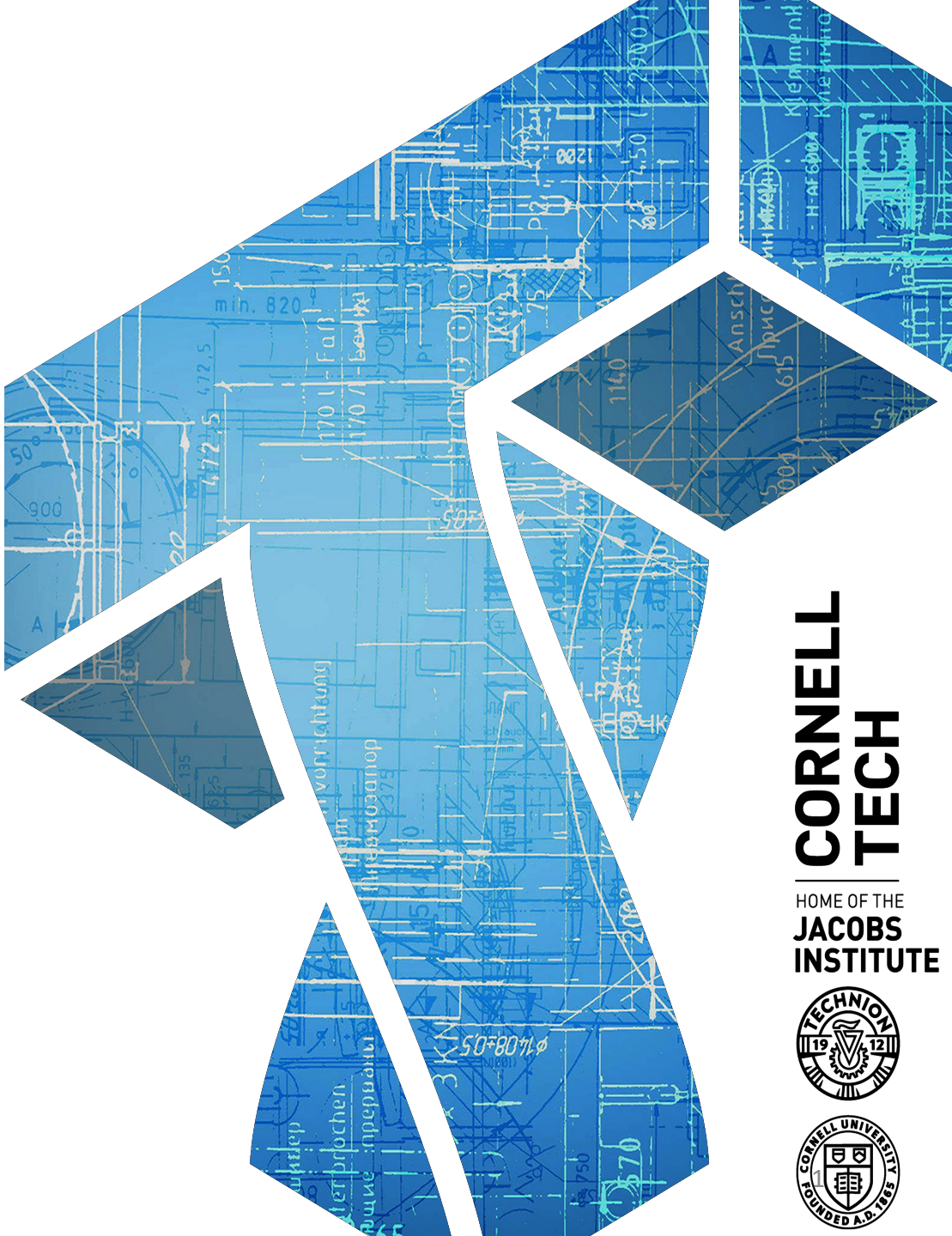
CS 5435: Cryptography

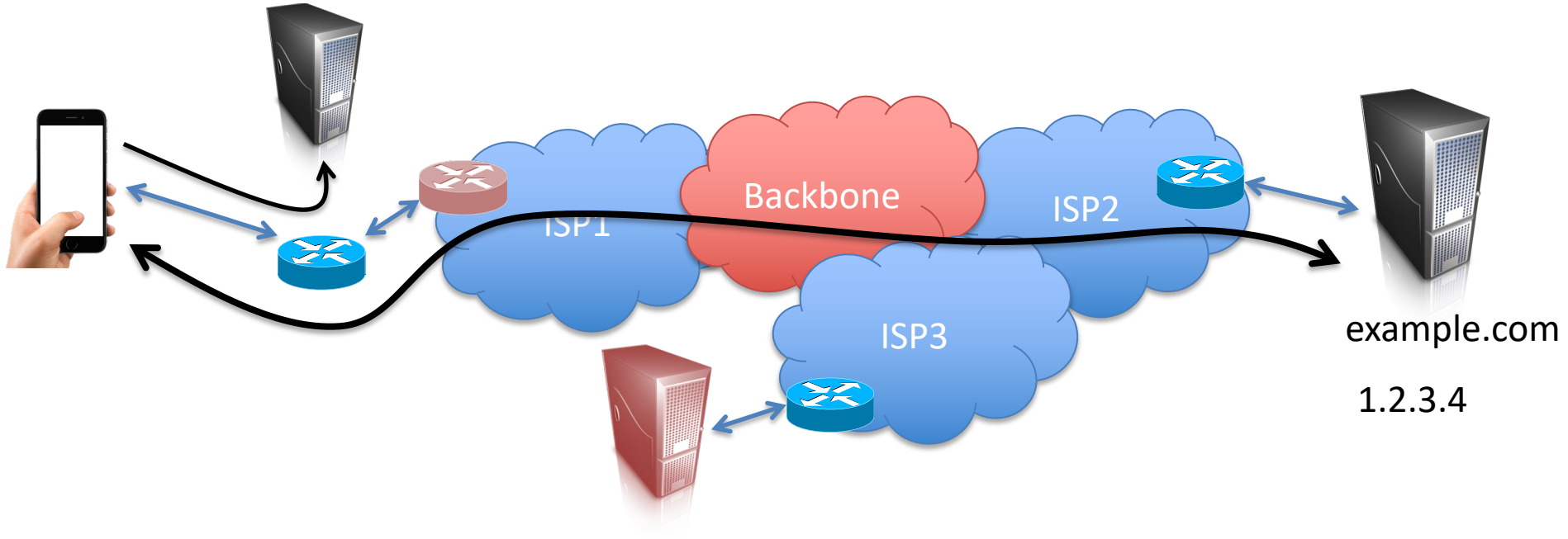
Instructor: Tom Ristenpart

https://github.com/tomrist/cs5435-spring2024



CORNELL TECH
HOME OF THE JACOBS INSTITUTE

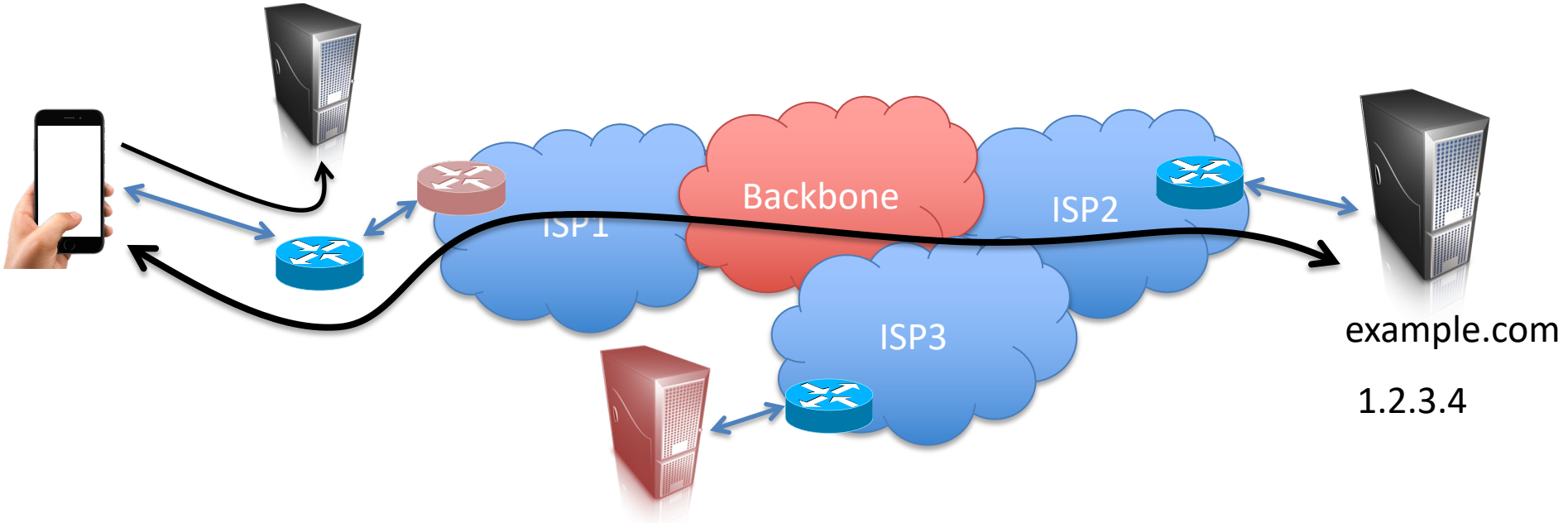
Network threat models



(1) Malicious hosts } Off-path attackers

On-path attackers {
(2) Subverted routers or links
(3) Malicious ISPs or backbone

Network threat models



Client and server want to communicate securely:

