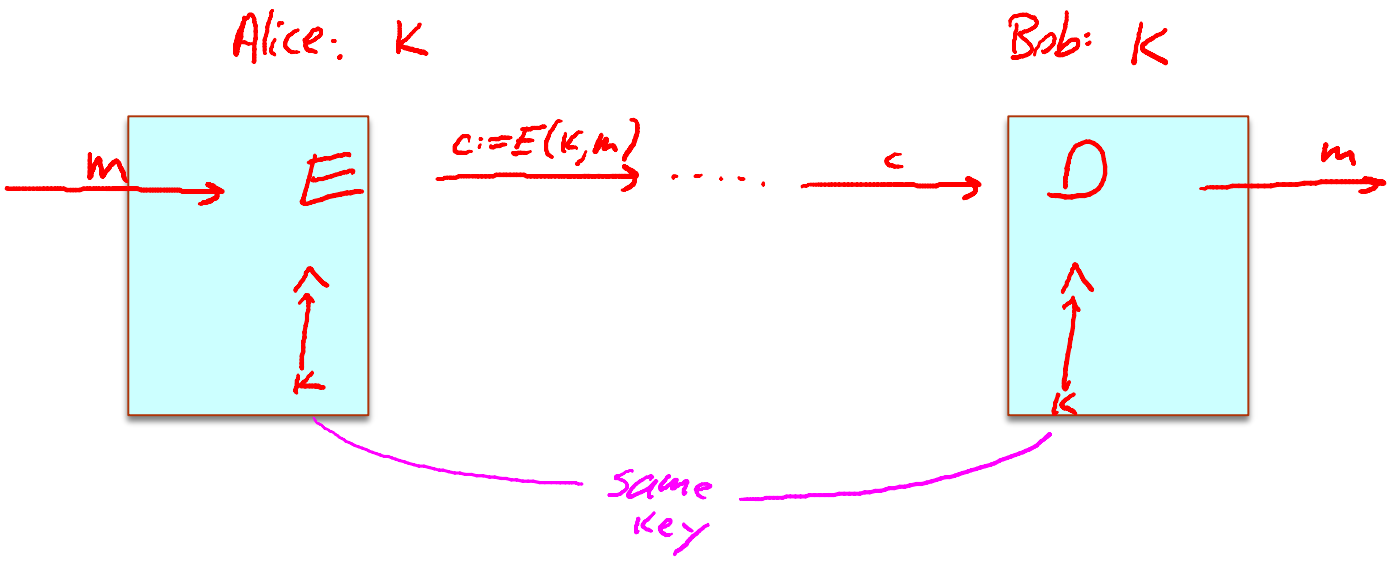
CS-3002: Information Security

Lecture # 3: One Time Pad (OTP), Pseudorandom Generator (PRG) and Stream Ciphers

Prof. Dr. Sufian Hameed
Department of Computer Science
FAST-NUCES

Symmetric Cryptosystems



Formal Definition:

Cryptosystem is defined over (K,M,C) and a pair of "efficient" algorithms (***E***, ***D***) s.t.

$$\forall\, m \in M,\ k \in K \text{ and } c \in C : \boldsymbol{E}(k, m) = c,\ \boldsymbol{D}(k, \boldsymbol{E}(k, m)) = m$$

Efficient means run in polynomial time

E *is often randomized.*

D *is always deterministic.*

One Time Pad (Vernam 1917)

One Time Pad has perfect secrecy (i.e. no CT only attacks)

Based on simple XOR operation

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

$C := E(k, m) = k \oplus m$

$D(k, c) = k \oplus c$

msg:	0	1	1	0	1	1	1
key:	1	0	1	1	0	1	0
CT:	1	1	0	1	1	0	1

Indeed

$$D(k, E(k, m)) = D(k, k \oplus m) = k \oplus (k \oplus m) = (k \oplus k) \oplus m$$
$$= 0 \oplus m = m$$

- **One-time pad** = XOR cipher with constraints:
  - *Key length equals message length*
  - *Key bits are truly random (not pseudo-random)*
  - *Key is used only once and destroyed*

You are given a message (m) and its OTP encryption (c).

Can you compute the OTP key from m and c ?

No, I cannot compute the key.

Yes, the key is $k = m \oplus c$. ⇐

I can only compute half the bits of the key.

Yes, the key is $k = m \oplus m$.

The One Time Pad (Vernam 1917)

Very fast enc/dec !!

… but long keys (as long as plaintext)

Is the OTP secure? What is a secure cipher?

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 PT

Information Theoretic Security

Basic Idea: CT should reveal no "info" about PT

Def: A cipher ***(E,D)*** over (K,M,C) has **perfect secrecy** if

$\forall m_0, m_1 \in M \quad (|m_0| = |m_1|) \quad \text{and} \quad \forall c \in C$

$$Pr[E(k,m_0)=c] = Pr[E(k,m_1)=c] \qquad \text{where } k \xleftarrow{R} K$$

- Given CT can't tell if msg is m0 or m1 (for all m0, m1)
- Most powerful adversary learns nothing about PT from CT
- no CT only attacks !!! (but other attacks possible)

The bad news …

Thm: perfect secrecy $\Rightarrow$ $|\mathcal{K}| \geq |\mathcal{M}|$

- i.e. perfect secrecy ➔ key-len >= msg-len
- Hard to use in practice !!!

