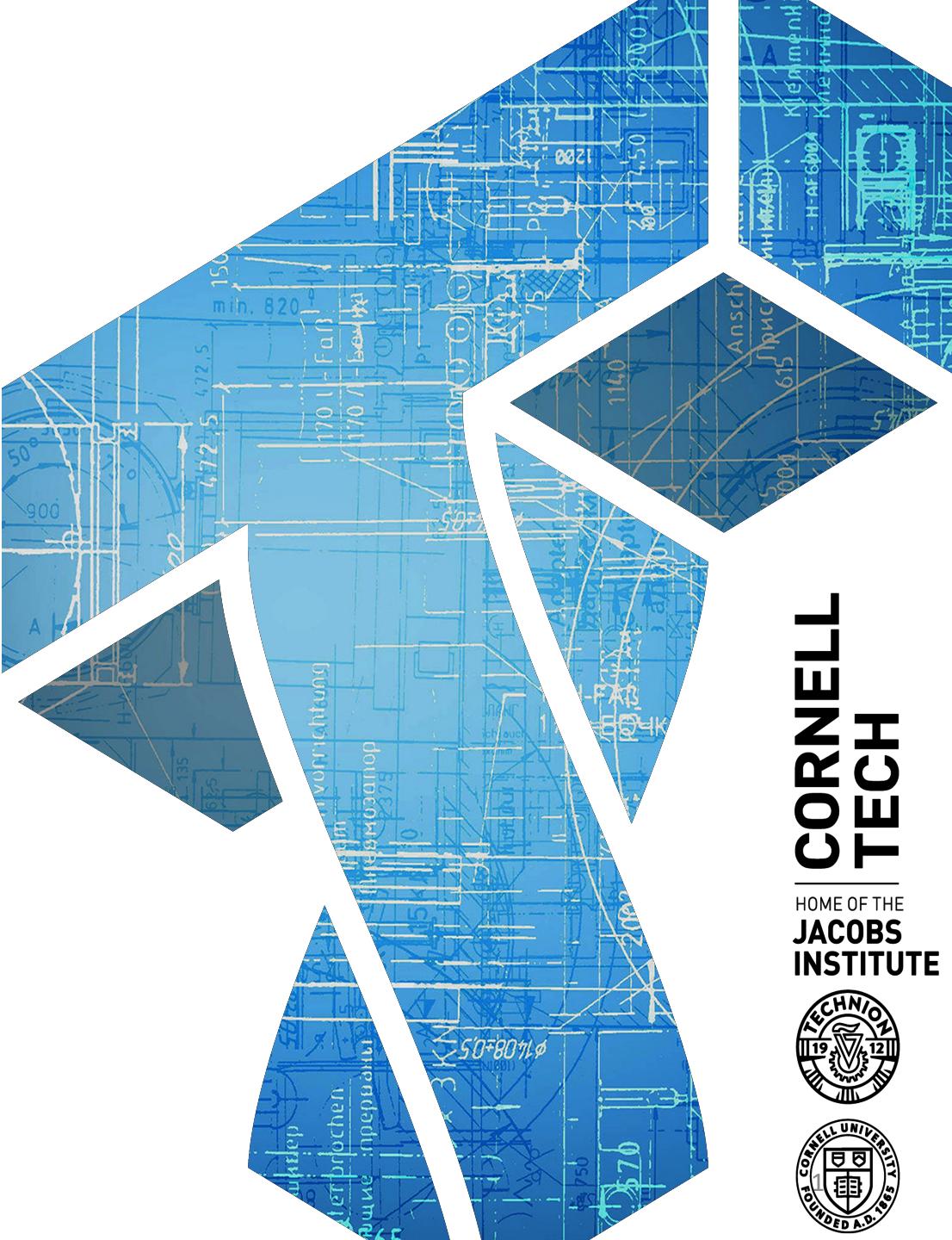


CS 5830

Cryptography

Instructor: Tom Ristenpart

TAs: Andres Fabrega,
Dhrumil Kumar Patel,
Sanketh Menda



**CORNELL
TECH**

HOME OF THE
**JACOBS
INSTITUTE**



Administrivia

- Homework 0 up
 - Don't need to turn in, but you should do it! Gets you setup with environment needed for homeworks
- Homework 1 will be up next Tuesday (we pushed back schedule slightly)
- Questions, ask on Slack!

Recap and where we're at

- Shannon and perfect secrecy
 - Good confidentiality against passive adversaries, but impractical
- Computational security
 - Relax to allow tiny success probability or limited adversarial resources
- Stream ciphers as computationally secure one-time pads

IND(SE, \mathcal{A}):

