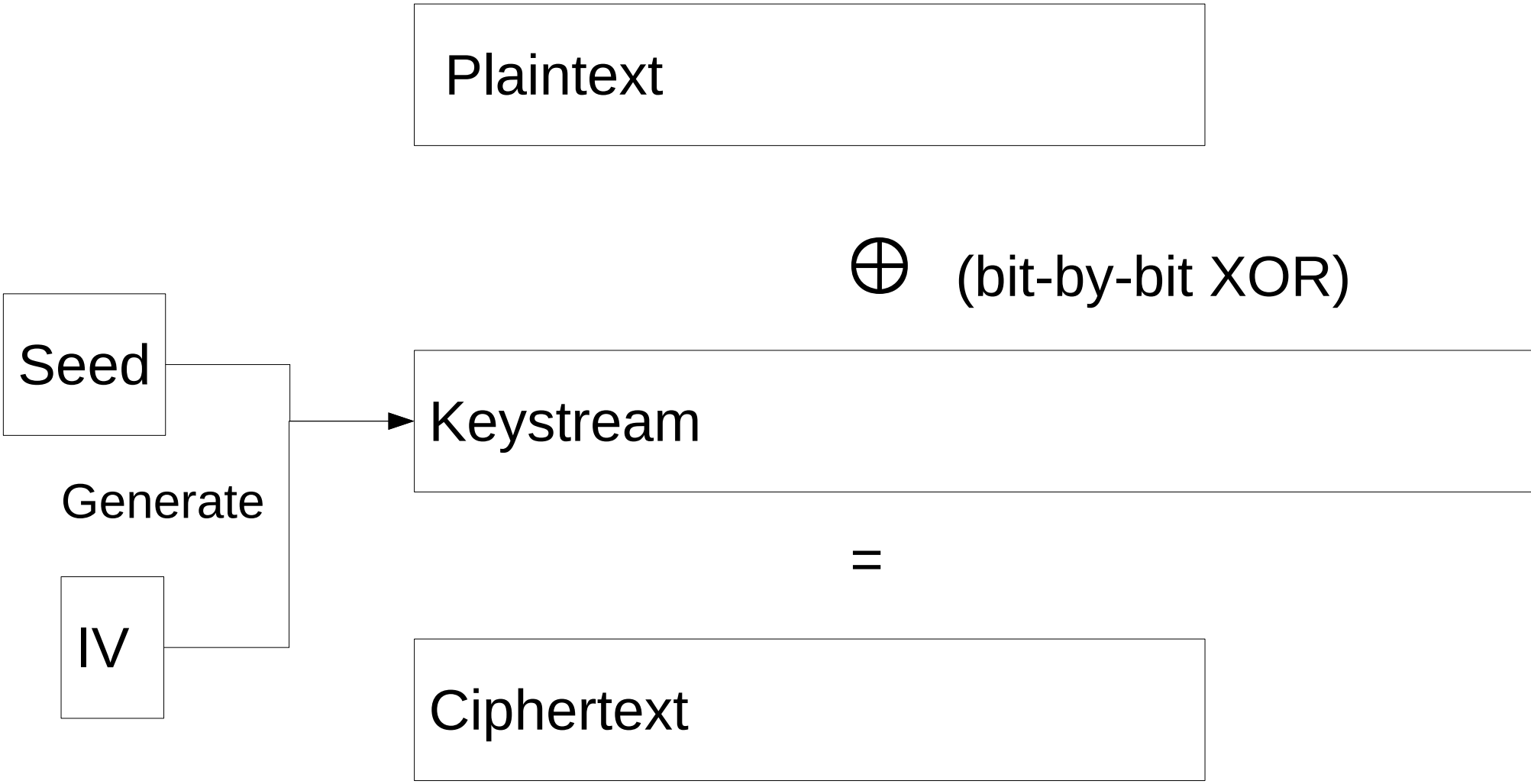
Generates keystream of any length from **random seed**

- Keystream is pseudorandom
- ~~Key is truly uniformly random~~
- Seed and IV are only used once, ever

$$\text{Enc}_{\text{seed, IV}}(M) = \text{Keystream}_{\text{seed, IV}} \oplus M$$

Currently used: A5/1 (cell phones), Salsa20 (TLS)

Stream cipher (Enc/Dec)