One Time Pad in practice

- **Intelligence and military services**
  - Regular usage by KGB spies
  - Hotline between USA and USSR
- **Major problems**
  - Key exchange difficult
  - True randomness required
- **Not very practical today**
  - Inspiration for other methods,
  - e.g. stream ciphers

Stream Cipher: Pseudorandom Generators

Stream Ciphers: making OTP practical

Idea: replace "random" key by "pseudorandom" key

PRG is a Function **G: $\{0,1\}^s \longrightarrow \{0,1\}^n$**

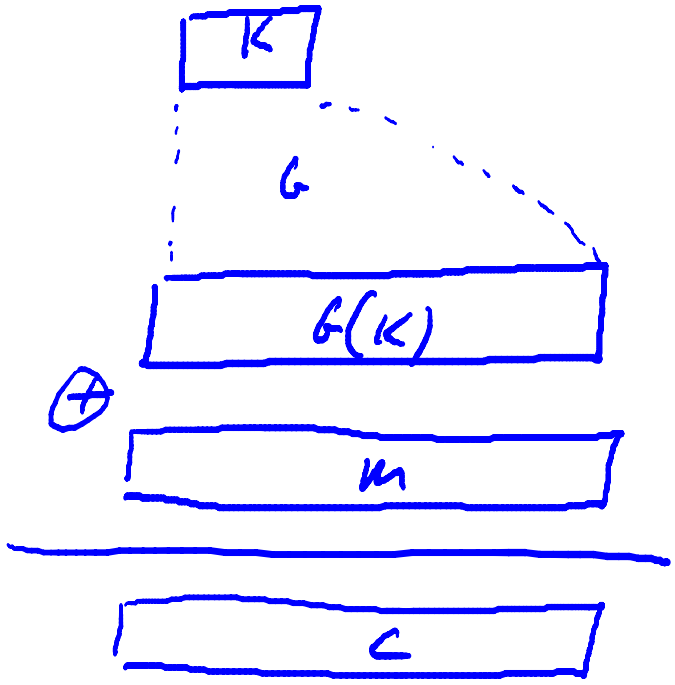
s.t **n >> s**

PRG is efficiently computable by a deterministic algorithm

Stream Ciphers: making OTP practical

$$C := E(K, m) = m \oplus G(K)$$

$$D(K, C) = C \oplus G(K)$$

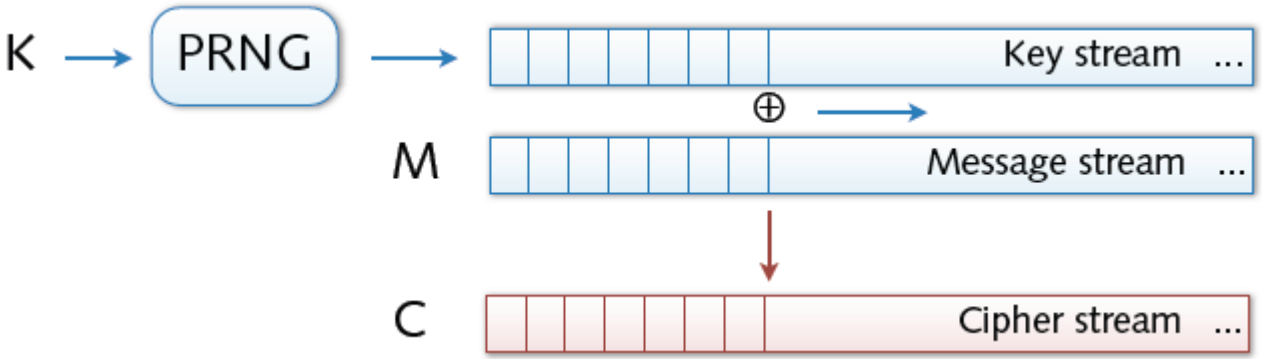


Key (K) is the seed used by G to generate the PRG

Security: PRG must be unpredictable

Stream Ciphers

- **Stream ciphers**
  - Bit-wise encryption and decryption of data
  - Application of pseudo-random number generator (PRG)
  - XOR operation on pseudo-random keystream



- Security solely depends on randomness of PRG

Stream Ciphers

Stream ciphers cannot have perfect secrecy !!