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

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

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

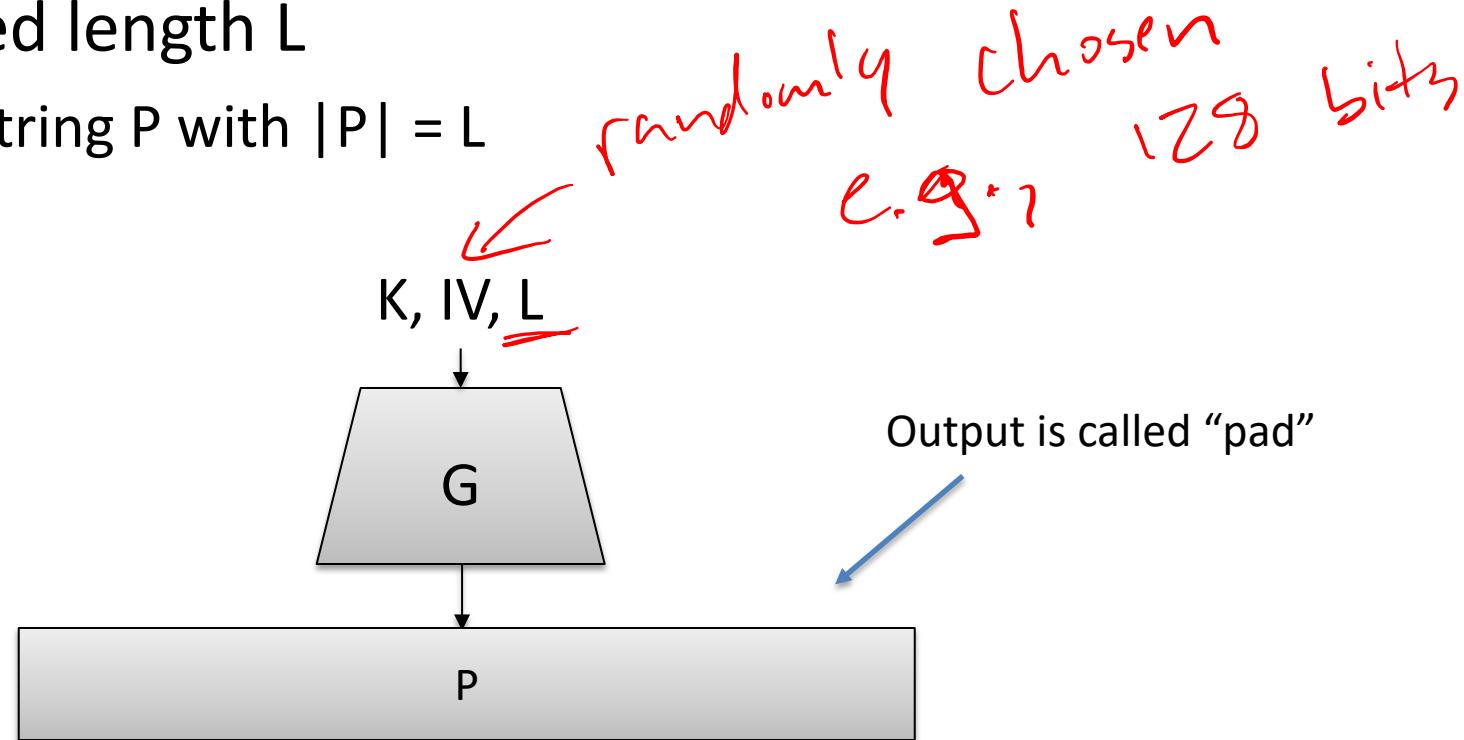
1) Tiny adversarial success

2) Computationally limited adversary

Stream ciphers

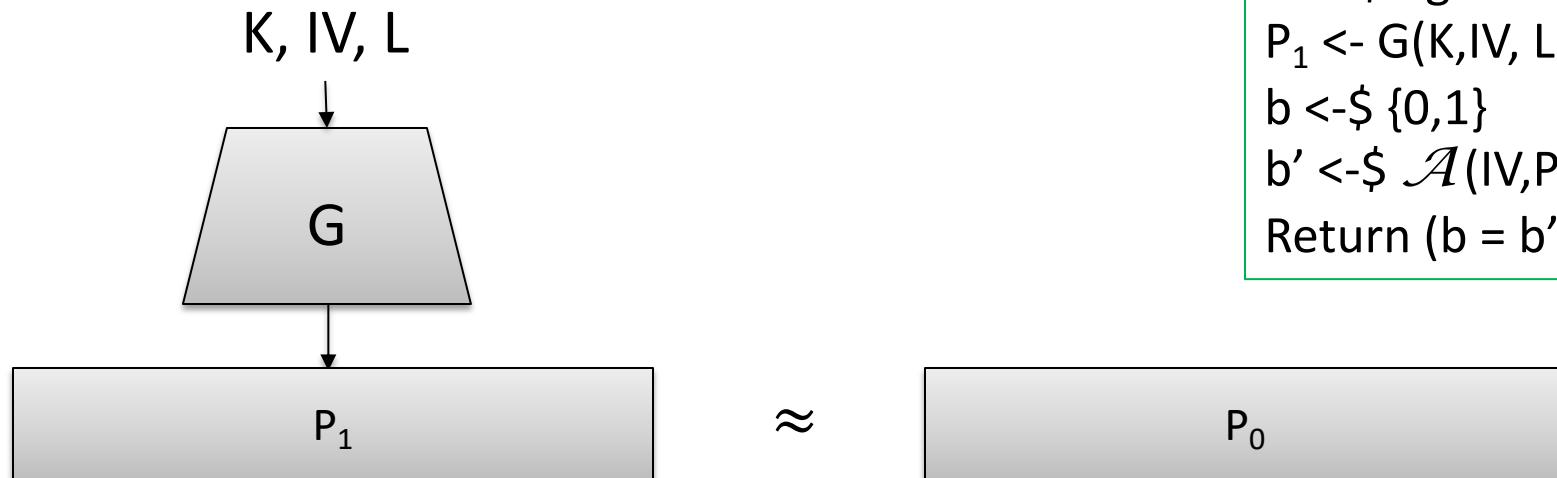
Stream cipher (aka pseudorandom generator, PRG) is pair of algorithms:

- K_g 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 $\text{IV}, G(K, \text{IV}, L)$ and IV, P_0 for P_0 random length- L bit string, with probability greater than ϵ



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

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

SE from a stream cipher

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

Kg():

$K \leftarrow \{0,1\}^k$

Pick a random key

Enc(K,M):

$L \leftarrow |M|$

$IV \leftarrow \{0,1\}^n$

Return $(IV, G(K, IV, L) \oplus M)$

Pick a per-message random IV

Encrypt using PRG-generated pad

Dec(K,(IV,C)):

$L \leftarrow |C|$

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:

Stream-cipher G is (t, ϵ) -pseudorandom



Stream-cipher based SE scheme is (t, ϵ) -indistinguishable

What we would show:

Stream-cipher G is *not* (t, ϵ) -pseudorandom



Stream-cipher based SE scheme is *not* (t, ϵ) -indistinguishable



Reduction-based analysis: high level idea

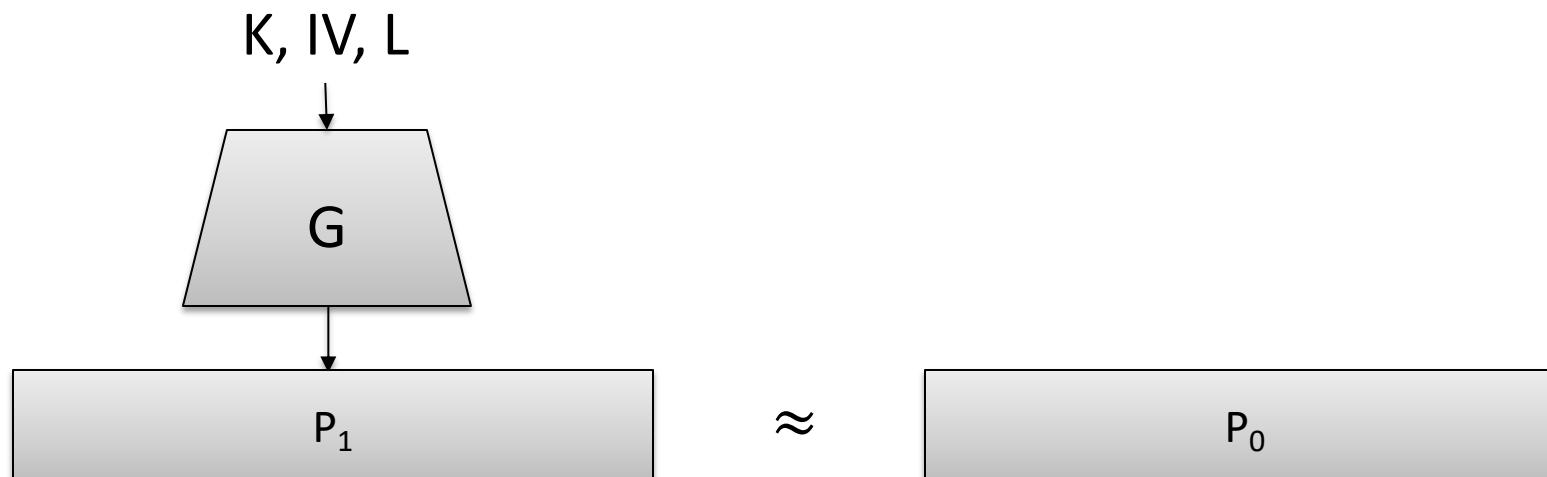
- **Reductionist analyses powerful tool:** rules out attacks that don't also break some (simpler to state) underlying hard computational problem
 - Often referred to as “provable security”
- Not a panacea:
 - Security definitions must capture relevant adversarial capabilities/goals
 - Proofs can sometimes be wrong
 - Must 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.: $\Pr[\underline{P_1[0]} = \textcircled{0x00}] = \frac{1}{256} + \delta$ *net for zero*



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,...,K_q

Adversary given C₁,...,C_q

$$C_1 = \underline{G(K_1)} \oplus M$$

$$C_2 = G(K_2) \oplus M$$

$$C_3 = G(K_3) \oplus M$$

:

$$C_q = G(K_q) \oplus M$$

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? $\sim q/3$

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,...,K_q

Adversary given C₁,...,C_q

$$C_1 = G(K_1) \oplus M$$

Alfardan et al. 2013 point out better approach

$$C_2 = G(K_2) \oplus M$$

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

$$C_3 = G(K_3) \oplus M$$

:

$$C_q = G(K_q) \oplus M$$

In words:

Choose value M that most likely explains the C₁,...,C_q seen given the known biases of G

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

You'll see PRGs used in different ways

- No nonce / initialization vector
 - $\text{PRG}(K)$
 - Can only ever use K to encrypt a single message
 - Otherwise two-time pad
- With nonce / initialization vector
 - $\text{PRG}(K, IV)$
 - Can use K many times, with distinct IV (usually randomly chosen, sometimes counters)
 - Repeat K, IV = two-time pad
- Length L is sometimes implicit and meant to be clear from context
 - $\text{PRG}(K) \oplus M$ means use PRG to generate $|M|$ bits

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
     $j \leftarrow j + \mathcal{S}[i] + K[i \bmod l]$ 
    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  $(i, j, \mathcal{S}) \leftarrow st_r$ 
   $i \leftarrow i + 1$ 
   $j \leftarrow j + \mathcal{S}[i]$ 
  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](http://www.isg.rhul.ac.uk/tls/RC4passwords.pdf)

RC4 stream cipher

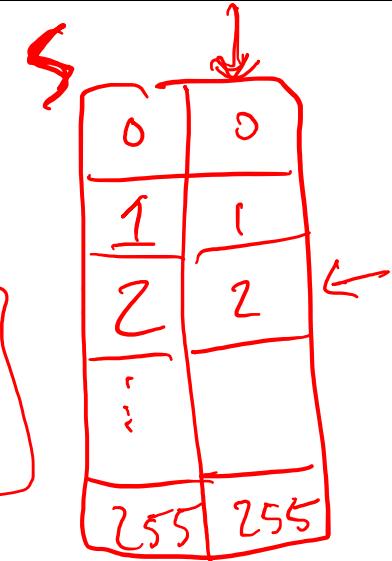
- Key scheduling algorithm uses K to build permutation S on $\{0, \dots, 255\}$ and initialize internal state (i, j, S)

