CS 578 – Cryptography

Prof. Michael Backes

Stream Ciphers

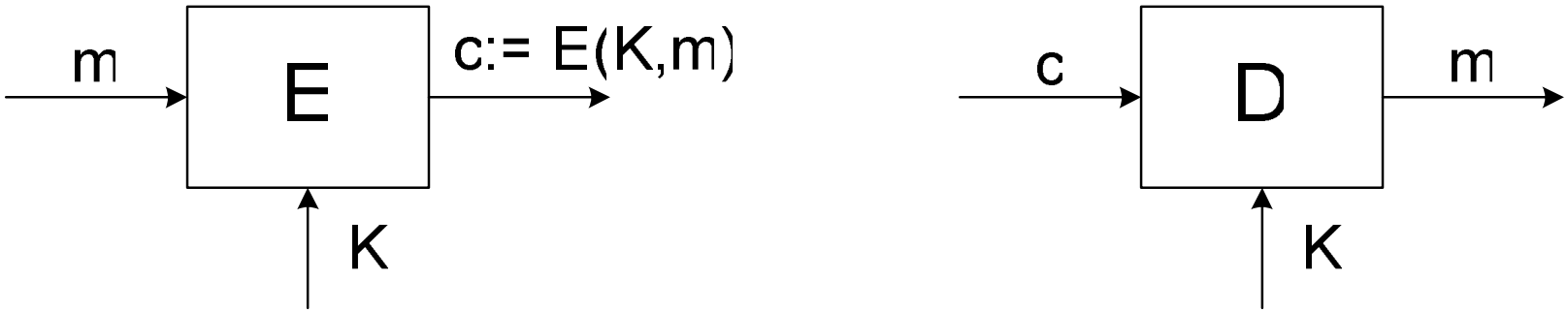
April 25, 2006

Organization of Practical Classes

- More practical classes:
  - Wednesday 4 x 1-2pm, 4 x 4-5pm (start: 1.10pm and 4.15pm)
- New teaching assistants:
  - Stefan Lorenz, Kim Pecina, Oana Ciabotaru
  - (in addition to Michaela Götz, Dirk Heine, Esfandiar Mohammadi)
  - cs578@mail-infsec.cs.uni-sb.de
- Tutor Office Hours (in E 1 1, U 19)
  - Wed 9-11, Thu 14-16, Fri 14-16 (might still change)
- New discussion board (can also be used for discussion current exercises, please use English)
  - http://infsec.cs.uni-sb.de/wbb2/

Recall: Ciphers

- Recall: Symmetric ciphers as pair (E,D) of algorithms defined over $(\mathcal{K},\mathcal{M},\mathcal{C})$



s.t. for all K,m: D(K,E(K,m)) = m.

Recall: The One-time Pad (OTP)

- First “proven secure” cipher: One-time Pad (Vernam 1917, proven in 1949)

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

- Secret key K = random bitstring as long as the message
- Encryption: c = E(K,m) = K $\oplus$ m
- Decryption: m = D(K,c) = K $\oplus$ c

Recall: Types of Adversary Success

- Here only for encryption (authentication later in the course)
  1. Total break: find the key
  2. Universal-break: find equivalent method to being able to decrypt with key
  3. Successfully decrypt only selected ciphertexts, but those completely
  4. Successfully learn partial information about single plaintexts (individual bits, checksum, etc.)
- 1. , 2. and 3. clearly unacceptable
- 4. strong, but on the safe side

Recall: Perfect Secrecy

- A cipher (E,D) defined over $(\mathcal{M},\mathcal{K},\mathcal{C})$ has perfect secrecy if for all $m_0, m_1 \in \mathcal{M}$, and for all $c \in \mathcal{C}$:

  $\Pr[c = c'; K \leftarrow_R \mathcal{K}, c' \leftarrow E(K,m_0)]$
  $= \Pr[c = c'; K \leftarrow_R \mathcal{K}, c' \leftarrow E(K,m_1)]$

- Perfect Secrecy implies that there are no ciphertext-only attacks!

Recall: Perfect Secrecy of the OTP

- Lemma: OTP has perfect secrecy (for fixed-size messages).

→ no ciphertext-only attacks on the OTP

→ But keys as long as the message, thus fully unpractical

Bad News: OTP is optimal

- Theorem: If any cipher (E,D) defined over $(\mathcal{M},\mathcal{K},\mathcal{C})$ has perfect secrecy, then $|\mathcal{K}| \geq |\mathcal{M}|$.