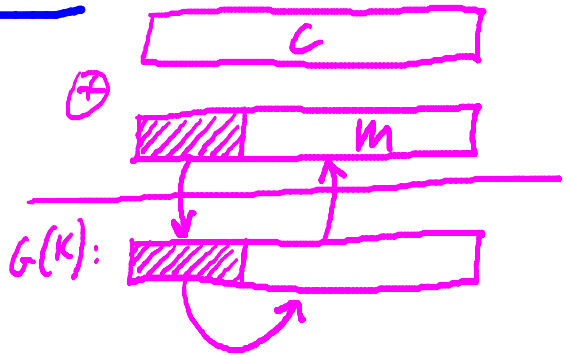
- Need a different definition of security
- Security will depend on specific PRG

PRG must be unpredictable

Suppose PRG is predictable:

$\exists i$: $G(k)|_{1,\ldots,i} \xrightarrow{alg} G(k)|_{i+1,\ldots,n}$

Then:



even $G(k)|_{1,\ldots,i} \rightarrow G(k)|_{i+1}$

is a problem!

PRG must be unpredictable

We say that $G: K \longrightarrow \{0,1\}^n$ is **predictable** if:

$\exists$ "eff" alg. A and $\exists\, 0 \le i \le n-1$ s.t.

$$\Pr_{k \xleftarrow{R} K}\left[A(G(k))\big|_{1,\ldots,i} = G(k)\big|_{i+1} \right] > \frac{1}{2} + \varepsilon$$

For non-negligible ε (e.g. $\varepsilon = 1/2^{30}$)

Def: PRG is **unpredictable** if it is not predictable

$\Rightarrow$ $\forall i$: no "eff" adv. can predict bit (i+1) for "non-neg" ε

Suppose $G:K \longrightarrow \{0,1\}^n$ is such that for all k: **XOR(G(k)) = 1**

Is G predictable ??

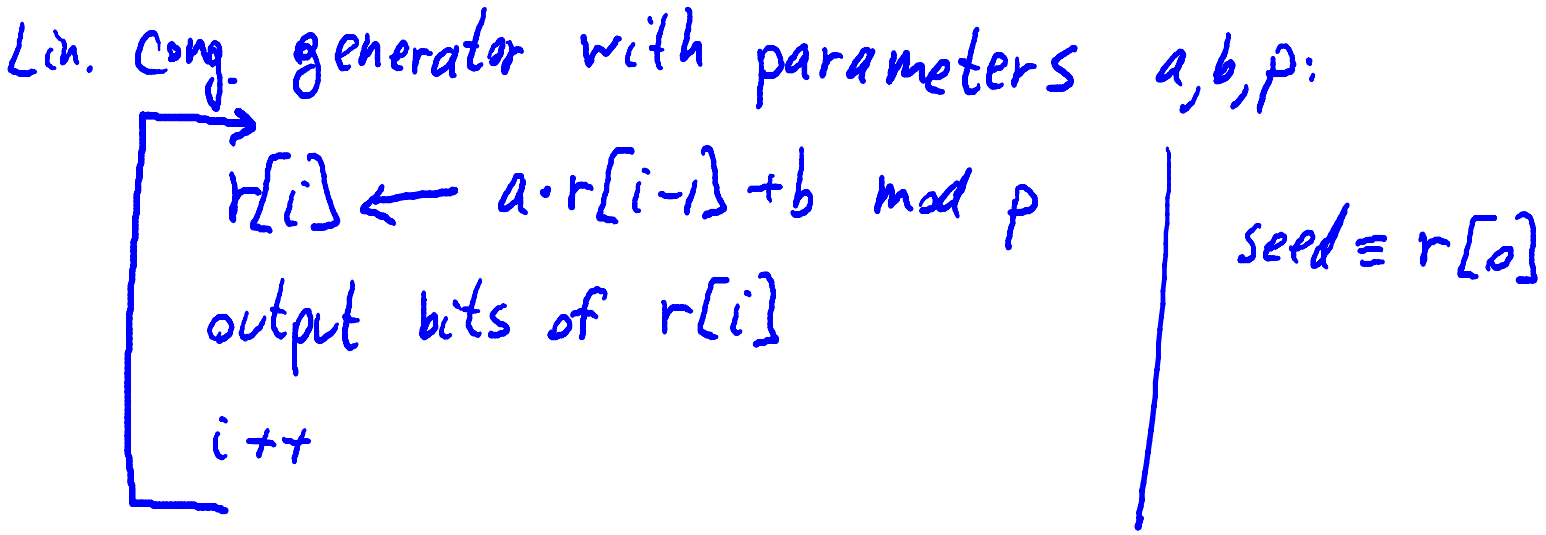
Yes, given the first bit I can predict the second

No, G is unpredictable

Yes, given the first (n-1) bits I can predict the nth bit 

It depends

Weak PRGs (do not use for crypto)



With LCG small number of outputs can be use to predict remaining bits

glibc random():

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

output $r[i] >> 1$

never use random() for crypto !! (e.g. Kerberos V4)

Negligible and non-negligible

- In practice: ε
  - ε non-neg: $\varepsilon \geq 1/2^{30}$ (likely to happen over 1GB of data)
  - ε negligible: $\varepsilon \leq 1/2^{80}$ (won't happen over life of key)

Attacks on OTP and Stream Ciphers

Attack: **two time** pad is insecure !!