[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
         $S[i] \leftarrow i$ 
     $j \leftarrow 0$ 
    for  $i = 0$  to 255 do
         $j \leftarrow j + S[i] + K[i \bmod l]$ 
        swap( $S[i], S[j]$ )
     $i, j \leftarrow 0$ 
     $st_0 \leftarrow (i, j, S)$ 
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  $(i, j, S) \leftarrow st_r$ 
     $i \leftarrow i + 1$ 
     $j \leftarrow j + S[i]$ 
    swap( $S[i], S[j]$ )
     $Z_{r+1} \leftarrow S[S[i] + S[j]]$ 
     $st_{r+1} \leftarrow (i, j, S)$ 
return  $(Z_{r+1}, st_{r+1})$ 
```

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

RC4 stream cipher

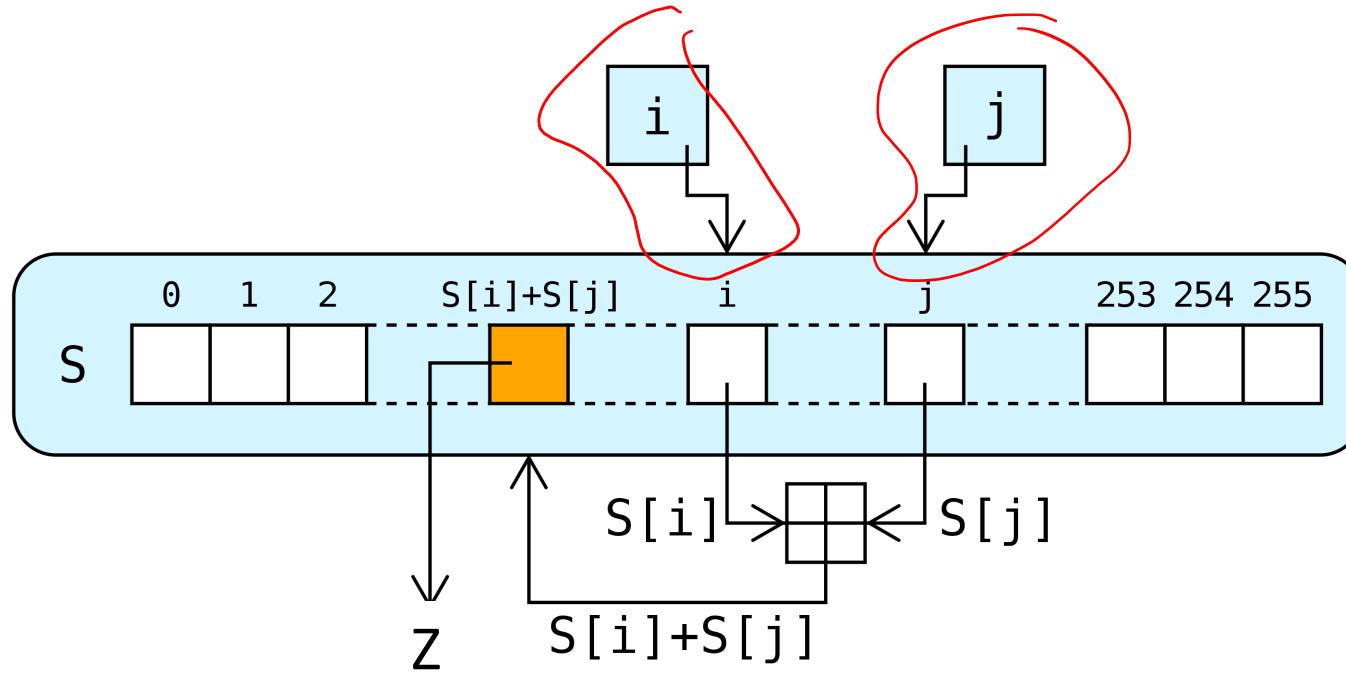


Diagram courtesy of <https://en.wikipedia.org/wiki/RC4>

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
     $j \leftarrow j + \mathcal{S}[i] + K[i \bmod l]$ 
    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  $(i, j, \mathcal{S}) \leftarrow st_r$ 
   $i \leftarrow i + 1$ 
   $j \leftarrow j + \mathcal{S}[i]$ 
  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

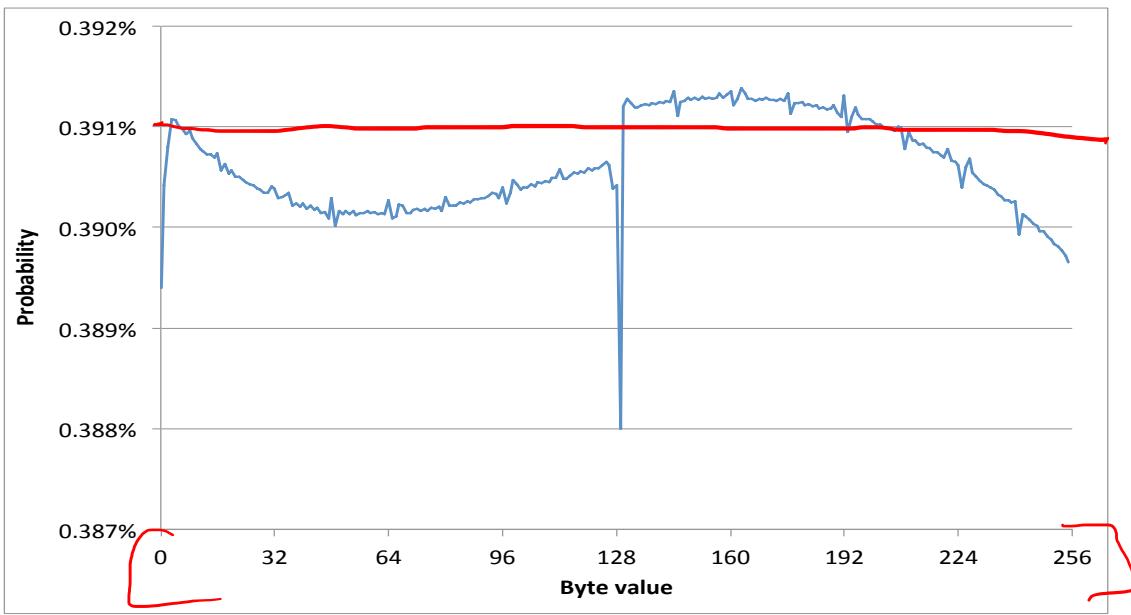
Algorithm 1: RC4 key scheduling (KSA)

