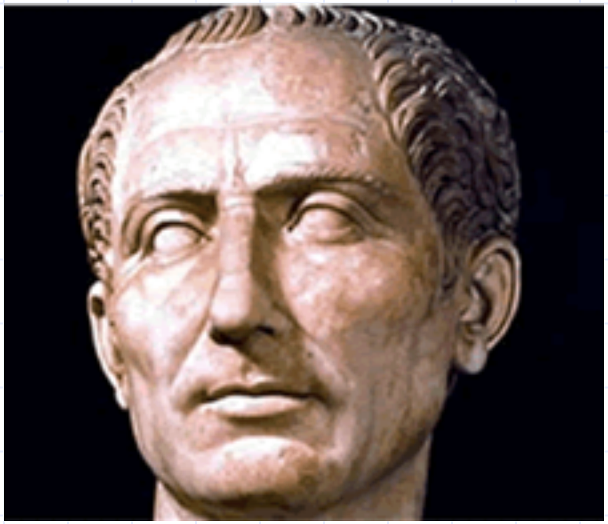
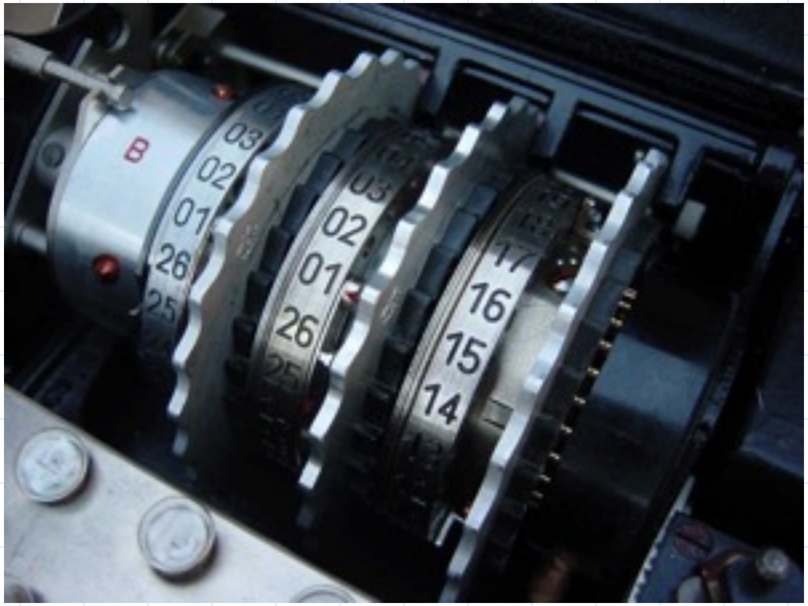
Cryptography Overview

John Mitchell

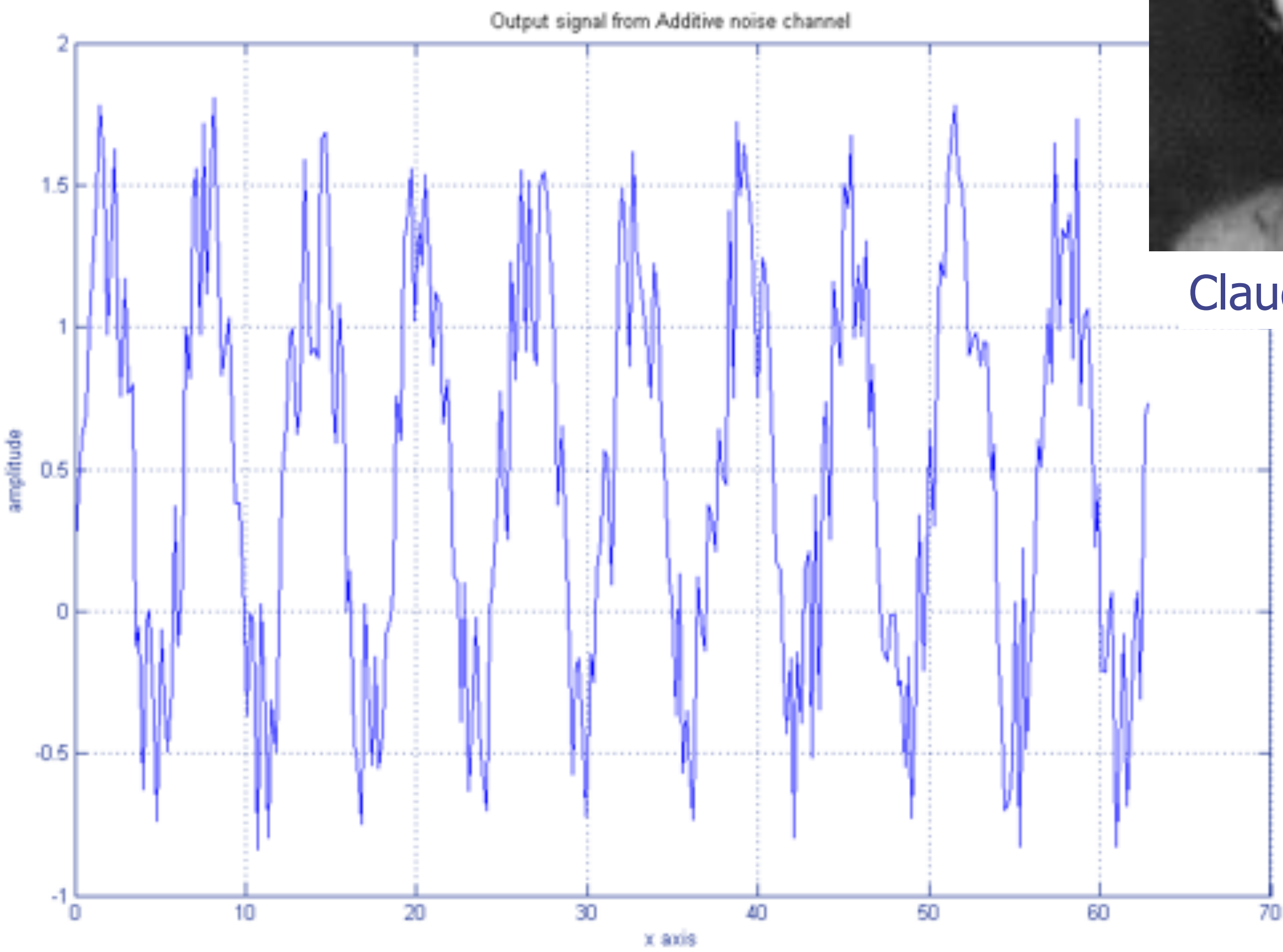
Caesar cipher

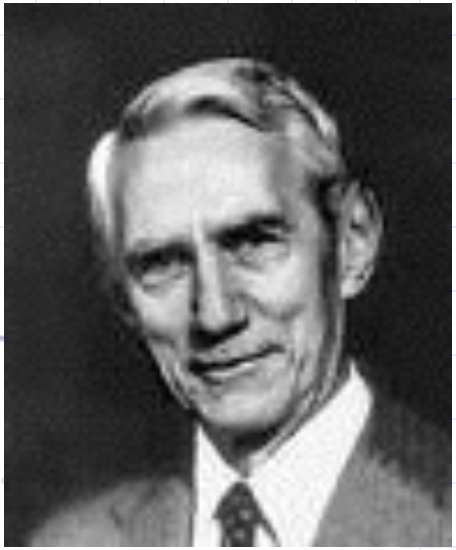


German Enigma



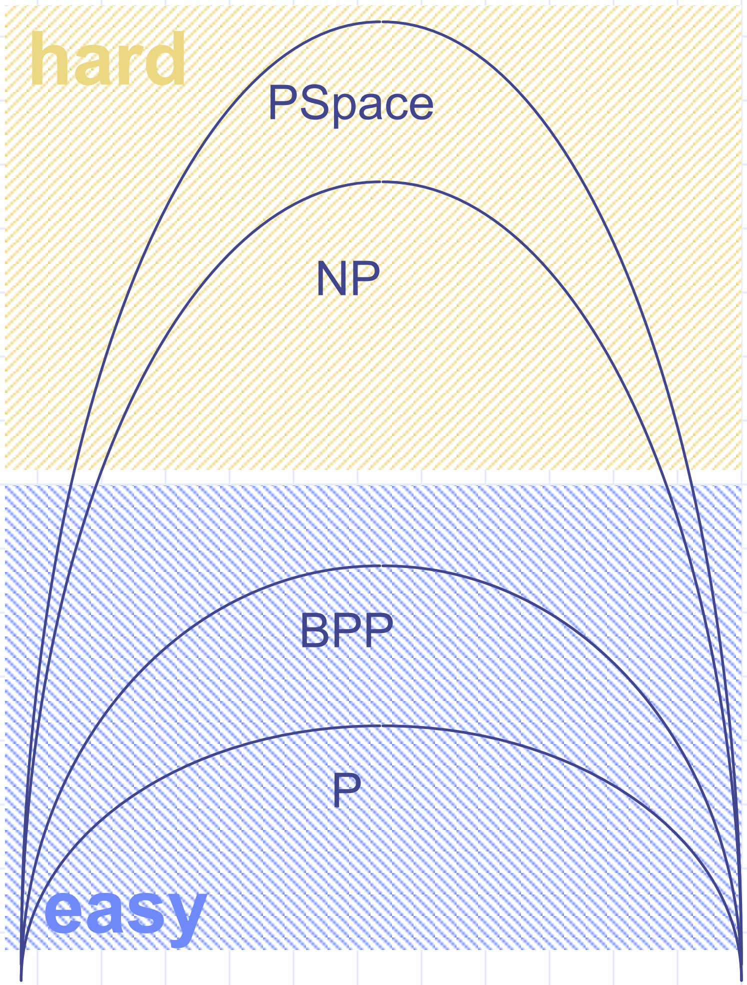
Information theory

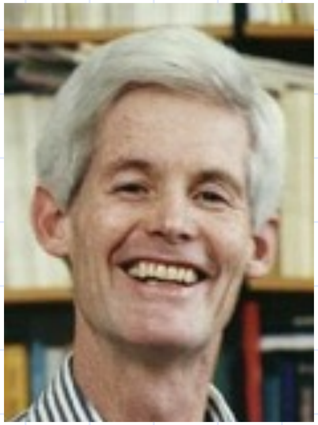




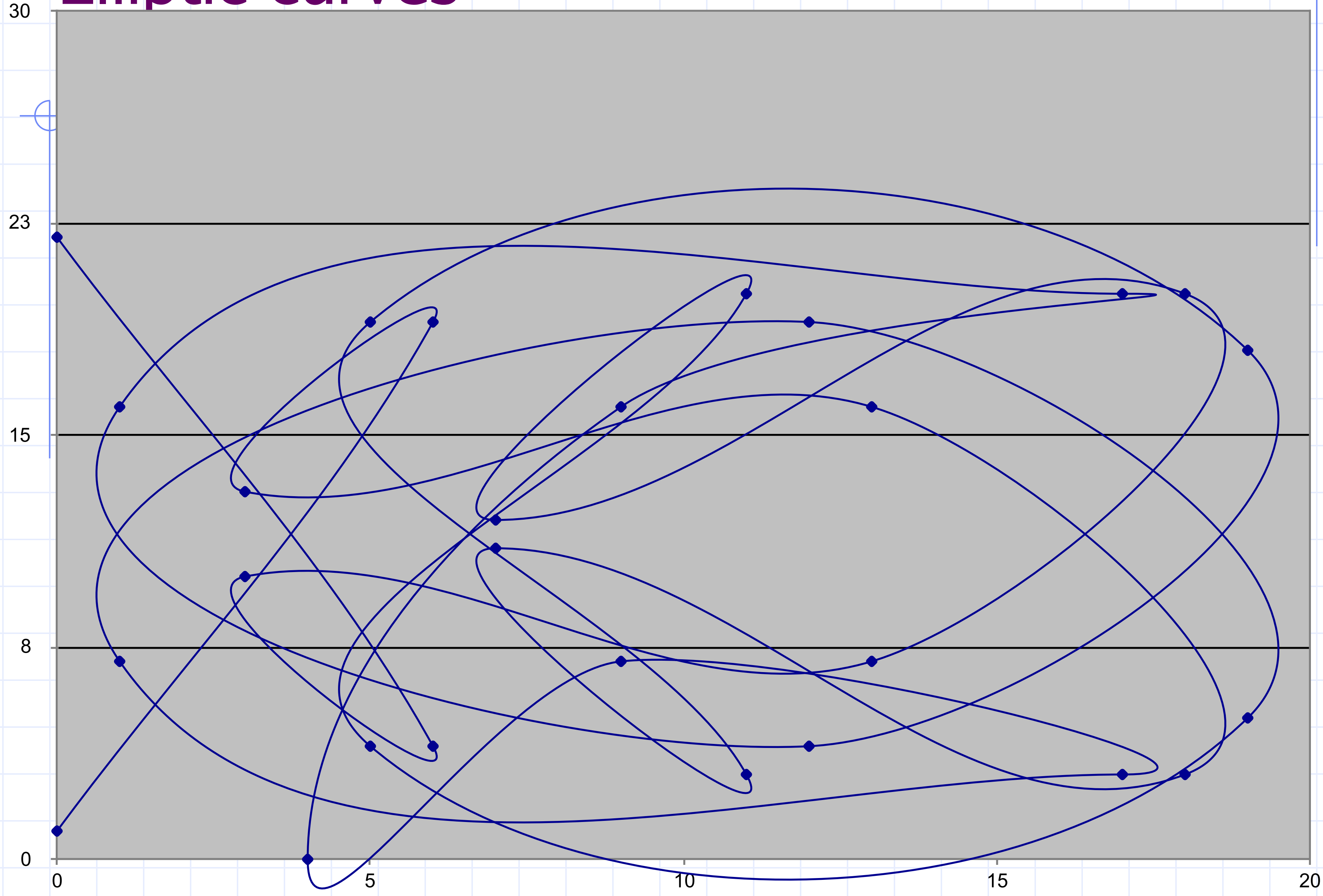
Claude Shannon

Complexity theory





Elliptic curves



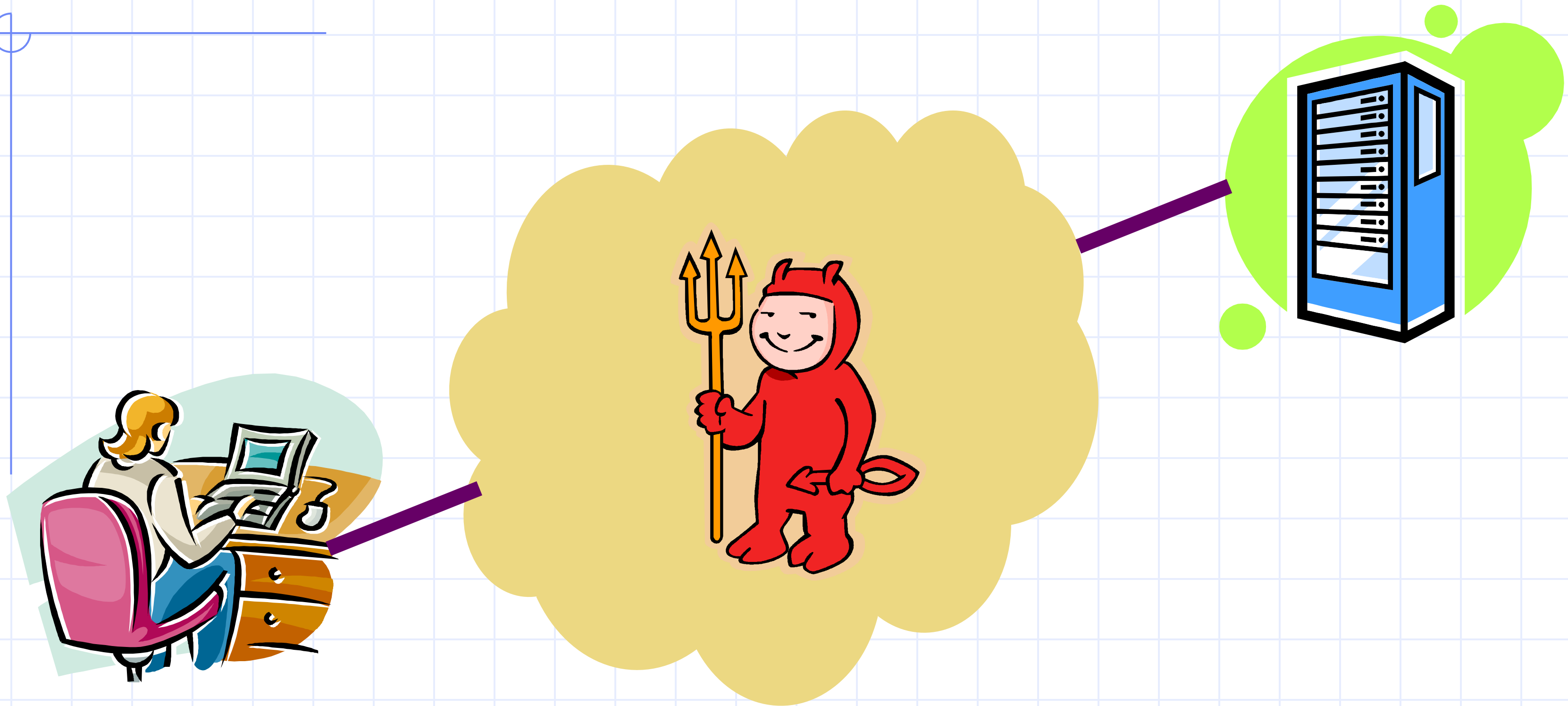
Cryptography

- Is
  - A tremendous tool
  - The basis for many security mechanisms
- Is not
  - The solution to all security problems
  - Reliable unless implemented properly
  - Reliable unless used properly
  - Something you should try to invent yourself unless
    - you spend a lot of time becoming an expert
    - you subject your design to outside review

