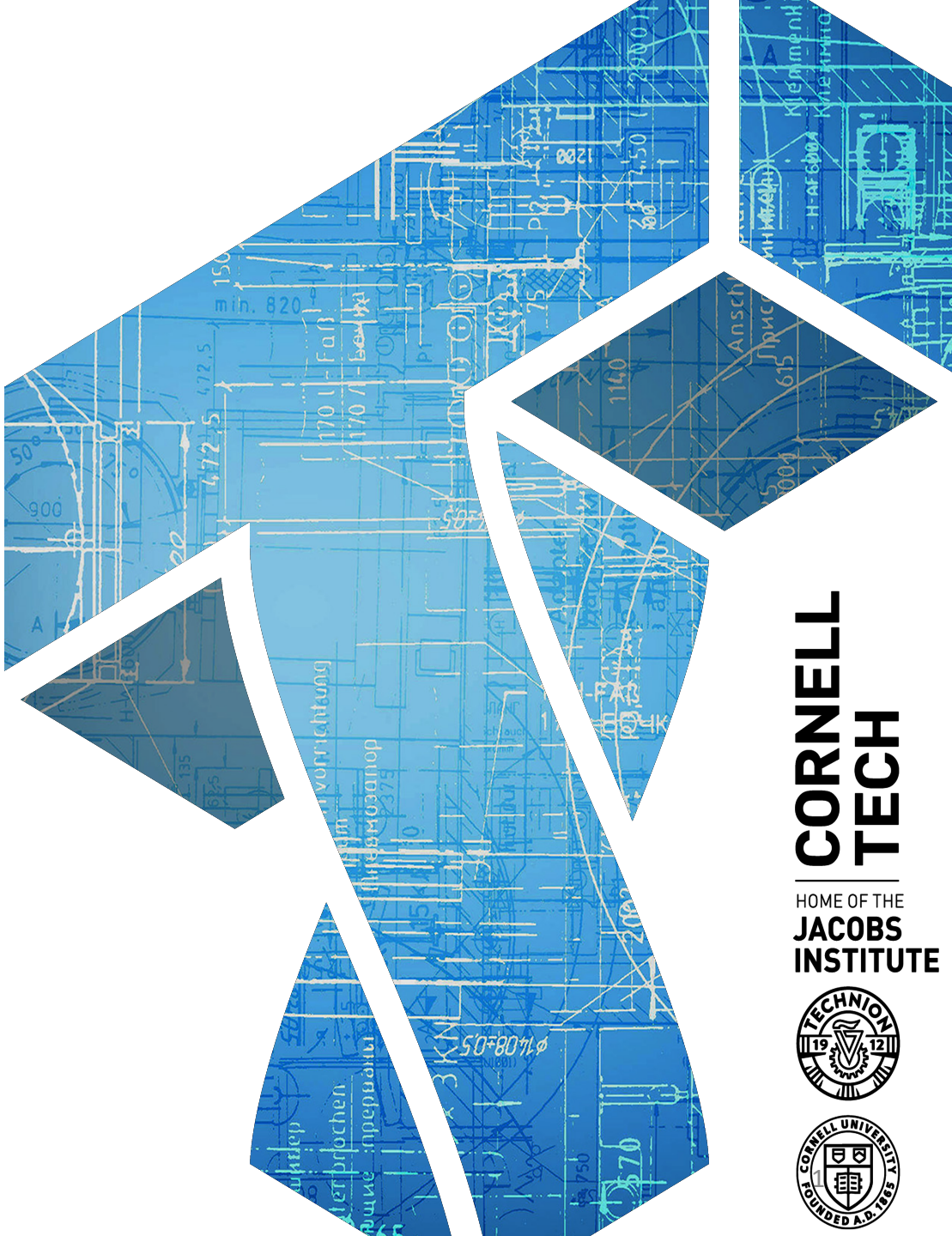
CS 5830
Cryptography

Instructor: Tom Ristenpart
TAs: Yan Ji, Sanketh Menda



CORNELL TECH
HOME OF THE JACOBS INSTITUTE

Administrivia

- Slack channel invitation (email one of the TAs if you have trouble)
  - Will use for announcements
  - Feel free to use general channels for questions, DM is fine too. We will try to be responsive and let you know if we can't
- Office hours will be announced, in mean time just message us
- Tentative homework schedule on course schedule now
  - Will try to stick to announced schedule
  - Let us know if there are large conflicts

The many types of attackers & threat models

- Threat model describes adversarial capabilities and goals
- **Example capabilities:**
  - Passive attackers observe communications / data
  - Active attackers may additionally interact with systems
    - Chosen-plaintext attacks can see input-output examples for adversarial plaintexts
    - Chosen-ciphertext attacks allow submitting mauled ciphertexts to decryption box
- **Example goals:**
  - Confidentiality violation (recover keys, recover plaintexts)
  - Integrity violation (trick recipient into accepting modified plaintext)
- Modern cryptography tries to formalize threat models

Shannon's security notion (1949)

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

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

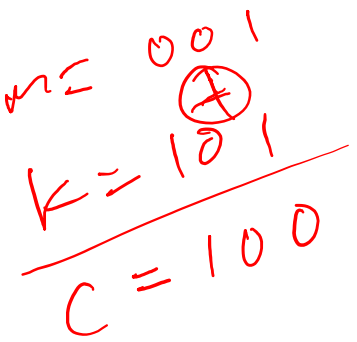
where probabilities are over choice of K

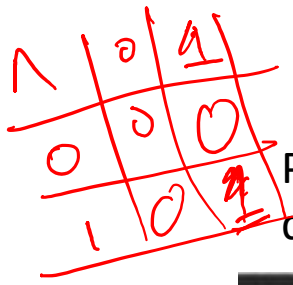
In words:
each message is equally likely to map to a given ciphertext