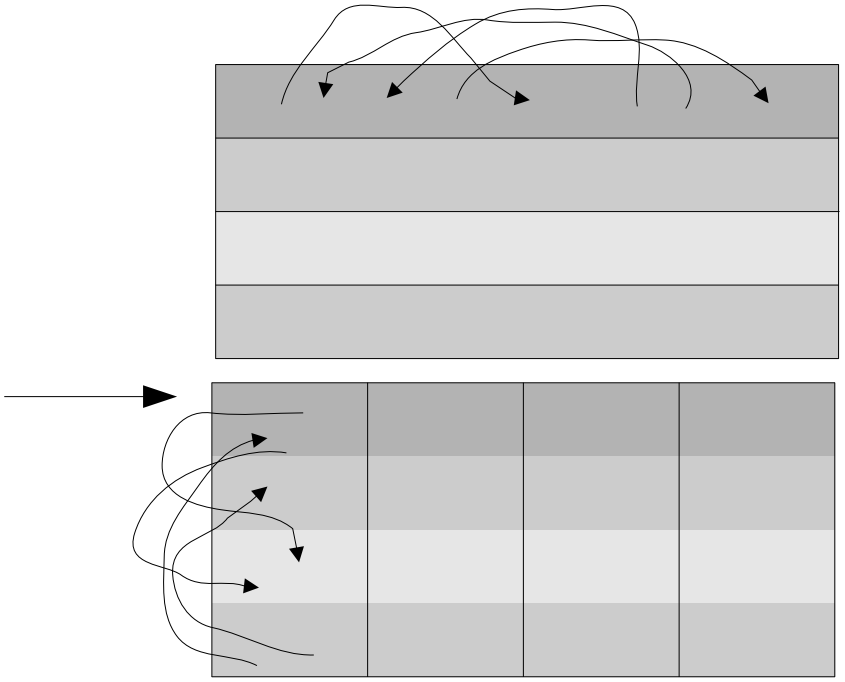


Salsa20 example

Place seed, IV, and position in a 16-by-16 matrix
Each entry is 4 bytes

“expa”	Seed	Seed	Seed
Seed	*“nd 3”*	IV	IV
Position	Position	*“2-by”*	Seed
Seed	Seed	Seed	*“te k”*

Alternatively, scramble each row and scramble each column (10 times each)

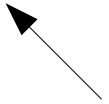


Output all bits as keystream

Block cipher

Difference from stream ciphers:

- There is a fixed block size (128 bits for AES)
- Plaintext is divided into blocks of this size
- We encrypt each block to produce ciphertext
- The “same” key is used for each block



We must change something,
or we run into the ciphertext repetition problem!

Block cipher

We use the **mode** to avoid the ciphertext repetition problem between blocks:

- Electronic codebook (ECB):

  All keys are the same (no defense against ciphertext repetition)

- Cipher Block Chaining (CBC):

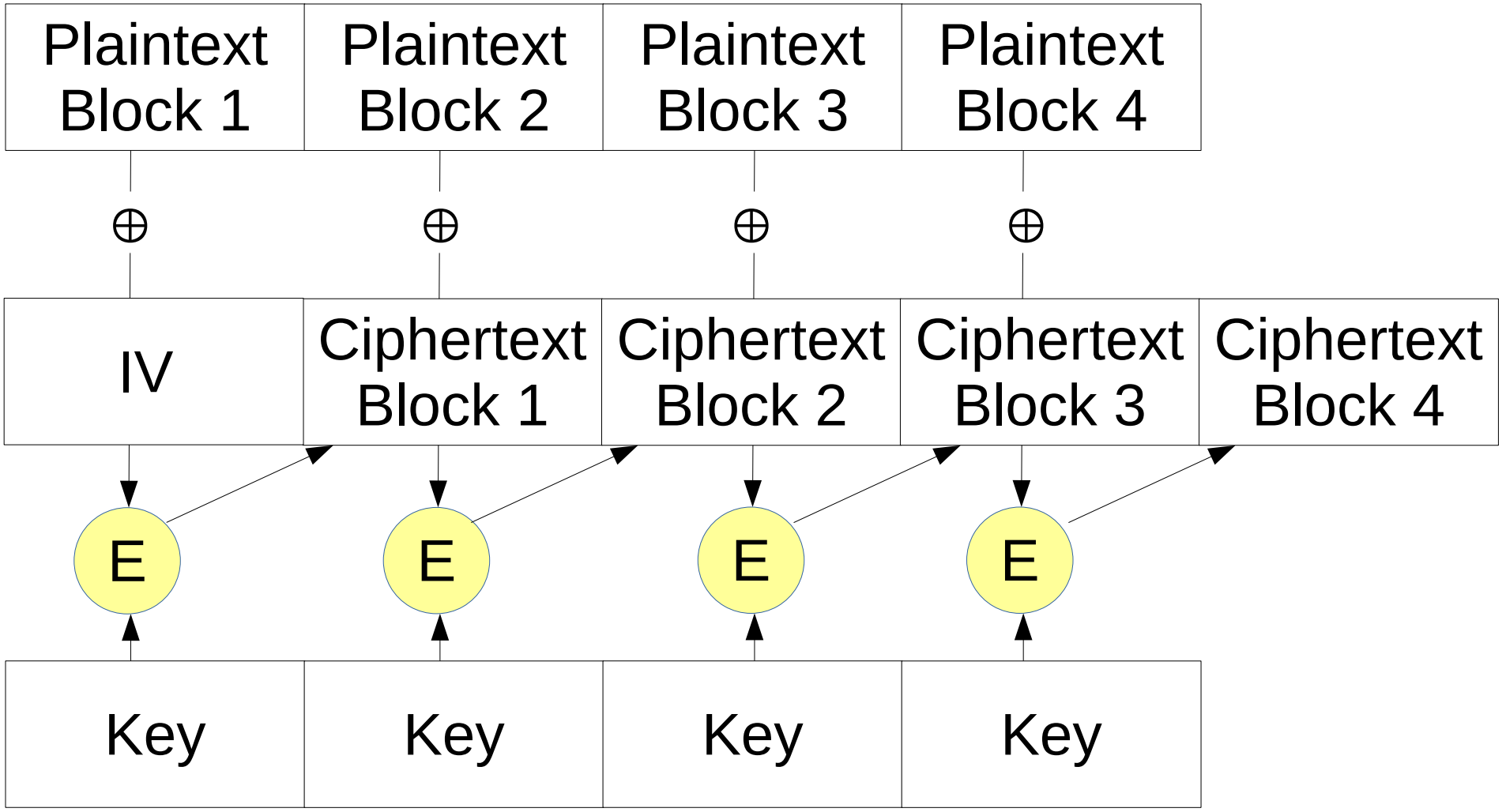
  Each plaintext block X is XOR'd with Ciphertext block (X-1), then encrypted

