# Symmetric-Key Encryption

Sushmita Ruj

# Symmetric Ciphers

Def: A **Cipher** defined over a message space, key space and Ciphertext space is a pair of efficient algorithms (E,D), where,

$E : \mathscr{M} \times \mathscr{K} \rightarrow \mathscr{C}$ and $D : \mathscr{C} \times \mathscr{K} \rightarrow \mathscr{M}$,

Such that,

$$\forall m \in \mathscr{M} \textbf{ and } k \in \mathscr{K}, D(E(m,k)) = m$$

This is also called Shannon Cipher

E is often randomised, D is always deterministic

# One Time Pad

$$\mathscr{M} = \mathscr{C} = \{0,1\}^n \qquad \mathscr{K} = \{0,1\}^n$$

$$C = E(M, K) = M \oplus K, D(C, K) = C \oplus K$$

$$D = C \oplus K$$

Eg: M = 0111100101

K = 1100100100

C = 1011000001

# One Time Pad

$\mathscr{M} = \mathscr{C} = \{0,1\}^n$ $\mathscr{K} = \{0,1\}^n$ **Vernam 1971**

$$C = E(M, K) = M \oplus K, \quad D(C, K) = C \oplus K$$

Eg: M = 0111100101

K = 1100100100

M XOR K = C = 1011000001

**Advantages =?**

**Simple, Fast,**

**Disadvantages=?**

**Key as large as message**

# Attack Models

- Ciphertext-only attack: The adversary possesses a string of ciphertext, y
- Known plaintext attack: The adversary possesses a string of plaintext, x, and the corresponding ciphertext, y
- Chosen plaintext attack: The opponent has obtained temporary access to the encryption machinery. Hence he can choose a plaintext string, x, and construct the corresponding ciphertext string, y.
- Chosen ciphertext attack: The adversary has obtained temporary access to the decryption machinery. Hence he can choose a ciphertext string, y, and construct the corresponding plaintext string, x.

# What is a secure cipher?

Attacker's abilities: **CT only attack** (for now)

Possible security requirements:

attempt #1: **attacker cannot recover secret key**

attempt #2: **attacker cannot recover all of plaintext**

**Shannon's idea: CT should reveal no "info" about plaintext**

Secure cipher is one for which an encrypted message remains "well hidden," even after seeing its encryption.

# Information Theoretic Security (Shannon 1949)

- A cipher (***E***, ***D***) over ($\mathcal{K}$, $\mathcal{M}$, $\mathcal{C}$) has **perfect secrecy** if $\forall m_0, m_1 \in \mathcal{M}, len(m_0) = len(m_1)$ and $\forall c \in \mathcal{C}$
- $P[E(m_0, k) = c] = P[E(m_1, k) = c]$
- Where, k is chosen uniformly at random from $\mathcal{K}$ (meaning $k \xleftarrow{R} \mathcal{K}$ )
- What does this mean?
- Given a CT c, you cannot tell if the message is $m_0$ or $m_1$.
- Even the most powerful adversary cannot tell PT by looking at the CT
- No CT only attack possible (Of course there might be other attacks)

# OTP has Perfect Secrecy

- In OTP, $\forall m, c,\ if\ E(m,k) = c,\ then\ c = m \oplus k,$
- $\implies k = m \oplus c$
- $|\{k \in \mathcal{K} : E(m,k) = c\}| =$ 1
- $\forall m, c, P[E(m,k) = c] = \dfrac{\text{no.of keys } k \in \mathcal{K}, \text{ such that } E(m,k) = c}{|\mathcal{K}|} = 1/\mathcal{K}|$
- Therefore OTP has perfect secrecy