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

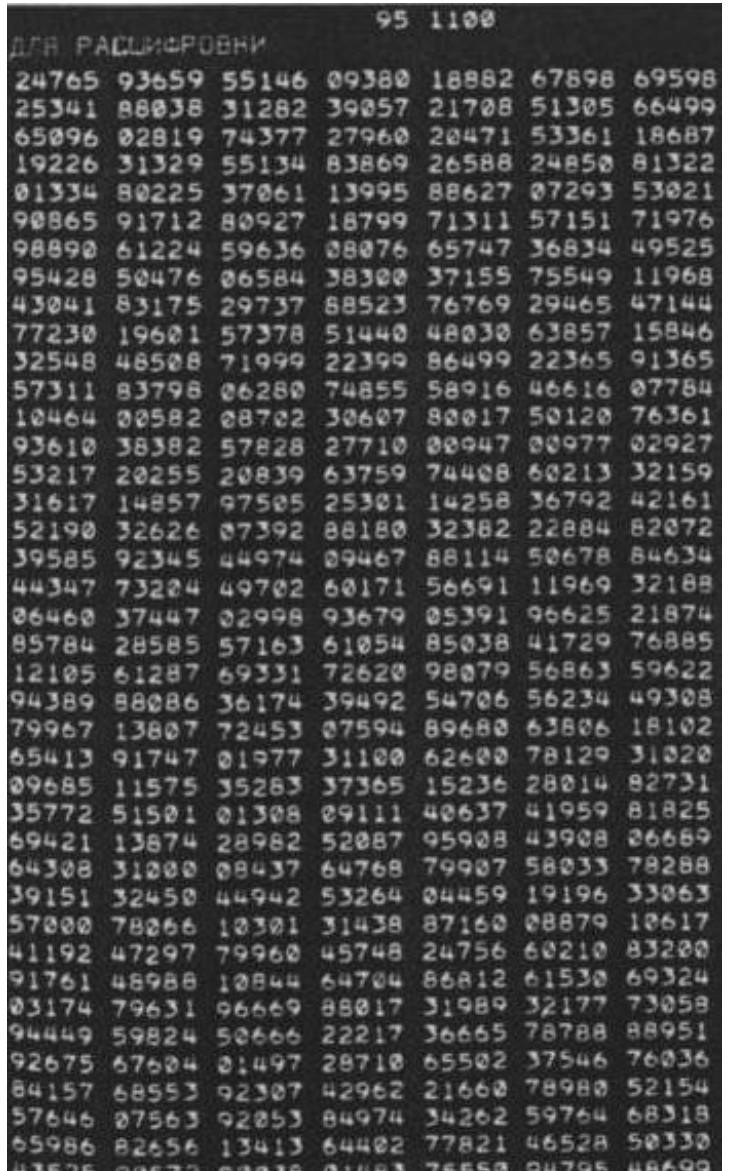
Does a substitution cipher meet this definition? No!

One-time pad (OTP)





Part of a CIA OTP used by Soviet diplomat spying for CIA



Kg():

K <-$ $\{0,1\}^L$

Pick a random bit string

Enc(K,M):

Return M $\oplus$ K

Assume M is L-bit string

Dec(K,C):

Return C $\oplus$ K

Assume C is L-bit string

Shannon's security notion (1949)

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

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

where probabilities are over choice of K

Thm. OTP is perfectly secure

For any C and M of length L bits

$$\Pr[\, K \oplus M = C \,] = 1 / 2^L$$

$$\Pr[\, K \oplus M = C \,] = \Pr[\, K \oplus M' = C \,]$$

Shannon's security notion (1949)

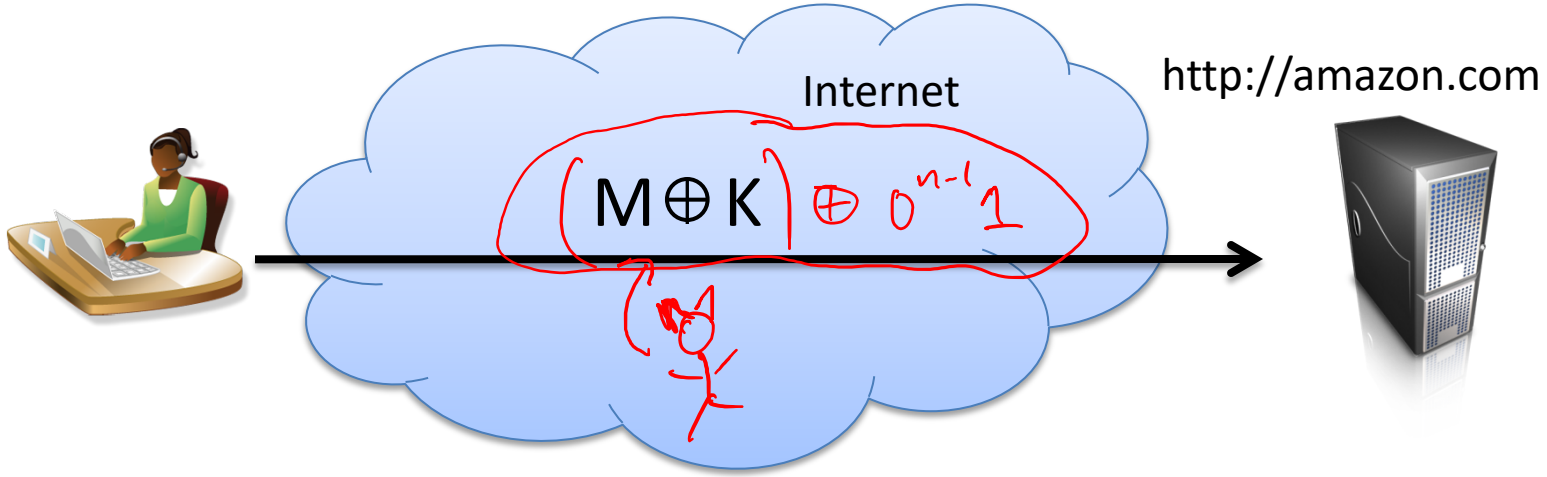
Def. A symmetric encryption scheme Enc is perfectly secure if for all messages M, M' and ciphertexts C