  Ciphertext block 0 is the IV

- Counter (CTR):

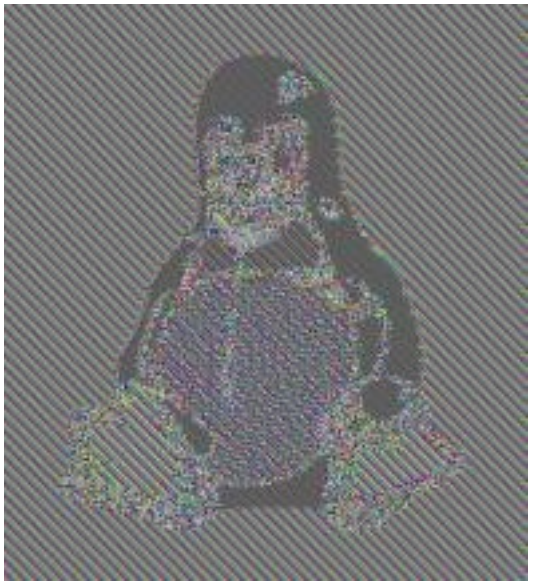
  Each plaintext block X is XOR'd with a "keyblock" that is generated by an encryption of counter X with an IV

CBC mode (AES):



E is the 128-bit encryption mechanism

Block cipher

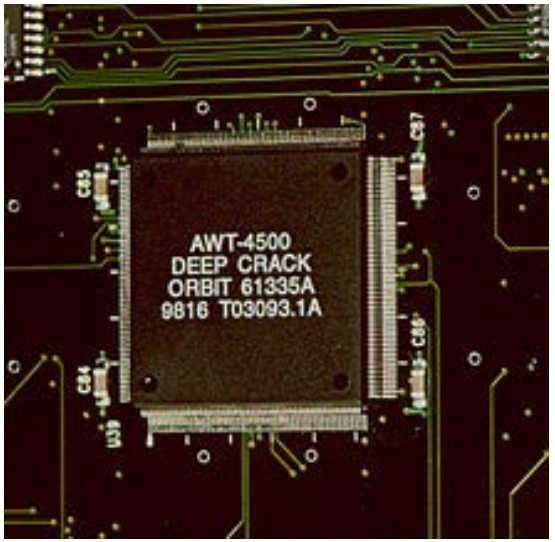


Plaintext ECB mode

ECB mode is insecure!

Block cipher

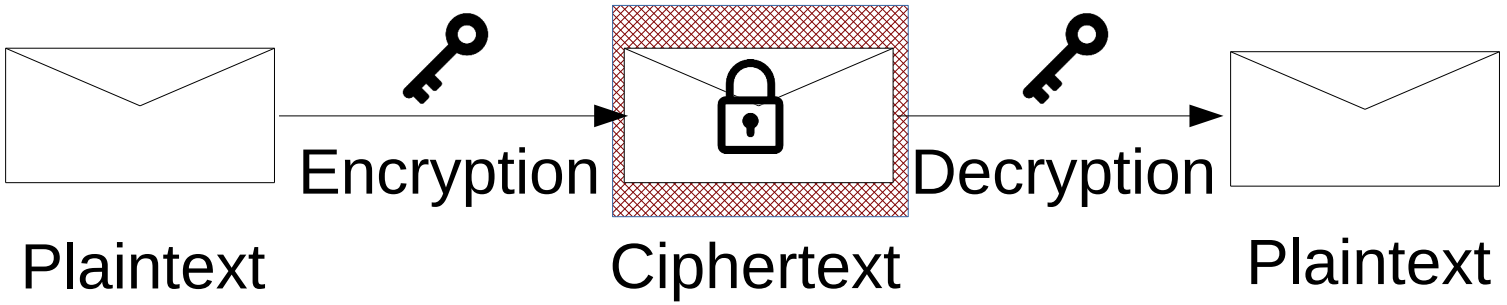
- Includes DES (56-bit), AES (128-bit)
- DES was shown to be too weak in 1998
- AES is the current standard; widely used
- Stream ciphers are generally faster (and keystream can be generated ahead of time)