# Property

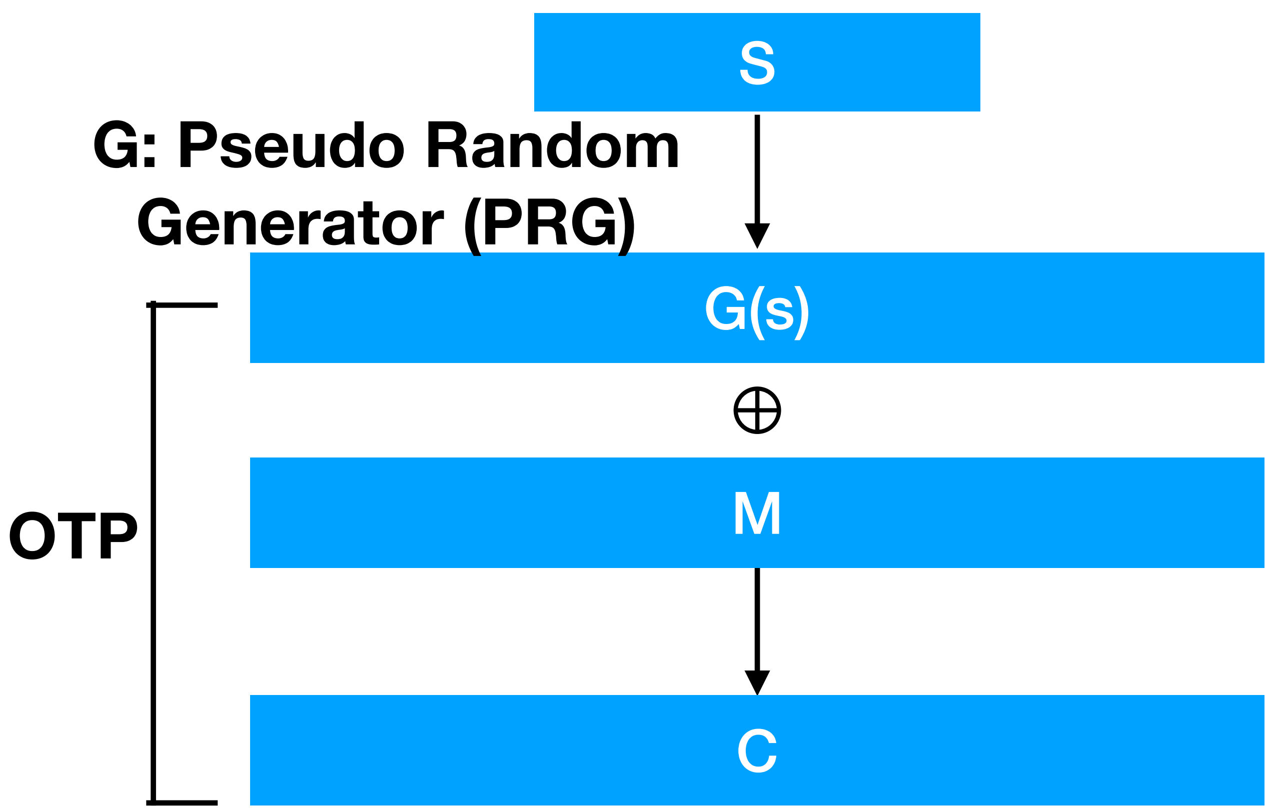
- Perfect secrecy => $|\mathscr{K}| \geq |\mathscr{M}|$
- Proof: Practice Exercise ( Boneh-Shoup Section 2.1.3)
- This means that key length should be more than message length, which is impractical in most applications.
- To encrypt a 1 GB message you’s need a 1 GB key, inconvenient :(

# Practical OTP

- Use PRG instead of OTP
- Stream ciphers
- The security is not "perfect secrecy" because the size of the key is smaller
- We prove semantic security (weaker notion than perfect secrecy )
- Intuitively an adversary cannot say the difference between a pseudo random string and a truely random string.
- Design effective statistical tests

# Practical OTP

No
Size of Key is
smaller than
message




Q1: Does this have prefect secrecy?

Size of message is L

Q1: What is G? What properties does it have?

|s| < |M|. K should look like a random string r of length L .