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

where probabilities are over choice of K

Thm. OTP is perfectly secure

Thm. Perfectly secure encryption requires $|K| \geq |M|$



$C' = m \oplus k \oplus 0^{n-1}1$

$= (m \oplus 0^{n-1}1) \oplus k$

$C' \oplus k$

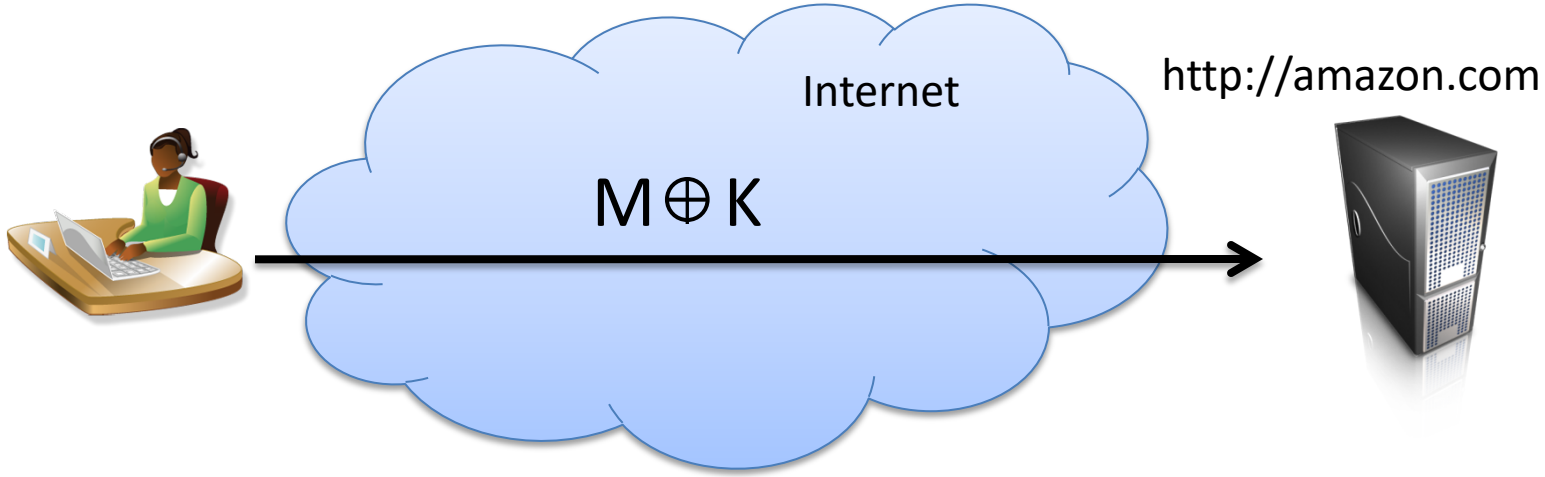
Does OTP suffice for securing communications online?

- Integrity easily violated
- Reuse of K for messages M,M' leaks $M \oplus M'$
  - Encrypting same message twice under K leaks the message equality
  - K must be as large as message
  - Message length revealed

Formal security notions are hard to get right

Simplifying abstraction to allow rigorous analysis

Amount of deployment details needed to capture attacks



Does OTP suffice for securing communications online?

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

Beyond Shannon

Thm. Perfectly secure encryption requires $|K| \geq |M|$

We will ***relax*** the definition of confidentiality:

1. Allow tiny adversarial success probability (often written as ϵ, e.g., $\epsilon = 2^{-80}$)
2. Focus on resource-efficient adversaries (e.g., only those given run time at most $t = 2^{80}$)

Intuition: we are ok if adversaries require prohibitive time to win, or only win with vanishingly small probability

Towards computational indistinguishability

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

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

where probabilities are over choice of K

Let's give a game-based formulation of this using an adversary

Let SE = (Kg,Enc,Dec) be a symmetric encryption scheme

Let $\mathcal{A}$ Let be a randomized algorithm, called the adversary

IND(SE,$\mathcal{A}$):
(M_0,M_1) <-$ $\mathcal{A}$
K <-$ Kg ; b <-$ {0,1}
b' <-$ $\mathcal{A}$(Enc(K,M_b))
Return (b = b')

IND(SE,$\mathcal{A}$) output is 1 if (b = b').

We say then that the adversary succeeded

Def. A scheme SE is perfectly secure if for every $\mathcal{A}$ it is the case that

$$\Pr[\text{IND}(SE,\mathcal{A}) = 1] = 1/2$$

Computational indistinguishability

Def. A symmetric encryption scheme is (t, ϵ)-indistinguishable if for any adversary $\mathcal{A}$ running in time at most t it holds that

$$\Pr[\text{IND}(\text{SE}, \mathcal{A}) = 1] < 1/2 + \epsilon$$

1) Tiny adversarial success

2) Computationally limited adversary

IND(SE, $\mathcal{A}$):
$(M_0, M_1) \leftarrow\$ \mathcal{A}$
$K \leftarrow\$ Kg$; $b \leftarrow\$ \{0,1\}$
$b' \leftarrow\$ \mathcal{A}(Enc(K, M_b))$
Return $(b = b')$

Discussion questions:

1) Does (t, ϵ)-indistinguishability model known, chosen message attack? What about chosen ciphertext?
2) Is OTP (t, ϵ)-indistinguishable?
3) Is a substitution cipher (t, ϵ)-indistinguishable?