```
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
         $j \leftarrow j + \mathcal{S}[i] + K[i \bmod l]$ 
        swap( $\mathcal{S}[i], \mathcal{S}[j]$ )
     $i, j \leftarrow 0$ 
     $st_0 \leftarrow (i, j, \mathcal{S})$ 
return  $st_0$ 
```

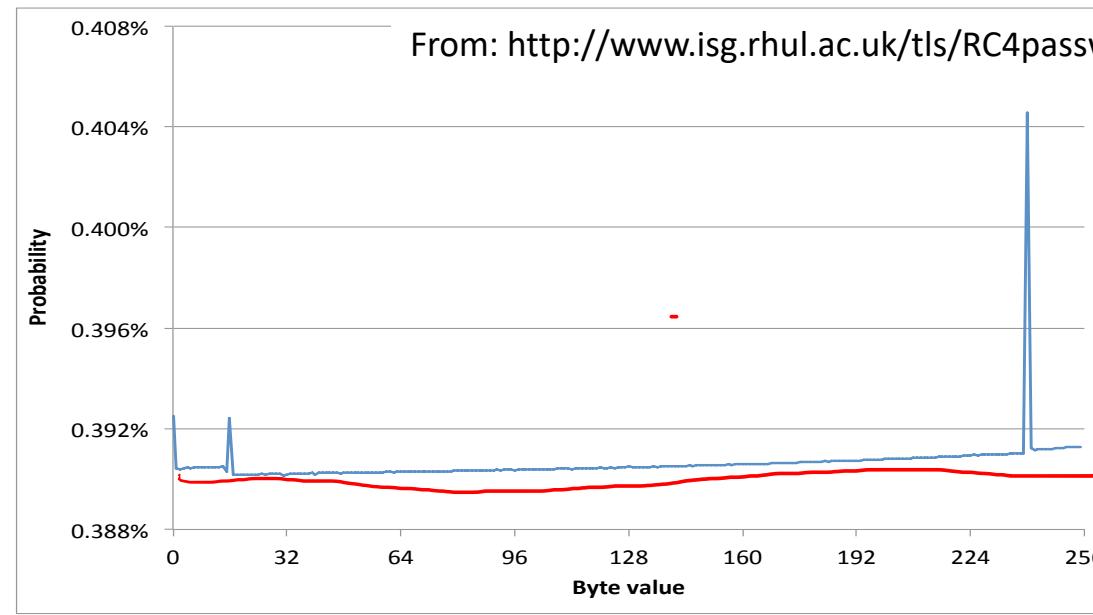
Algorithm 2: RC4 keystream generator (PRGA)

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

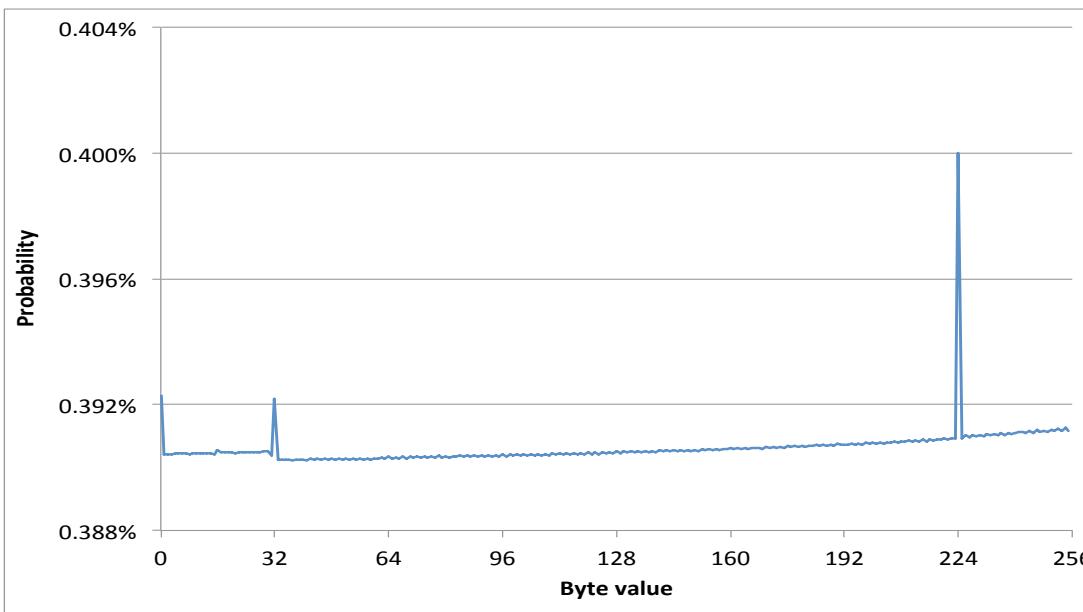
From:
<http://www.isg.rhul.ac.uk/tls/RC4passwords.pdf>



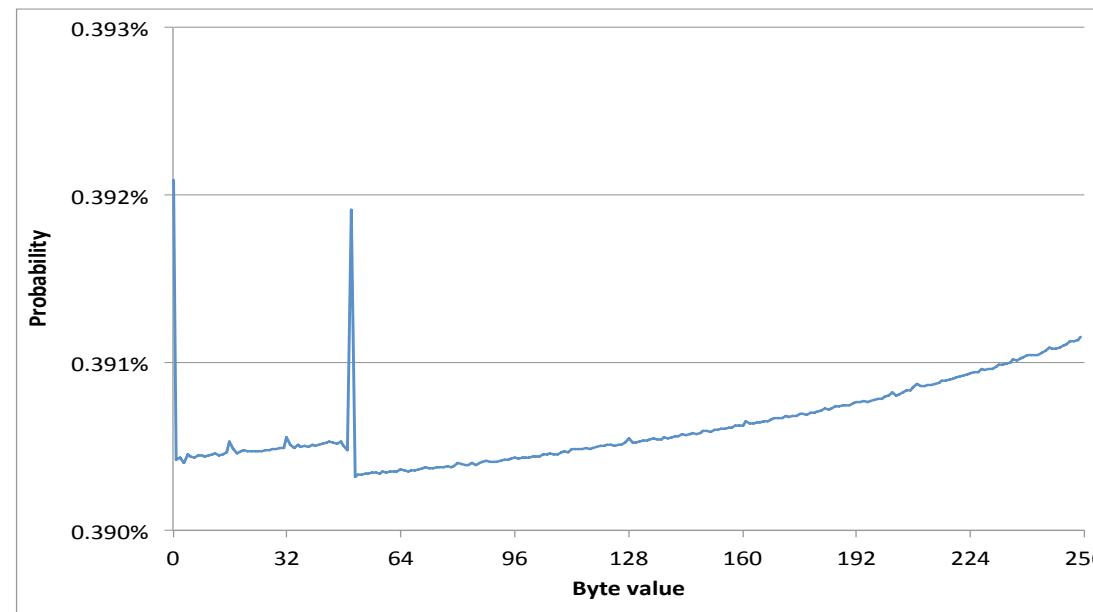
(a) Byte Z_1



(b) Byte Z_{16}

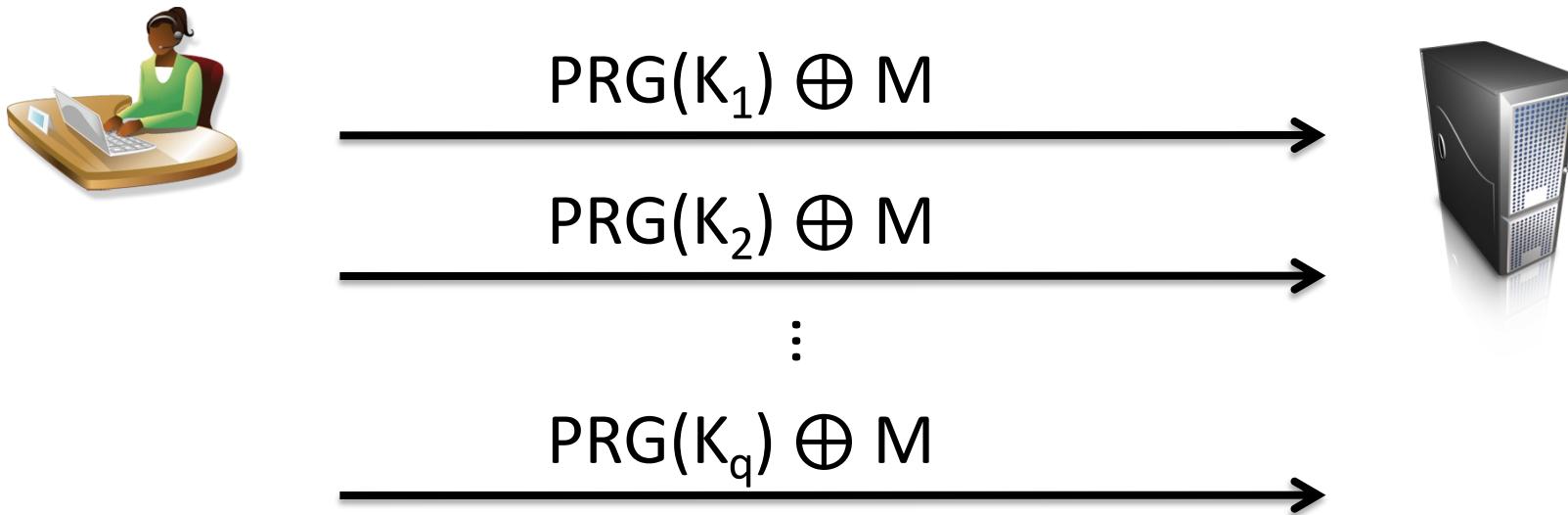


(c) Byte Z_{32}



(d) Byte Z_{50}

Bias-abusing attacks



Broadcast attack:

Unknown message M

Encrypted q times under q different K (or different K, IV pairs)

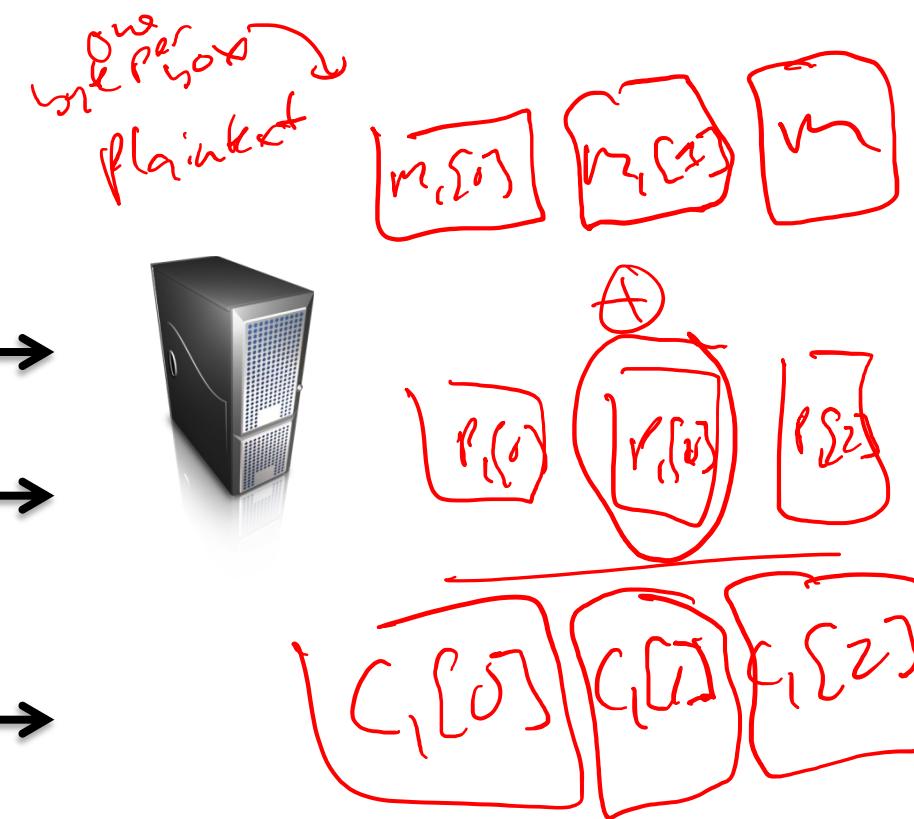
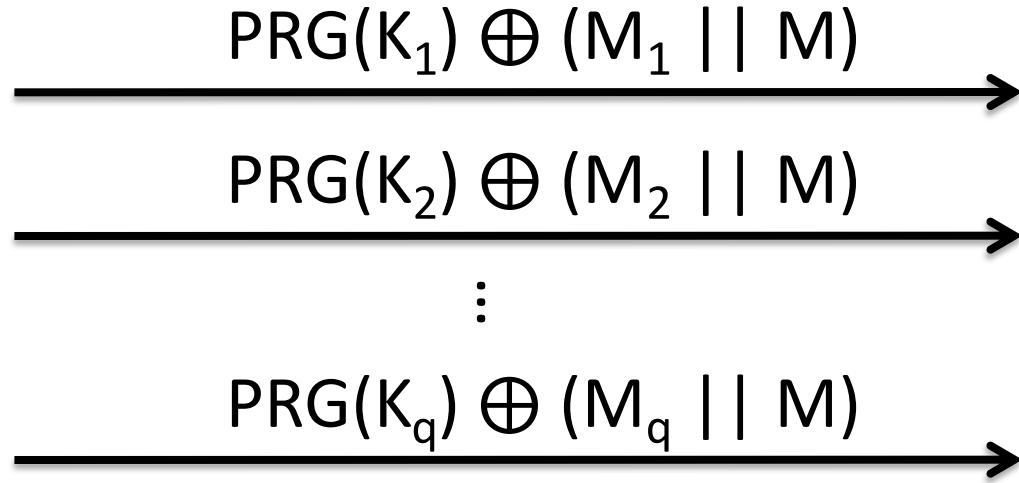
Use ciphertexts and biases to recover (portions of) M

E.g., M is password
and this is login-over-TLS
Each encryption is fresh session

Say we know bias, e.g. $\Pr[\text{PRG}(K)[0] = 0x00] > \delta$

- What settings would give rise to broadcasts?
- How could you estimate δ ?
- How do you exploit?

Bias-abusing with shifting