Never use stream cipher key more than once !!

$$C_1 \leftarrow m_1 \oplus PRG(k)$$
$$C_2 \leftarrow m_2 \oplus PRG(k)$$

Eavesdropper does:

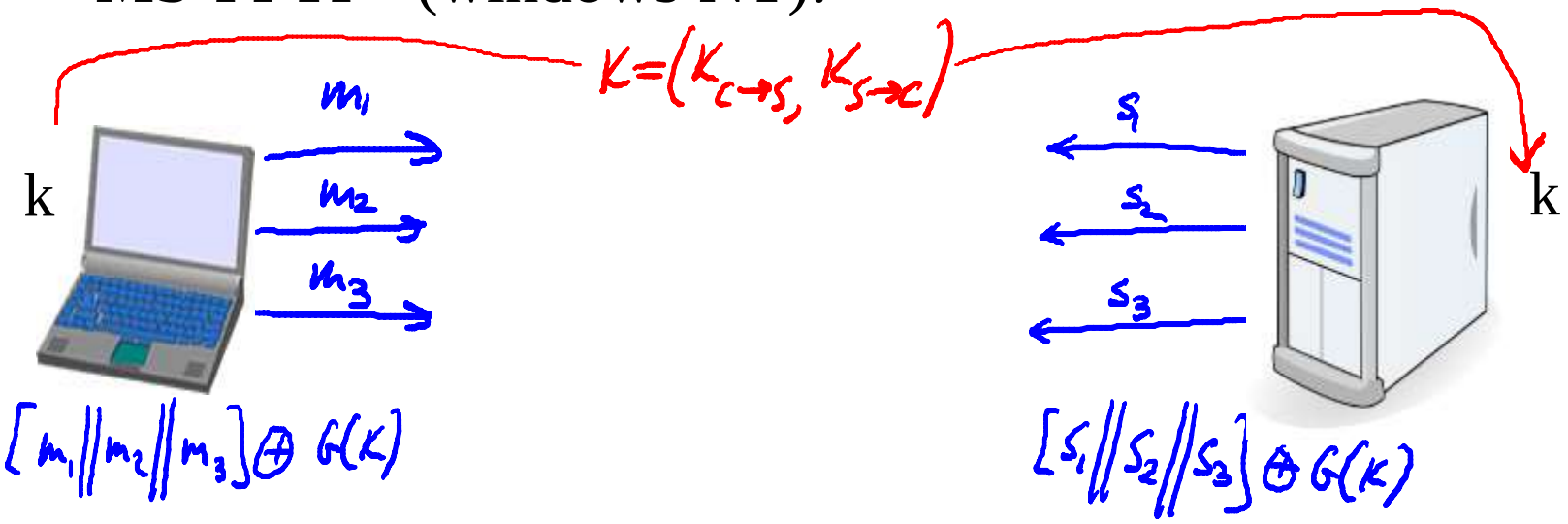
$$C_1 \oplus C_2 \rightarrow m_1 \oplus m_2$$

Enough redundancy in English and ASCII encoding that:

$$m_1 \oplus m_2 \rightarrow m_1 , m_2$$

Real world examples

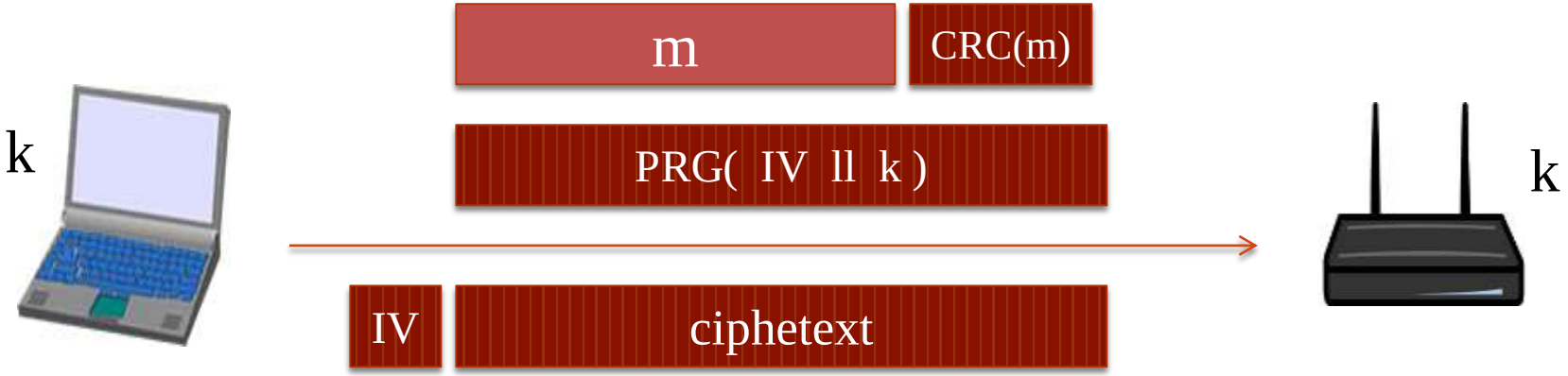
- Project Venona (1941-1946)
  - Roll a dice a compute a pad. This seems boring and they started using the same pads again and again. Soviet messages sent from 1941 to 1946, decrypted about 3000 messages.
- MS-PPTP (windows NT):