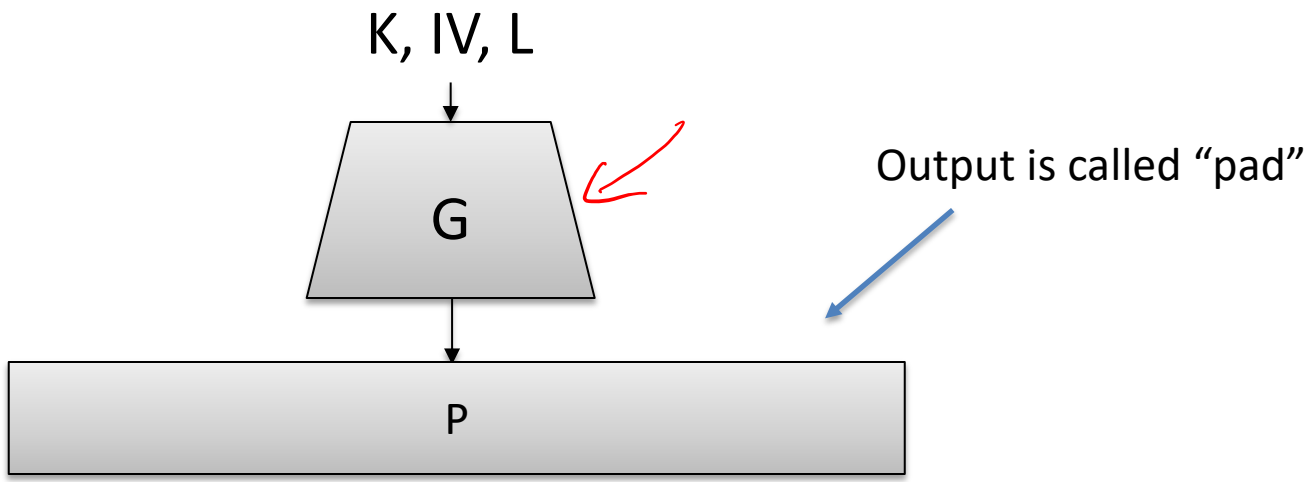
How do we build computationally-secure SE schemes?

- Game plan: need more tools
  - Stream ciphers
  - Block ciphers

Stream ciphers

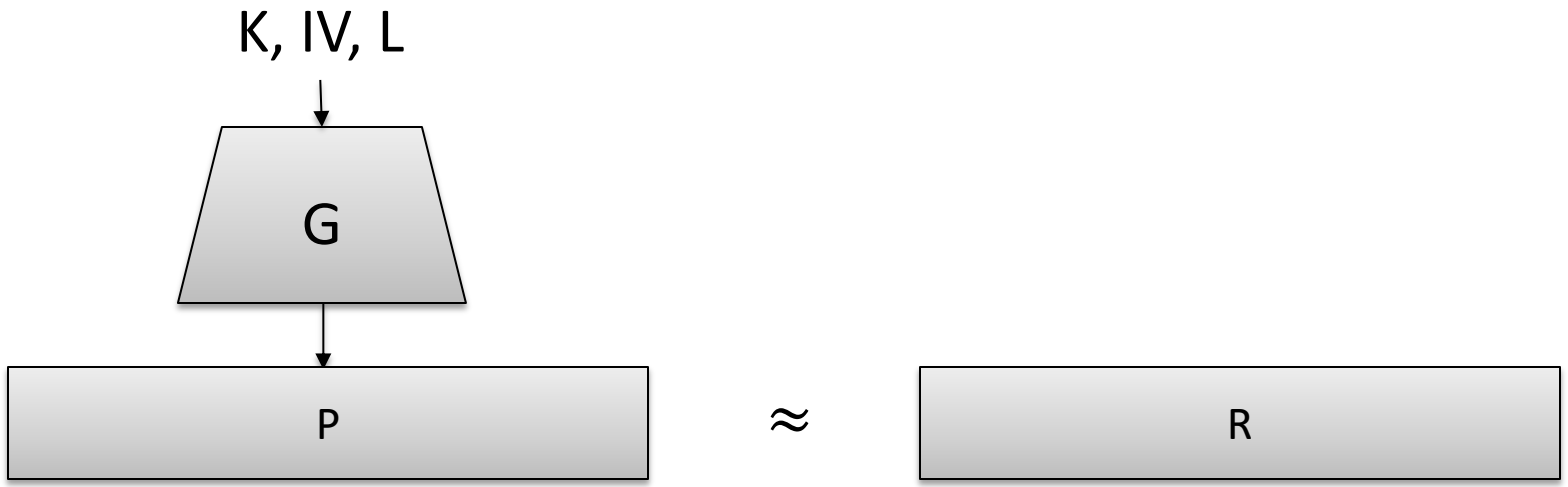
Stream cipher (aka pseudorandom generator) is pair of algorithms:

- 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

(t, ϵ)-pseudorandom: no attacker limited to time t can distinguish between IV, G(K,IV,L) and random bitstring R of length L with probability greater than ϵ



Intuition: P is as good as a true one-time pad R (uniformly sampled)

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

First byte of P

0000 0000

SE from a stream cipher

Say we have a secure stream cipher. How do we build an SE scheme?