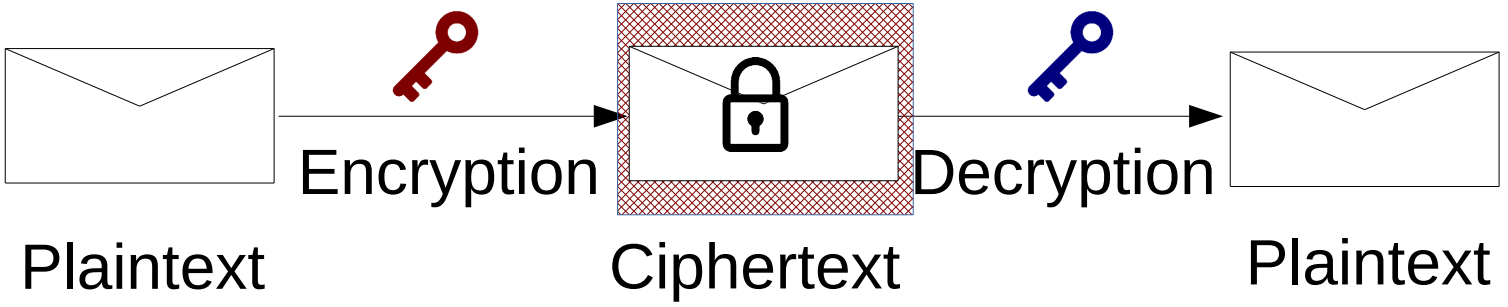
"Deep crack" DES cracker

Public Key Encryption (PKE)

In SKE, locking and opening require the *same key*



What if we want them to require *different keys*?



This is known as *Public Key Encryption*

Public Key Encryption (PKE)

Has two keys for two procedures:

 *Public key* is used for encryption

 *Private key* is used for decryption

Alice generates both keys.

(They are mathematically related.)

Then, Alice publishes her public key: 

 *Anyone* can encrypt

 *Only Alice* can decrypt

Anyone can write a message that only Alice can read.

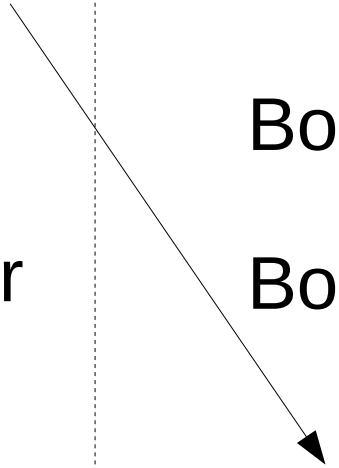
Examples: RSA, ElGamal, ECC

RSA

- First PKE (1977), widely used now in encryption
- Requires much longer keys (2048/4096 bits)
- Less efficient than SKE
- No “perfect security”; can be broken by quantum computers

PKE and SKE

	PKE	SKE
Key	Two: public/private	One: secret
Key setup	Share public key	Need safe channel
Encrypt	Anyone	Both participants
Decrypt	Only key generator	Both participants
Efficiency	Costly to encrypt/decrypt	Cheap



We can combine PKE and SKE to cover their weaknesses

Key Exchange (using PKE)

1. Alice generates a public/private key pair 
2. Alice shares the public encryption key 
3. Bob generates a secret key, 
   encrypts it with PKE  , and sends it to Alice
4. Alice decrypts the secret key, 
   and uses it for SKE from now on

What if the private key  *is leaked?*

In practice, the public/private key pair is **short-lived** to guranatee **forward secrecy**

Key Establishment (using Diffie-Hellman)

1. Alice and Bob use some g and prime p, where g generates integers modulo p
2. Alice generates and sends $g^A \bmod p$
3. Bob generates and sends $g^B \bmod p$
4. Alice and Bob compute secret key $g^{AB} \bmod p$
   Alice: $(g^B \bmod p)^A = g^{AB} \bmod p$
   Bob: $(g^A \bmod p)^B = g^{AB} \bmod p$

Other cryptographic tools

We may also want **integrity** and **authenticity**

- Confidentiality: The message is secret
- Integrity: The message is correct
- Authenticity: The sender/receiver's identity is correct

For this, we need other tools:

- Cryptographic hash
- Message Authentication Code (MAC)
- Digital signature

Cryptographic Hash

Cryptographic hashes are irreversible *one-way functions:*

MESSAGE —Hash→ b194 d920

Properties:

- Output is small, fixed size
- Different inputs may give same output
- Function is publicly known