Need different keys for C⟶S and S⟶C

Real world examples

802.11b WEP:

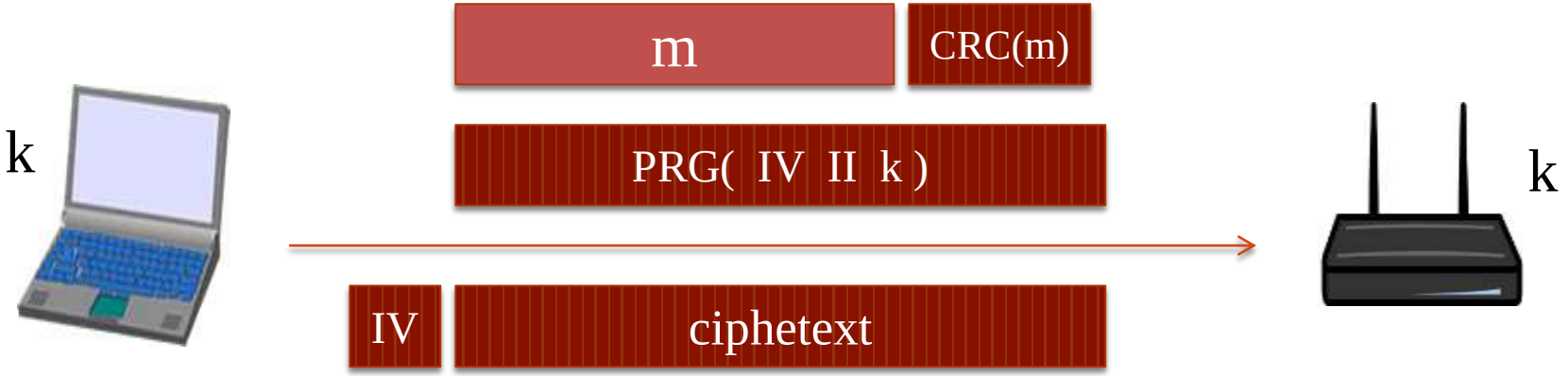


Length of IV: 24 bits

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

Avoid related keys

802.11b WEP:



key for frame #1: (1 II k)

key for frame #2: (2 II k)

⋮

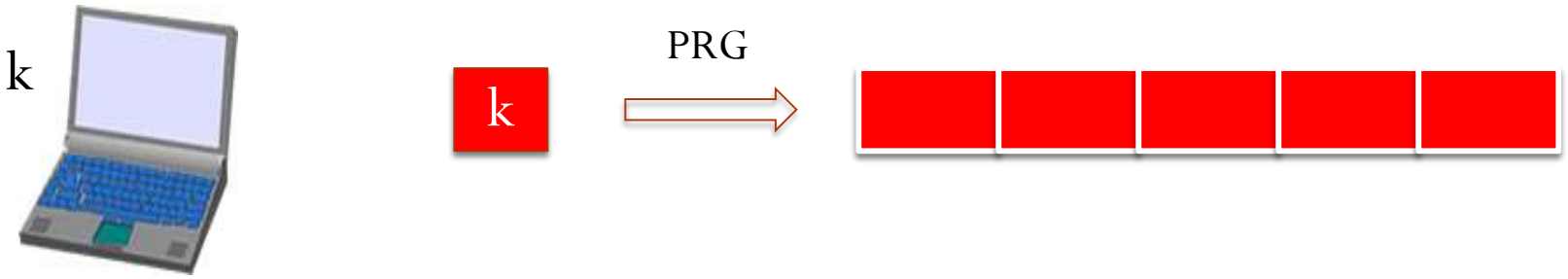
24 bits → ← 104 bits

For the RC4 PRG:

FMS2001 ⇒ can recover k after 10^6 frames

Recent attacks ≈ 40,000 frames

A better construction



⇒ now each frame has a pseudorandom key

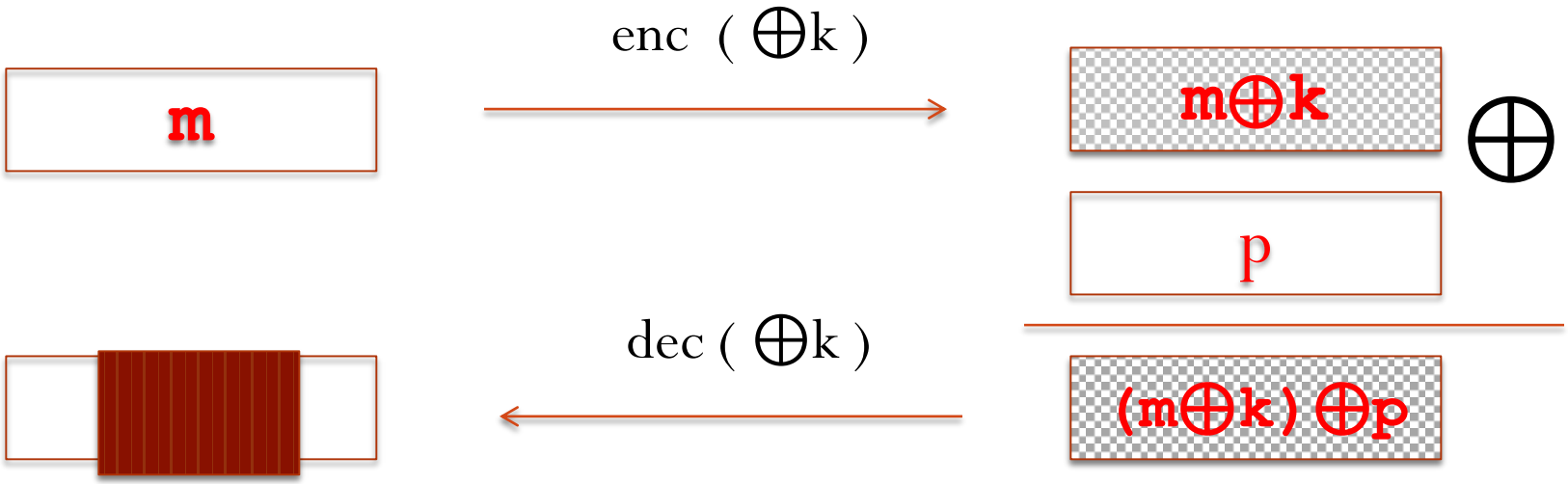
better solution: use stronger encryption method (as in WPA2)

Two time pad: summary

Never use stream cipher key more than once !!

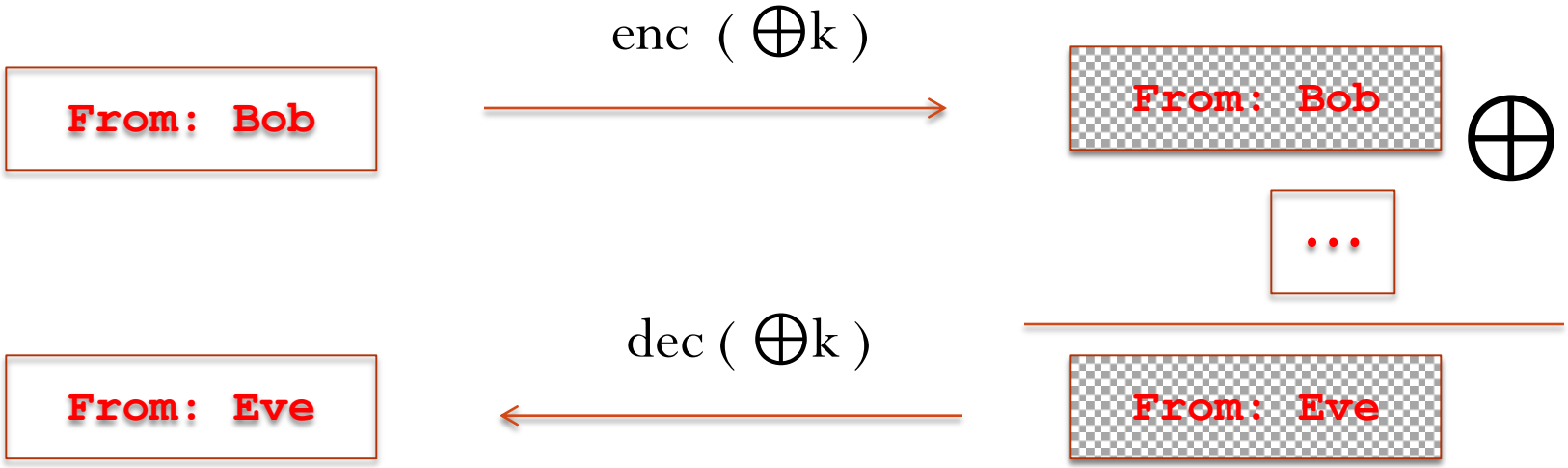
- Network traffic: negotiate new key for every session (e.g. TLS)
- Disk encryption: typically do not use a stream cipher

Attack 2: No Integrity (OTP is malleable)



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

Attack 2: No Integrity



Modifications to ciphertext are undetected and have predictable impact on plaintext