<u>Kg():</u>
$K \xleftarrow{\$} \{0,1\}^k$

Pick a random key

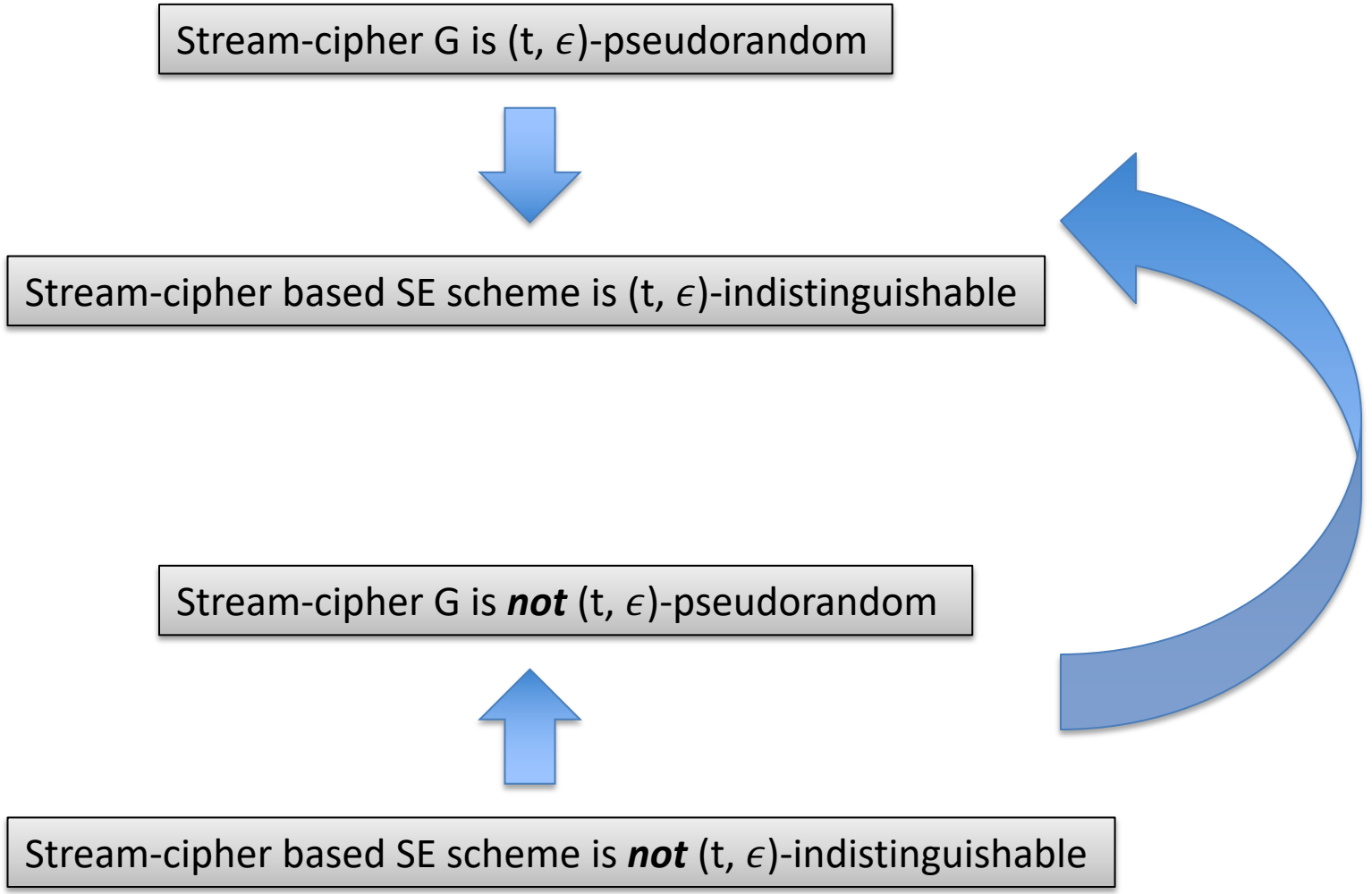
<u>Enc(K,M):</u>
$L \leftarrow |M|$
$IV \xleftarrow{\$} \{0,1\}^n$
Return $(IV, G(K,IV,L) \oplus M)$

<u>Dec(K,(IV,C)):</u>
$L \leftarrow |C|$
Return $G(K,IV,L) \oplus C$

Assume ciphertext can be parsed into IV and remaining ciphertext bits

Reduction-based analysis: high level idea

What we would want to show:



What we would show:

Reduction-based analysis: high level idea

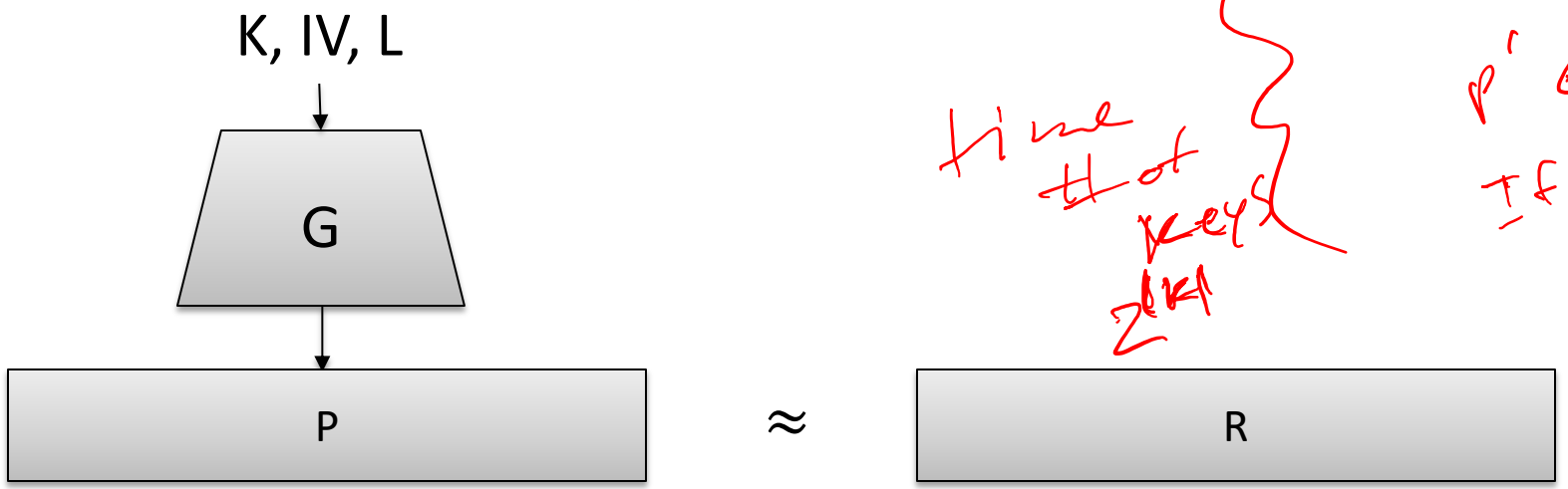
- **Reductionist analyses powerful tool**: rules out attacks that don't also break some (simpler to state) underlying hard computational problem
  - Often referred to as "provable security"
- Not a panacea:
  - Security definitions must capture relevant adversarial capabilities/goals
  - Proofs can sometimes be wrong
  - Couple with other approaches such as direct cryptanalysis