- **Confidentiality** (messages are private)
- **Integrity** (accepted messages are as sent)
- **Authenticity** (is it really example.com?)
- Sometimes: anonymity (hide identities)
- Sometimes: steganography (hide that communication took place)

TLS, SSH, IPsec, PGP

Cryptography: “Hidden writing”

- Study and practice of building security protocols that resist adversarial behavior
- Blend of mathematics, engineering, computer science

- Powerful tool for confidentiality, authenticity, and more
- But:
  - must design securely
  - must implement designs securely
  - must use properly (e.g., key management)

Auguste Kerckhoffs' (Second) Principle

"The system must not require secrecy and can be stolen by the enemy without causing trouble"

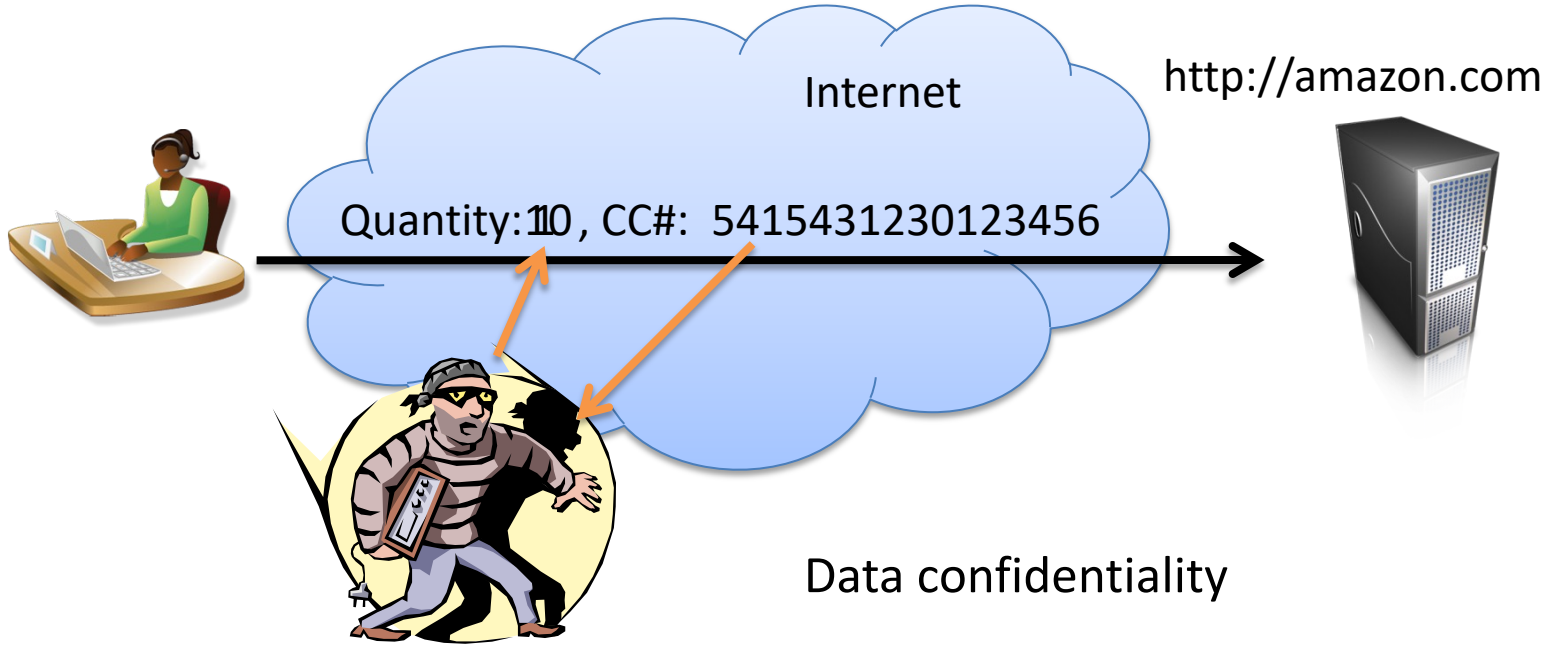
A cryptosystem should be secure even if its algorithms, implementations, configuration, etc. is made public --- the only secret should be a key

Why?

Some cryptographic primitives

- Symmetric cryptography (shared key K)
  - encryption & decryption using K
  - message authentication using K
  - pseudorandom functions (PRF)
- Public-key cryptography (public key pk, secret key sk)
  - encrypt with pk and decrypt with sk
  - digitally sign using sk and verify with pk
- Hash functions (no keys)
  - used to “compress” messages in a secure way

An example: On-line shopping with TLS

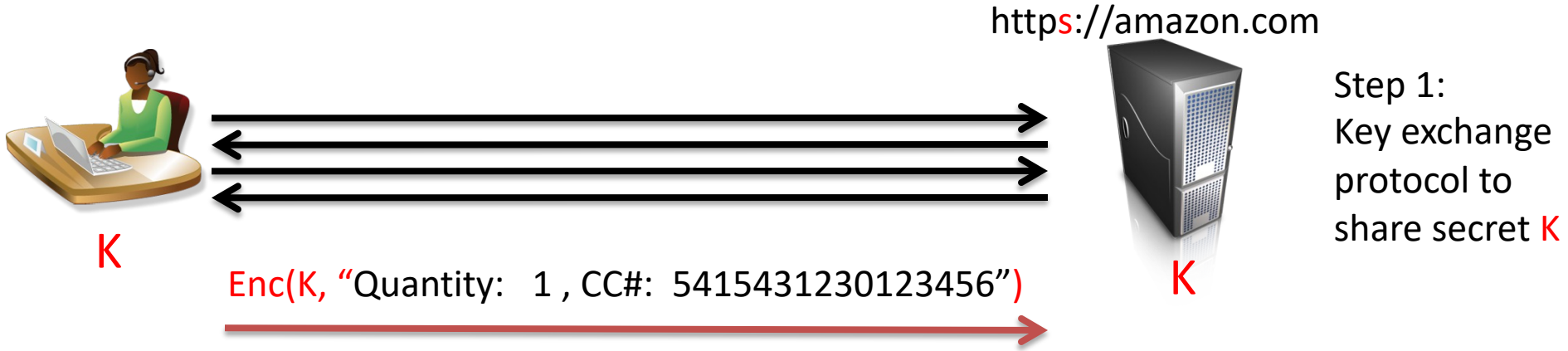


Data confidentiality

Data integrity

We need secure channels for transmitting data

An example: On-line shopping with TLS



Step 1:
Key exchange protocol to share secret K

Step 2:
Send data via secure channel

TLS uses many cryptographic primitives:

key exchange: hash functions, digital signatures, public key encryption

secure channel: symmetric encryption, message authentication

Mechanisms to resist replay attacks, man-in-the-middle attacks, truncation attacks, etc…

TLS 1.2 handshake (key transport)

Client

Server

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

Pick random Ns

← ServerHello, Ver, Ns, SessionID, Cipher/CompMethod

Check CERT using CA public verification key

← CERT = (pk of server, signature over it)

Pick random PMS
C <- Enc(pk,PMS)

C →

PMS <- Dec(sk,C)

ChangeCipherSpec,
{ Finished, PRF(MS, “Client finished” || H(transcript)) } →

ChangeCipherSpec,
{ Finished, PRF(MS, “Server finished” || H(transcript’)) } ←

Bracket notation means contents encrypted

MS <- PRF(PMS, “master secret” || Nc || Ns)

TLS 1.2 Record layer

Client

Server

MS <- PRF(PS, “master secret” || Nc || Ns)

K1,K2 <- PRF(MS, “key expansion” || Ns || Nc)

C1 <- Enc(K1,Message)

C1

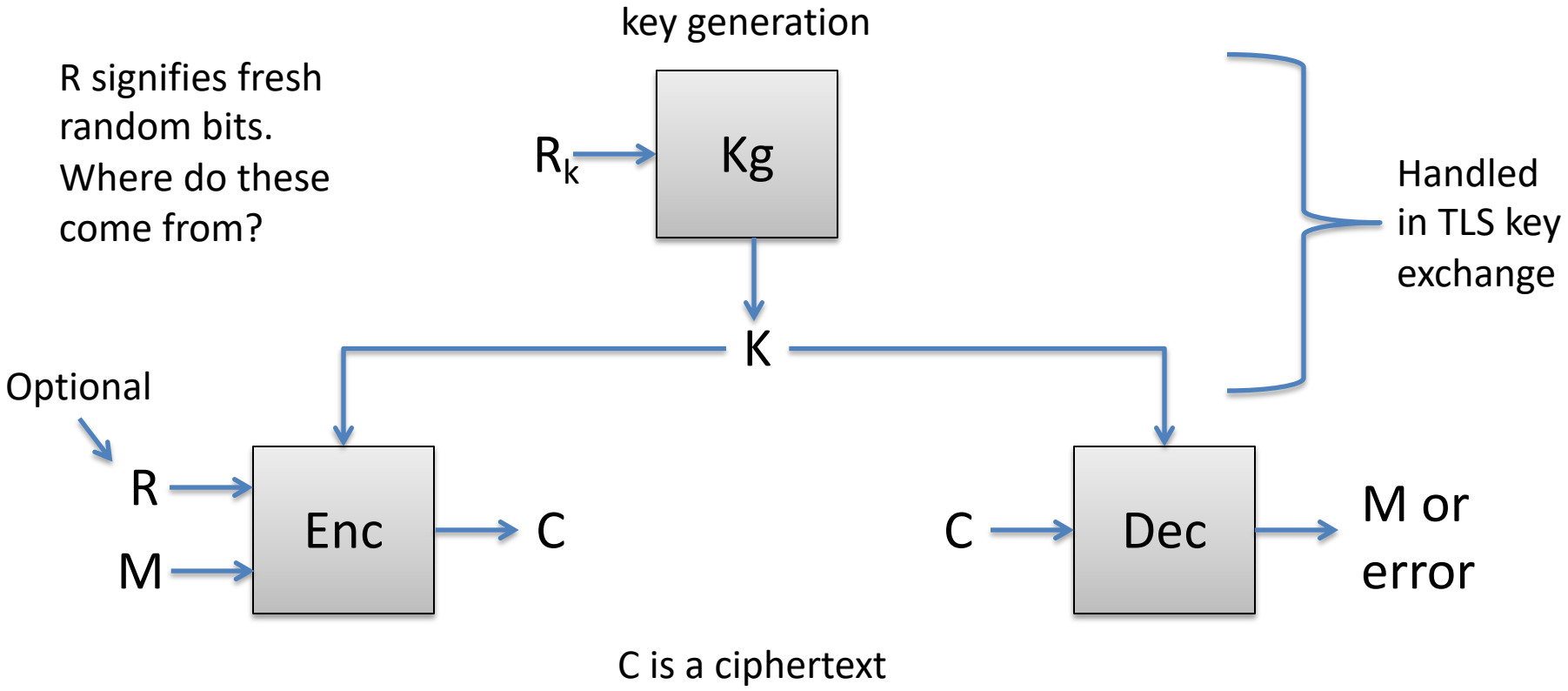
Message <- Dec(K1,C1)