Broadcast attack with plaintext shifting:

Unknown message M

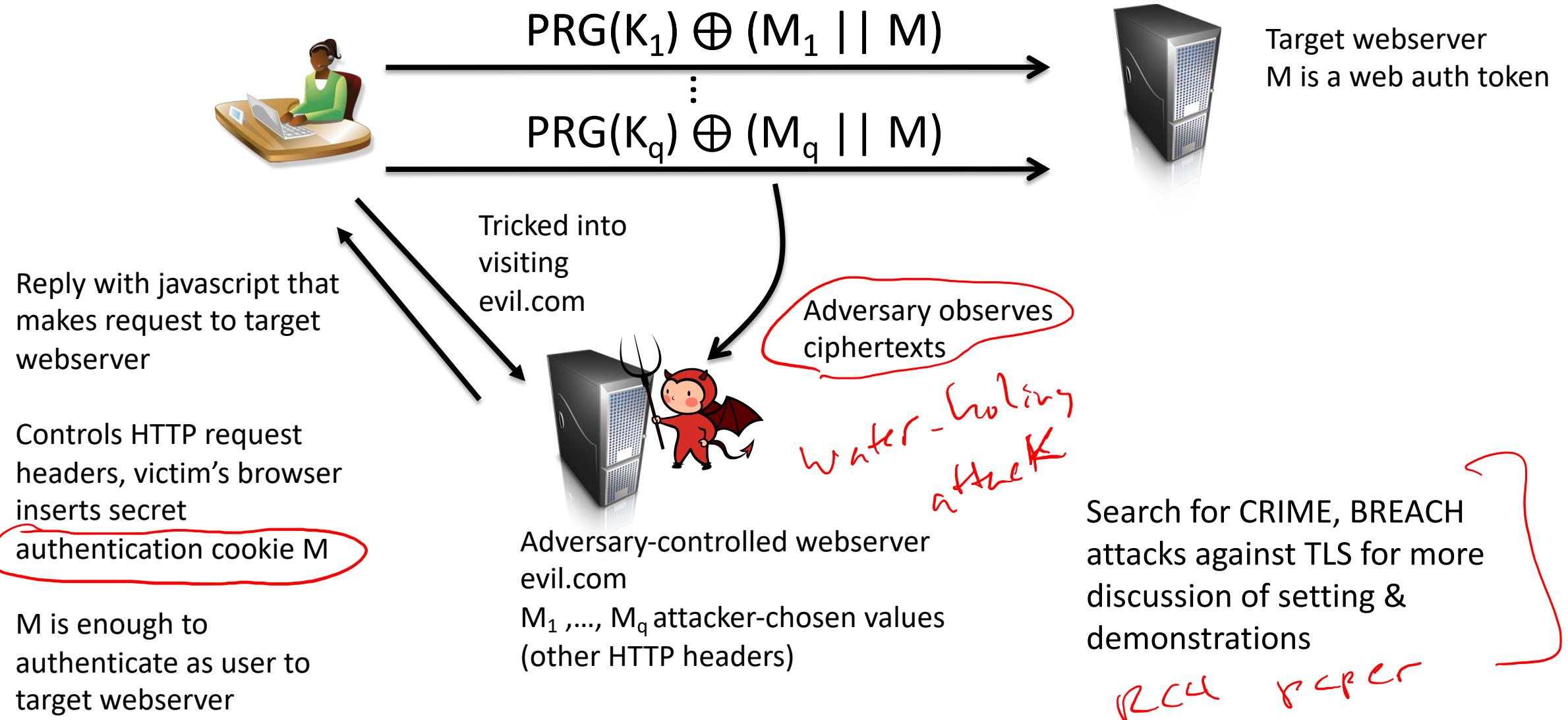
Attacker controls message prefix M_i

Encrypted q times under q different K (or different K, N pairs)

Use ciphertexts and biases to recover all of M

In what kind of situations could this arise?

Bias-abusing with shifting



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

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

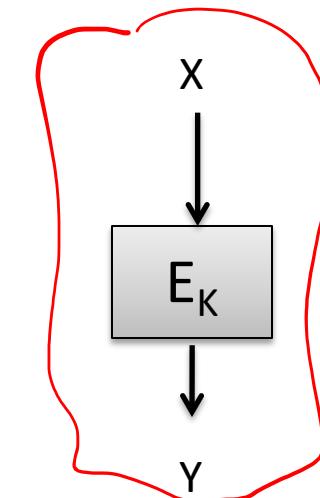
Use notation $E(K, X) = E_K(X) = Y$

Define inverse $D(K, Y) = D_K(Y) = X$

$$\text{s.t. } D(K, E(K, X)) = X$$

E, D must be efficiently computable

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



A 4x2 grid with columns labeled X and Y . The first column has rows 00, 01, 10, 11. The second column has rows 10, 11, 01, 00. A large red bracket on the right side groups the four rows of the second column. A red arrow points from the top of the first column to the top of the second column.

X	Y
00	10
01	11
10	01
11	00

A 4x2 grid with columns labeled X and Y . The first column has rows 00, 01, 10, 11. The second column has rows 11, 10, 00, 01. A large red bracket on the left side groups the four rows of the first column. A red arrow points from the top of the second column to the top of the first column.