Reduces task of building indistinguishable encryption to building pseudorandom stream cipher

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.: $\Pr[\, P[0] = 0x00 \,] = \frac{1}{256} + \delta$



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

- Rivest Cipher 4 (1987)
  - No IV: can't reuse K
  - L is implicit: call generate L times
  - All addition mod 256
- Originally proprietary secret
- Leaked to CypherPunks mail list (1994)
  - "Alleged RC4" or ARCFOUR
  - Rivest confirmed history later
- Very simple, fast to implement in software
- 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

RC4 stream cipher

- Key scheduling algorithm uses K to build permutation *S* on {0,…,255} and initialize internal state (i,j,S)

  Interesting discussion of permutation generation:
  https://blog.codinghorror.com/the-danger-of-naivete/

- Keystream generator outputs byte Z_{r+1} and updated internal state
  - *S* remains permutation throughout
  - i is a counter
  - j varies as function of i, *S*, previous j

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

RC4 stream cipher

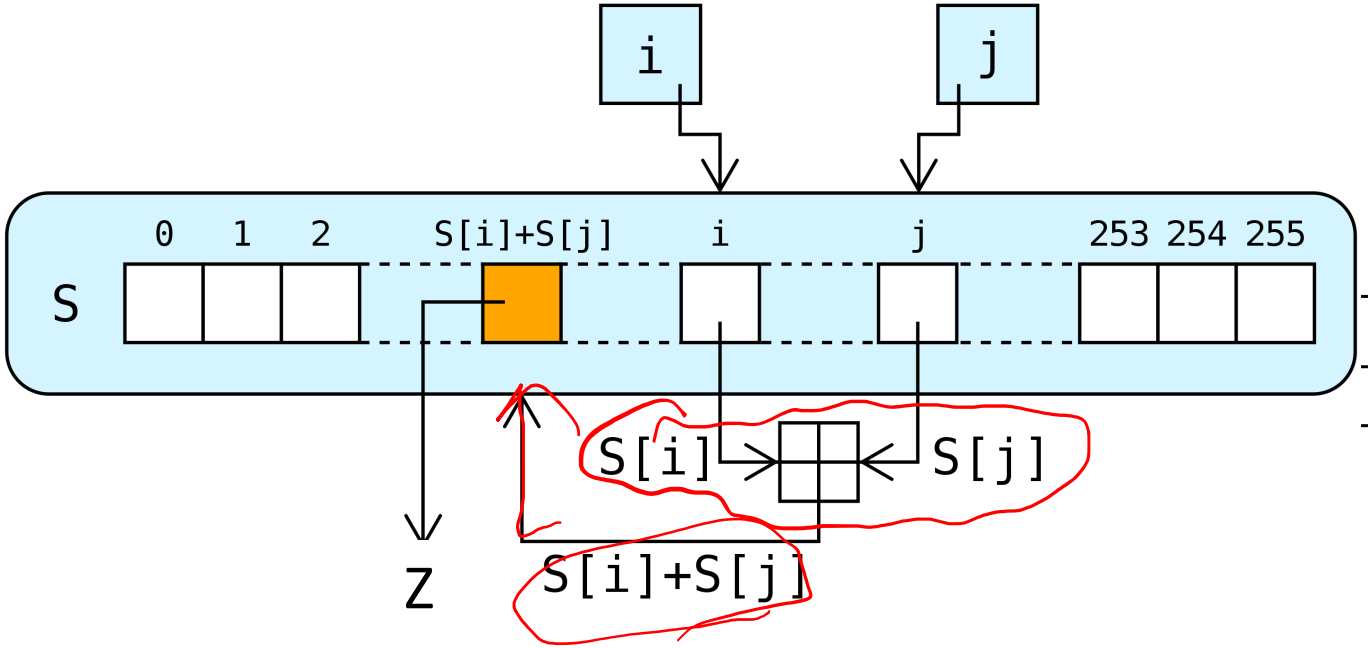


Diagram courtesy of https://en.wikipedia.org/wiki/RC4

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

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)

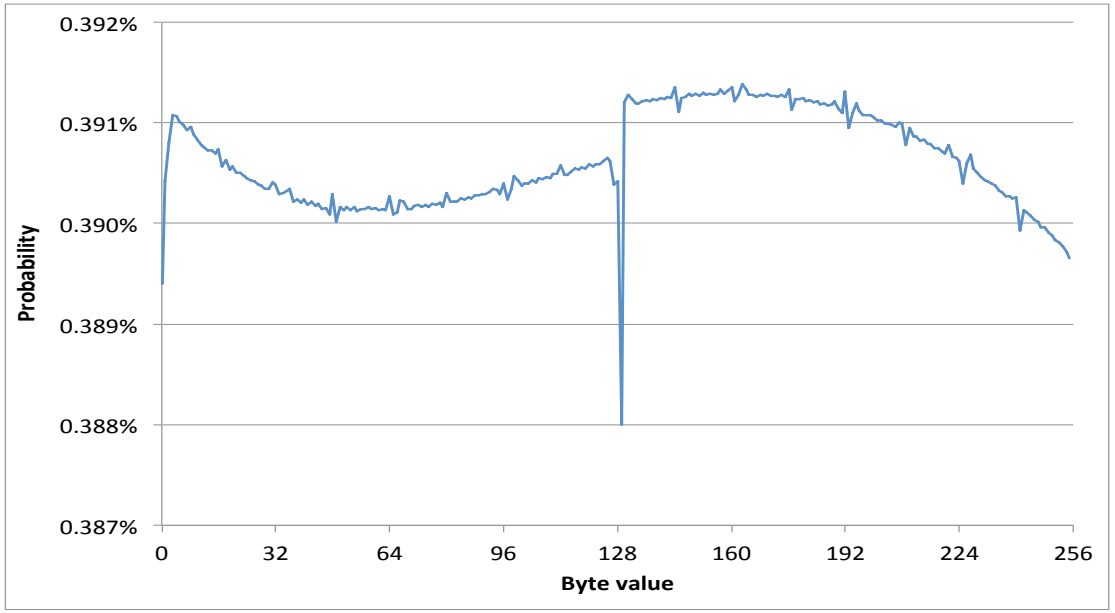
input : key K of l bytes
output : initial internal state st_0
begin
  for $i = 0$ **to** 255 **do**
    $\mathcal{S}[i] \leftarrow i$
  $j \leftarrow 0$
  for $i = 0$ **to** 255 **do**
    $j \leftarrow j + \mathcal{S}[i] + K[i \bmod l]$
    $\text{swap}(\mathcal{S}[i], \mathcal{S}[j])$
  $i, j \leftarrow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