C2 <- Enc(K2,Message’)

C2

Message’ <- Dec(K2,C2)

Symmetric encryption



Correctness: Dec(K , Enc(K,R,M)) = M with probability 1 over randomness R used

Kerckhoffs' principle: what parts are public and which are secret?

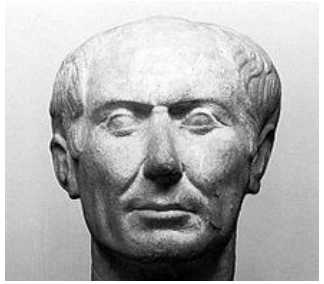
Some attack settings, high level

Given ciphertext C of unknown plaintext, want to recover (information about) plaintext

- Unknown plaintext
  - attacker only sees some ciphertexts
- Known plaintext
  - Attacker gets some plaintext-ciphertext pairs
- Chosen plaintext
  - Attacker can see encryption examples of chosen plaintexts
- Chosen plaintext & ciphertext
  - Attacker can see encryption/decryption examples of chosen plaintexts/ciphertexts

Substitution ciphers

Julius Caeser



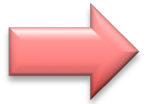
Kg: output randomly chosen permutation of digits

K =

0	1	2	3	4	5	6	7	8	9	plaintext digit
8	2	7	4	1	6	0	5	9	3	ciphertext digit

E(K, 2321-4232-1340-1410) = 7472-1747-2418-2128

Jane Doe	2414-2472-2742-7428
Thomas Ristenpart	3612-4260-2478-7243
John Jones	6020-7412-7412-2728
Eve Judas	7472-1747-2418-2128



1343-1321-1231-2310

Knowing one plaintext, ciphertext pair leaks key material!

Attacker knows 2321-4232-1340-1410

7472-1747-2418-2128

0	1	2	3	4	5	6	7	8	9
?	?	?	?	?	?	?	?	?	?

WWII Enigma machine built by Germans

Polyalphabetic substitution cipher

- Substitution table changes from character to character
- Rotors control substitutions

Allies broke Enigma (even before the war), significant intelligence impact

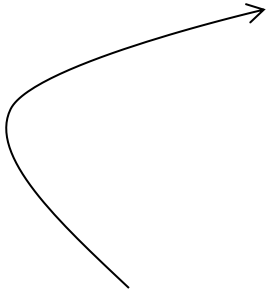
Computers were built to break WWII ciphers, by Alan Turing and others

More modern approach to cryptography

Supplement "design-break-redesign-break…" with a more mathematical approach

1. Design a cryptographic scheme
2. Provide proof that no one is able to break it

} Shannon 1949



Formal definitions

Scheme semantics

Security notions

Security proofs

Show it is mathematically impossible to break security

Analogous to (mathematical or physics) models

- Necessarily abstract view of computers & adversarial capabilities
- We can prove things, but only relative to model
- Must interpret results via utility of model to security in applications

One-time pads

Fix some message length L

Kg: output random bit string K of length L

$$Enc(K,M) = M \oplus K \qquad Dec(K,C) = C \oplus K$$

Exclusive OR operation

Shannon's security notion

Def. A symmetric encryption scheme is perfectly secure if for all messages M,M' and ciphertexts C

$$\Pr[\,\mathrm{Enc}(K,M) = C\,] = \Pr[\,\mathrm{Enc}(K,M') = C\,]$$

where probabilities are over choice of K

In words:
each message is equally likely to map to a given ciphertext

In other words:
seeing a ciphertext leaks nothing about what message was encrypted

Does a substitution cipher meet this definition? ***No!***

Shannon's security notion

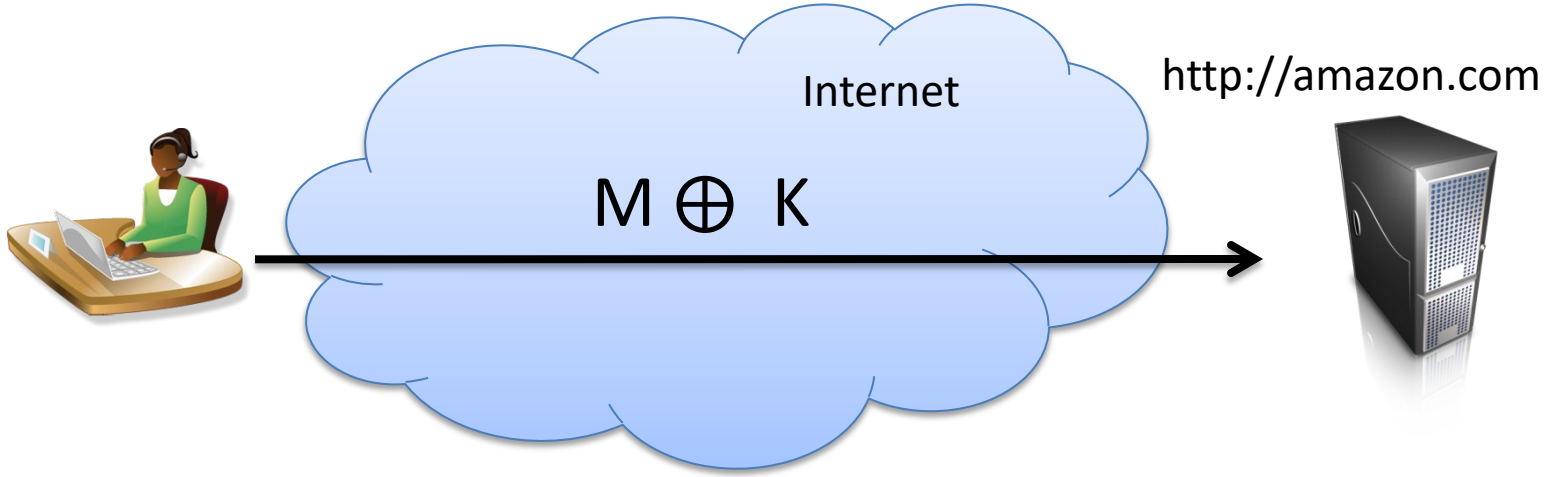
Def. A symmetric encryption scheme is perfectly secure if for all messages M,M' and ciphertexts C

$$\Pr[\,\mathrm{Enc}(K,M) = C\,] = \Pr[\,\mathrm{Enc}(K,M') = C\,]$$

where probabilities are over choice of K

Thm. OTP is perfectly secure

Thm. To be perfectly secure, encryption key must be as long as message



OTP is perfectly secure. Does OTP suffice for protecting Internet communications?

Integrity easily violated

Reuse of K for messages M,M' leaks M ⊕ M'

Encrypting same message twice under K leaks the message equality

K must be as large as message

Message length revealed

Cryptography as computational science

Use computational intractability as basis for confidence in systems

1. Design a cryptographic scheme
2. Provide proof that no attacker with limited computational resources can break it

} Goldwasser, Micali and Blum circa 1980's

Formal definitions

Scheme semantics

Security

Security proofs (reductions)

Breaking scheme

Breaking assumptions

Example:

Attacker can not recover credit card

Can fact factor large composite numbers

But no one knows how to do this. It's been studied for a very long time!

As long as assumptions hold and security model is good we believe in security of scheme!