Basic Cryptographic Concepts

- Encryption scheme:
  - functions to encrypt, decrypt data
- Symmetric encryption
  - Block, stream ciphers
- Hash function, MAC
  - Map any input to short hash; ideally, no collisions
  - MAC (keyed hash) used for message integrity
- Public-key cryptography
  - PK encryption: public key does not reveal key^{-1}
  - Signatures: sign data, verify signature

Example: network transactions

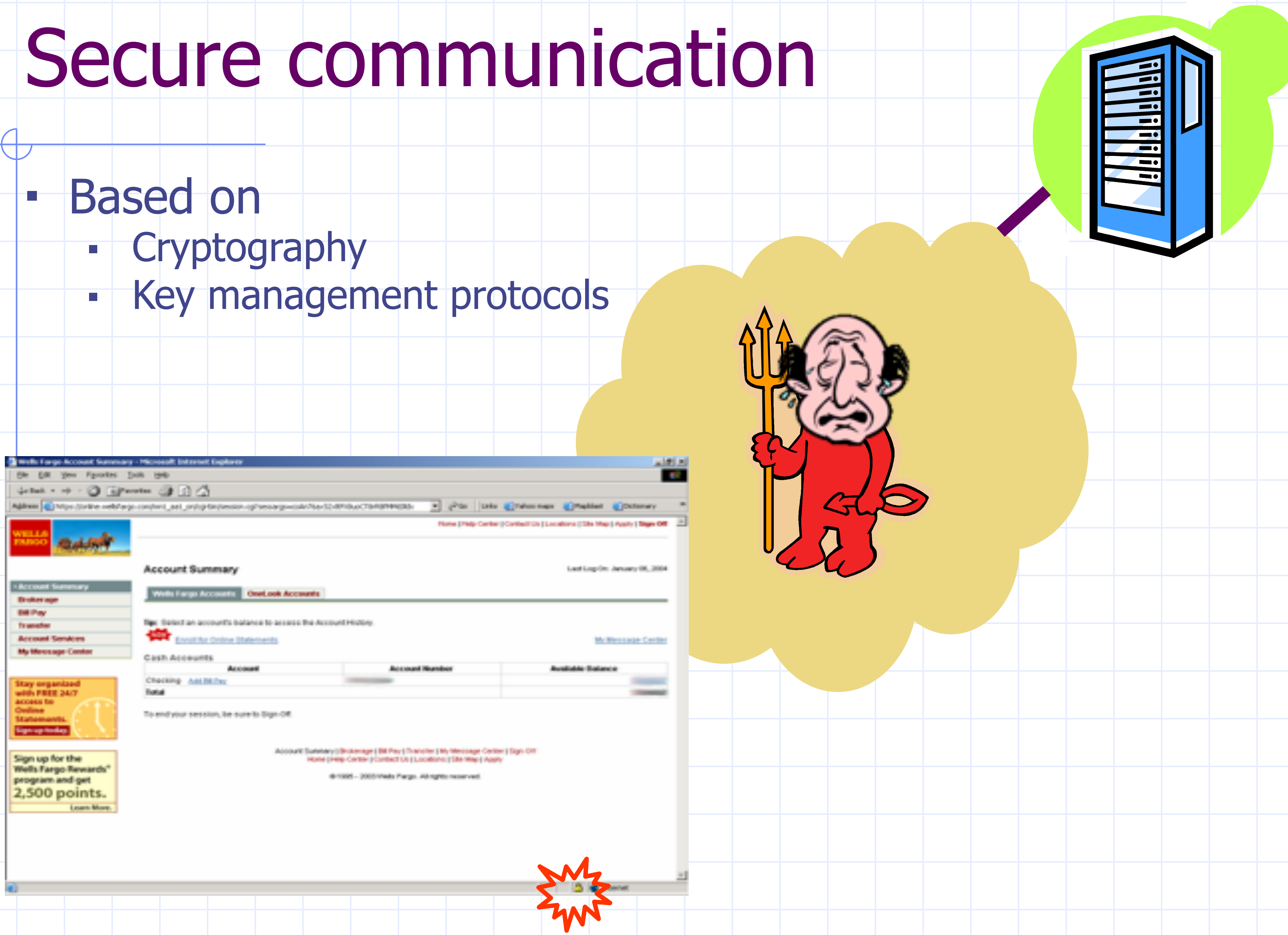


Assume attackers can control the network

- We will talk about how they do this in a few weeks

Secure communication

- Based on
  - Cryptography
  - Key management protocols



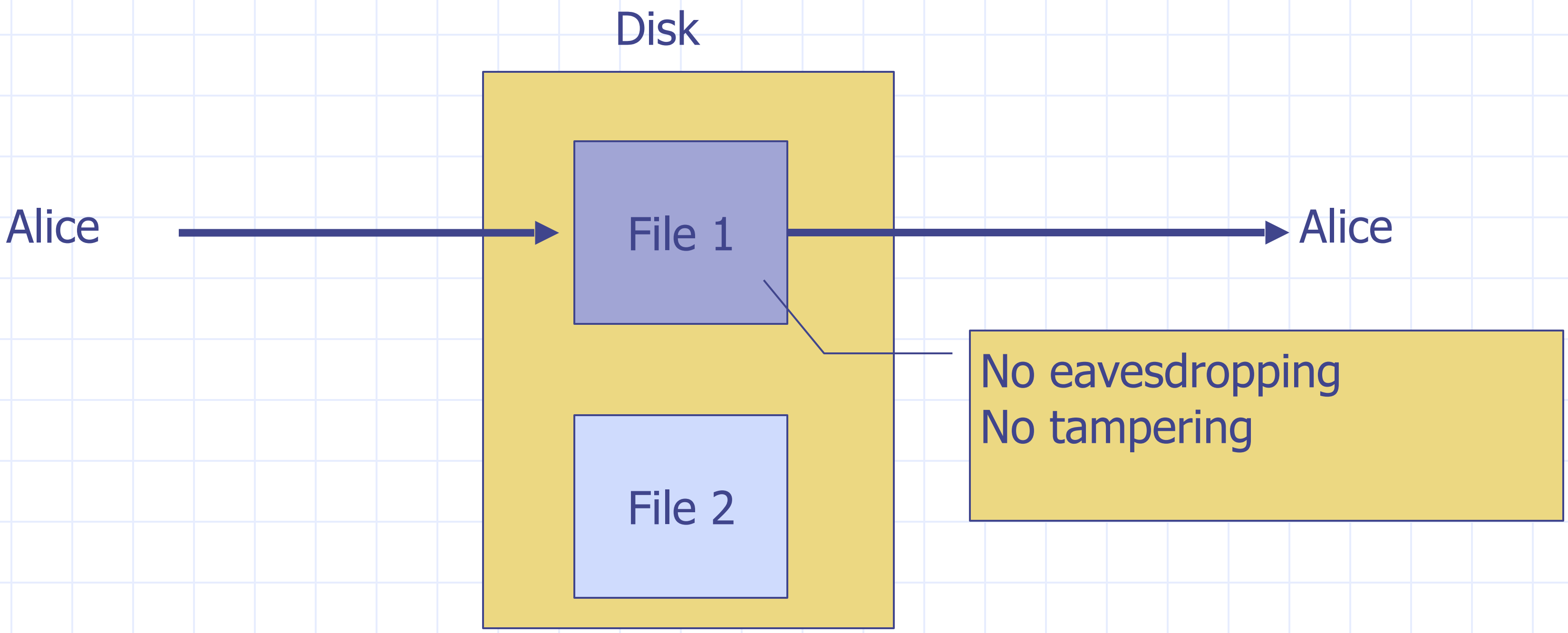
Secure Sockets Layer / TLS

- Standard for Internet security
  - Originally designed by Netscape
  - Goal: "... provide privacy and reliability between two communicating applications"
- Two main parts
  - Handshake Protocol
    - Establish shared secret key using public-key cryptography
    - Signed certificates for authentication
  - Record Layer
    - Transmit data using negotiated key, encryption function

SSL/TLS Cryptography

- Public-key encryption
  - Key chosen secretly (handshake protocol)
  - Key material sent encrypted with public key
- Symmetric encryption
  - Shared (secret) key encryption of data packets
- Signature-based authentication
  - Client can check signed server certificate
  - And vice-versa, if client certificates used
- Hash for integrity
  - Client, server check hash of sequence of messages
  - MAC used in data packets (record protocol)

Goal 2: protected files

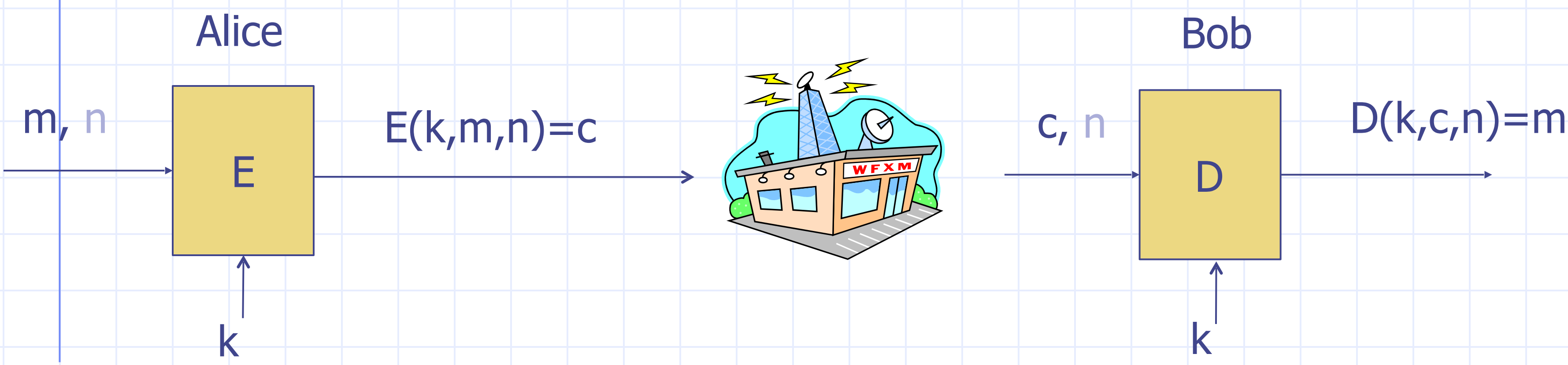


Analogous to secure communication:

Alice today sends a message to Alice tomorrow

Symmetric Cryptography

Symmetric encryption



E, D: cipher   k: secret key (e.g., 128 bits)

m, c: plaintext, ciphertext   n: nonce (aka IV)

Encryption algorithm is publicly known

- Never use a proprietary cipher

First example: One Time Pad (single use key)

- **Vernam** (1917)

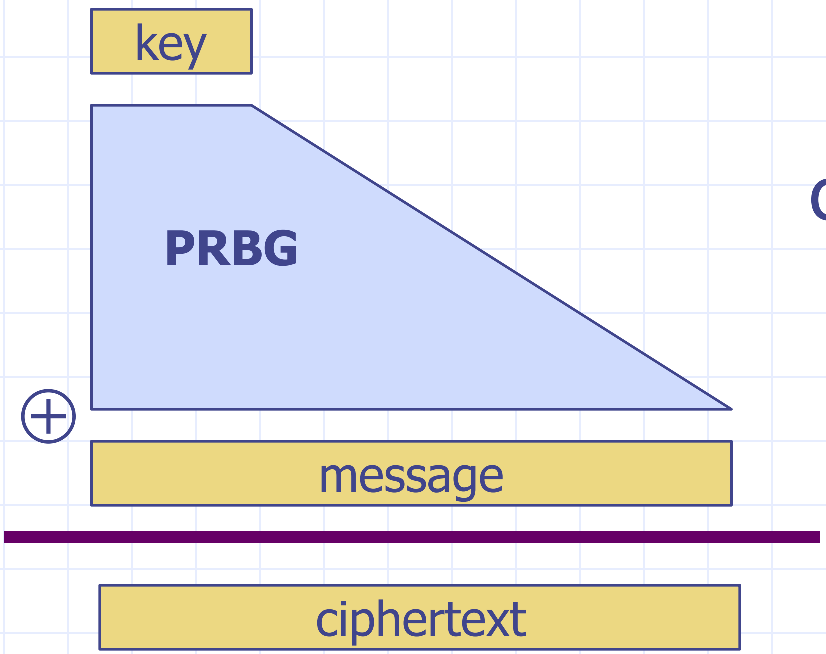
Key:	0	1	0	1	1	1	0	0	1	0	$\oplus$
Plaintext:	1	1	0	0	0	1	1	0	0	0	
Ciphertext:	1	0	0	1	1	0	1	0	1	0	

- **Shannon '49:**
  - OTP is "secure" against ciphertext-only attacks

Stream ciphers (single use key)

Problem: OTP key is as long the message

Solution: Pseudo random key -- stream ciphers



$c \leftarrow \text{PRBG}(k) \oplus m$

Stream ciphers: RC4 (113MB/sec) , SEAL (293MB/sec)

Dangers in using stream ciphers

One time key !!   "Two time pad" is insecure:

$$\begin{cases} C_1 \leftarrow m_1 \oplus \text{PRBG}(k) \\ C_2 \leftarrow m_2 \oplus \text{PRBG}(k) \end{cases}$$

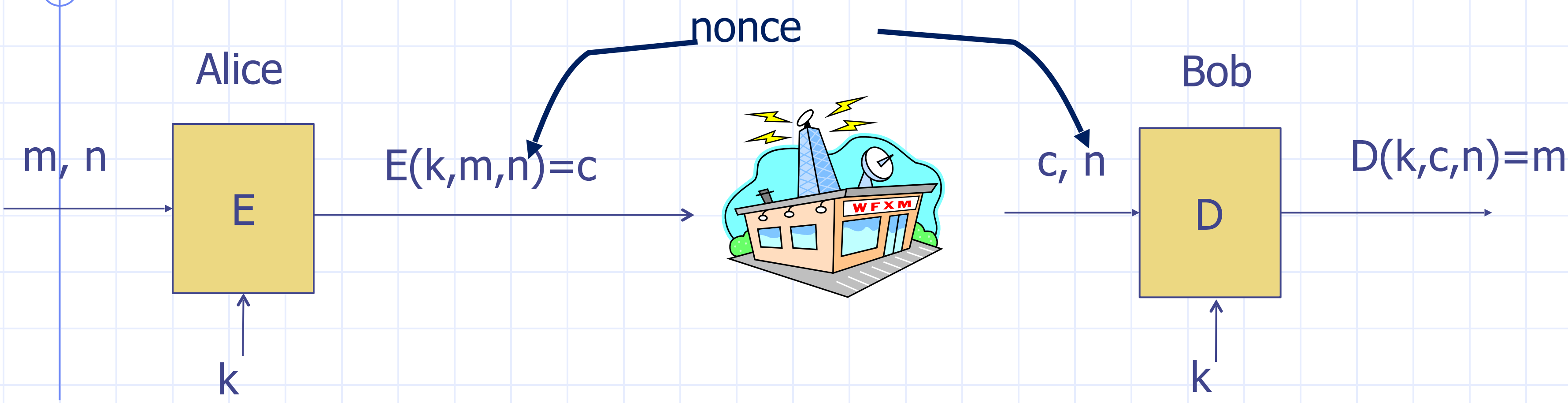
Eavesdropper does:

$$C_1 \oplus C_2 \quad \rightarrow \quad m_1 \oplus m_2$$

Enough redundant information in English that:

$$m_1 \oplus m_2 \rightarrow \quad m_1 , \; m_2$$

Symmetric encryption: nonce (IV)