Dan Bernstein's ChaCha (Read if interested)

Q3: What can we say about the security of this cipher?

$$C = M \oplus G(K)$$

$$M = C \oplus G(K)$$

This does not have prefect secrecy, so we define a new type of security called "semantic Security"

**These are called stream ciphers**

# Statistical Tests

- To distinguish a pseudo-random string G(s) from a truely random string r of L bits

- Algorithm takes a string and outputs 0 or 1

- Such a test is called effective if the probability that it outputs 1 on a pseudo-random input is significantly different than the probability that it outputs 1 on a truly random input

- Count number of 1's appearing in the input, if this is $\approx L/2$

- Count the pairs 00, 01, 10, 11, each of these should be $\approx L/4$

- Count the pairs 000, 001…10, 111, each of these should be $\approx L/8$

- But … this is not enough, Try the next bit test

# Next Bit Test and Unpredictability

- If an adversary can figure out the L-th bit from the L-1 bits of G(s), then it can know that LSB of the message. Could be disastrous. (5 diplomats casts their votes, 4 have last names ending in even number, 1 in odd number alphabet)

- We say that a PRG $G: k \rightarrow \{0,1\}^L$ is predictable if there exists an efficient algorithm A, and $0 \leq i < L$, $Pr_{k \xleftarrow{R} \mathcal{K}}[A(G(k))|_{1,2,\cdots i} A(G(k)|_{i+1}] > 1/2 + \epsilon$, (eg $\epsilon =$ 1/10000 non-negligible)

- PRGs must be unpredictable (if not, then PRG is insecure)

- Def: $\forall i$, there are no efficient adversary that can can predict bit $(i+1)$ with “non-neg” $\epsilon$

# PRG in Practice

- linear congruential generators (LCG) generate pseudo-random numbers not for Crypto

```
r[0] ←seed
r[i] ← ( a.r[ i-1]+ b )  % p;
      output  r[i] >> x;
      i++;
```

**glibc/random()**

$r[i] \leftarrow ( r[i-3] + r[i-31] ) \ \% \ 2^{32}$

output r[i] >> 1

**Never use**
**Kerberos 4 used and was hacked.**

A good PRG should pass all statistical tests.
The existence of provably secure PRG is UNKNOWN