Typical assumptions

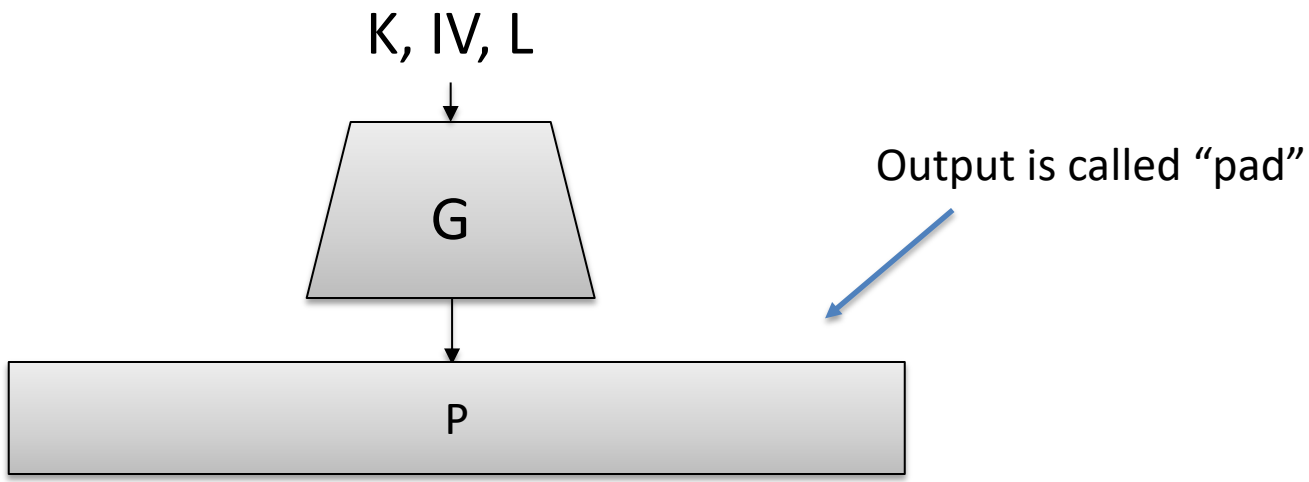
- Basic primitives that we believe are hard to break:
  - Large composites hard to factor
  - RSA permutation hard-to-invert
  - Discrete log of elliptic curve group points hard to recover
  - Block ciphers (AES, DES) are good pseudorandom permutations (PRPs)
  - Hash functions are collision resistant

Confidence in low-level primitives gained by cryptanalysis, public design competitions (AES & SHA-3 competitions)

Stream ciphers

Stream cipher (aka pseudorandom generator, PRG) is pair of algorithms:

- Kg outputs a random key K
- G(K,IV,L) takes K, optionally an additional random value IV (initialization vector), desired length L
  - Outputs bit string P with |P| = L

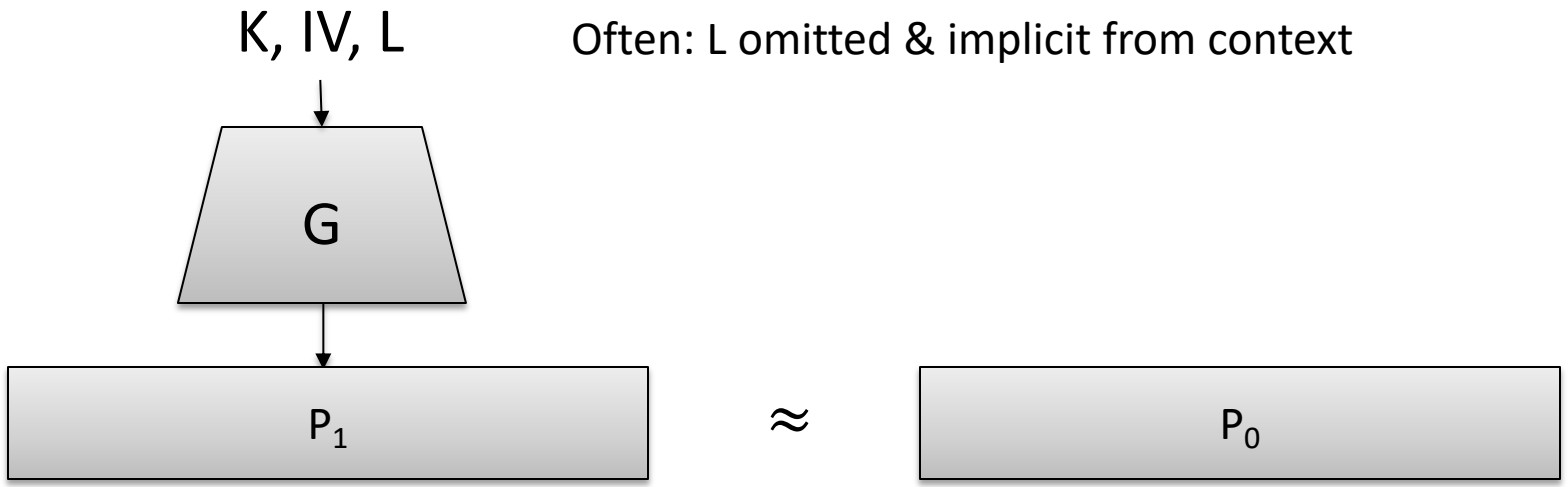


Stream cipher security goal

No computationally-limited attacker can distinguish between

IV, G(K,IV,L) and IV, P_0

for P_0 random length-L bit string, with good probability



Intuition: G output as good as one-time pad R (uniformly sampled)

Implies that no large ***biases*** exist, e.g. Pr[$P_1[0]$ = 0x00] = $\frac{1}{256} + \delta$

RC4 stream cipher

- Rivest Cipher 4 (1987)
  - No explicit IV input
    - RC4(K,IV) uses K || IV as key with RC4
  - Outputs one byte at a time
    - L is implicit: call L times to get L bytes
  - All addition mod 256
- Originally proprietary secret
  - Leaked to CypherPunks mail list (1994)
  - "Alleged RC4" or ARCFOUR
  - Rivest confirmed history later
- Very simple and fast
- Used widely, only recently deprecated

Algorithm 1: RC4 key scheduling (KSA)

input : key K of l bytes
output : initial internal state st_0
begin
  for $i = 0$ **to** 255 **do**
    $\mathcal{S}[i] \leftarrow i$
  $j \leftarrow 0$
  for $i = 0$ **to** 255 **do**
    $j \leftarrow j + \mathcal{S}[i] + K[i \bmod l]$
    $\text{swap}(\mathcal{S}[i], \mathcal{S}[j])$
  $i, j \leftarrow 0$
  $st_0 \leftarrow (i, j, \mathcal{S})$
  return st_0

Algorithm 2: RC4 keystream generator (PRGA)

input : internal state st_r
output : keystream byte Z_{r+1} updated internal state st_{r+1}
begin
  parse $(i, j, \mathcal{S}) \leftarrow st_r$
  $i \leftarrow i + 1$
  $j \leftarrow j + \mathcal{S}[i]$
  $\text{swap}(\mathcal{S}[i], \mathcal{S}[j])$
  $Z_{r+1} \leftarrow \mathcal{S}[\mathcal{S}[i] + \mathcal{S}[j]]$
  $st_{r+1} \leftarrow (i, j, \mathcal{S})$
  return (Z_{r+1}, st_{r+1})

From:
http://www.isg.rhul.ac.uk/tls/RC4passwords.pdf

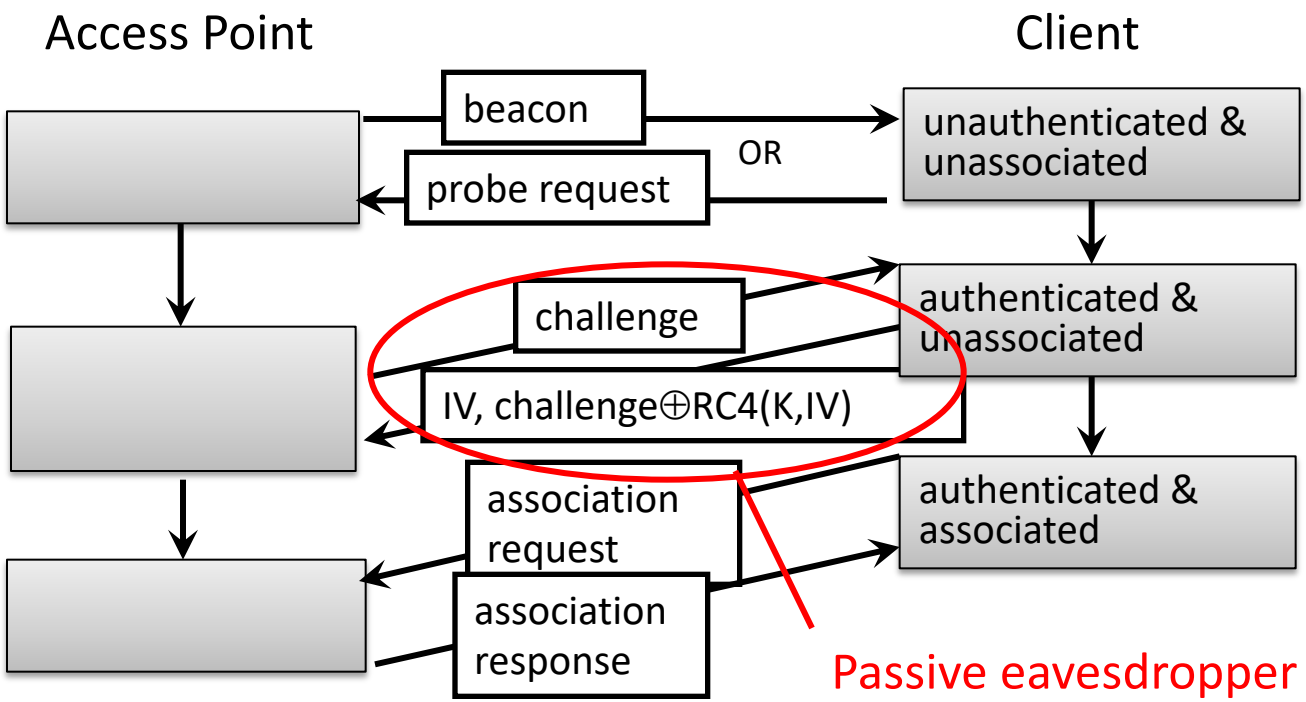
Misusing stream cipher case study: WEP (Wired Equivalent Privacy)