X	Y
00	11
01	10
10	00
11	01

A 4x2 grid with columns labeled X and Y . The first column has rows 00, 01, 10, 11. The second column has rows 11, 10, 00, 01. A large red bracket on the top row groups the two columns. A red arrow points from the top of the first column to the top of the second column.

X	Y
00	11
01	10
10	00
11	01

Block cipher design

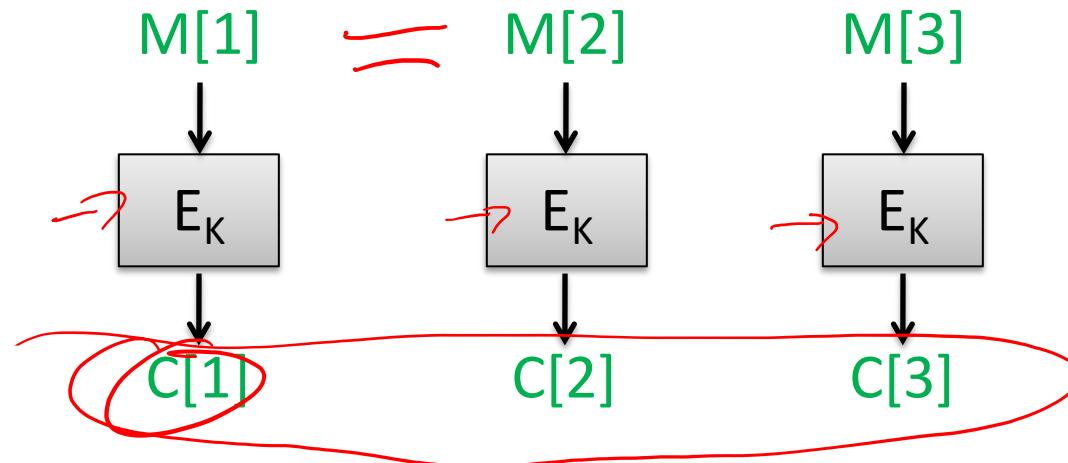
- A big topic with ~50 year history
- Data Encryption Standard (DES) designed in 1970s
 - Uses Feistel network structure
- Advanced Encryption Standard (AES) designed in 1990s
 - Uses substitution-permutation network structure
- We will dig into this more next lecture
- First: How to encrypt messages with block ciphers?

Block cipher modes of operation

Electronic codebook (ECB) mode

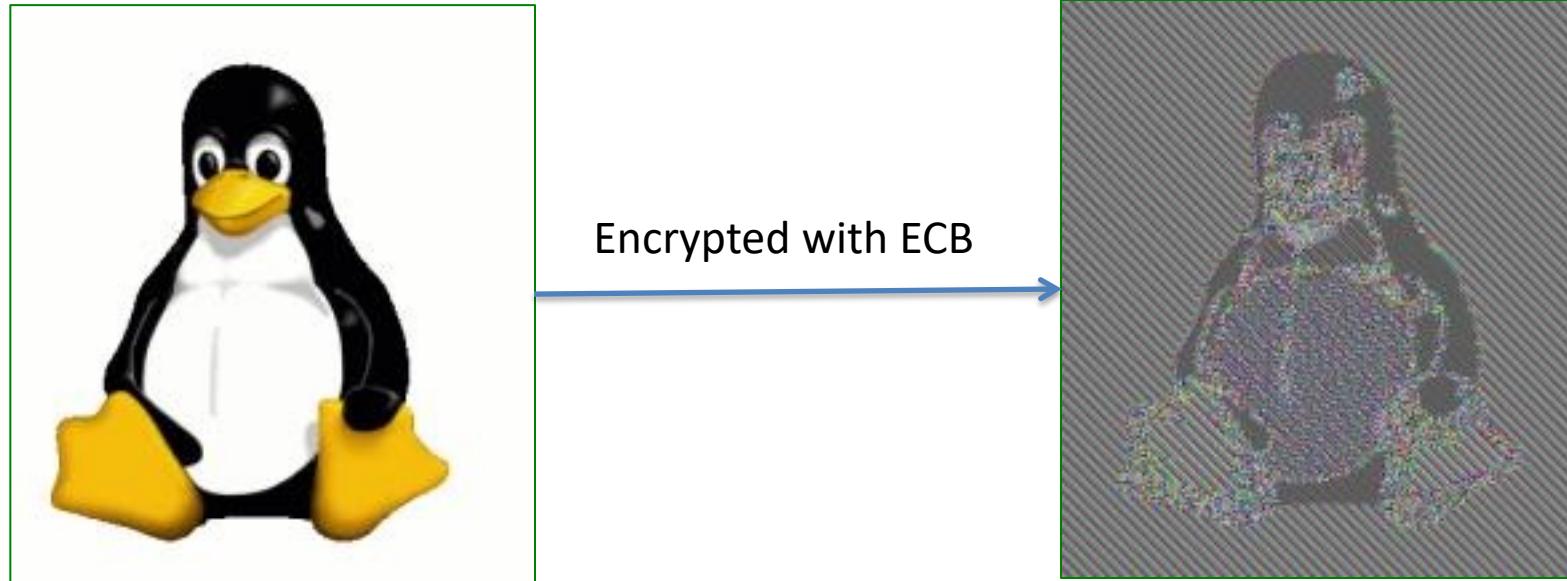
Pad message M to $M[1], M[2], M[3], \dots$ where each block $M[i]$ is n bits

Then:



How can we decrypt?

ECB mode is a more complicated looking substitution cipher

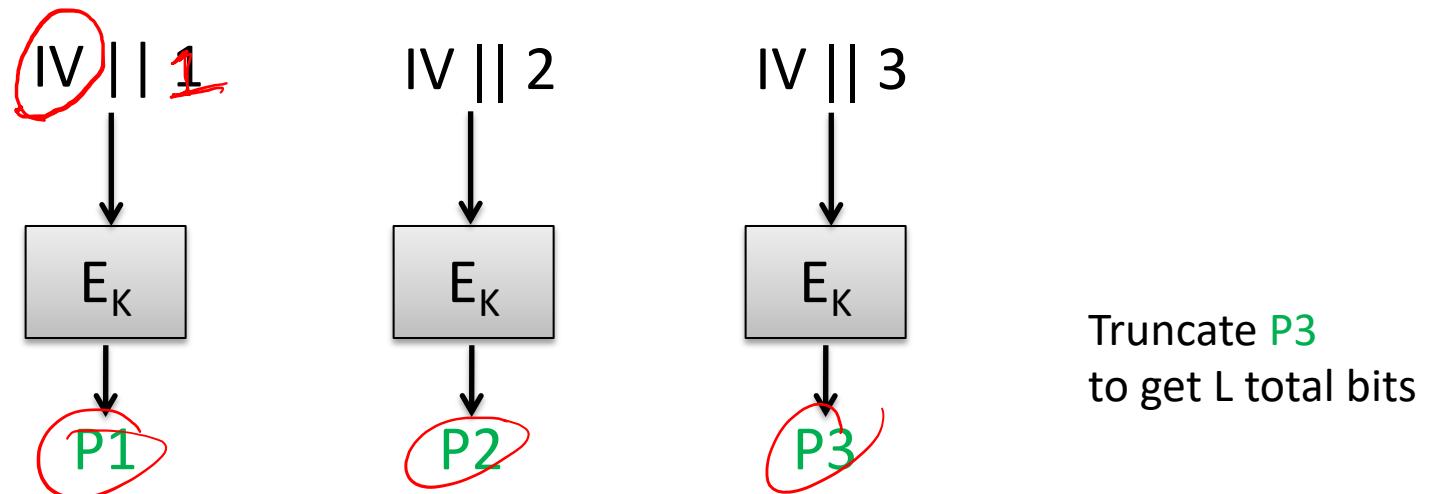


Images courtesy of