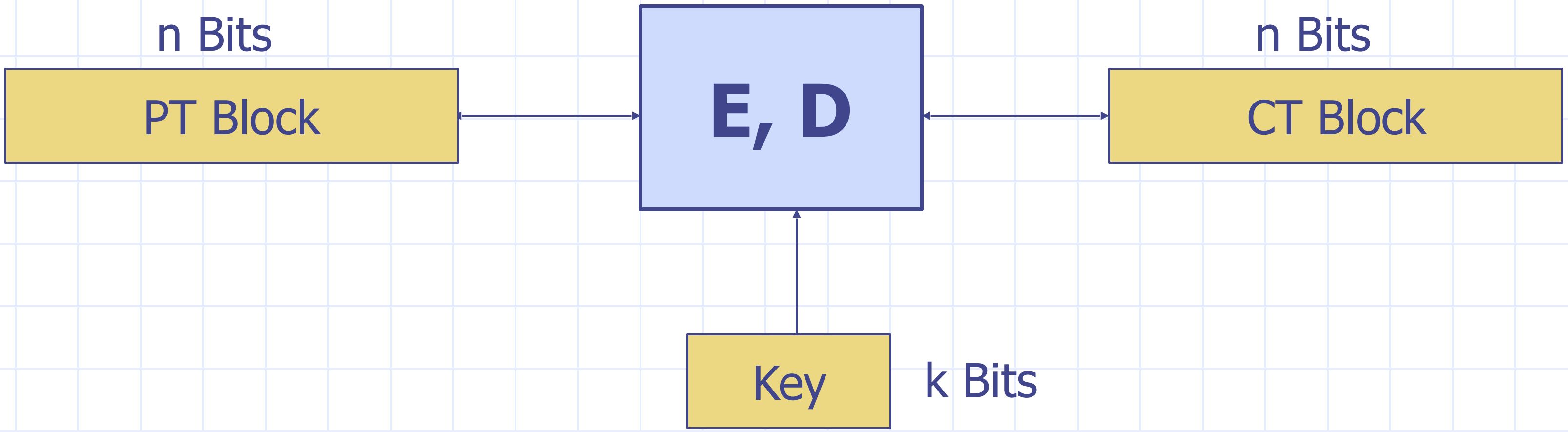
E, D: cipher k: secret key (e.g., 128 bits)

m, c: plaintext, ciphertext n: nonce (aka IV)

Use Cases

- Single use key: (one time key)
  - Key is only used to encrypt one message
    - encrypted email: new key generated for every email
  - No need for nonce (set to 0)
- Multi use key:
  - Key used to encrypt multiple messages
    - SSL: same key used to encrypt many packets
  - Need either unique nonce or random nonce
- Multi use key, but all plaintexts are distinct:
  - Can eliminate nonce (use 0) using special mode (SIV)

Block ciphers: crypto work horse



Canonical examples:

1. 3DES: n= 64 bits, k = 168 bits
2. AES: n=128 bits, k = 128, 192, 256 bits

IV handled as part of PT block

Building a block cipher

Input: (m, k)

Repeat simple mixing operation several times

- DES: Repeat 16 times:

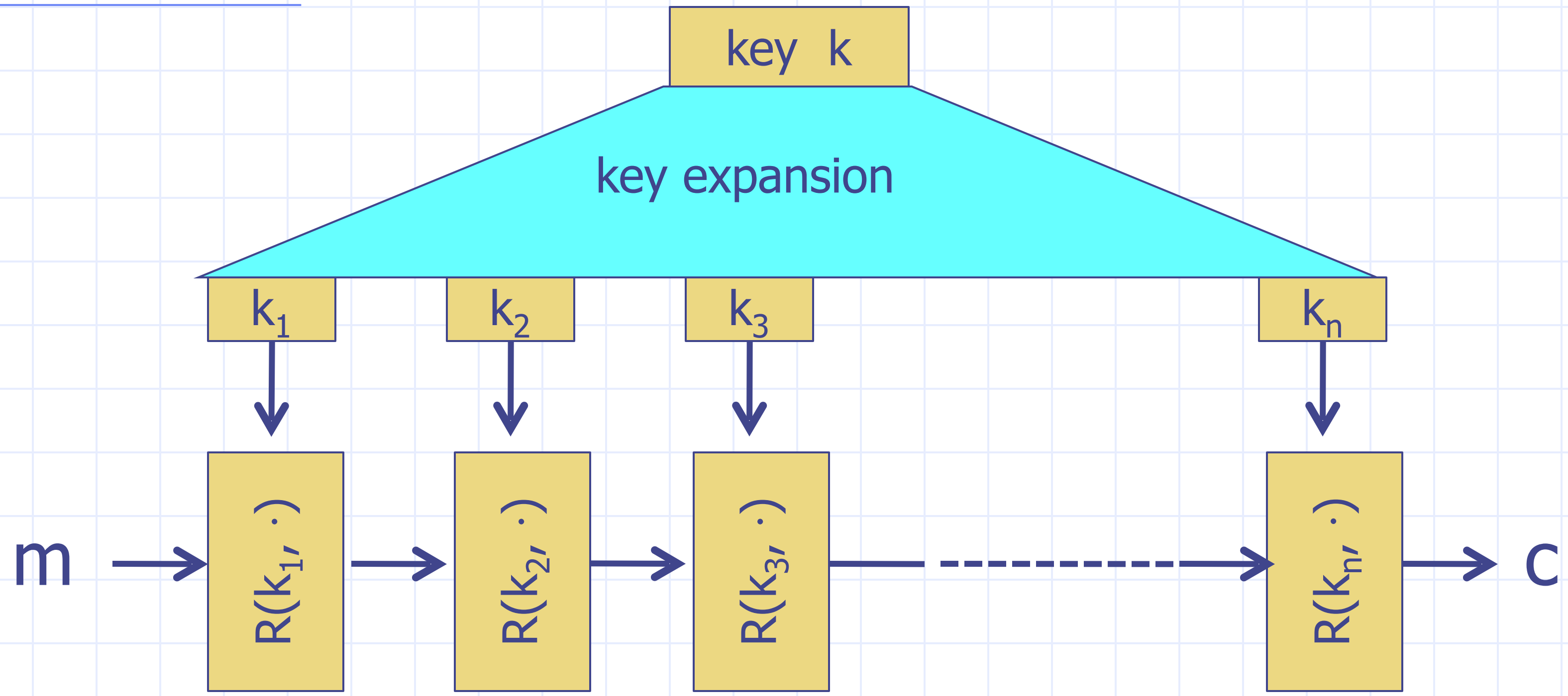
$$\begin{cases} m_L \leftarrow m_R \\ m_R \leftarrow m_L \oplus F(k, m_R) \end{cases}$$

- AES-128: Mixing step repeated 10 times

Difficult to design: must resist subtle attacks

- differential attacks, linear attacks, brute-force, …

Block Ciphers Built by Iteration

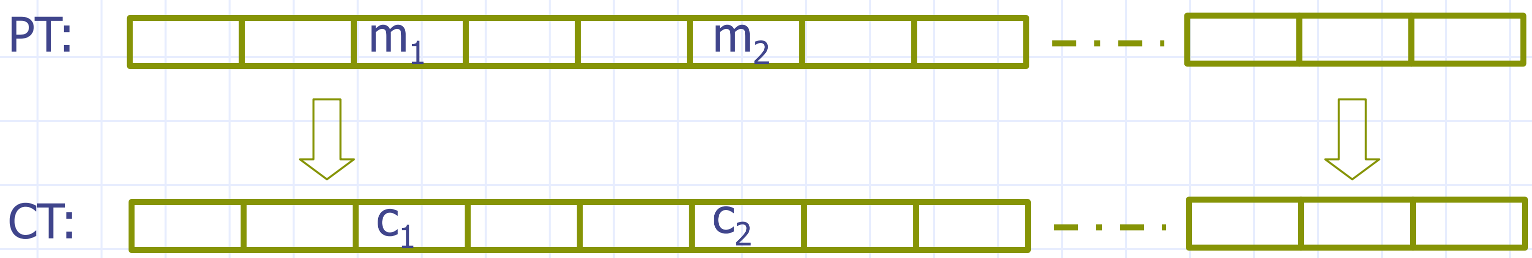


R(k,m): round function

for DES (n=16), for AES (n=10)

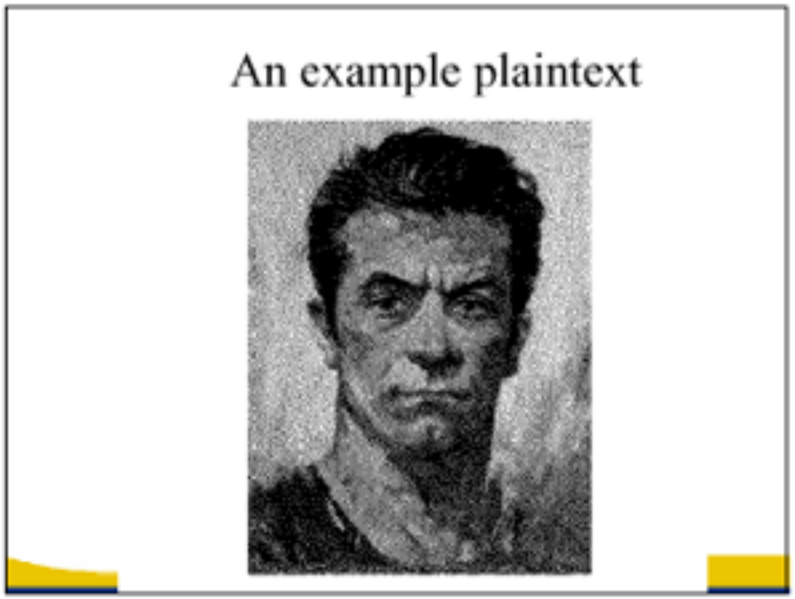
Incorrect use of block ciphers

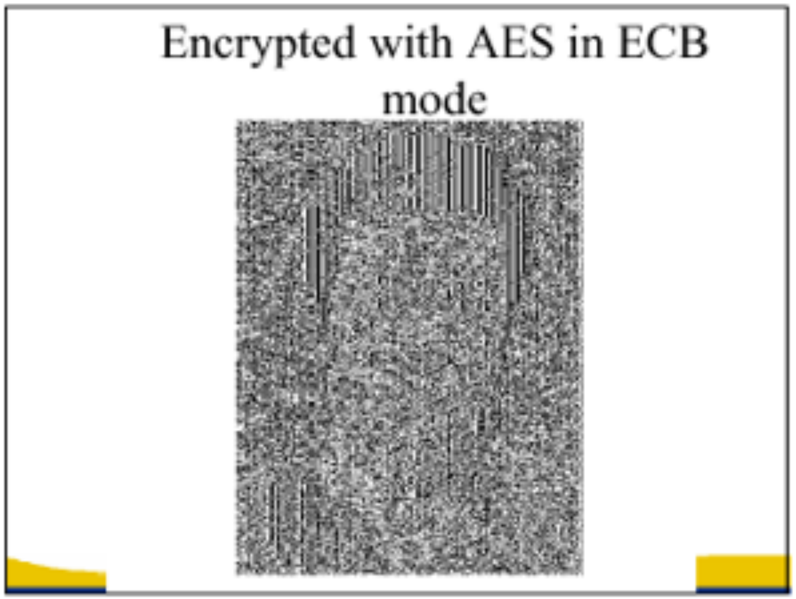
- Electronic Code Book (ECB):



- Problem:
  - if $m_1=m_2$ then $c_1=c_2$

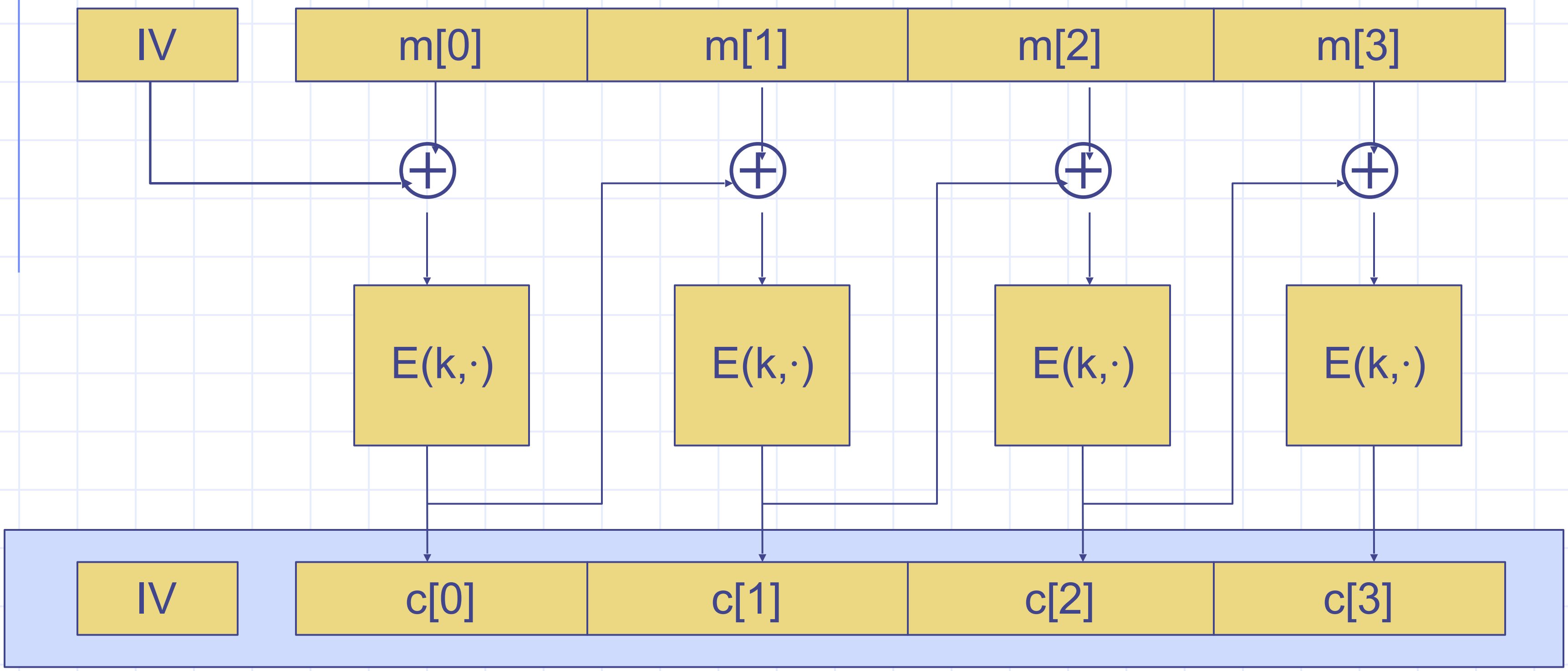
In pictures





Correct use of block ciphers I: CBC mode

E a secure PRP. Cipher Block Chaining with IV:



ciphertext

Q: how to do decryption?