- Original Wi-Fi encryption in the 802.11b standard
- Goals: confidentiality, integrity, authentication
  - Intended to make wireless as secure as wired network
- Assumes that a secret key is shared between access point and client
- Uses RC4 stream cipher seeded with 24-bit initialization vector and 40-bit key

Bad design choice for wireless environment

Shared Key Authentication

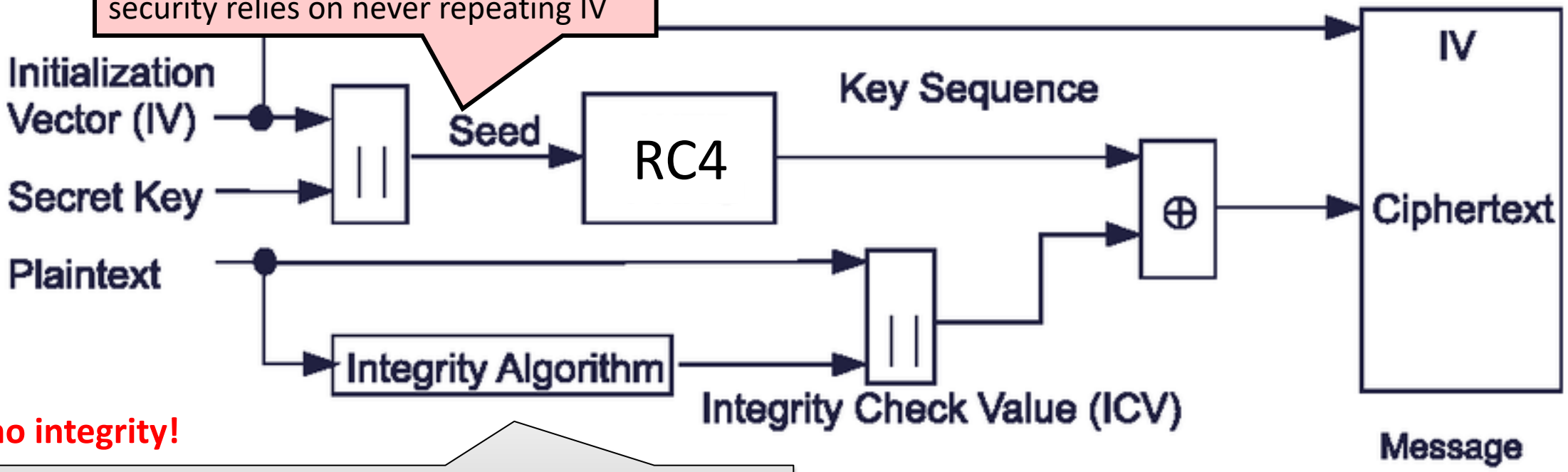
Prior to communicating data, access point may require client to authenticate



Passive eavesdropper recovers RC4(IV,K), can respond to any subsequent challenge without knowing K

How WEP encryption works

(IV, shared key) used as RC4 seed
Must never be repeated (why?)
There is no key update protocol, so security relies on never repeating IV



no integrity!

CRC-32 checksum is linear in ⊕:
if attacker flips some plaintext bits, knows which bits of CRC to flip to produce the same checksum

RC4 Was a Bad Choice for Wireless

Stream ciphers require sender and receiver to be at the same place in the keystream

- Not suitable when packet losses are common

WEP solution: a separate keystream for each packet (requires a separate seed for each packet)

- Can decrypt a packet even if a previous packet was lost

But there aren't enough possible seeds!

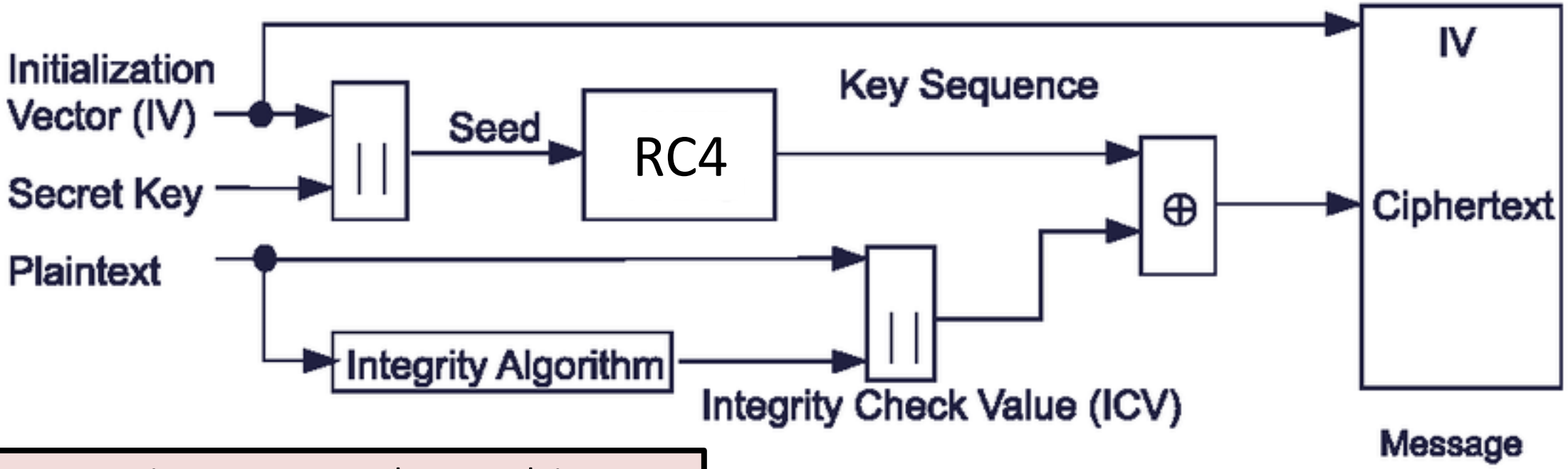
- RC4 seed = 24-bit initialization vector + fixed key
- Assuming 1500-byte packets at 11 Mbps,

  2^{24} possible IVs will be exhausted in about 5 hours

Seed reuse is catastrophic for stream ciphers

Recovering the key stream

Not a problem if the keystream is never re-used!



Get access point to encrypt a known plaintext M

- Send spam, AP will encrypt and forward
- Get victim to send an email with known content

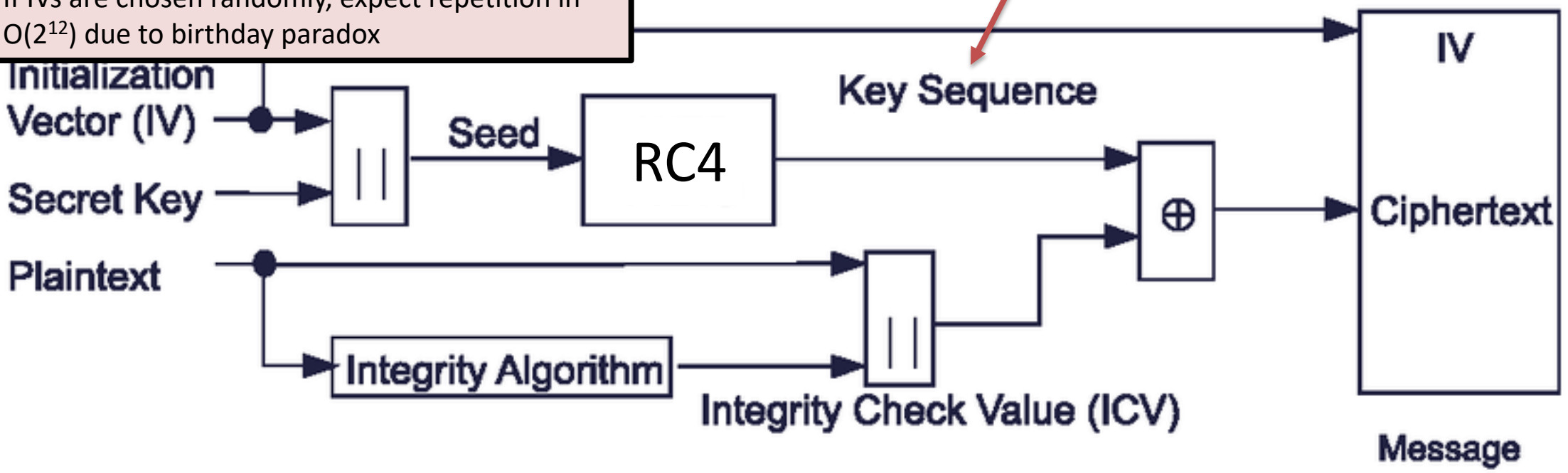
Recover keystream:
$C \oplus M = (M \oplus RC4(key,IV)) \oplus M = RC4(key,IV)$

Key stream reuse attack

Busy network will repeat IVs often

- Many cards reset IV to 0 when re-booted, then increment by 1 ⇒ expect re-use of low-value IVs
- If IVs are chosen randomly, expect repetition in $O(2^{12})$ due to birthday paradox