http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

CTR mode stream cipher

Counter mode stream cipher:

- K_g outputs random k -bit key for block cipher
- $G(K, IV, L) = E_K(IV \parallel 1) \parallel E_K(IV \parallel 2) \parallel \dots \parallel \text{trunc}(E_K(IV \parallel m))$
where $m = \text{ceil}(L / n)$
- Here $|IV|$ smaller than n bits, say $n/2$ bits



CTR-mode SE scheme

Kg():

$K \leftarrow \{0,1\}^k$

Enc(K,M):

$L \leftarrow |M| ; m \leftarrow \text{ceil}(L/n)$

$\text{IV} \leftarrow \{0,1\}^{n/2}$

$P \leftarrow E_K(\text{IV} || 1) \parallel \dots \parallel \text{trunc}(E_K(\text{IV} || m))$

Return $(\text{IV}, P \oplus M)$

Dec(K,(IV,C)):

$L \leftarrow |C| ; m \leftarrow \text{ceil}(L/n)$

$P \leftarrow E_K(\text{IV} || 1) \parallel \dots \parallel \text{trunc}(E_K(\text{IV} || m))$

Return $(\text{IV}, P \oplus C)$

Pick a random key

Should be able to call Enc with same K for many messages.
What could go wrong?

What security properties do we need from the block cipher?

Assume ciphertext can be parsed into IV and remaining ciphertext bits

*if this refers
we are in trouble*

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