Algorithm 2: RC4 keystream generator (PRGA)

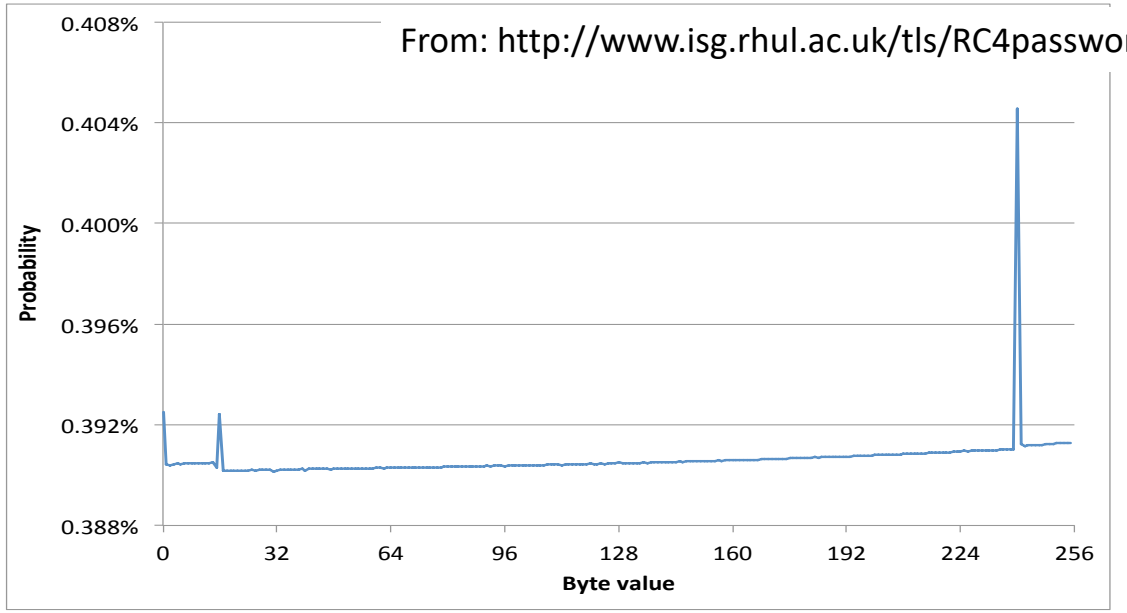
input : internal state st_r
output : keystream byte Z_{r+1} updated internal state st_{r+1}
begin
  parse $(i, j, \mathcal{S}) \leftarrow st_r$
  $i \leftarrow i + 1$
  $j \leftarrow j + \mathcal{S}[i]$
  $\text{swap}(\mathcal{S}[i], \mathcal{S}[j])$
  $Z_{r+1} \leftarrow \mathcal{S}[\mathcal{S}[i] +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

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

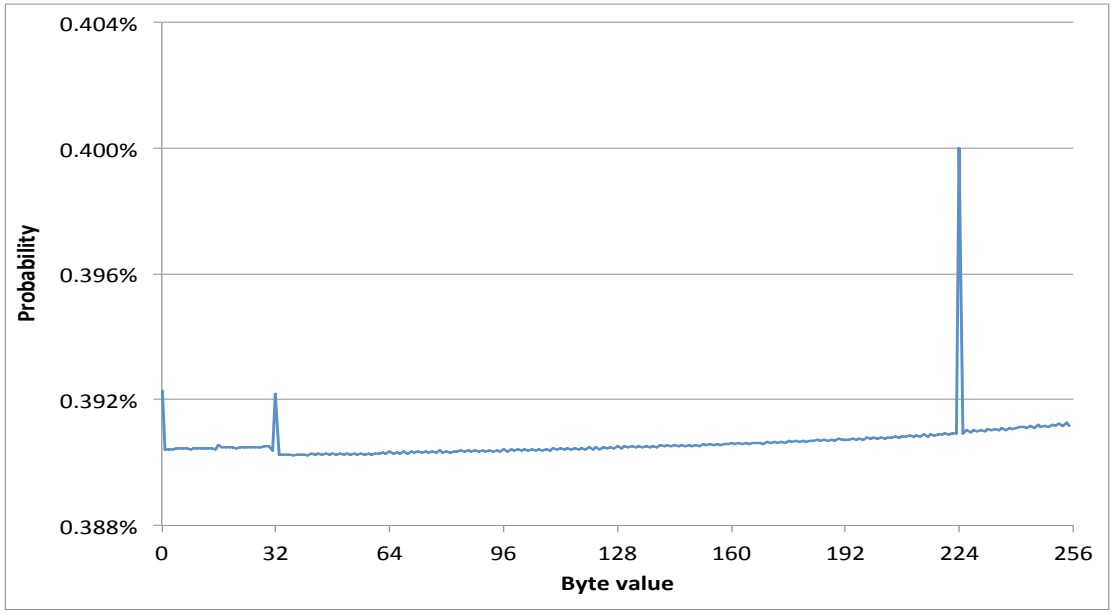
From: http://www.isg.rhul.ac.uk/tls/RC4passwords.pdf