In WEP, repeated IV = repeated keystream



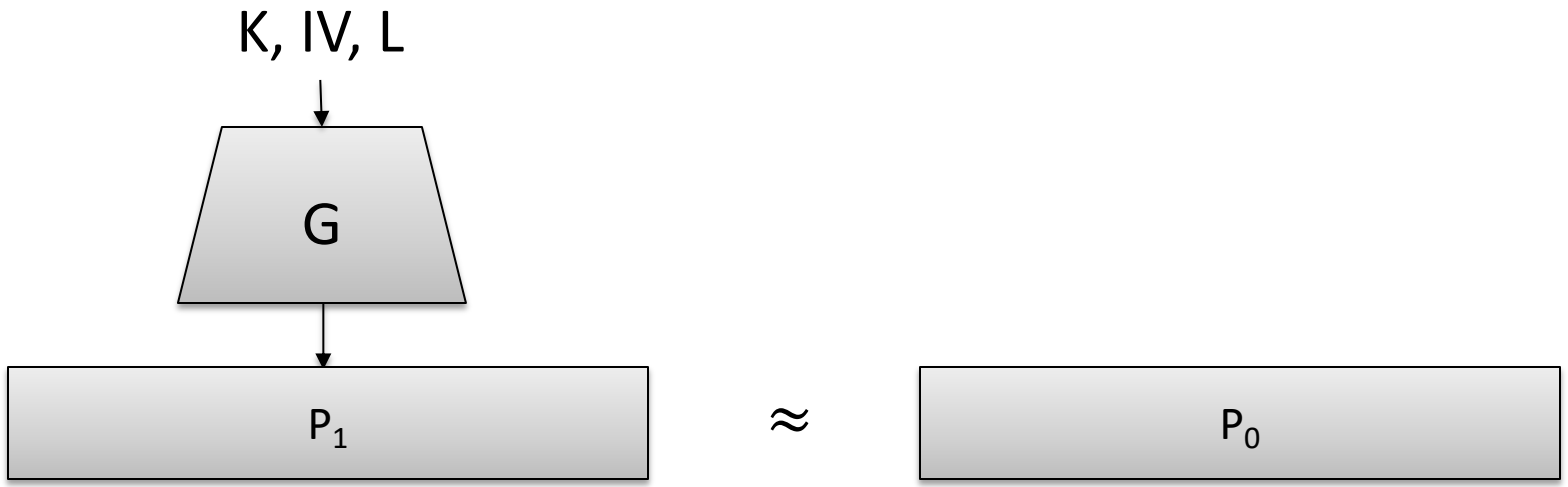
Recover keystream for each IV, store in table:
(KnownM ⊕ RC4(key,IV)) ⊕ KnownM = RC4(key,IV)

Wait for IV to repeat, decrypt, enjoy plaintext:
(M' ⊕ RC4(IV,key)) ⊕ RC4(IV,key) = M'

Beyond misuse: stream cipher security

- Someone gives you a stream cipher G
- How would you go about analyzing its security?
  1. **Key recovery attacks**: given IV, P (for some hidden K), recover K. How would you do it? What is run time of attack?
  2. **Bias-finding attacks**: can we find biases? How would you go about it?

E.g.: Pr[$P_1[0]$ = 0x00] = $\frac{1}{256} + \delta$



Simplified view of bias-abusing attack

Assume stream cipher G with no IV and outputting single byte
Unknown byte M encrypted under q different keys K1,…,Kq
Adversary given C1,…,Cq

$C1 = G(K1) \oplus M$

$C2 = G(K2) \oplus M$

$C3 = G(K3) \oplus M$

$\vdots$

$Cq = G(Kq) \oplus M$

Suppose Pr[G(K) = 0x00] > Pr[G(K) = B] for any other B
Pad byte is most likely to be 0x00

Say Pr[G(K) = 0x00] = 1/3
How many ciphertexts do we expect to equal M? ~q/3

Simple attack:
Let N_r = number of ciphertexts that equal byte value r
Output r such that N_r is highest

Simplified view of bias-abusing attack

Assume stream cipher G with no IV and outputting single byte
Unknown byte M encrypted under q different keys K1,…,Kq
Adversary given C1,…,Cq

$C1 = G(K1) \oplus M$

$C2 = G(K2) \oplus M$

$C3 = G(K3) \oplus M$

$\vdots$

$Cq = G(Kq) \oplus M$

Alfardan et al. 2013 point out better approach

Build maximum likelihood estimator (MLE) that takes advantage of all known biases in G's output

In words:
Choose value M that most likely explains the C1,…,Cq seen given the known biases of G

http://www.isg.rhul.ac.uk/tls/RC4biases.pdf

RC4 stream cipher

- How do evaluate RC4 security?
  - Manual analysis:

    walk through code and analyze probabilities

    [Mantin-Shamir 2001]: $\Pr[Z_2 = 0x00] \approx 1/128$

  - Statistical tests:

    for a random key, do you get expected distribution of output bytes?

    for many random keys, do you get expected distribution for each byte?

    [AlFardan et al. 2013] calculated outputs for 2^{44} keys and reported on empirical distributions

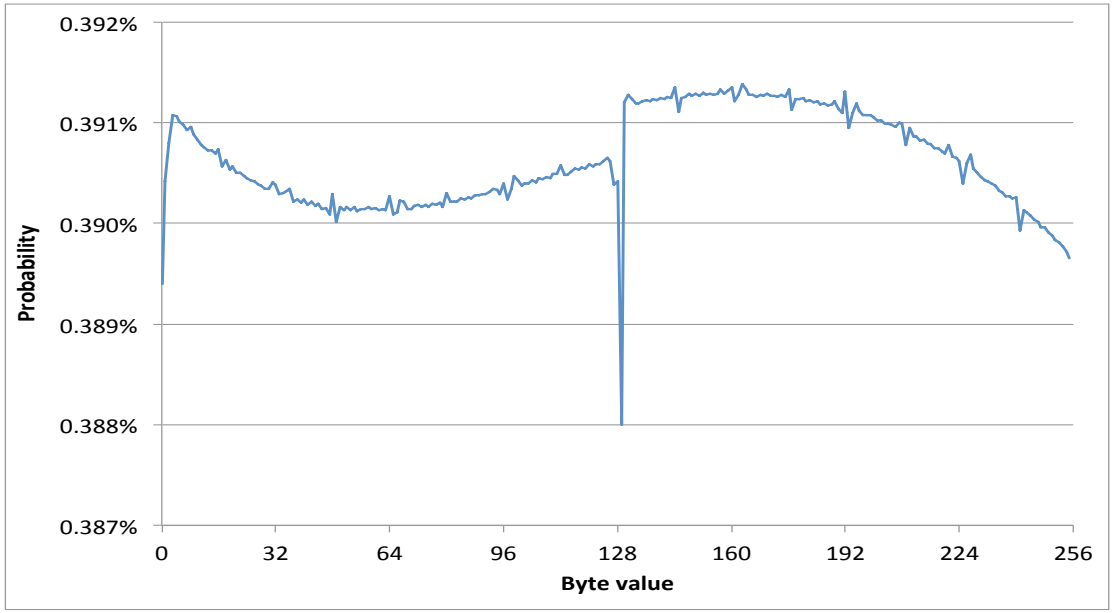
Algorithm 1: RC4 key scheduling (KSA)

input : key K of l bytes
output : initial internal state st_0
begin
  for $i = 0$ **to** 255 **do**
    $\mathcal{S}[i] \leftarrow i$
  $j \leftarrow 0$
  for $i = 0$ **to** 255 **do**
    $j \leftarrow j + \mathcal{S}[i] + K[i \bmod l]$
    $\text{swap}(\mathcal{S}[i], \mathcal{S}[j])$
  $i, j \leftarrow 0$
  $st_0 \leftarrow (i, j, \mathcal{S})$
  return st_0

Algorithm 2: RC4 keystream generator (PRGA)

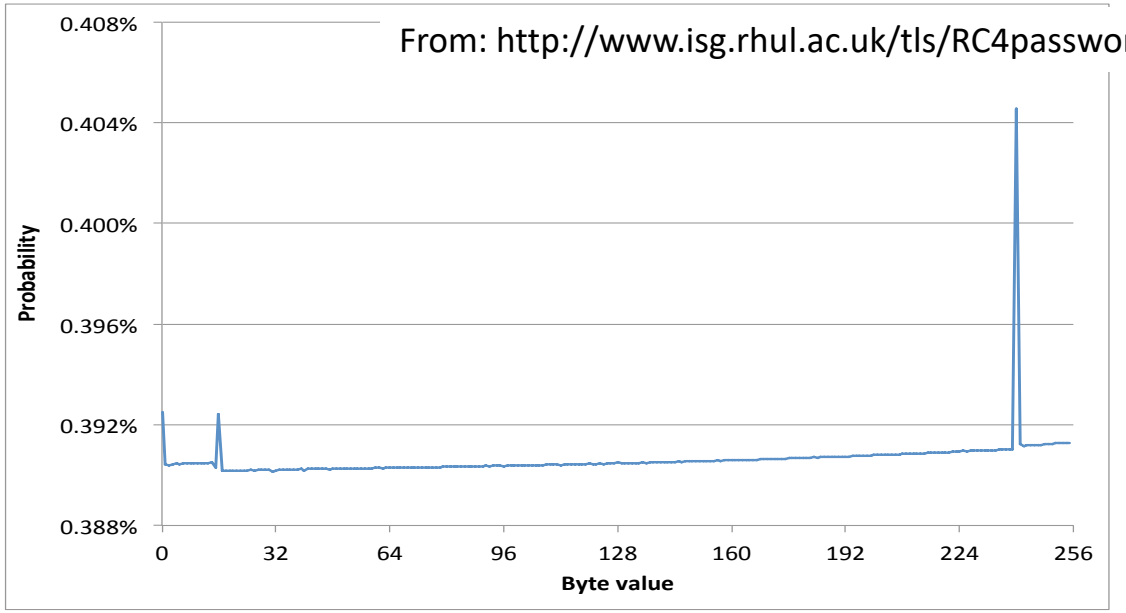
input : internal state st_r
output : keystream byte Z_{r+1} updated internal state st_{r+1}
begin
  parse $(i, j, \mathcal{S}) \leftarrow st_r$
  $i \leftarrow i + 1$
  $j \leftarrow j + \mathcal{S}[i]$
  $\text{swap}(\mathcal{S}[i], \mathcal{S}[j])$
  $Z_{r+1} \leftarrow \mathcal{S}[\mathcal{S}[i] + \mathcal{S}[j]]$
  $st_{r+1} \leftarrow (i, j, \mathcal{S})$
  return (Z_{r+1}, st_{r+1})