Use cases: how to choose an IV

Single use key: no IV needed (IV=0)

Multi use key: (CPA Security)

Best: use a fresh <u>random</u> IV for every message (IV ⇽ X)

Can use <u>unique</u> IV (e.g counter) [Bitlocker]

but then first step in CBC <u>must be</u> IV′ ⇽ E(k,IV)

benefit: may save transmitting IV with ciphertext

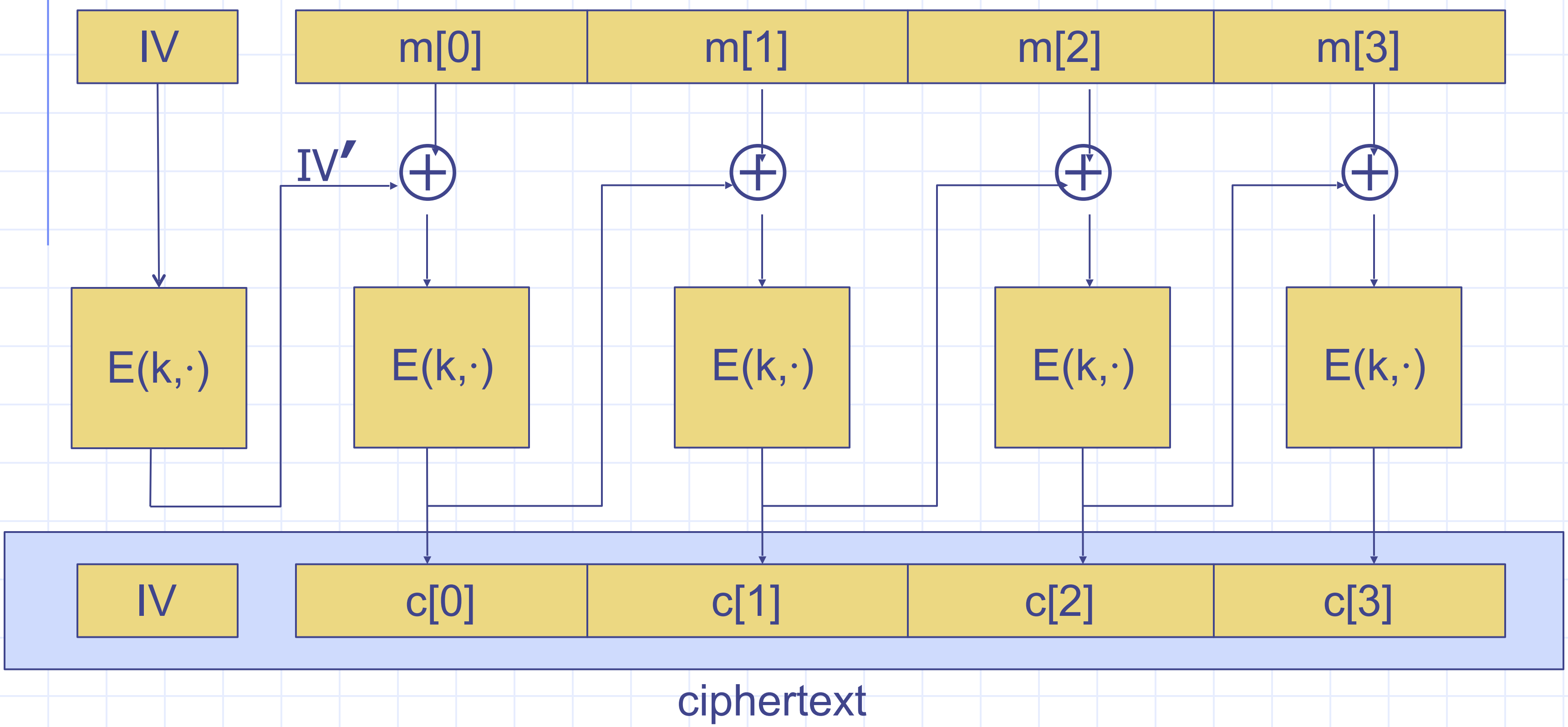
Multi-use key, but unique messages

SIV: eliminate IV by setting IV ⇽ F(k′, PT)

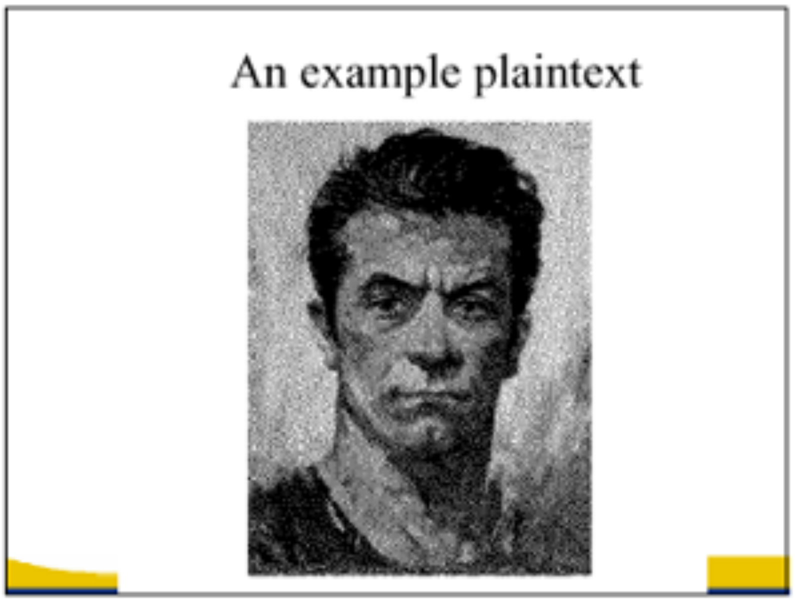
F: secure PRF with key k′

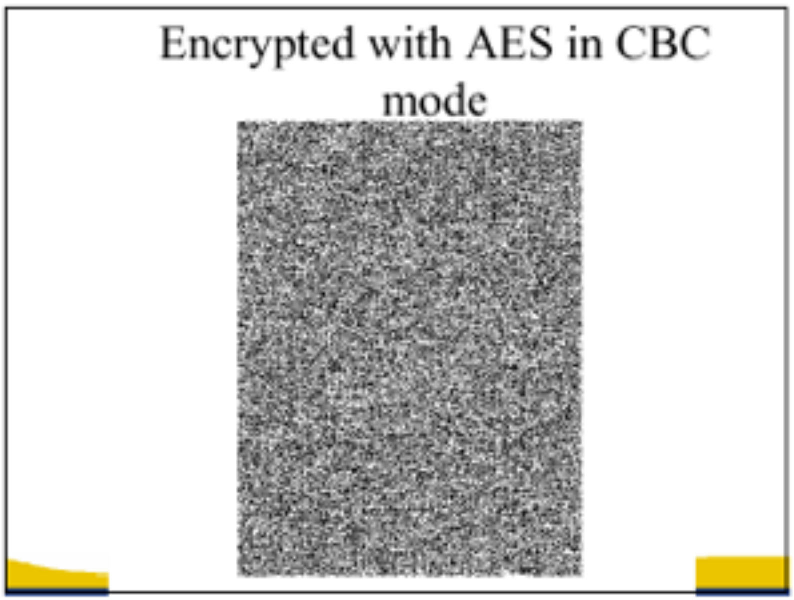
CBC with Unique IVs

unique IV means: (k,IV) pair is used for only one message
may be predictable so use E(k,·) as PRF



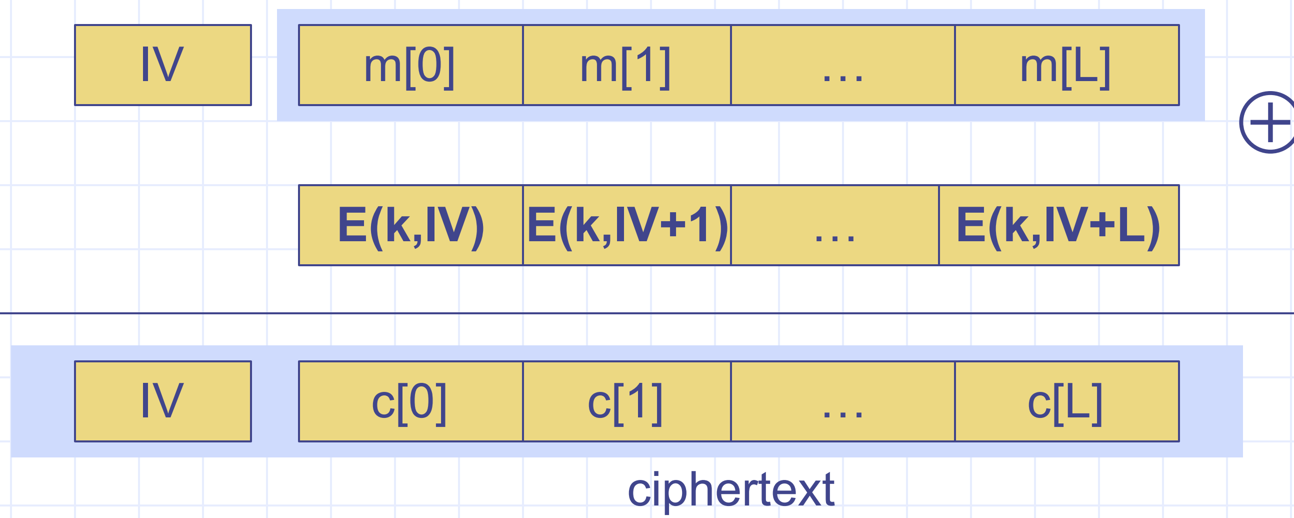
In pictures





Correct use of block ciphers II: CTR mode

Counter mode with a random IV: (parallel encryption)



- Why are these modes secure? not today.

Performance: Crypto++ 5.2.1 [Wei Dai]

Pentium 4, 2.1 GHz (on Windows XP SP1, Visual C++ 2003)

Cipher	Block/key size	Speed (MB/sec)
RC4		113
SEAL		293
3DES	64/168	9
AES	128/128	61
IDEA	64/128	19
SHACAL-2	512/128	20

Hash functions and message integrity

Cryptographic hash functions

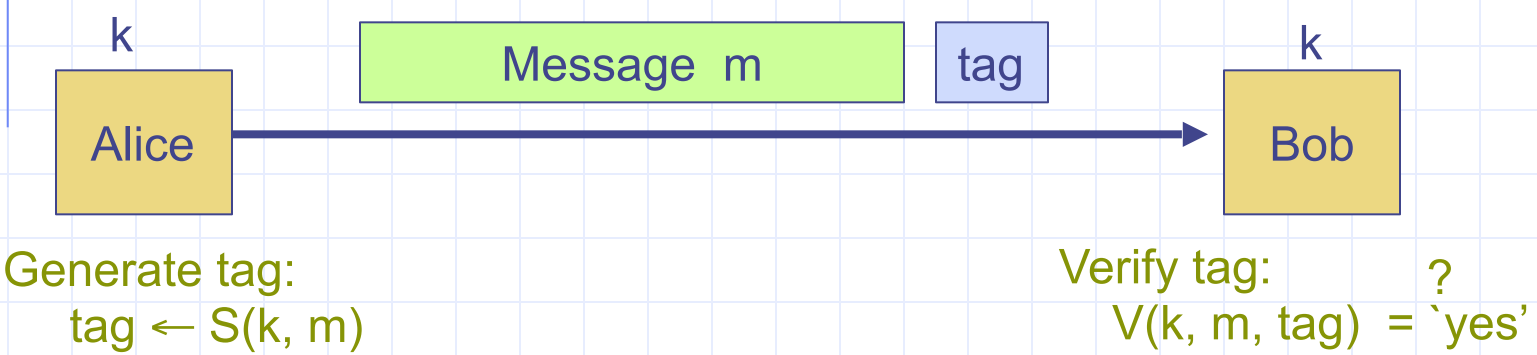
- Length-reducing function h
  - Map arbitrary strings to strings of fixed length
- One way ("preimage resistance")
  - Given y, hard to find x with h(x)=y
- Collision resistant
  - Hard to find any distinct m, m′ with h(m)=h(m′)
- Also useful: 2nd preimage resistance
  - Given x, hard to find x′≠x with h(x′)=h(x)
  - Collision resistance $\Rightarrow$ 2nd preimage resistance

Applications of one-way hash

- Password files (one way)
- Digital signatures (collision resistant)
  - Sign hash of message instead of entire message
- Data integrity
  - Compute and securely store hash of some data
  - Check later by recomputing hash and comparing
- Keyed hash for message authentication
  - MAC – Message Authentication Code

Message Integrity: MACs

- Goal: message integrity. No confidentiality.
  - ex: Protecting public binaries on disk.



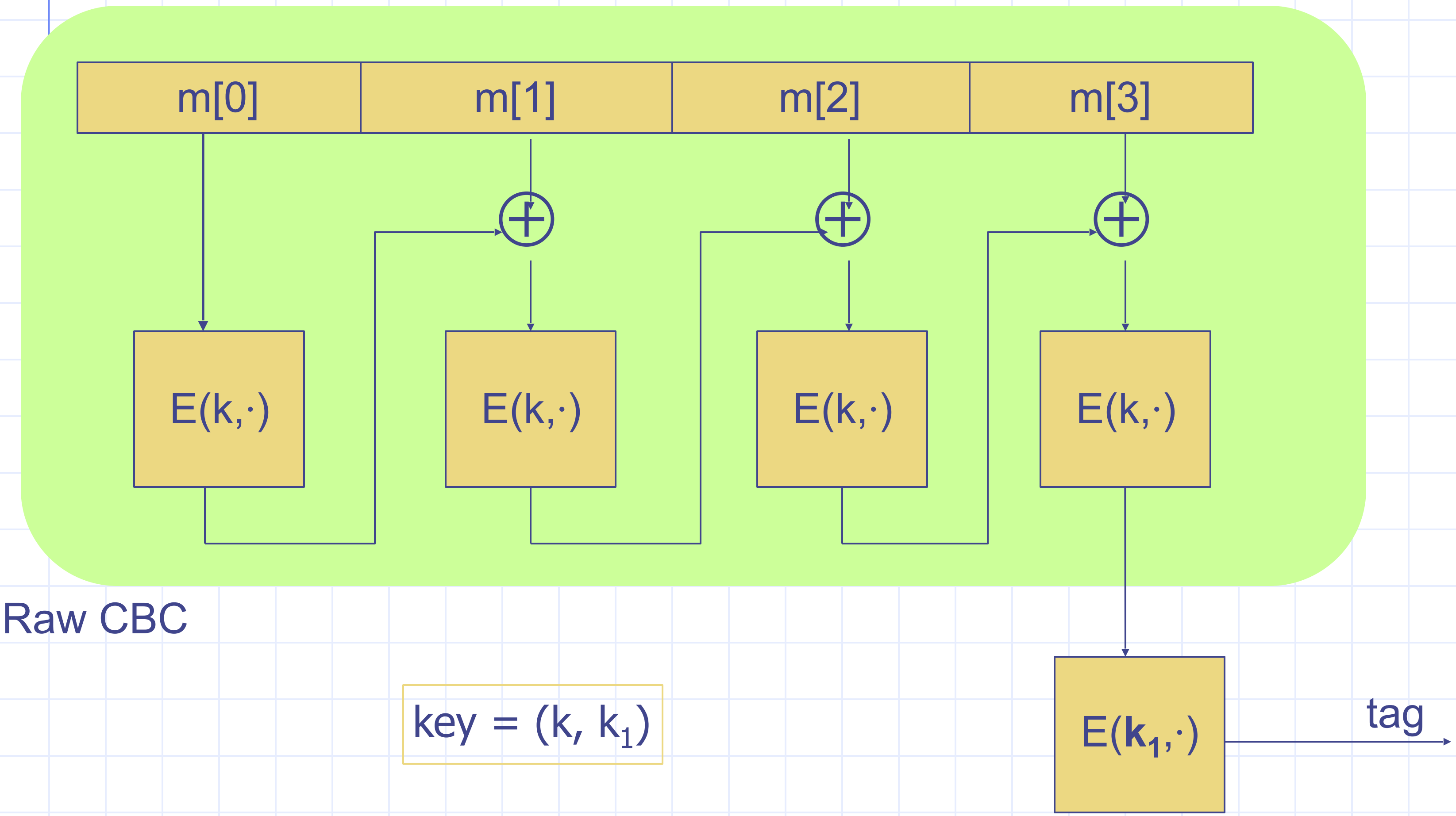
note: non-keyed checksum (CRC) is an insecure MAC !!

Secure MACs

- Attacker's power: chosen message attack.
  - for $m_1, m_2, \ldots, m_q$ attacker is given $t_i \leftarrow S(k, mi)$
- Attacker's goal: existential forgery.
  - produce some **new** valid message/tag pair (m,t).

    $(m,t) \notin \{ (m_1,t_1), \ldots, (m_q,t_q) \}$
- A secure PRF gives a secure MAC:
  - $S(k,m) = F(k,m)$
  - $V(k,m,t)$: \`yes' if $t = F(k,m)$ and \`no' otherwise.

Construction 1: ECBC



Construction 2: HMAC (Hash-MAC)

Most widely used MAC on the Internet.