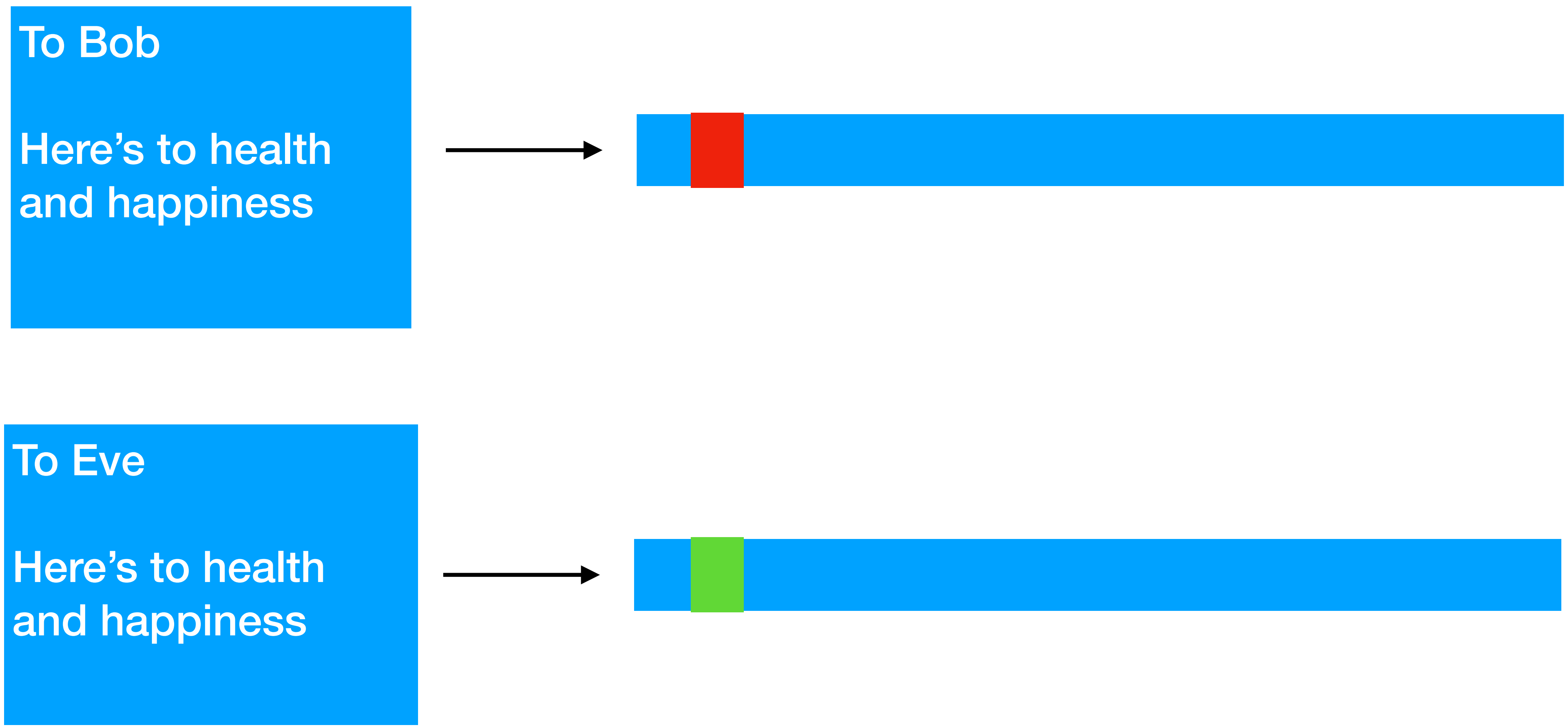
# Attacks on Ciphers

# Attacks 1: Never use same Key Twice in Stream ciphers

- Suppose $K_1$ is used to encrypt two messages $m_1\ and\ m_2$
- Therefore, $c_1 = m_1 \oplus k_1$ and $c_2 = m_2 \oplus k_1$
- Hence, $c_1 \oplus c_2 = m_1 \oplus m_2$
- If you know some bits of $m_1$, you can infer corresponding bits $m_2$
- For example if there are a sequence of same bits in two messages, $m_1$ and $m_2$, this can be known

# Real-world attacks

**Messages encrypted on disc with the same key k**




**Easily infer that both messages are the same**

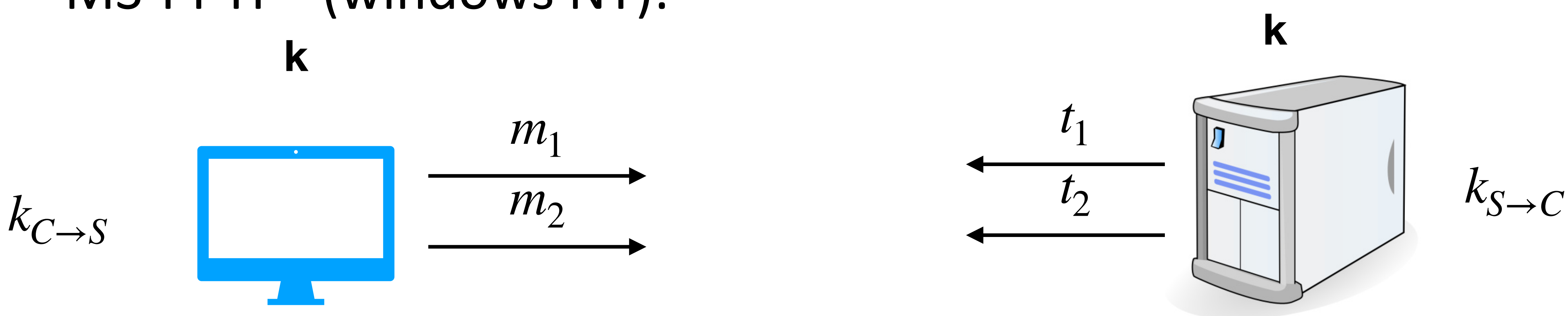
**DO NOT USE STREAM CIPHERS FOR DISK ENCRYPTION**