Examples: MD5 (insecure!), SHA1, SHA2, SHA3

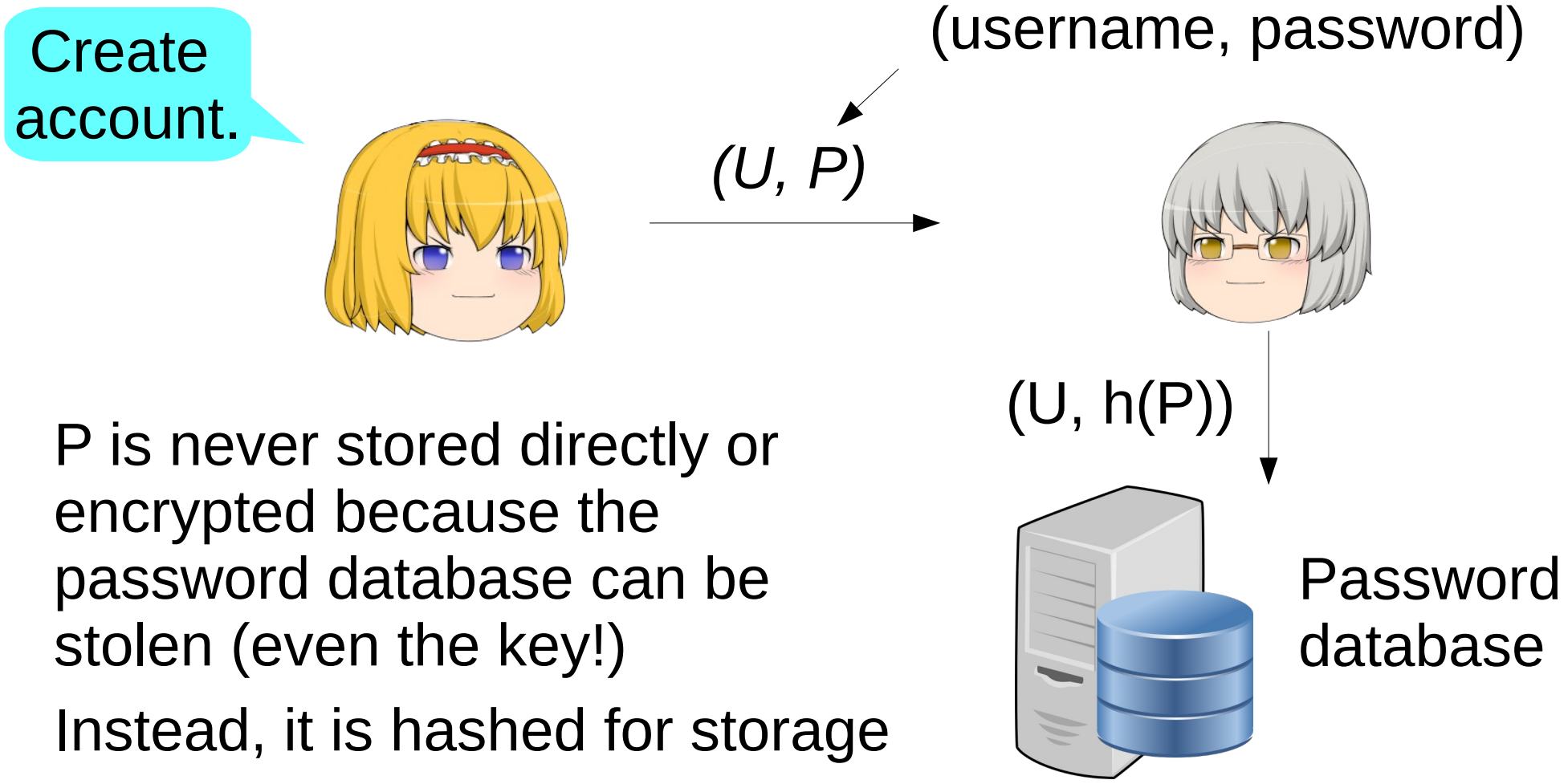
Cryptographic Hash

Cryptograhic hashes need to be difficult for the attacker to reverse or manipulate:

1) Given the output, it is hard to find an input hashing to that output

2) A small input change should produce an unpredictable output change

Cryptographic Hash Password Storage



P is never stored directly or encrypted because the password database can be stolen (even the key!)

Instead, it is hashed for storage

What if two users have the same password?

Cryptographic Hash Password Storage

Attacker can *precompute* a hash table:

Guess password	Hash
123456	h(123456)
abc	h(abc)
...	...

When hashed passwords are stolen, attacker simply has to do a matching exercise to “invert” the hash!

This is exactly like the ciphertext repetition problem

Cryptographic Hash Password Storage

Create account.

(username, password)

(U, P)

(U, h(P +S), S)

Add a random *Salt* to the password

Password database

Cryptographic Hash Verifying Integrity

I would like to download file M.

verify h(M)

M, h(M)

Sure, here you go.

Good against unintentional errors, random errors

What about a malicious MITM attacker?

Message Authentication Code

A MAC is attached to messages for authentication:

- The two parties both need to have the secret key (like SKE)
- An attacker cannot “forge” a MAC
- **Authenticates** the message
- Can be built from a hash (this is called HMAC):

$$h(K||M)$$

Message Authentication Code

Alice sends M, h(K||M) to Bob

Verification: Bob, using his key, verifies h(K||M) is correct.

Resistance against MITM: Mallory, who does not have the key, cannot produce the HMAC. Specifically, if Mallory changes M to M', he cannot also replace h(K||M) with h(K||M'). If he attempts to change any part of the message, Bob's verification will fail.

Signatures

What if we reversed the roles of  and  ?

~~"Encrypting"~~ would be limited but everyone could ~~"decrypt"~~

Signing verify

 Private signing key: signs the message

 Public verification key: verifies the message

Achieves authentication if you know the correct public verification key

In practice, sign/verify keys are long-lasting while encrypt/decrypt keys are short-lived

Blockchain

Problem: We want a distributed transaction ledger so that **no single entity has control over the ledger**

- Global transaction ledger:
  - Contains all transactions between all participants
  - Distributed to all participants (=> open)
  - Can be sent by anyone, and therefore requires some verifiability
- Solution: Use cryptographic techniques so that participants can verify the ledger

Global Transaction Ledger

Sender	Receiver	Bitcoin amount	
Alice	Bob	0.31	Block 1
Carol	Bob	1.21	
Alice	Bob	0.4	
Bob	Alice	0.532	Block 2
Carol	Bob	0.01	Block 4
Bob	Carol	0.01	
...	...	...	
...	...	…	

- Everyone agrees on the same ledger
  - Update each other on new transactions
- Divided into blocks, 1 block every 10 minutes
- Can someone lie?
- e.g. “Alice: Block 1 has no transactions!”