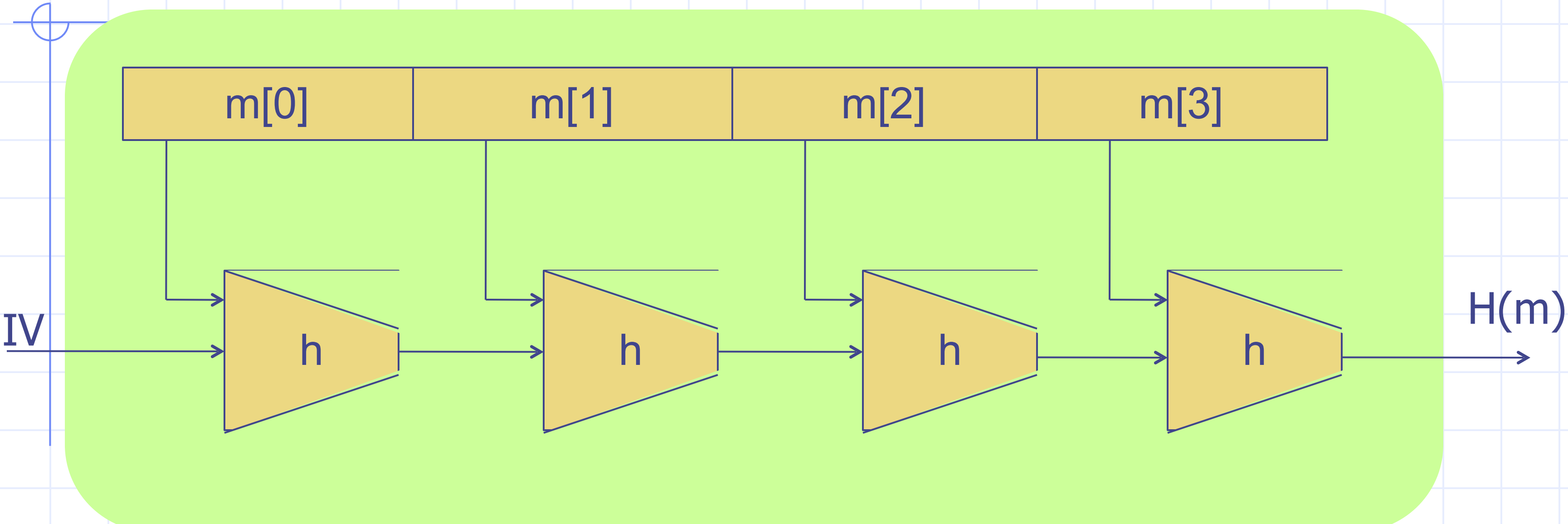
H: hash function.
example: SHA-256 ; output is 256 bits

Building a MAC out of a hash function:

Standardized method: HMAC

$$S(k, m) = H(k \oplus opad \,\|\, \mathbf{H(k \oplus ipad \,\|\, m)})$$

SHA-256: Merkle-Damgard



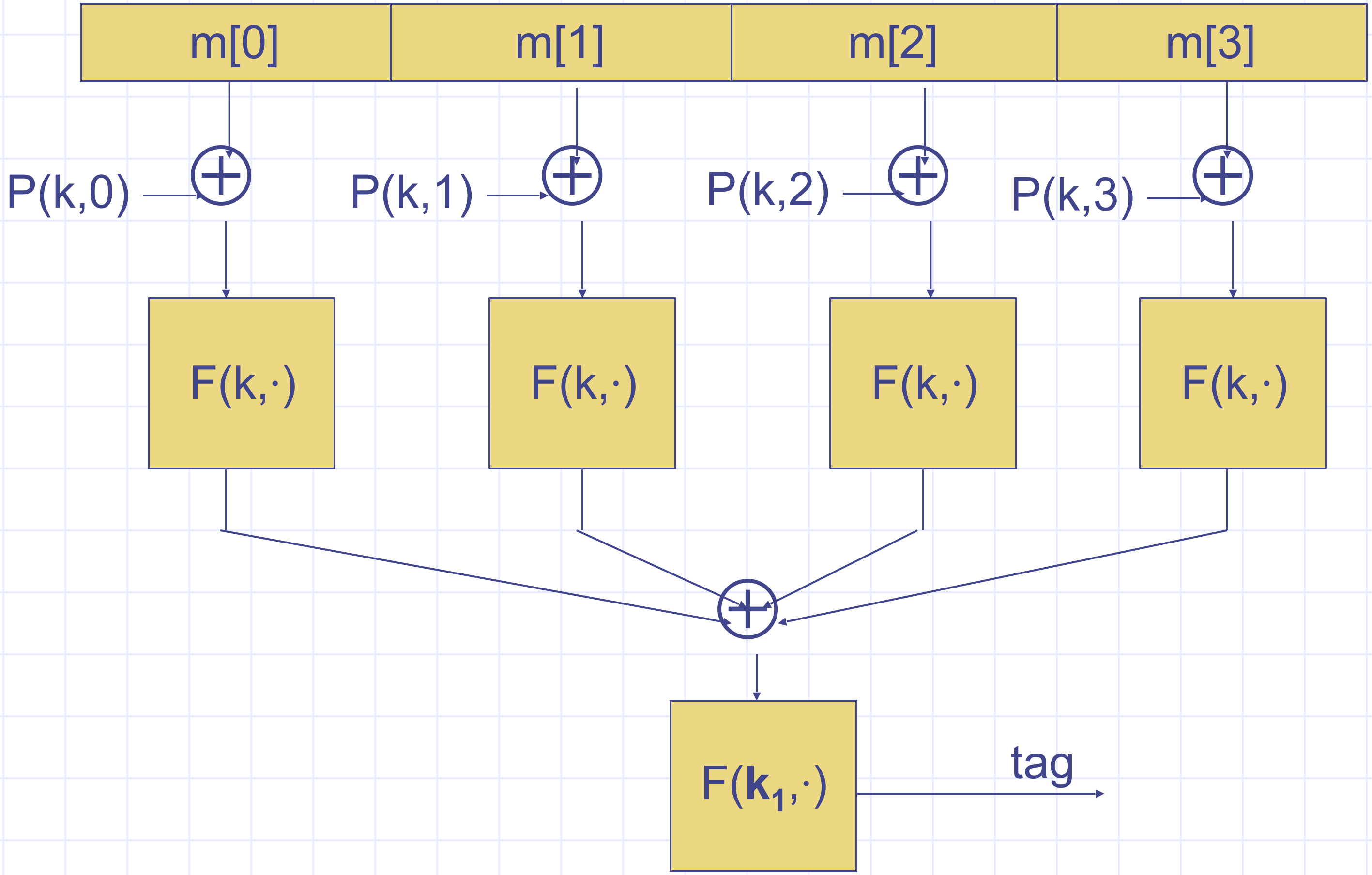
h(t, m[i]): compression function

Thm 1: if h is collision resistant then so is H

"Thm 2": if h is a PRF then HMAC is a PRF

Construction 3: PMAC – parallel MAC

ECBC and HMAC are sequential.   PMAC:



- Why are these MAC constructions secure?
  - … not today – take CS255

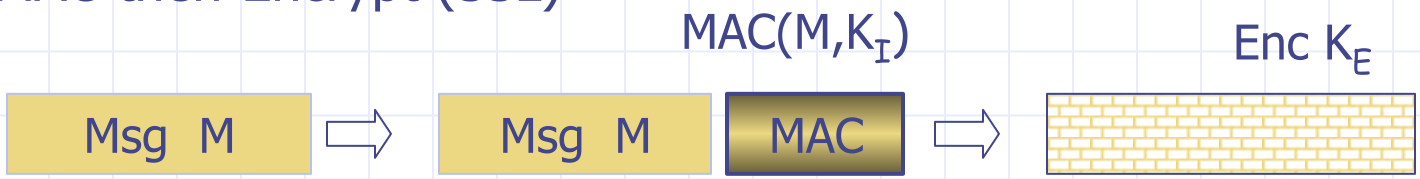
- Why the last encryption step in ECBC?
  - CBC (aka Raw-CBC) is not a secure MAC:
    - Given tag on a message m, attacker can deduce tag for some other message m’
    - How: good exercise.

Authenticated Encryption: Encryption + MAC

Combining MAC and ENC (CCA)

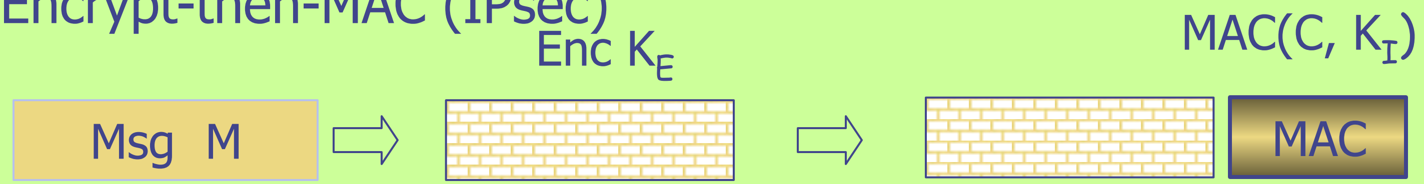
Encryption key K_E MAC key = K_I

Option 1: MAC-then-Encrypt (SSL)

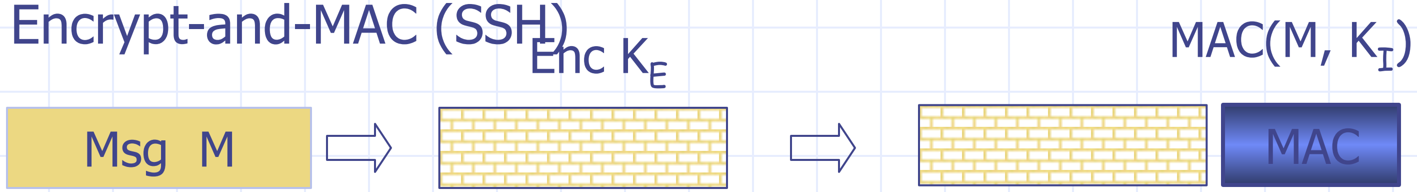


Option 2: Encrypt-then-MAC (IPsec)

Secure on general grounds



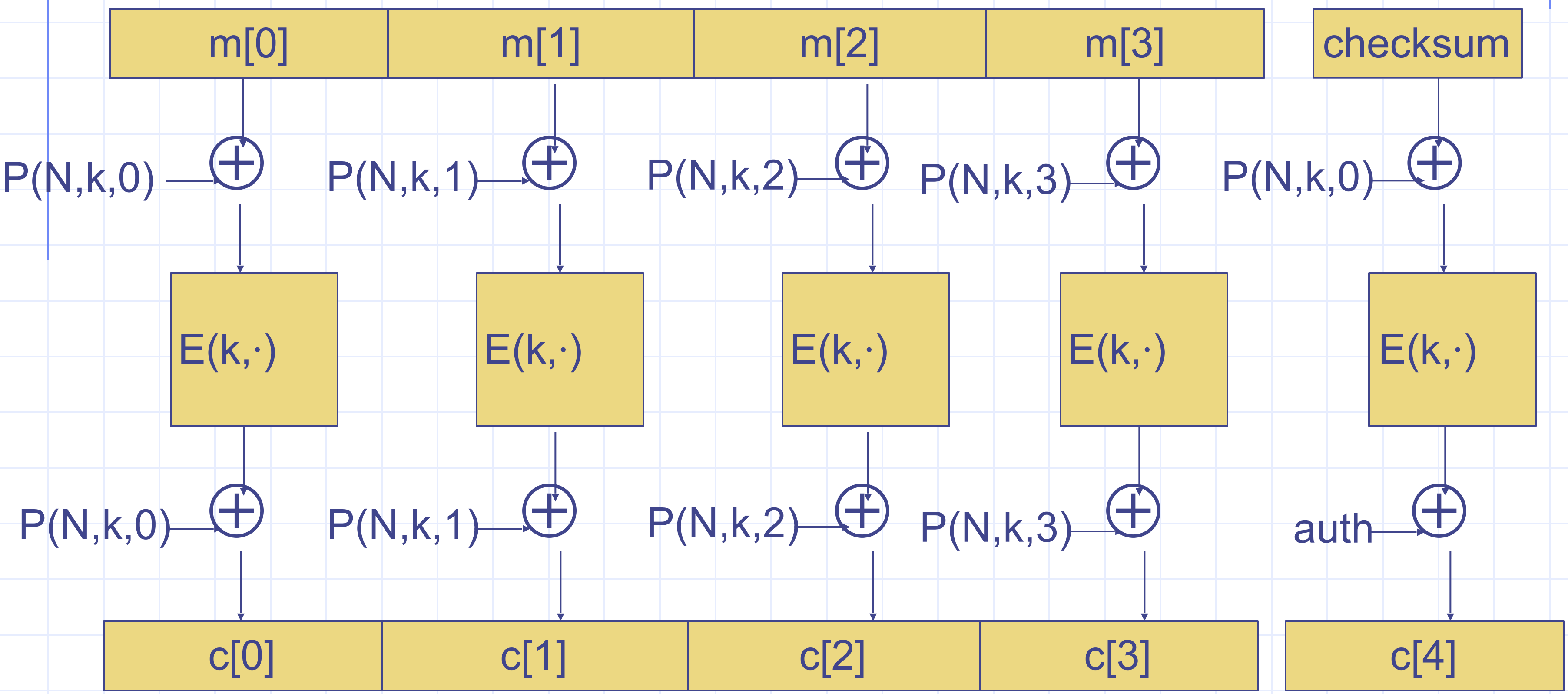
Option 3: Encrypt-and-MAC (SSH)



Recent developments: OCB

offset codebook mode

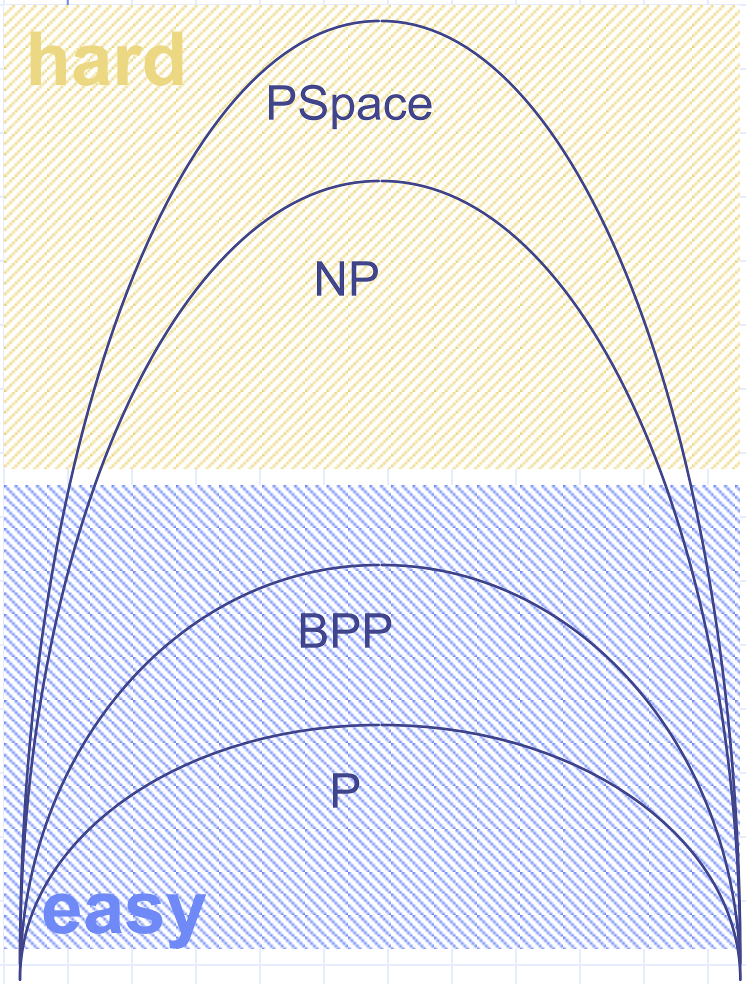
More efficient authenticated encryption



Rogaway, …

Public-key Cryptography

Complexity Classes



Answer in polynomial space
may need exhaustive search

If yes, can guess and check in polynomial time

Answer in polynomial time, with high probability

Answer in polynomial time compute answer directly

Example: RSA

- Arithmetic modulo pq
  - Generate secret primes p, q
  - Generate secret numbers a, b with $x^{ab} \equiv x \bmod pq$ $\overbrace{}^{n}$
- Public encryption key $\langle n, a \rangle$
  - $\text{Encrypt}(\langle n, a \rangle, x) = x^a \bmod n$
- Private decryption key $\langle n, b \rangle$
  - $\text{Decrypt}(\langle n, b \rangle, y) = y^b \bmod n$
- Main properties
  - This appears to be a "trapdoor permutation"
  - Cannot compute b from n,a
    - Apparently, need to factor n = pq

Why RSA works (quick sketch)

- Let p, q be two distinct primes and let n=p*q
  - Encryption, decryption based on group Z_n^*
  - For n=p*q, order $\phi(n) = (p-1)*(q-1)$
    - Proof: $(p-1)*(q-1) = p*q - p - q + 1$
- Key pair: $\langle a, b \rangle$ with $ab \equiv 1 \mod \phi(n)$
  - Encrypt(x) = $x^a \mod n$
  - Decrypt(y) = $y^b \mod n$
  - Since $ab \equiv 1 \mod \phi(n)$, have $x^{ab} \equiv x \mod n$
    - Proof: if gcd(x,n) = 1, then by general group theory, otherwise use "Chinese remainder theorem".