# Real-world attacks

- Project Venona (1941-1946): Russian project which used

OTP (insecurely) and intercepted by NSA (3000 msgs decrypted)

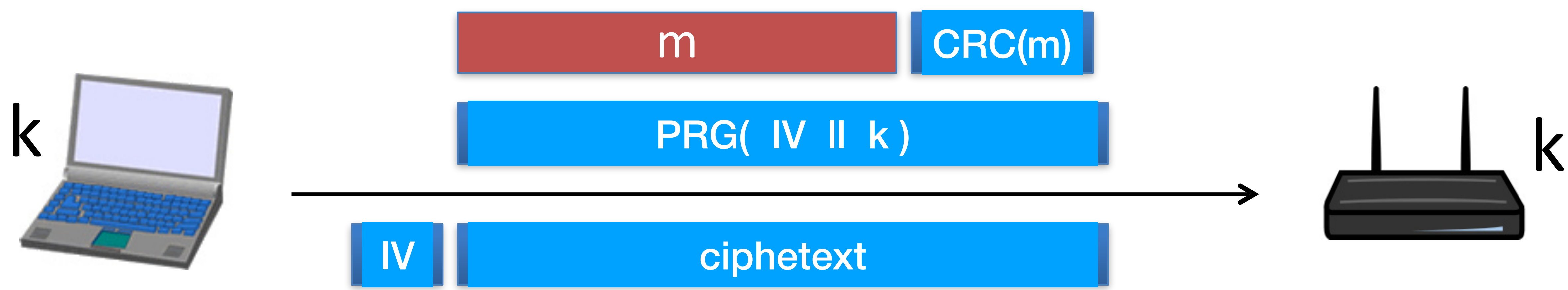
MS-PPTP (windows NT):




$m_1 || m_2 .. || G(k)$ $\qquad$ $t_1 || t_2 .. || G(k)$

**In any bidirectional channel use two keys**

# 802.11b WEP Vulnerabilities




Length of IV: 24 bits

- Repeated IV after $2^{24}$ ≈ 16M frames
- On some 802.11 cards: IV resets to 0 after power cycle

key for frame #1: (1 II k) (4 + 104) bits

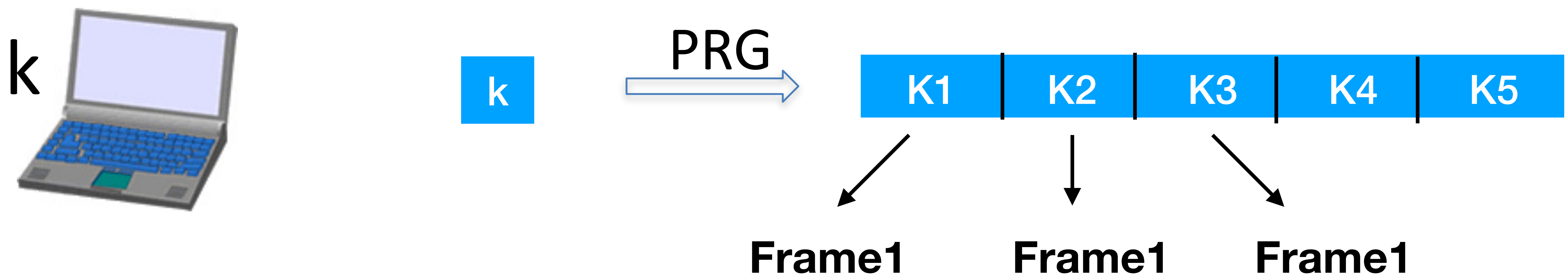
key for frame #2: (2 II k) (4 + 104) bits