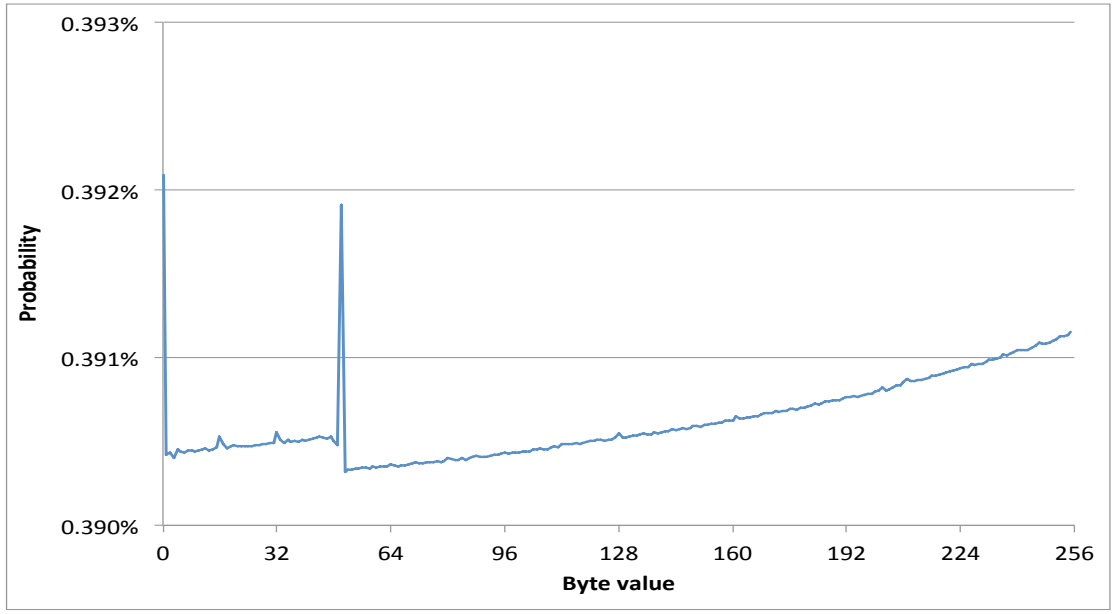
(a) Byte Z_1



(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

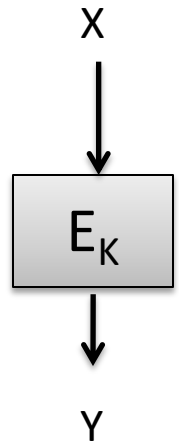
Family of permutations, one permutation for each key

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

Use notation $E(K,X) = Y$

Define inverse $D(K,Y) = X$ such that $D(K,E(K,X)) = X$

E,D must be efficiently computable



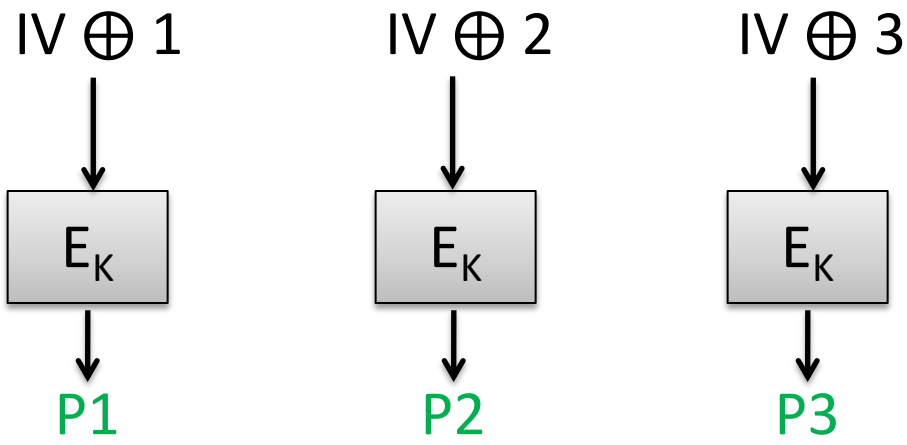
Pick K uniformly at random from $\{0,1\}^k$

CTR mode stream cipher

Counter mode stream cipher:

- Kg outputs random k-bit key for block cipher
- $G(K,IV,L) = E_K(IV \oplus 1) \| E_K(IV \oplus 1) \| \ \ldots \ \| \text{trunc}(E_K(IV \oplus m))$

  where $m = \text{ceil}(L / n)$



Truncate P3
to get L total bits

CTR-mode SE scheme

Kg():
K <-$ $\{0,1\}^k$

Pick a random key

Enc(K,M):
L <- |M| ; m <= ceil(L/n)
IV <-$ $\{0,1\}^n$
P <- $E_K(IV \oplus 1) \| \cdots \| \text{trunc}(E_K(IV \oplus m))$
Return (IV, P $\oplus$ M)

What security properties do we need from the block cipher?

Dec(K,(IV,C)):
L <- |C| ; m <= ceil(L/n)
P <- $E_K(IV \oplus 1) \| \cdots \| \text{trunc}(E_K(IV \oplus m))$
Return (IV, P $\oplus$ C)

Assume ciphertext can be parsed into IV and remaining ciphertext bits

Summary

- Target security against computational attackers
- Stream ciphers are computational equivalent of OTP
  - Keys must be large enough to avoid brute force
  - Even moderate biases in output stream must be avoided
- Stream ciphers give efficient symmetric encryption secure against passive adversaries
- None achieve security against active adversaries (stay tuned)

Next up

- Block cipher security goals
  - Pseudorandom functions and permutations
- Building & analyzing block ciphers
  - Feistel networks and DES
  - AES cipher