Textbook RSA is insecure

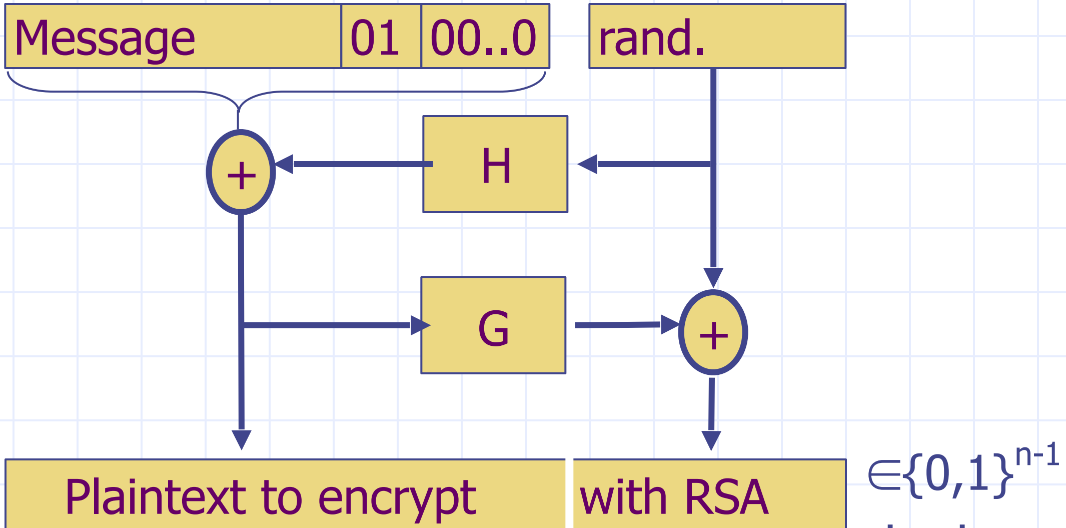
- What if message is from a small set (yes/no)?
  - Can build table
- What if I want to outbid you in secret auction?
  - I take your encrypted bid c and submit

    c (101/100)e mod n
- What if there's some protocol in which I can learn other message decryptions?

OAEP [BR94, Shoup '01]

Preprocess message for RSA

Check pad
on decryption.
Reject CT if invalid.



- If RSA is trapdoor permutation, then this is chosen-ciphertext secure (if H,G "random oracles")
- In practice: use SHA-1 or MD5 for H and G

Digital Signatures

- Public-key encryption
  - Alice publishes encryption key
  - Anyone can send encrypted message
  - Only Alice can decrypt messages with this key
- Digital signature scheme
  - Alice publishes key for verifying signatures
  - Anyone can check a message signed by Alice
  - Only Alice can send signed messages

Properties of signatures

- Functions to sign and verify
  - $\text{Sign}(\text{Key}^{-1}, \text{message})$
  - $\text{Verify}(\text{Key}, x, m) = \begin{cases} \text{true} & \text{if } x = \text{Sign}(\text{Key}^{-1}, m) \\ \text{false} & \text{otherwise} \end{cases}$
- Resists forgery
  - Cannot compute $\text{Sign}(\text{Key}^{-1}, m)$ from m and Key
  - Resists existential forgery:

    given Key, cannot produce $\text{Sign}(\text{Key}^{-1}, m)$
    for any random or arbitrary m

RSA Signature Scheme

- Publish decryption instead of encryption key
  - Alice publishes decryption key
  - Anyone can decrypt a message encrypted by Alice
  - Only Alice can send encrypt messages
- In more detail,
  - Alice generates primes p, q and key pair $\langle a, b \rangle$
  - $\text{Sign}(x) = x^a \bmod n$
  - $\text{Verify}(y) = y^b \bmod n$
  - Since $ab \equiv 1 \bmod \phi(n)$, have $x^{ab} \equiv x \bmod n$

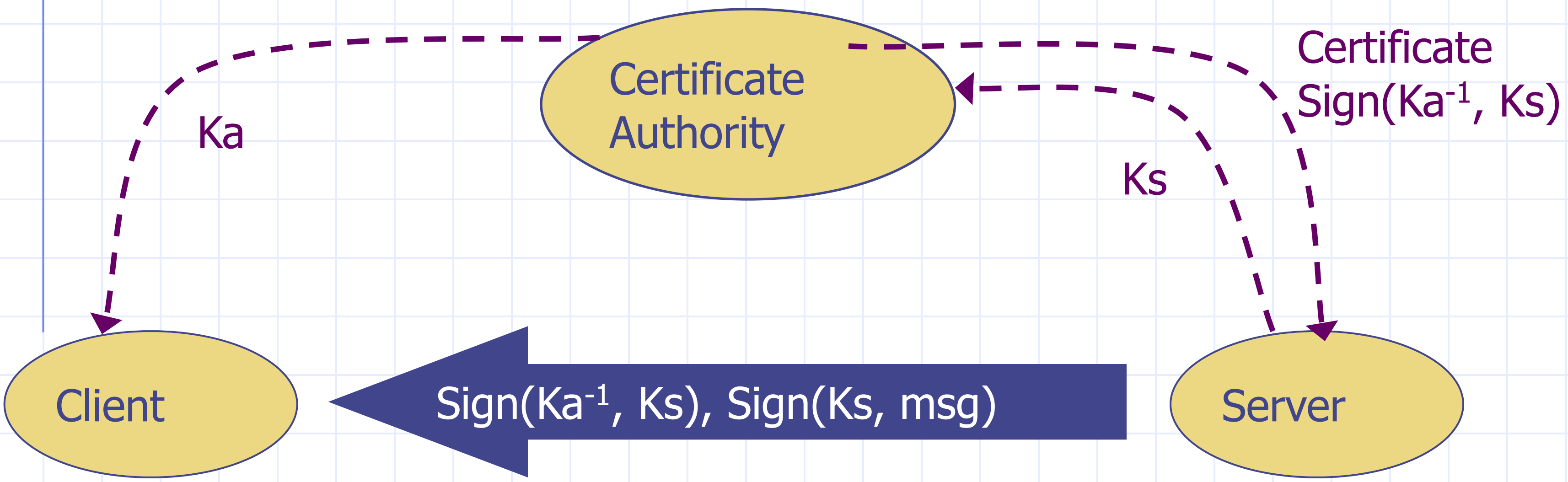
Generally, sign hash of message instead of full plaintext

Public-Key Infrastructure (PKI)

- Anyone can send Bob a secret message
  - Provided they know Bob's public key
- How do we know a key belongs to Bob?
  - If imposter substitutes another key, can read Bob's mail
- One solution: PKI
  - Trusted root authority (VeriSign, IBM, United Nations)
    - Everyone must know the verification key of root authority
    - Check your browser; there are hundreds!!
  - Root authority can sign certificates
  - Certificates identify others, including other authorities
  - Leads to certificate chains

Public-Key Infrastructure

Known public signature verification key Ka

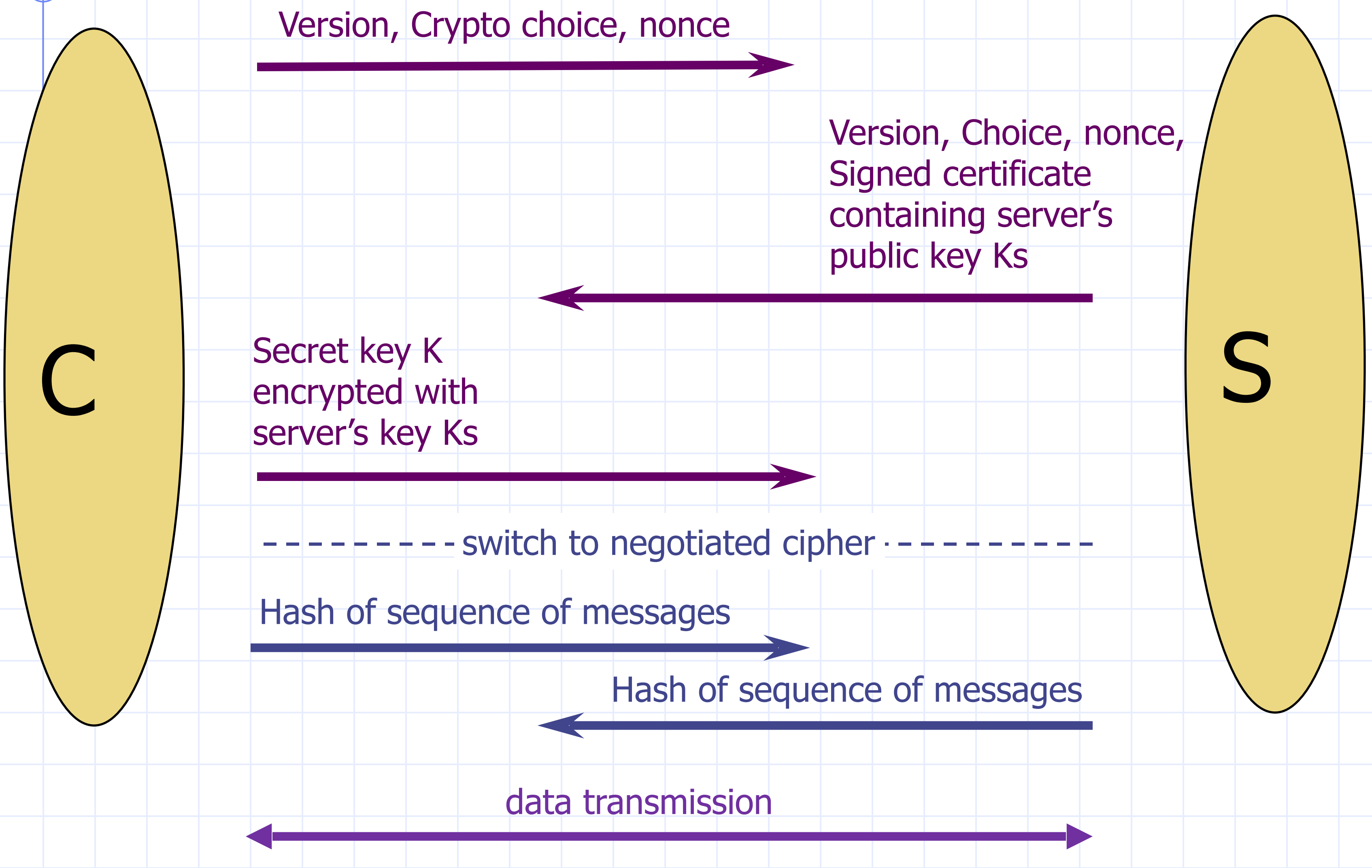


Server certificate can be verified by any client that has CA key Ka

Certificate authority is "off line"


Certificate Manager
Your Certificates
Other People's
Web Sites
Authorities
You have certificates on file that identify these certificate authorities:
Certificate Name
Security Device
Comodo CA Limited
Digital Signature Trust Co.
Entrust.net
Equifax
Equifax Secure
Equifax Secure Inc.
GTE Corporation
GeoTrust Inc.
GlobalSign nv-sa
Government Root Certification A...
IPS Internet publishing Services s.l.
IPS Seguridad CA
NetLock Halozatbiztonsagi Kft.
QuoVadis Limited
RSA Data Security, Inc.
RSA Security Inc
SECOM Trust.net
Sonera
View
Edit
Import
Delete
OK

Back to SSL/TLS

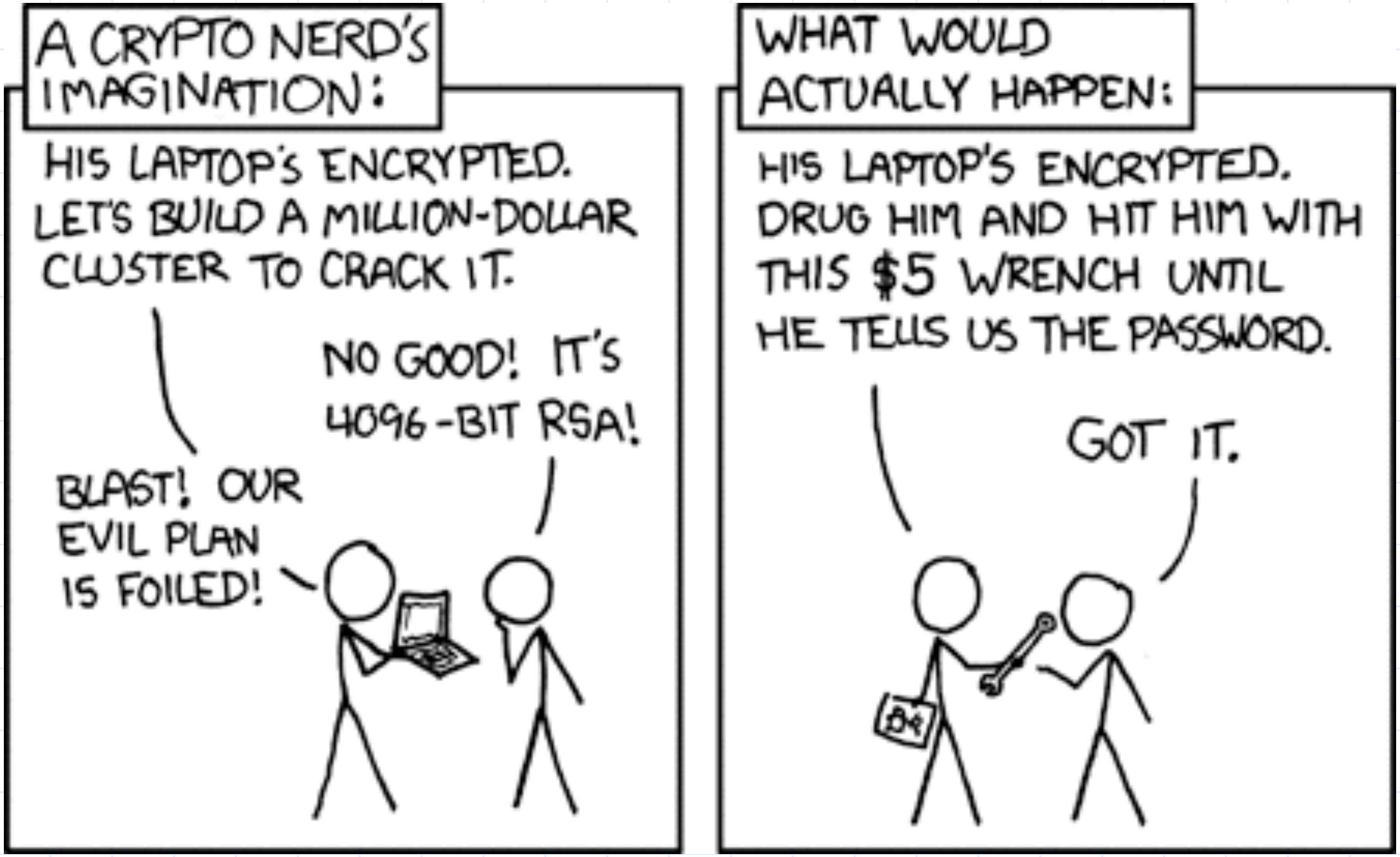


Crypto Summary

- Encryption scheme:
  - functions to encrypt, decrypt data
- Symmetric encryption
  - Block, stream ciphers
- Hash function, MAC
  - Map any input to short hash; ideally, no collisions
  - MAC (keyed hash) used for message integrity
- Public-key cryptography
  - PK encryption: public key does not reveal key^{-1}
  - Signatures: sign data, verify signature

Limitations of cryptography

- Most security problems are not crypto problems
  - This is good
    - Cryptography works!
  - This is bad
    - People make other mistakes; crypto doesn't solve them
- Misuse of cryptography is fatal for security
  - WEP – ineffective, highly embarrassing for industry
  - Occasional unexpected attacks on systems subjected to serious review


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

Auguste Kerckhoffs

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

baptised as **Jean-Guillaume-Hubert-Victor-François-Alexandre-Auguste Kerckhoffs von Nieuwenhof**

Example cryptosystems

- One-time pad
  - "Theoretical idea," but leads to stream cipher
- Feistel construction for symmetric key crypto
  - Iterate a "scrambling function"
  - Examples: DES, Lucifer, FREAL, Khufu, Khafre, LOKI, GOST, CAST, Blowfish, …
  - AES (Rijndael) is also block cipher, but different …
- Complexity-based public-key cryptography
  - Modular exponentiation is a "one-way" function
  - Examples: RSA, El Gamal, elliptic curve systems, ...

Symmetric Encryption

- Encryption keeps communication secret
- Encryption algorithm has two functions: E and D
  - To communicate secretly, parties share secret key K
- Given a message M, and a key K:
  - M is known as the plaintext
  - E(K,M) → C (C known as the ciphertext)
  - D(K, C) → M
  - Attacker cannot efficiently derive M from C without K
- Note E and D use same key K
  - Reason for the name "symmetric encryption"

One-time pad

- Share a random key K
- Encrypt plaintext by xor with sequence of bits
  - encrypt(key, text) = key $\oplus$ text (bit-by-bit)
- Decrypt ciphertext by xor with same bits
  - decrypt(key, text) = key $\oplus$ text (bit-by-bit)
- Advantages
  - Easy to compute encrypt, decrypt from key, text
  - This is an information-theoretically secure cipher
- Disadvantage
  - Key is as long as the plaintext
    - How does sender get key to receiver securely?