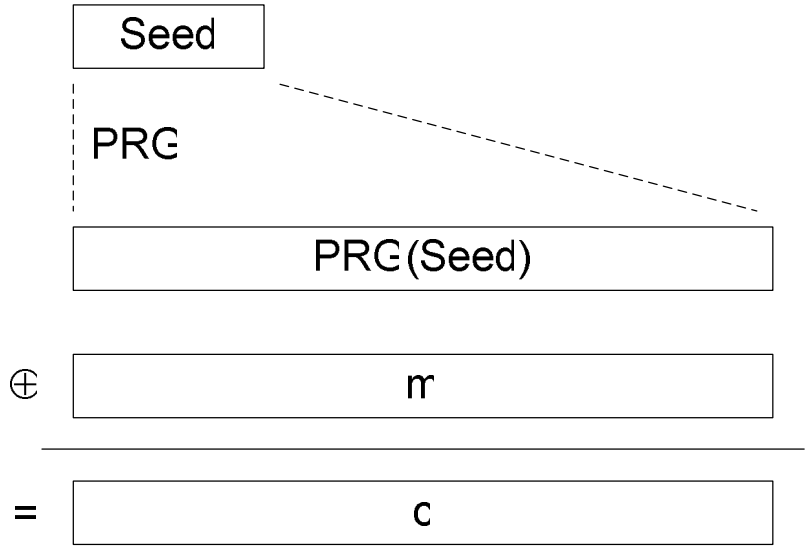
  [proof on the board]

- Thus perfect secrecy implies that (key length ≥ message length)

Stream Ciphers

- OTP: secret key = random string in $\{0,1\}^n$
- Idea: Replace “random” by “pseudo random”

- Stream cipher: secret key = “seed”



Formal Definition of Stream Ciphers

- Definition (Synchronous Stream Cipher):A synchronous stream cipher over $(\mathcal{K},\mathcal{L},\mathcal{M},\mathcal{C})$ is a tuple of algorithms (E,D,G) where

  G: $\mathcal{K} \rightarrow \mathcal{L}^*$

  E: $\mathcal{L} \times \mathcal{M} \rightarrow \mathcal{C}$

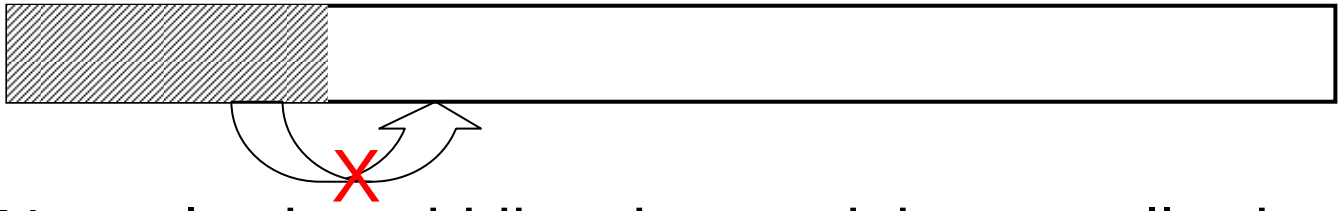
  D: $\mathcal{L} \times \mathcal{C} \rightarrow \mathcal{M}$

  such that for all m $\in$ $\mathcal{M}$, z $\in$ $\mathcal{L}$:
  D(z,E(z,m)) = m.

Stream Ciphers (cont'd)

- No more perfect secrecy for stream ciphers!
- Security depends on the PRG
- Question: What sort of property do PRG need to satisfy so that a stream cipher is "secure"?

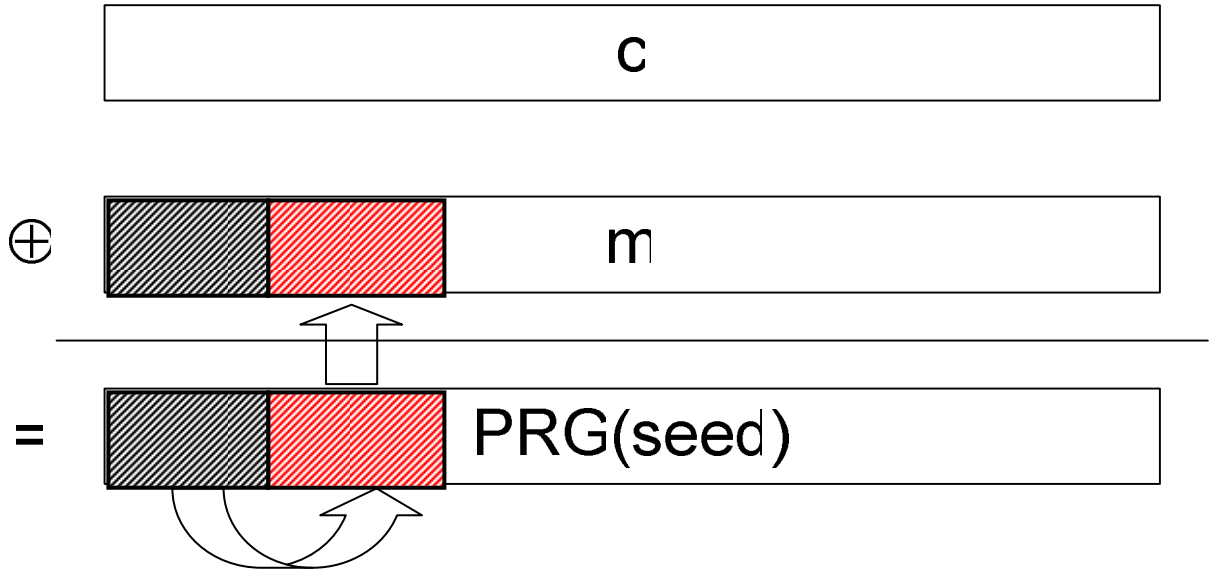
- PRG should be unpredicatable:



- Not only about hiding the seed, but not allowing to "look forward" which randomness will be created

Why Unpredictability of PRGs?