From: http://www.isg.rhul.ac.uk/tls/RC4passwords.pdf

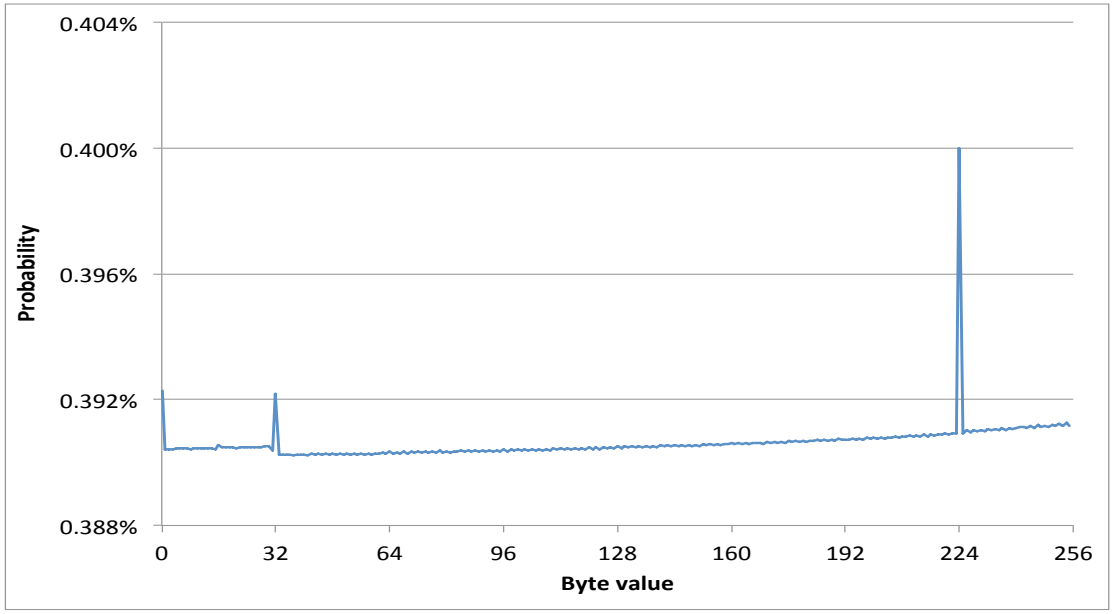


(a) Byte Z_1

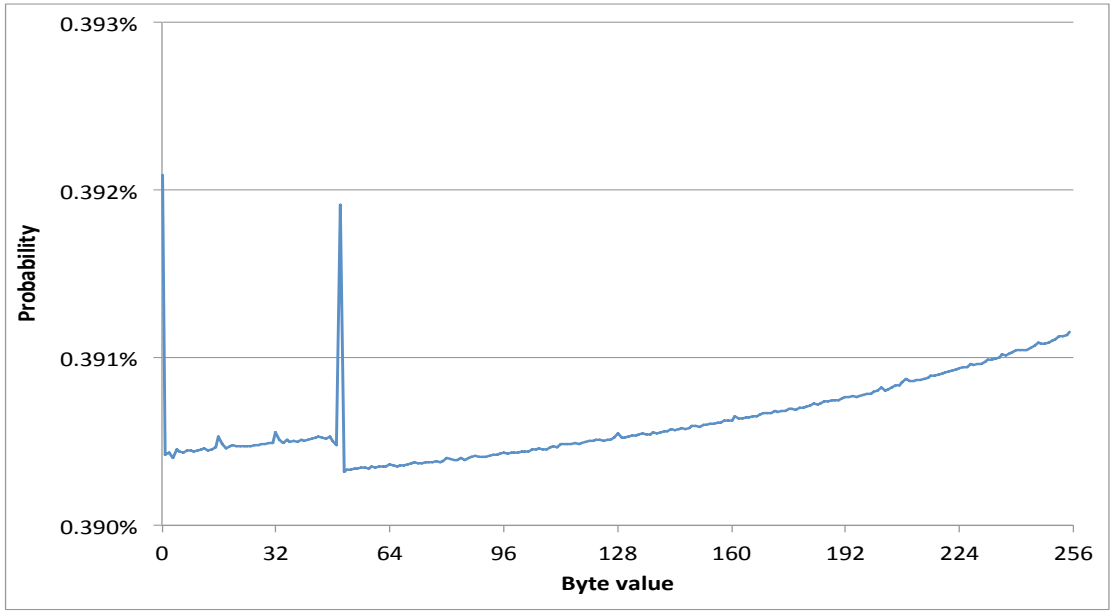
From: http://www.isg.rhul.ac.uk/tls/RC4passwords.pdf



(b) Byte Z_{16}



(c) Byte Z_{32}



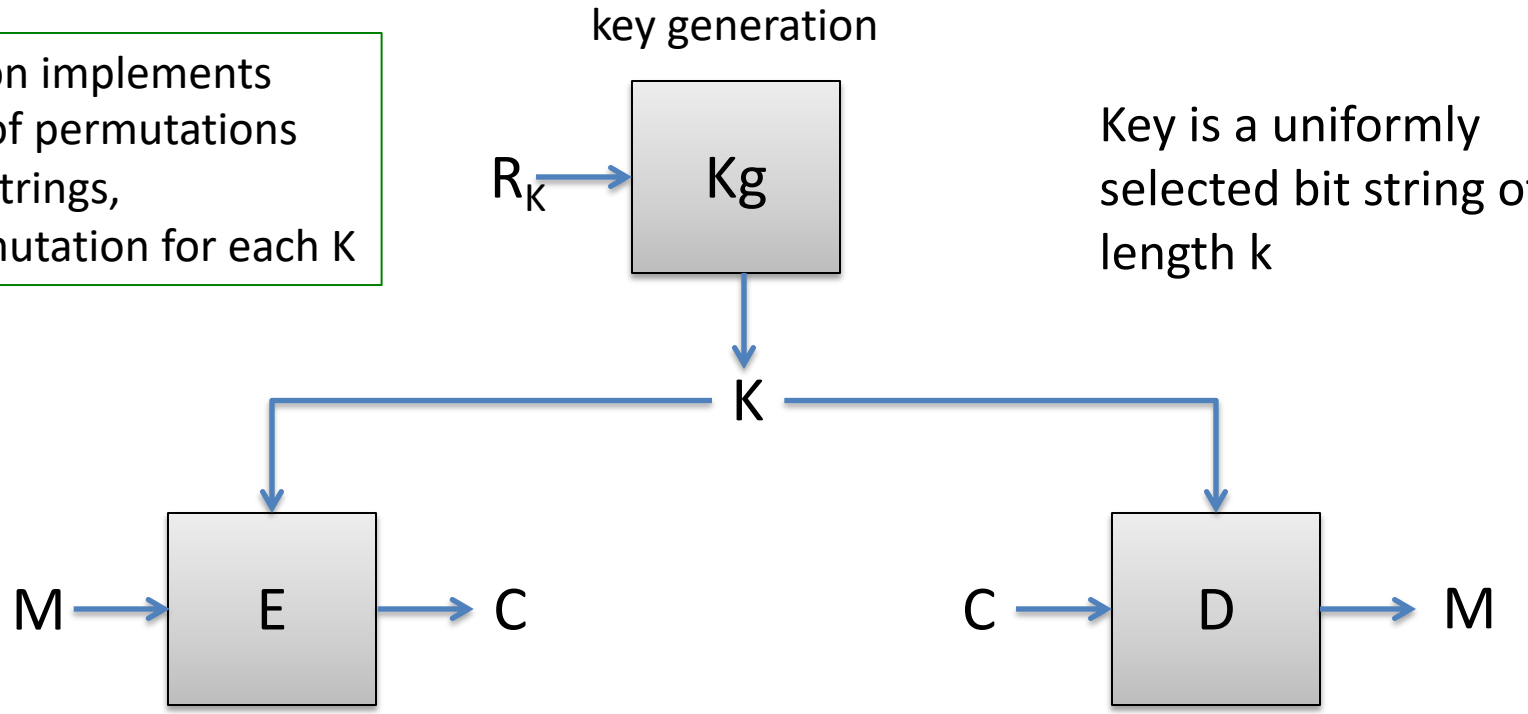
(d) Byte Z_{50}

Fallout from 2013 RC4 attacks

- RC4 was the most widely used encryption method for TLS at the time (circa 2013)
- Attack required about q = 2^{26} (67 million) to start recovering plaintexts
  - Not quite practical, but within realm of feasibility
  - Enough to be considered ***significant problem***, and potentially within reach of intelligence agencies
- Needed to move on from RC4, but all other TLS encryption methods had even more severe security problems (stay tuned)
- Have replaced now with encryption based on ***block ciphers***

Block ciphers

Encryption implements
a family of permutations
on n bit strings,
one permutation for each K



Key is a uniformly
selected bit string of
length k

$E: \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$

Data encryption standard (DES)

Originally called Lucifer
- team at IBM
- input from NSA
- standardized by NIST in 1976