Global Transaction Ledger

Three types of threats against the ledger's integrity:

Add: People refuse to add a real transaction to the ledger

Modify: Someone modifies a transaction in the ledger

- This is easiest to fix: Simply have the participants sign all transactions, and include the signature in the ledger

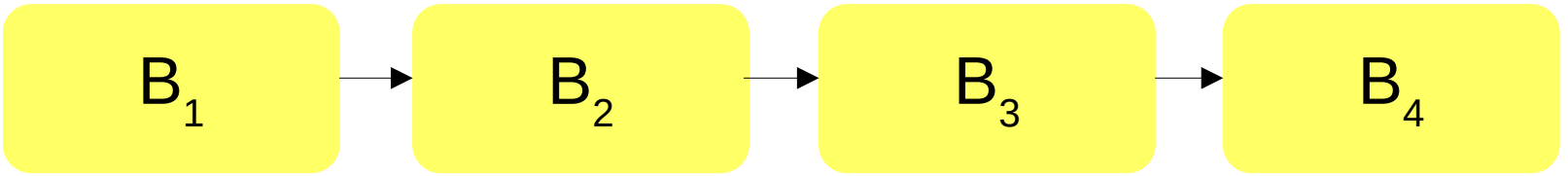
Delete: Someone removes a transaction from the ledger

- To prevent those, we use a *proof of work* to safeguard blocks

Two types of participants in a blockchain system:

- Users: Signs their transactions and announces them
- Miners: Helps add the transactions to the ledger by generating the proof of work (technically, miners can also be users)

Blockchain



$B_2 = <T_2, h(B_1), P_2(h(B_1))>$

$B_3 = <T_3, h(B_2), P_3(h(B_2))>$

$B_4 = <T_4, h(B_3), P_4(h(B_3))>$

Transactions in this block

Proof of work, based on hash and very hard to compute

Attacker claim: actually, B_1 should be B_1'

- The attacker needs to change B_1, which changes B_2, which...
- The attacker needs to generate 3 new proofs of work, because the network accepts the longest chain
- Trying to generate those essentially triggers a race: other miners are generating P_5, P_6, P_7...

Proof of Work

- Challenge: Given C, how can we find P such that

h(C||P) *ends with 32 zero bits?*

- Cryptographic hash: Best way = Brute force (one success per 2^{32} hashes on average)
- C is the Content and P is the Proof of Work: If Alice sends C to Miner and Miner sends P back to Alice, Miner has “proven” that they did the work of many hashes

Public Key Infrastructure

Hello! I am Alice. Here is my signature!

Sign  *(h(M))*

You can verify it with this public verification key.

How can you trust Alice?

Public Key Infrastructure

Delivering the right public verification key to users

We will examine PKI in three technologies:

- SSH tunneling
- PGP
- SSL/TLS

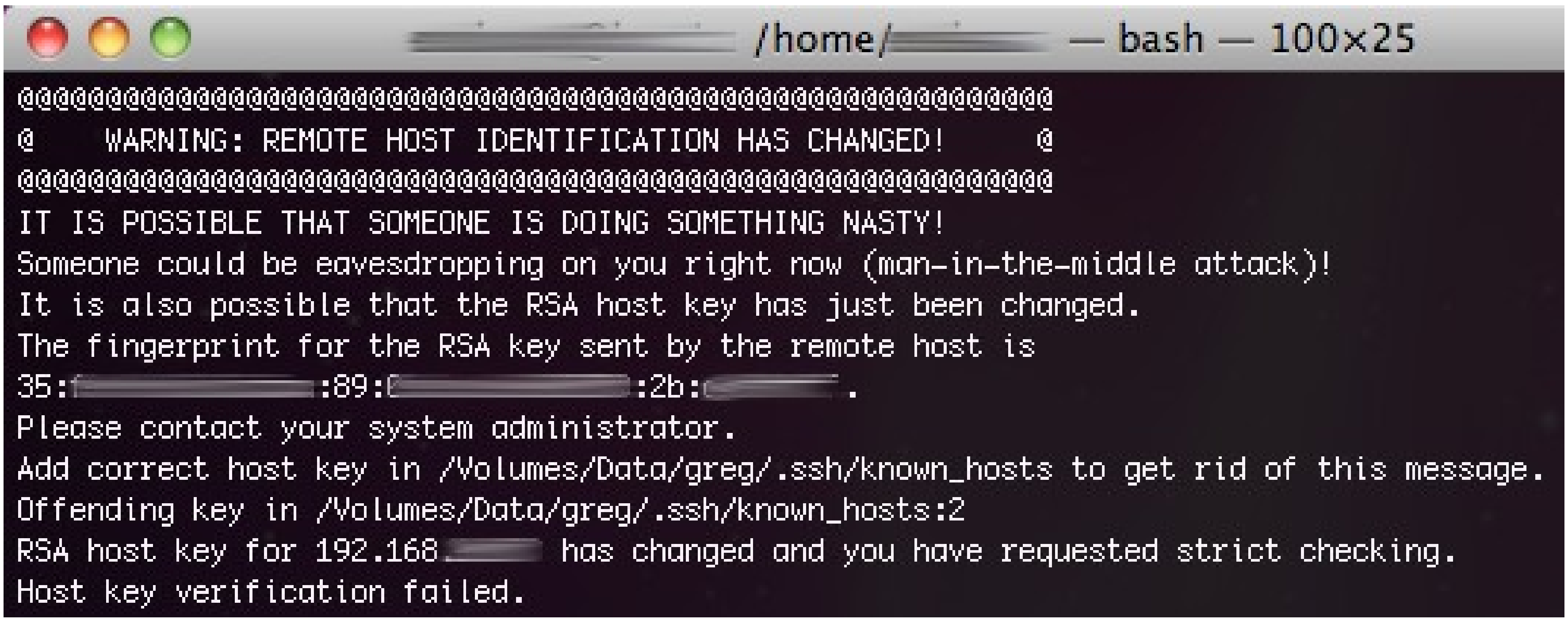
SSH tunneling

Used for connecting to remote machine

TOFU (Trust On First Use):

- When connecting for the first time, the server shows the public key
- You are asked if you trust the public key (yes/no)
- If “yes”, you will not be asked again unless the key changes
- If “no”, you will be disconnected

SSH tunneling



PGP

Used in e-mails

Pretty Good Privacy

- Developed in 1991
- Needs setup
- Used by some professionals, privacy-sensitive circles

PGP

Used in e-mails

Web of Trust:

- Trust is transitive
- Alice can trust Bob directly (like TOFU)
- Alice can trust Carol indirectly – if Alice trusts Bob, and Bob trusts Carol
- Bob signs Carol's key, and Alice verifies Bob's signature

SSL/TLS

Used in HTTP

- Most widely used crypto-technology
- First appeared in Netscape for e-commerce
- Used by default in (increasingly) many websites
- Uses almost all of the tools in this module
- Versions: SSL1, SSL2, SSL3, TLS1.0, TLS1.1, TLS1.2, TLS1.3
- Current trend: removing bad encryption

SSL/TLS

Certificate system:

- By default, browsers will trust a set of **Ceritifcate Authorities** (CA)
- CA can sign any website's public key; the CA's signature is called a certificate
- The website presents its certificate when you connect to it
- Certificates can also be transitive

SSL/TLS

A basic connection uses most of this module's tools.

Key:

Root CA's public verification key

Root CA's private signature key


Web server's public verification key

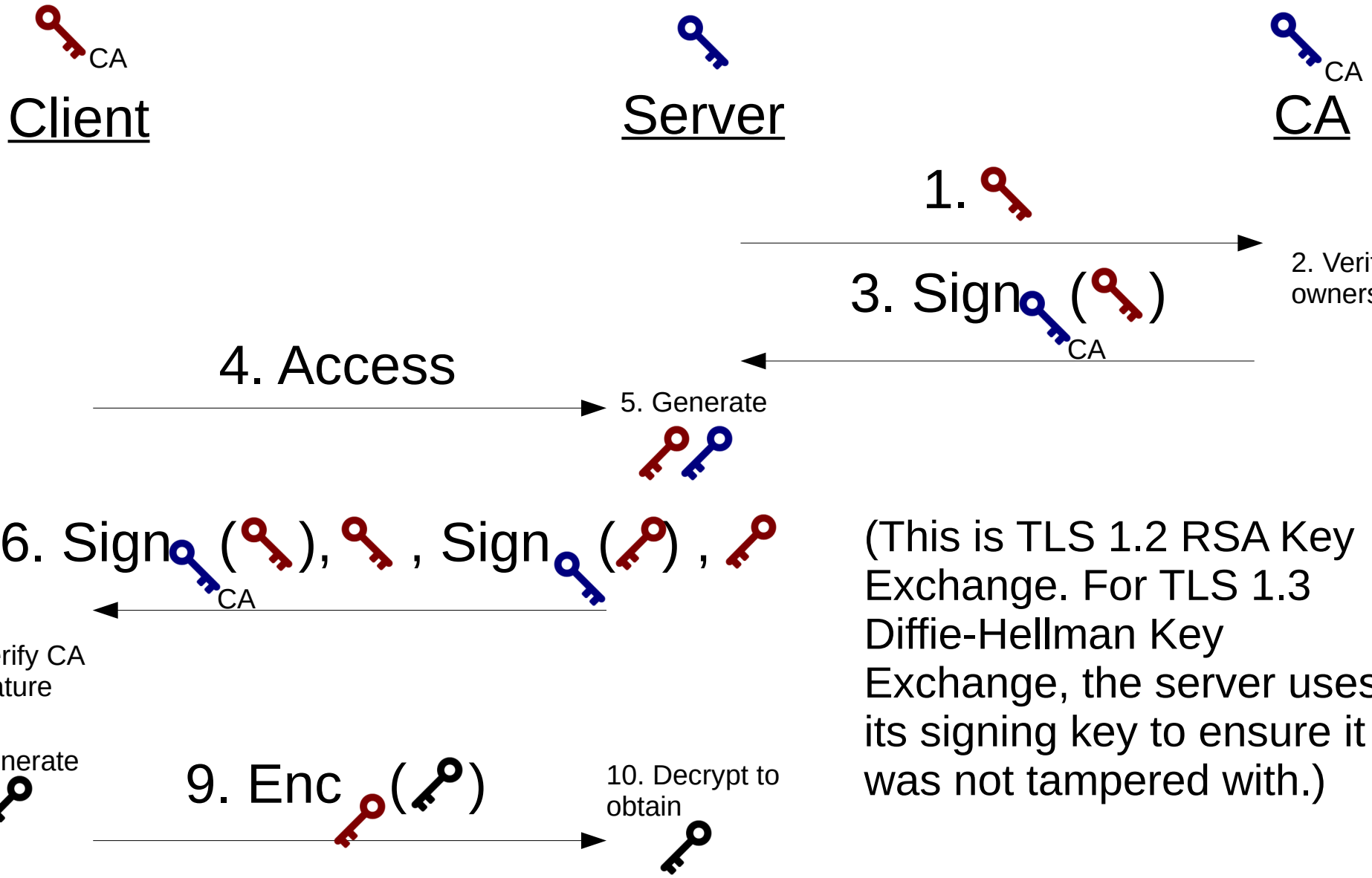

Web server's private signature key


Web server's public encryption key


Web server's private decryption key


Secret key negotiated between client and web server

SSL/TLS



(This is TLS 1.2 RSA Key Exchange. For TLS 1.3 Diffie-Hellman Key Exchange, the server uses its signing key to ensure it was not tampered with.)

SSL/TLS

1. Server sends its public verification key to the root CA.
2. Root CA checks that person really owns the web server.
3. Root CA signs the web server's public verification key and sends it back (the cert).

(After some time)

4. The client accesses the web server.
5. Server generates an ephemeral PKE key pair.
6. Server sends the cert to client, along with both public keys and a signed version of the public encryption key to avoid tampering.
7. Client checks signature on cert to verify the server's public verification key, then uses that to verify the server's public encryption key.
8. Client generates secret key.
9. Client encrypts secret key with server's public encryption key and sends it to server.
10. Server decrypts to obtain secret key.

From this point onward all communication will use that secret key (most likely 128-bit AES CBC with SHA-256 for HMAC).

Attacks on Cryptosystems

Cryptanalysis

Find mathematical weaknesses in cryptography

For example:

- DES key length is too short
- RC4 has correlations between keystream and key
  - Initially only some initial byte correlations were found, so the patch was to discard initial keystream bytes
- MD5, SHA-1 are vulnerable to a “collision attack”
  - This is a problem for hash-signatures

Attacks on Cryptosystems

Root CA compromise:

- DigiNotar, dutch root CA (2011)
  - Issued fake certs for google.com
  - Breach was hidden
  - Web browsers removed DigiNotar as root CA
- Comodo (2011)
  - Issued fake certs for google, yahoo, etc.
  - Certificates were immediately revoked
- Kazakhstan’s government issues all certificates (i.e. can read/intercept all HTTPS)

Attacks on Cryptosystems

BEAST attack (Duong and Rizzo, 2011)

- If an attacker can ask Alice to encrypt a plaintext block, then an attacker can make a guess on any ciphertext block and determine if it is correct
- Suppose the attacker wants to decrypt bytes {b_1, b_2, b_3...}
  - The attacker asks Alice to encrypt {r_1, r_2, ..., r_{15}, b_1, b_2, ...}
    - 16 bytes = 1 block, so the only unknown byte is b_1
  - This allows the attacker to guess byte b_1 after 256 tries
  - Next, the attacker asks Alice to encrypt {r_1, r_2, ..., r_{14}, b_1, b_2, ...}
  - Practically, this could be done with browser plugins to create POST requests

Attacks on Cryptosystems

Logjam (Adrian et al., 2015)

- For a fixed *prime* p, it is practically possible to perform enough pre-computation to make discrete log easy
  - For 512-bit p: About one week of pre-computation, then about one minute for any discrete log with that prime
  - Discrete log in general is still a hard problem
- Many implementations were using the same *p* – including almost all Apache servers (82% of traffic)

Attacks on Cryptosystems (WEP)

WEP: Uses a stream cipher (RC4)

A number of critical weaknesses

1) Short IV:

- Original design called for 24-bit IV
- IV is generated randomly for each message
- How long does it take to find two messages with the same key and IV?

Attacks on Cryptosystems (WEP)

2) RC4 issues

- Fluhrer, Martin, and Shamir discovered issues in RC4's initial randomization (key scheduling)
- Given a byte of the keystream, and if the IV satisfies a certain format, you can derive part of the key
- How can you get a byte of the keystream?

Attacks on Cryptosystems (WEP)

3) Bad integrity check:

- Integrity check is a checksum...
- You can inject a message into the channel
  - e.g. tricks the user into thinking the message came from the router
  - Authentication is done by asking the user to encrypt a message
- How do you generate a plausible ciphertext for a given plaintext?

Recap

SKE	is efficient and hard to break cryptographically
	but it needs a shared key
PKE	can be used to share the SKE key
	but text and public key are not authenticated
MAC	can authenticate text
	but it needs a shared key
PKI	can authenticate public key
TOFU *Web of Trust* *PKI*	*but they each have their own problems*