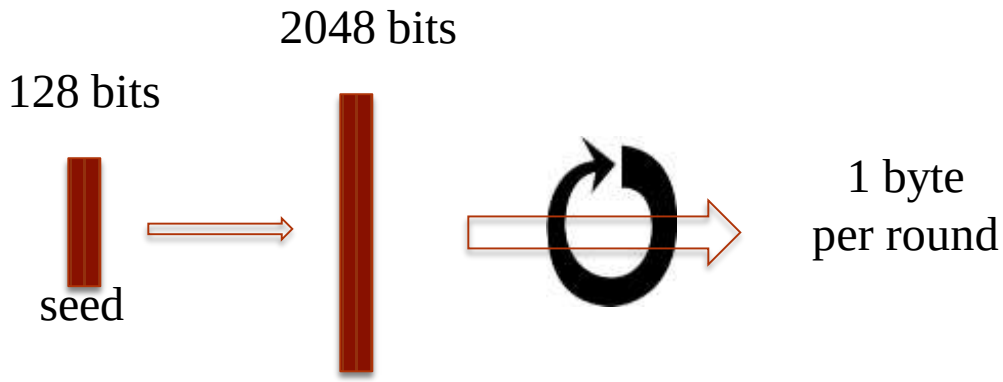
Real-world Stream Ciphers

RC4 Cipher

- **Common stream cipher**
  - Developed by Ron Rivest for RSA Security
  - Leaked to the public in 1994 (ARC4 = Alleged RC4)
  - Key size (seed): 40 to 256 bits

- **Some known weaknesses, e.g., in WEP implementation**

Old example (software): RC4 (1987)



- Used in HTTPS and WEP
- Weaknesses:
  1. Bias in initial output: Pr[2nd byte = 0] = 2/256
  2. Prob. of (0,0) is $1/256^2 + 1/256^3$ (happens in GBs of Data)
  3. Related key attacks

Old example (hardware): CSS (badly broken)

Linear feedback shift register (LFSR):

DVD encryption (CSS): 2 LFSRs

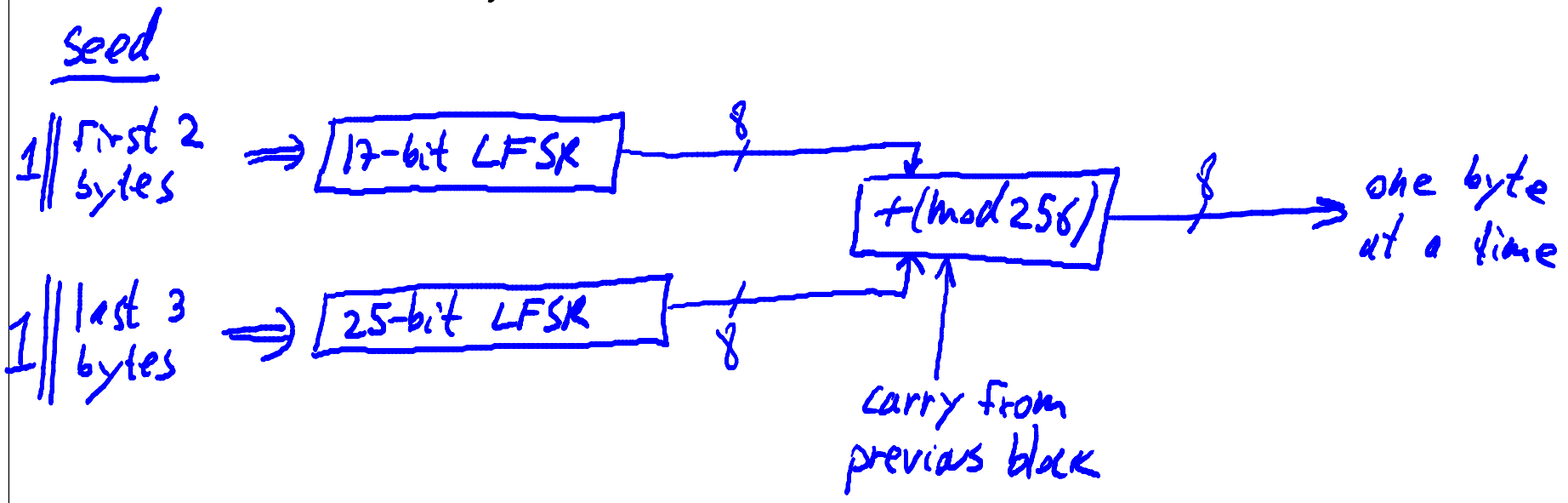
GSM encryption (A5/1,2): 3 LFSRs

Bluetooth (E0): 4 LFSRs

} all broken

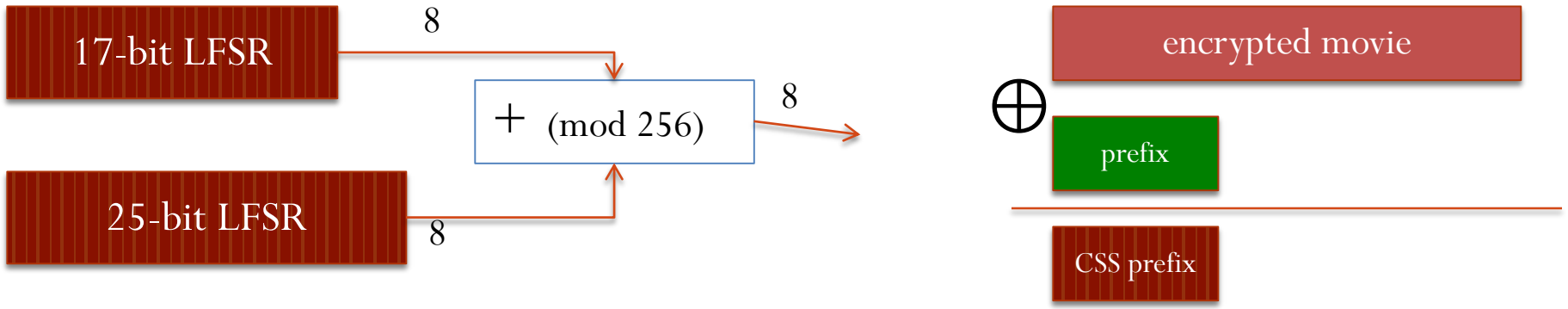
Old example (hardware): CSS (badly broken)