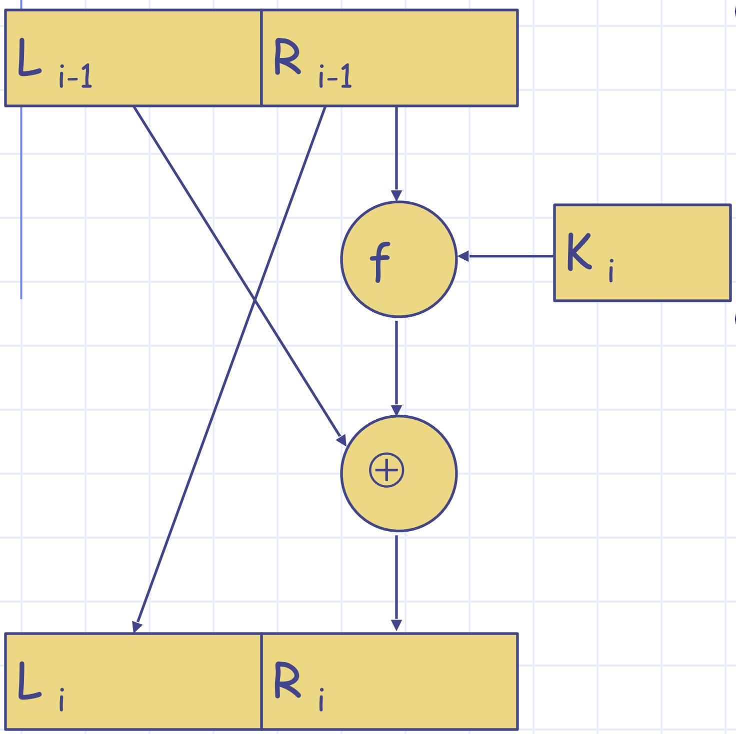
Idea for stream cipher: use pseudo-random generators for key …

Types of symmetric encryption

- Stream ciphers – pseudo-random pad
  - Generate pseudo-random stream of bits from short key
  - Encrypt/decrypt by XORing as with one-time pad
  - But NOT one-time PAD! (People who claim so are frauds!)
- Block cipher
  - Operates on fixed-size blocks (e.g., 64 or 128 bits)
  - Maps plaintext blocks to same size ciphertext blocks
  - Today use AES; other algorithms: DES, Blowfish, . . .

Feistel network: One Round

Divide n-bit input in half and repeat



- Scheme requires
  - Function $f(R_{i-1}, K_i)$
  - Computation for K_i
    - e.g., permutation of key K
- Advantage
  - Systematic calculation
    - Easy if f is table, etc.
  - Invertible if K_i known
    - Get R_{i-1} from L_i
    - Compute $f(R_{i-1}, K_i)$
    - Compute L_{i-1} by $\oplus$

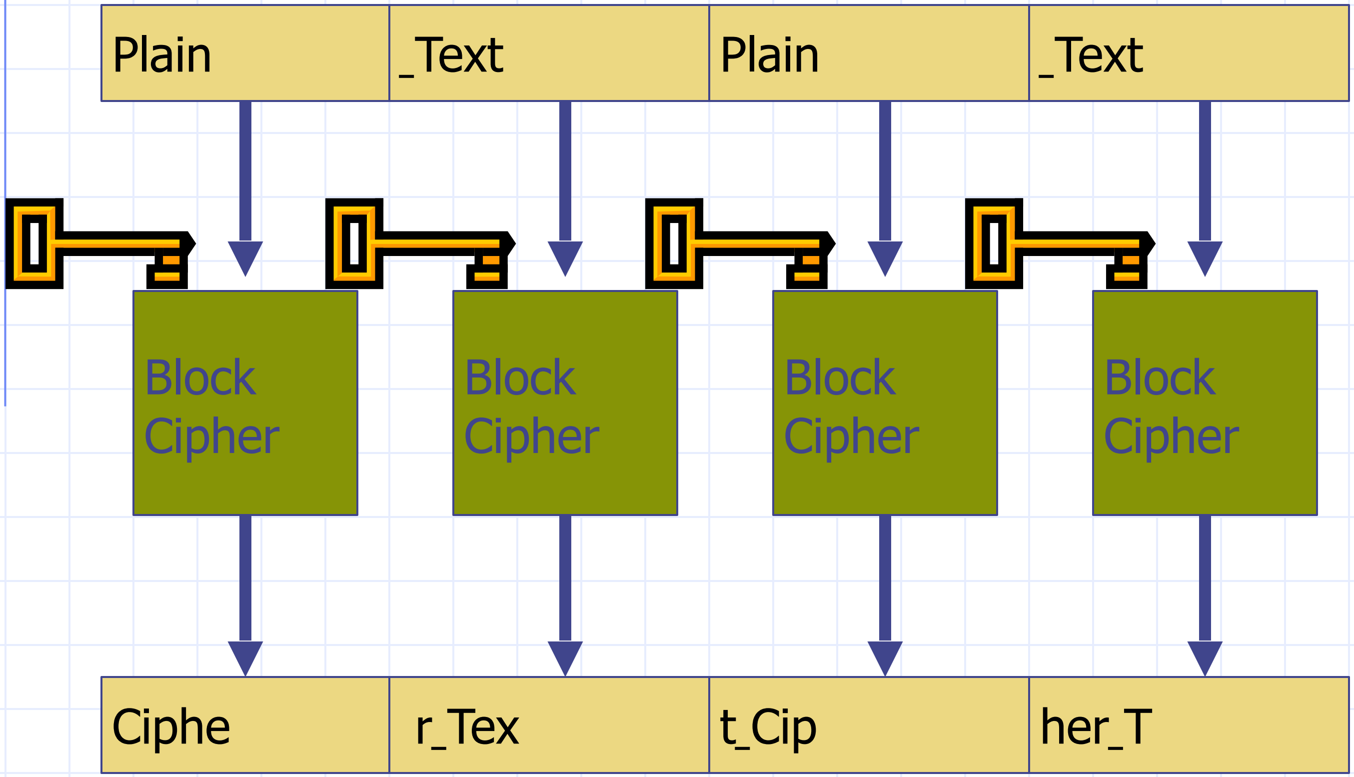
Data Encryption Standard

- Developed at IBM, some input from NSA, widely used
- Feistel structure
  - Permute input bits
  - Repeat application of a S-box function
  - Apply inverse permutation to produce output
- Worked well in practice (but brute-force attacks now)
  - Efficient to encrypt, decrypt
  - Not provably secure
- Improvements
  - Triple DES, AES (Rijndael)

Block cipher modes (for DES, AES, …)

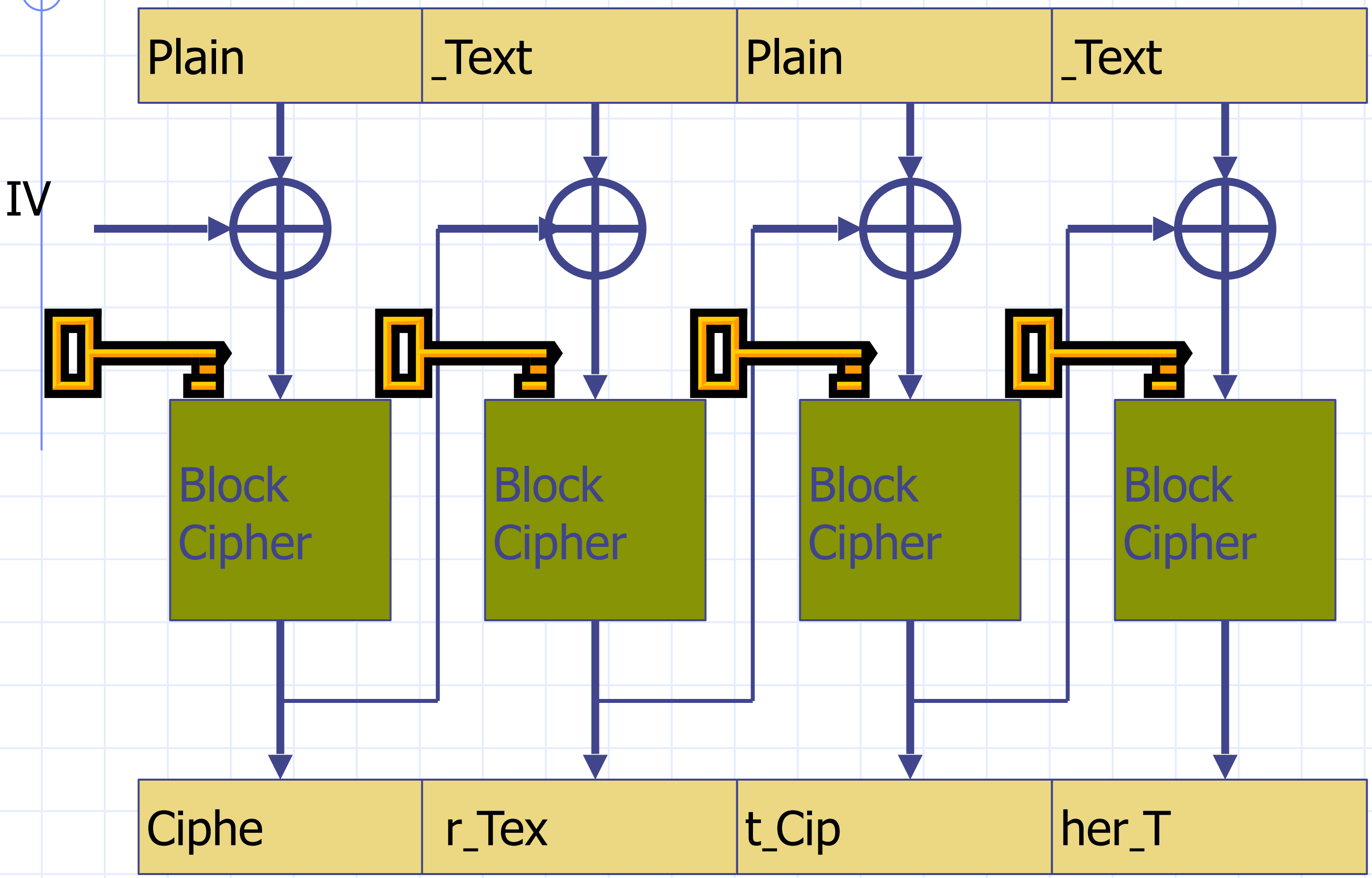
- ECB – Electronic Code Book mode
  - Divide plaintext into blocks
  - Encrypt each block independently, with same key
- CBC – Cipher Block Chaining
  - XOR each block with encryption of previous block
  - Use initialization vector IV for first block
- OFB – Output Feedback Mode
  - Iterate encryption of IV to produce stream cipher
- CFB – Cipher Feedback Mode
  - Output block y_i = input $x_i \oplus \text{encyrpt}_K(y_{i-1})$

Electronic Code Book (ECB)



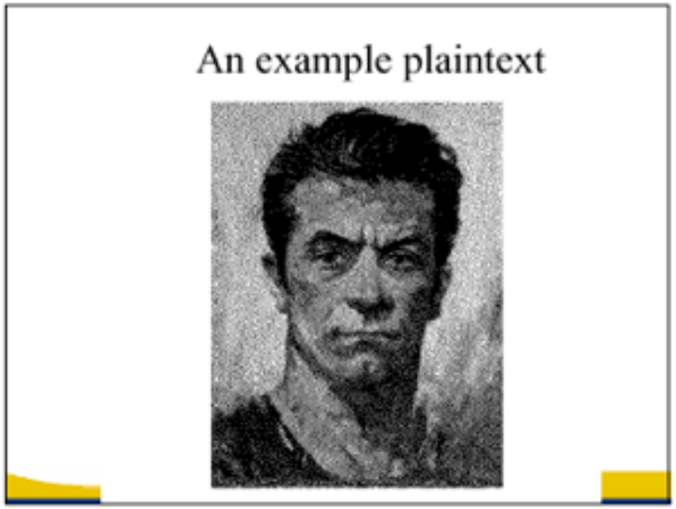
Problem: Identical blocks encrypted identically

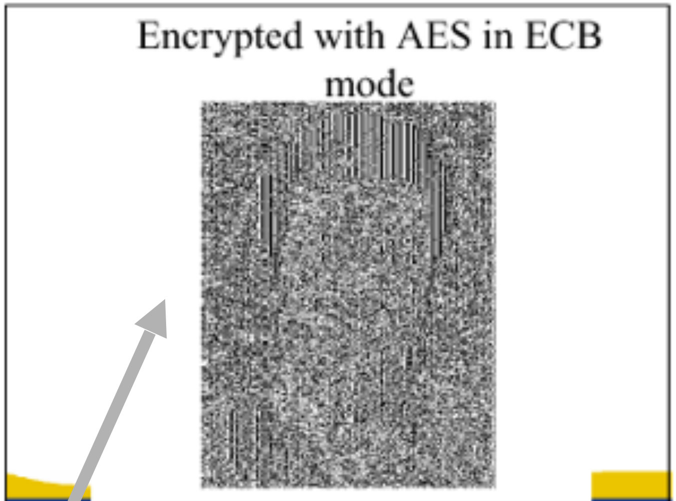
Cipher Block Chaining (CBC)

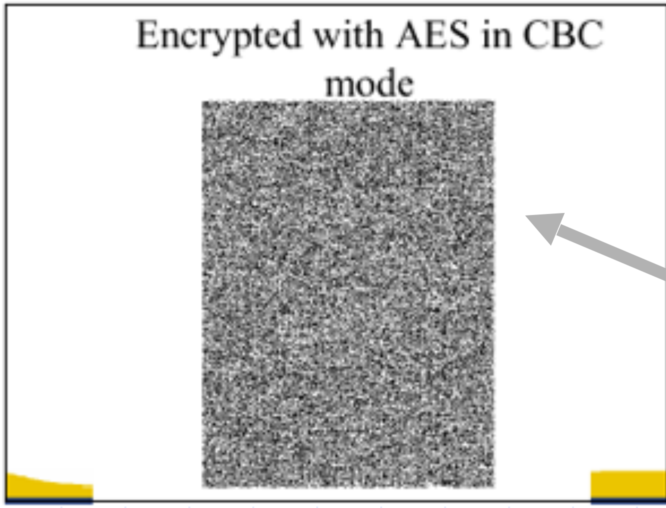


Advantages: Identical blocks encrypted differently
Last ciphertext block depends on entire input

Comparison (for AES, by Bart Preneel)