Related key attacks

More attacks later in context of RC4

**New keys for every session (As in TLS)**

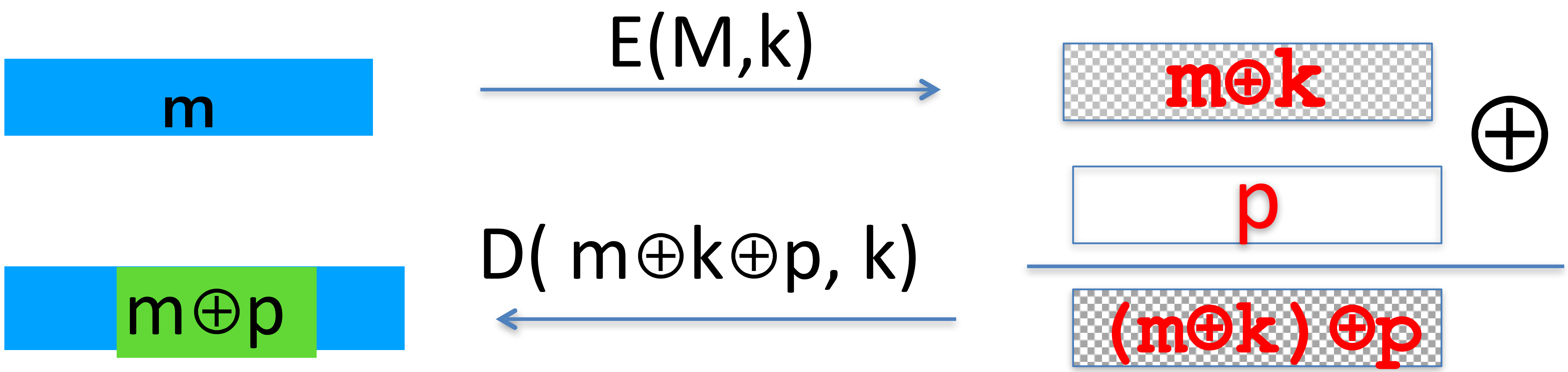
# Alternate Better Construction




⇒ now each frame has a pseudorandom key

better solution: use stronger encryption method (as in WPA2)

# Attack 2: No integrity




Modifications to ciphertext are undetected and have **predictable** impact on plaintext

(OTP is malleable)

# Stream Ciphers

# RC4 Rivest Cipher (1987)

Key scheduling algo, input key k

**Algorithm KSA**

*Initialization*:

For $i = 0, \ldots, N-1$

$S[i] = i;$

$j = 0;$

*Scrambling*:

For $i = 0, \ldots, N-1$

$j = (j + S[i] + K[i]);$

$\text{Swap}(S[i], S[j]);$

Pseudo Random Generator Algo,
Use Internal state S

**Algorithm PRGA**

*Initialization*:

$i = j = 0;$

*Output Keystream Generation Loop*:

$i = i + 1;$

$j = j + S[i];$

$\text{Swap}(S[i], S[j]);$

$t = S[i] + S[j];$

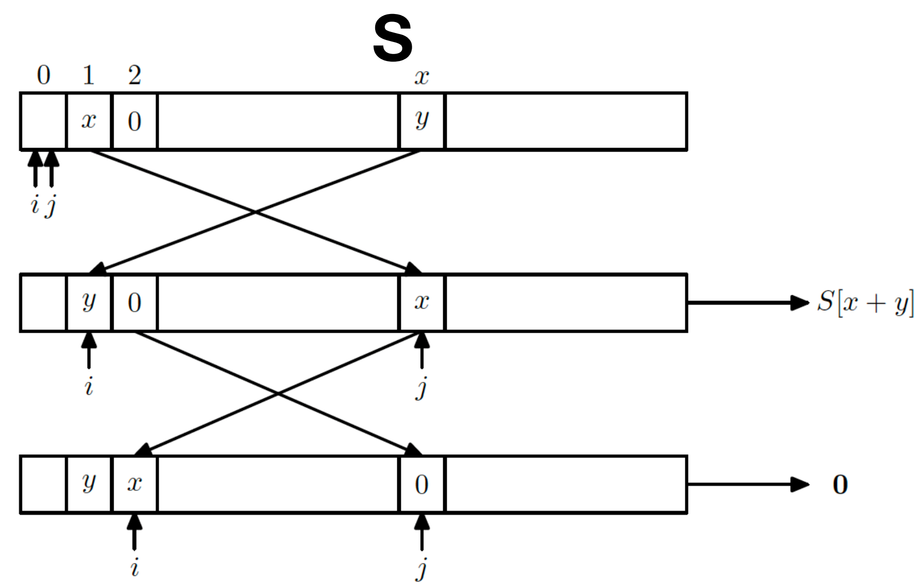
Output $z = S[t];$

- Weaknesses:

1. Bias in initial output: Pr[ $2^{nd}$ byte = 0 ] = 2/256
2. Prob. of (0,0) is $1/256^2 + 1/256^3$
3. Related key
4. No longer used

# RC4 Rivest Cipher (1987)

**Algorithm PRGA**
*Initialization*:
    $i = j = 0$;
*Output Keystream Generation Loop*:
        $i = i + 1$;
        $j = j + S[i]$;
        Swap($S[i]$, $S[j]$);
        $t = S[i] + S[j]$;
        Output $z = S[t]$;




Bias in initial output: Pr[ 2nd byte = 0 ] = 2/256

Let P be the event that $S[2] = 0 \ and \ S[1] \neq 2$, $Pr[z_2 = 0] = 1$

Otherwise, $z_2$ is evenly distributed in $\{0,1,\cdots,n-1\}$ .

$$Pr[z_2 = 0] = Pr[z_2 = 0 \,|\, P]Pr[P] + Pr[z_2 = 0 \,|\, \neg P]Pr[\neg P]$$

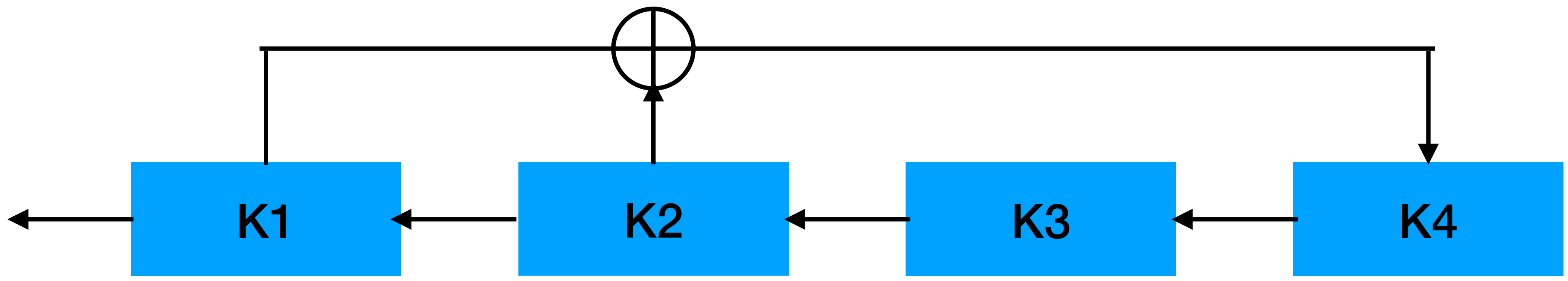
$$\approx 1.1/n + (1 - 1/n)1/n \approx 2/n$$

Later it is possible to recover the first 128 bytes of the plaintext

with probability close to 1.

Other attacks: Boneh-Shoup 3.9

# Linear Feedback Shift Register (LFSR)




- (1000) -> 1 0 0 0 1 0 0 1 1 0 1 0 1 1 1…
- Combine many LFSRs, but weak PRGs
- Trivium (eSTREAM), A5/1 (GSM), E0 (Bluetooth) use LFSR
- CSS: Content Scramble System, used in DVDs use 2 LFRS
- Broken (Read Section 3.8, Boneh-Shoup)

# Stream ciphers: eStream

PRG: $\{0,1\}^s \times R \longrightarrow \{0,1\}^n$

Nonce: a non-repeating value for a given key.

$E(k, m ; r) = m \oplus PRG(k ; r)$

The pair (k,r) is never used more than once.

| **Profile 1 (SW)** | **Profile 2 (HW)** |
| --- | --- |
| HC-128 | Grain v1 |
| Rabbit | Rabbit |
| Salsa20/12 | Trivium |
| SOSEMANUK | |

# Thank you