CSS: seed = 5 bytes = 40 bits



Easy to break in time $\simeq 2^{17}$

Cryptanalysis of CSS (2^{17} time attack)



For all possible initial settings of 17-bit LFSR do:

- Run 17-bit LFSR to get 20 bytes of output
- Subtract from CSS prefix ⇒ candidate 20 bytes output of 25-bit LFSR
- If consistent with 25-bit LFSR, found correct initial settings of both
- Using key, generate entire CSS output

Modern stream ciphers: eStream (2008)

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

seed

nonce

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.

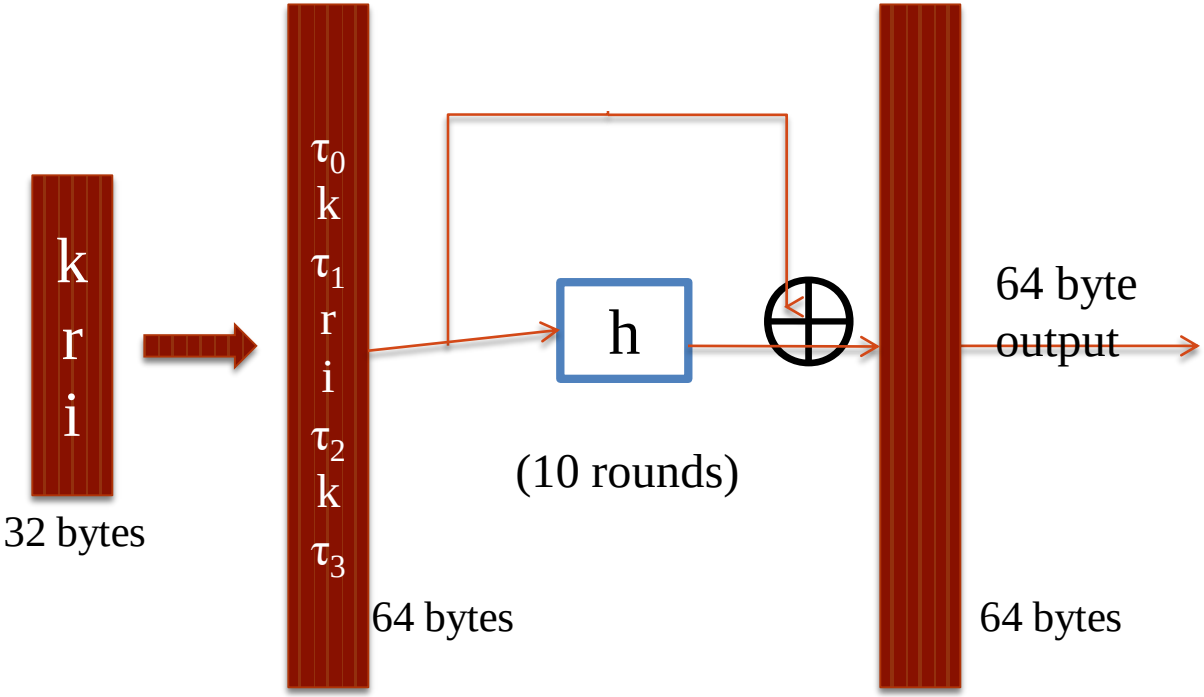
eStream: Salsa 20 (SW+HW)

nonce

Salsa20: $\{0,1\}^{128 \text{ or } 256} \times \{0,1\}^{64} \longrightarrow \{0,1\}^{n}$

Salsa20(k ; r) := H(k , (r, 0)) ll H(k , (r, 1)) ll …

0,1… are counters that goes from step to step



h: invertible function. designed to be fast on x86

Is Salsa20 secure (unpredictable) ?

- Unknown: no known **provably** secure PRGs
- In reality: no known attacks better than exhaustive search

Performance:

AMD Opteron, 2.2 GHz (Linux)

	PRG	Speed (MB/sec)
	RC4	126
eStream	Salsa20/12	643
eStream	Sosemanuk	727

Acknowledgements

Material in this lecture are taken from the slides prepared by:

- Prof. Dr. Konrad Rieck (Uni-Göttingen)
- Prof. Dan Boneh (Stanford)