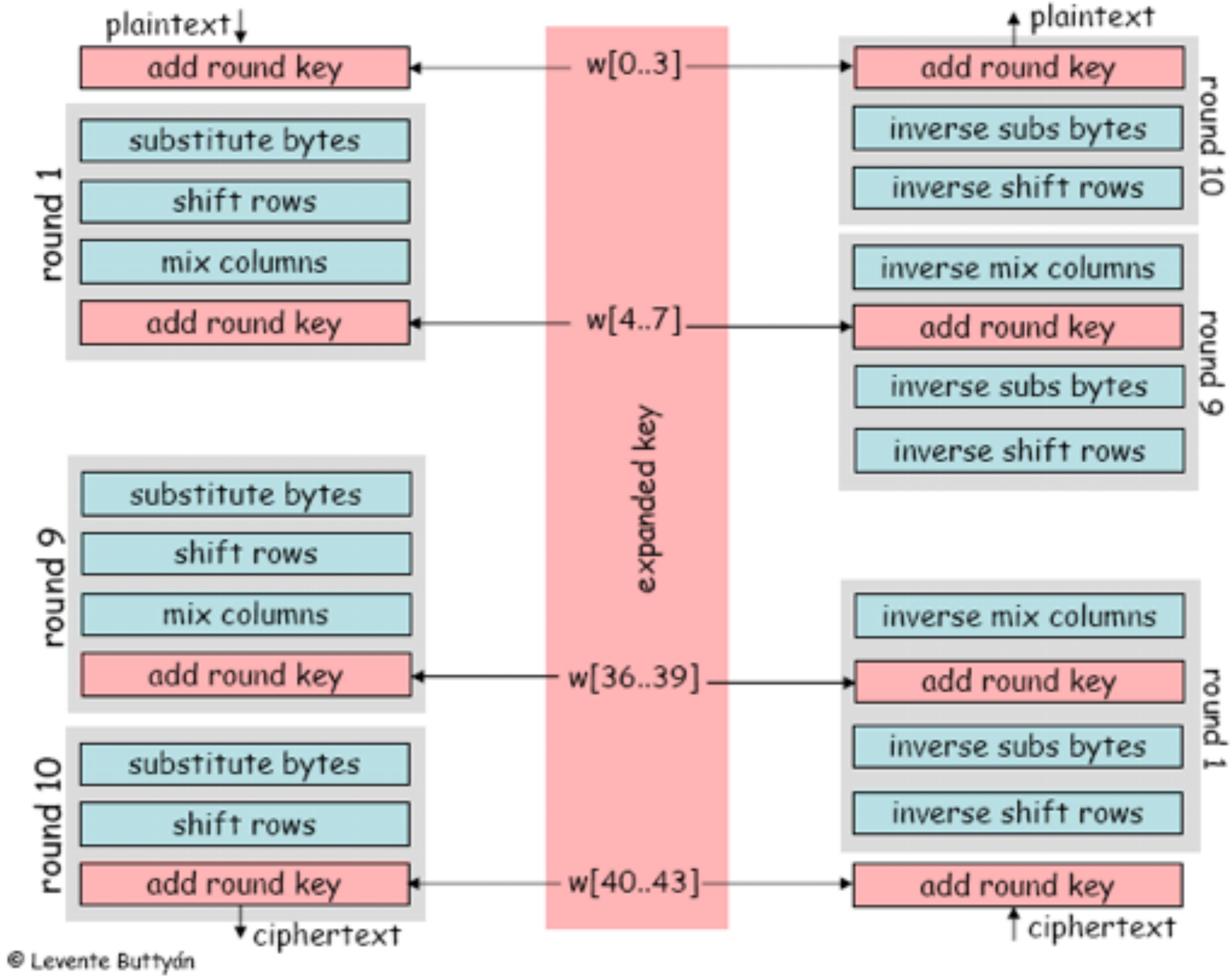




Similar plaintext blocks produce similar ciphertext (see outline of head)

No apparent pattern

Structure of AES



RC4 stream cipher – "Ron's Code"

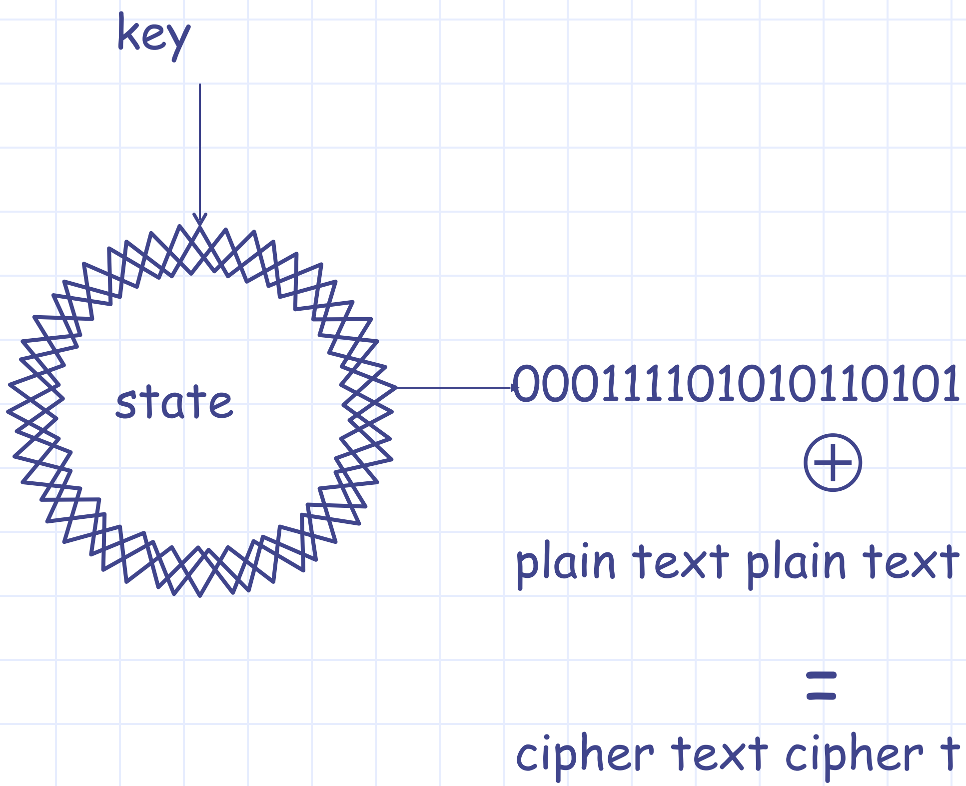
- Design goals (Ron Rivest, 1987):
  - speed
  - support of 8-bit architecture
  - simplicity (circumvent export regulations)
- Widely used
  - SSL/TLS
  - Windows, Lotus Notes, Oracle, etc.
  - Cellular Digital Packet Data
  - OpenBSD pseudo-random number generator

RSA Trade Secret

- History
  - 1994 – leaked to cypherpunks mailing list
  - 1995 – first weakness (USENET post)
  - 1996 – appeared in Applied Crypto as “alleged RC4”
  - 1997 – first published analysis

Weakness is predictability of first bits; best to discard them

Encryption/Decryption



Stream cipher: one-time pad based on pseudo-random generator

Security

- Goal: indistinguishable from random sequence
  - given part of the output stream, it is impossible to distinguish it from a random string
- Problems
  - Second byte [MS01]
    - Second byte of RC4 is 0 with twice expected probability
  - Related key attack [FMS01]
    - Bad to use many related keys (see WEP 802.11b)
- Recommendation
  - Discard the first 256 bytes of RC4 output [RSA, MS]

Complete Algorithm

(all arithmetic mod 256)

```
for i := 0 to 255    S[i] := i
j := 0
for i := 0 to 255
     j := j + S[i] + key[i]
     swap (S[i], S[j])

i, j := 0
repeat
     i := i + 1
     j := j + S[i]
     swap (S[i], S[j])
     output (S[ S[i] + S[j] ])
```

Key scheduling

0	1	2	3	4	5	6	…

Permutation of 256 bytes, depending on key

2	123	134	24	1	218	53	…

Random generator

2	123	134	24	9	218	53	…

i → j → +24

Example use of stream cipher?

- Share secret s with web vendor
- Exchange payment information as follows
  - Send: E(s, "Visa card #3273. . . ")
  - Receive: E(s, "Order confirmed, have a nice day")
- Now eavesdropper can't get out your Visa #

Wrong!

- Suppose attacker overhears
  - c1 = Encrypt(s, "Visa card #3273. . . ")
  - c2 = Encrypt(s, "Order confirmed, have a nice day")
- Now compute
  - m $\leftarrow$ c1 $\oplus$ c2 $\oplus$ "Order confirmed, have a nice day"
- Lesson: Never re-use keys with a stream cipher
  - Basic problem with one-time pads
  - This is why they're called one-time pads

Public-key Cryptosystem

- Trapdoor function to encrypt and decrypt
  - encrypt(key, message)

    key pair

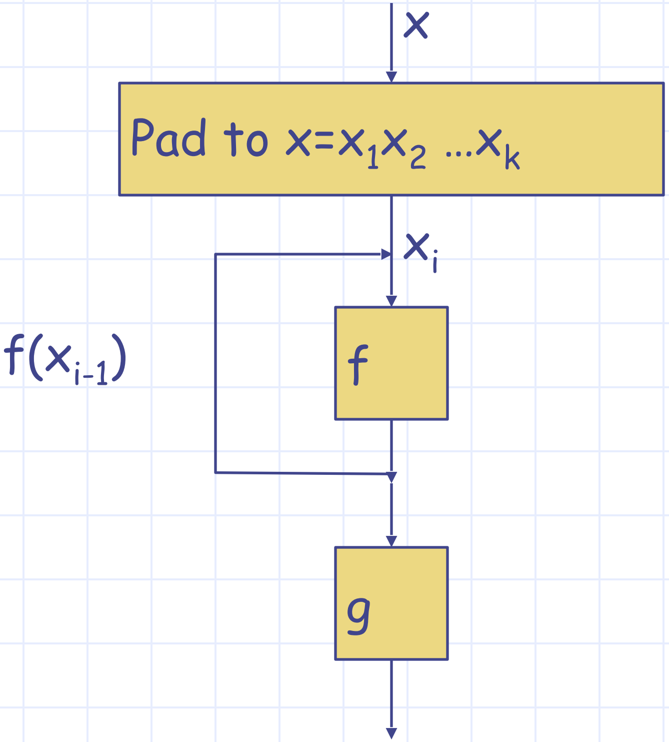
  - decrypt(key $^{-1}$, encrypt(key, message)) = message

- Resists attack
  - Cannot compute m from encrypt(key, m) and key, unless you have key^{-1}

Iterated hash functions

- Repeat use of block cipher or custom function
  - Pad input to some multiple of block length
  - Iterate a length-reducing function f
    - $f : 2^{2k} \text{ -> } 2^k$ reduces bits by 2
    - Repeat h_0= some seed

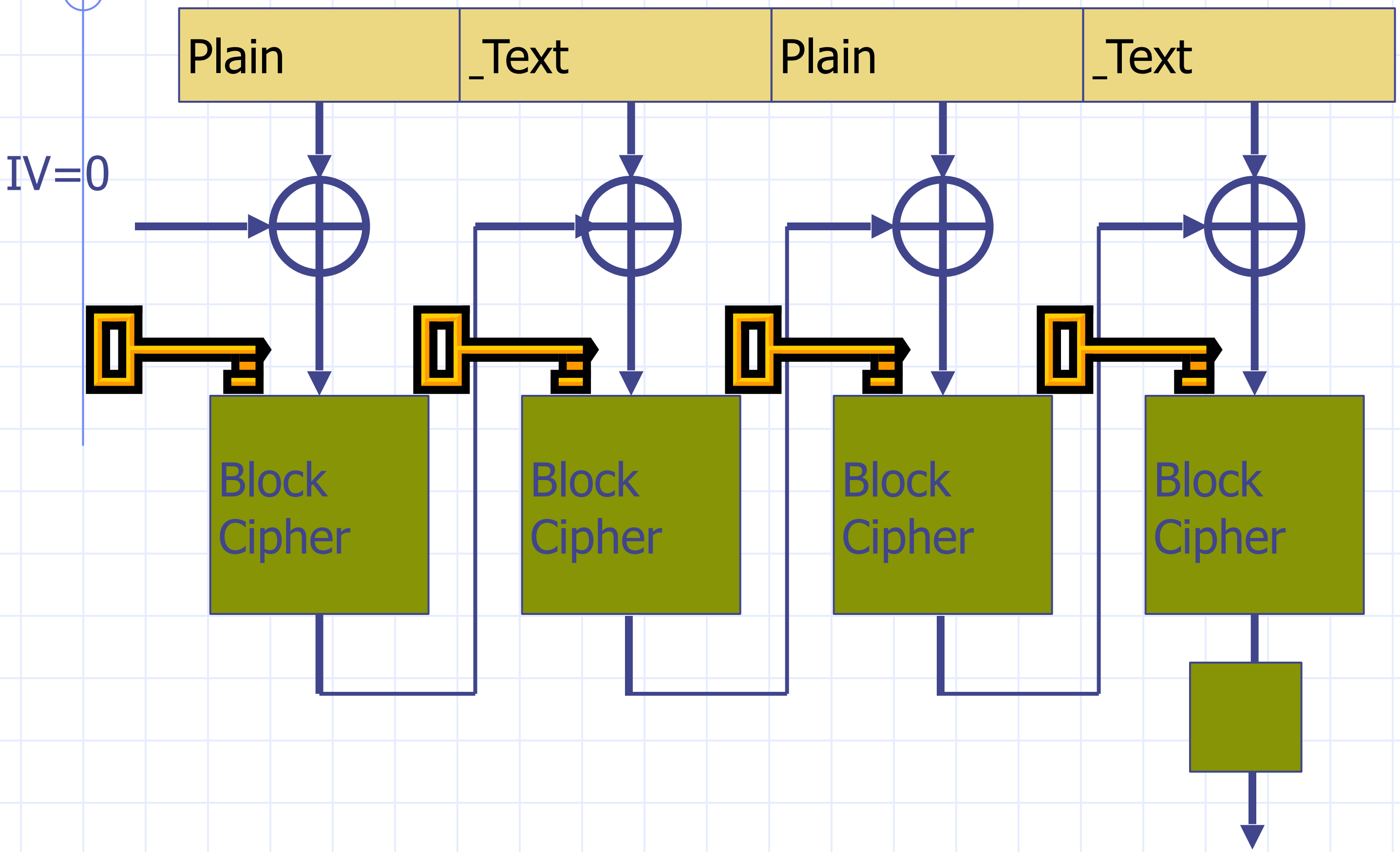
      $h_{i+1} = f(h_i, x_i)$
  - Some final function g
    completes calculation



MAC: Message Authentication Code

- General pattern of use
  - Sender sends Message and M1 = MAC(Message)
  - Receiver receives both parts
  - Receiver computes M2 = MAC(Message)
    - If M2 == M1, data is valid
    - If M2 != M1, data has been corrupted
- This requires a shared secret key
  - Suppose an attacker can compute MAC(x)
  - Intercept M and MAC(M), resend as M' and MAC(M')
  - Receiver cannot detect that message has been altered

Basic CBC-MAC

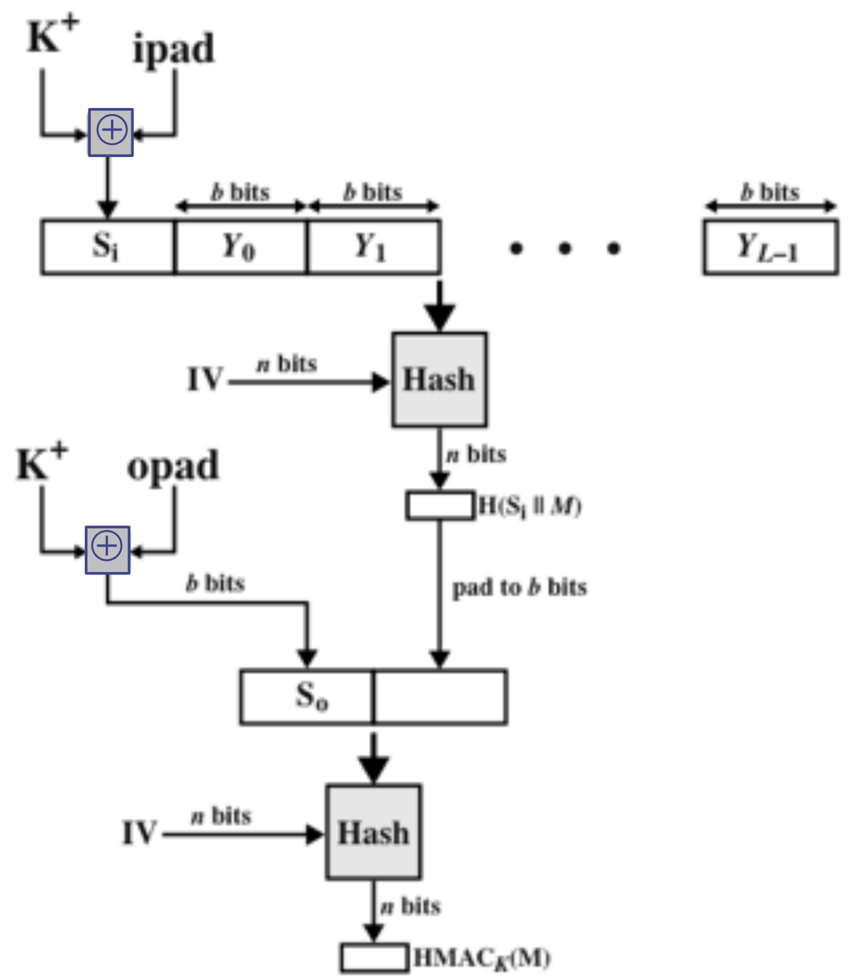


CBC block cipher, discarding all but last output block

Additional post-processing (e.g, encrypt with second key) can improve output

HMAC: Keyed Hash-Based MAC

- Internet standard RFC2104
- Uses hash of key, message:
  $HMAC_K(M)$
  $= Hash[(K^+ \text{ XOR } opad) || Hash[(K^+ \text{ XOR } ipad)||M)]]$
- Low overhead
  - **`opad, ipad`** are constants
- Any of MD5, SHA-1, … can be used



K+ is the key padded out to size

Order of Encryption and MACs

- Should you Encrypt then MAC, or vice versa?
- MACing encrypted data is always secure
- Encrypting {Data+MAC} may not be secure!