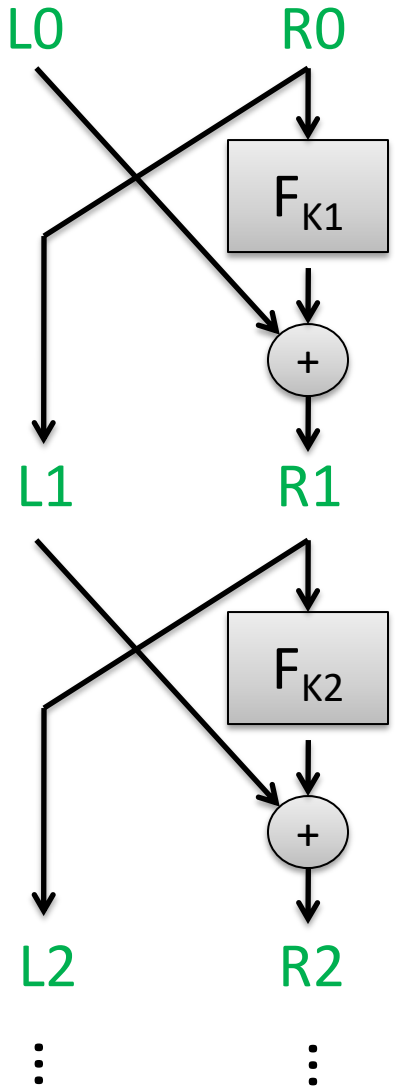
n = 64
k = 56

Number of keys:
72,057,594,037,927,936

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

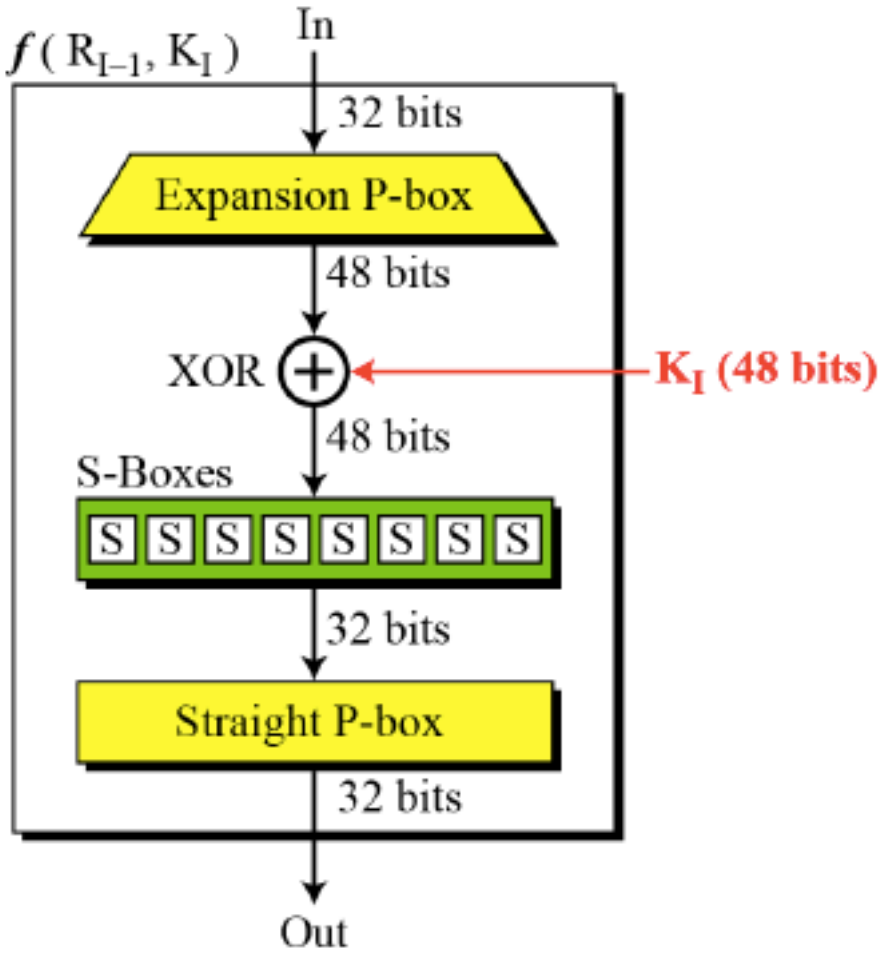
Repeat Feistel round 16 times

Each round applies function F using separate round key



DES round functions

- P-box expands 32 bits to 48 bits and permutes
- S-boxes: 6-bit to 4-bit lookup tables
- XOR in round key
  - 16 48-bit round keys derived via key schedule from 56 bit key deterministically
- How S-boxes chosen? Why particular permutations?
  - Resist cryptanalytic attacks known to NSA at the time (discovered later in 1990s)
  - Differential cryptanalysis



Best attacks against DES

Attack	Attack type	Complexity	Year
Biham, Shamir	Chosen plaintexts, recovers key	2^{47} plaintext, ciphertext pairs	1992
Matsui	Known plaintext, ciphertext pairs, recovers key	2^{42} plaintext, ciphertext pairs, ~2^{41} DES computations	1993
DESCHALL	Unknown plaintext, recovers key	$2^{56/4}$ DES computations 41 days	1997
EFF Deepcrack	Unknown plaintext, recovers key	~4.5 days	1998
Deepcrack + DESCHALL	Unknown plaintext, recovers key	22 hours	1999

- DES is still used in some places
- 3DES (use DES 3 times in a row with more keys) expands keyspace and still used widely in practice

Advanced Encryption Standard (AES)

Response to 1999 attacks:

- NIST has design competition for new block cipher standard
- 5 year design competition
- 15 designs, Rijndael design chosen

Advanced Encryption Standard (AES)

A form of key-alternating cipher

n = 128

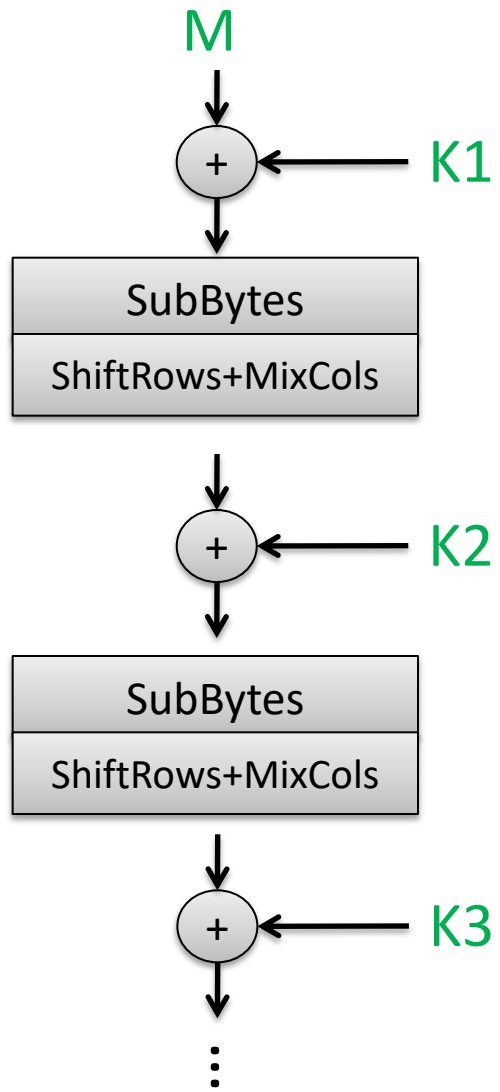
k = 128, 192, 256

Number of keys for k=128:
340,282,366,920,938,463,463,374,607,431,768,211,456

Substitution-permutation design.
For k=128 uses 10 rounds of:

1) SubBytes (non-linear 8-bit S-boxes)
2) ShiftRows & MixCols (linear permutation)
3) XOR'ing in a round key

(Last round skips MixCols)

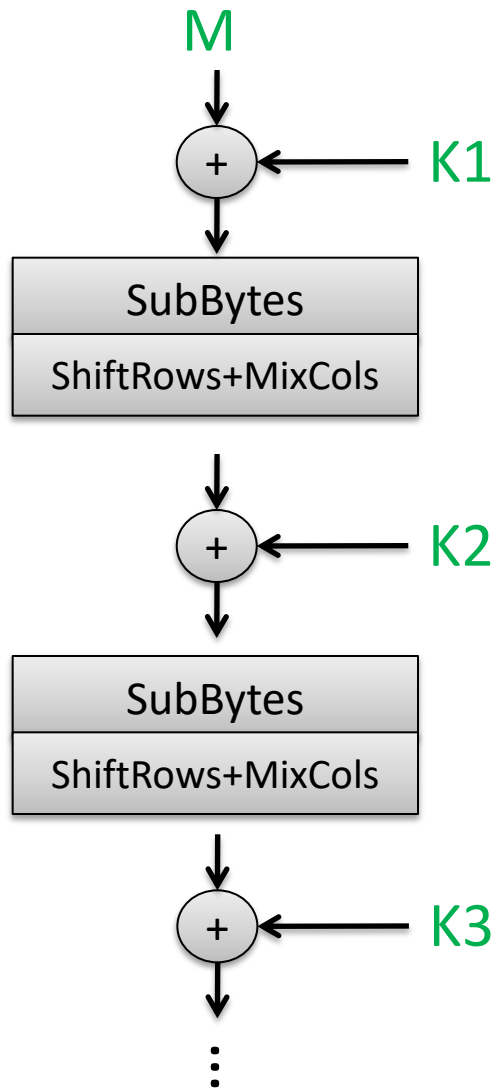


Advanced Encryption Standard (AES)

Designed to resist linear & differential cryptanalysis

"Wide-trail" strategy

- Ensure large # of Sboxes involved in multi-round trail (sequence of intermediate state bits)
- Use coding theory viewpoint to build permutations to ensure rapid ***diffusion***



Best attacks against AES

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

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

Summary and next time

- Crypto as computational science
- Overview of TLS
- Symmetric encryption, stream ciphers, and block ciphers
- Next time:
  - Symmetric encryption from block ciphers
  - Need for active attack security