- Often prefix of message known
  → Fixed headers if protocols known, etc.



- Lots of PRGs don't satisfy this
- Don't use UNIX rand for security (/dev/rnd better)!

Formal Definition of Unpredictability

- $G:\{0,1\}^k \rightarrow \{0,1\}^{p(k)}$ satisfies next-bit unpredictability iff for every $0 \leq i \leq p(k)$:

$\forall A \in PPT$:	(All efficient adv.'s)
$\Pr[b = b_i;$	(Attacker success)
$seed \leftarrow_R \{0,1\}^k,$	(Seed)
$(b_1, \ldots, b_{p(k)}) \leftarrow G(k,seed),$	(Bit generations)
$b \leftarrow A(b_1, \ldots, b_{i-1})]$	(Adv.'s guess)
$\leq 1/2 + \varepsilon(k)$	(Negligible)

Getting True Randomness

- How to get (weak) randomness in practice:
  - User throws coins,…
  - User types, derived from load/system parameters, …
- Stronger: Exploit different physical processes that are expected to be random (some provably random)
  - Thermal noise, air perturbation,
- Randomness from all inputs XORed and hashed to remove bias
- Works well but too slow for some purposes
  - For instance RG in INTEL designs 10Kbit/s

→ Use true randomness as seed for PRG

- Good PRGs also allow to add entropy (OpenSSL)

Attacks on OTP and Stream Ciphers

- "Two-time" pad:
  - $c_1 = E(K,m_1) = K \oplus m_1$
  - $c_2 = E(K,m_2) = K \oplus m_2$

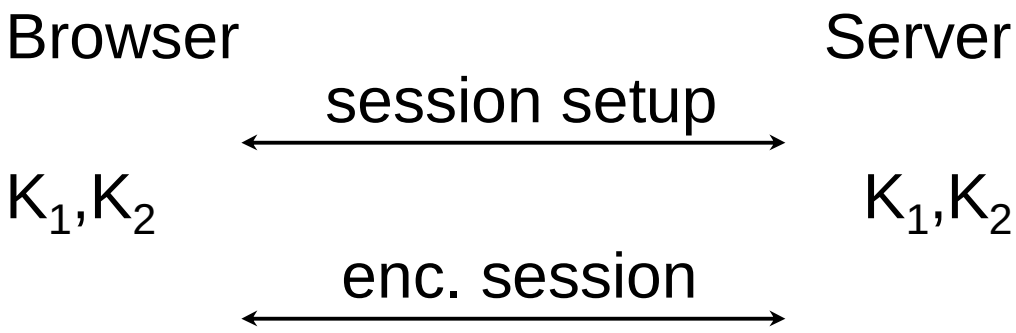
→ $c_1 \oplus c_2 = m_1 \oplus m_2$

→ Vulnerable to frequency analysis, etc.

→ Keys must only be used once!

How to Use a Stream Cipher

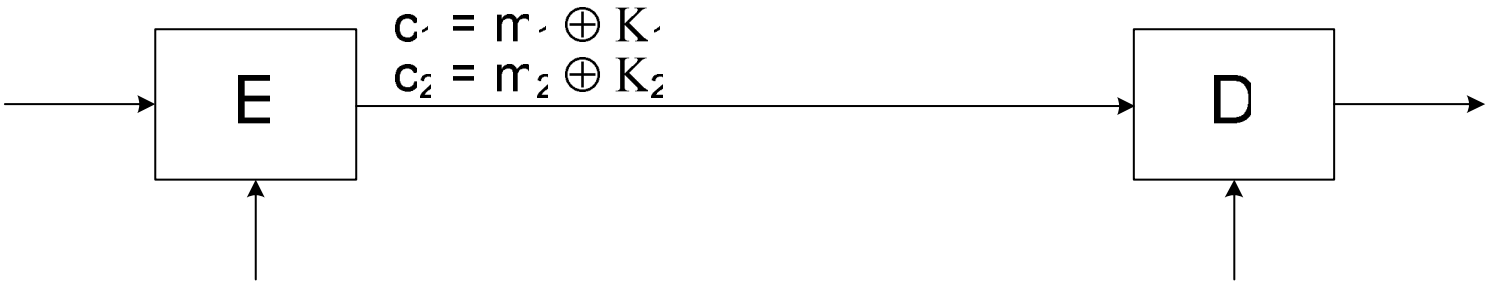
- SSL:



- New encryptions computed by later parts of the pseudo-random stream
  → Stream cipher is only used once in SSL
- File encryption: pick random seed
  $E(K,m) = \underbrace{E^*(K,seed)}_{\text{Strong cipher}} \,||\, \underbrace{M \oplus PRG(seed)}_{\text{Fast cipher}}$

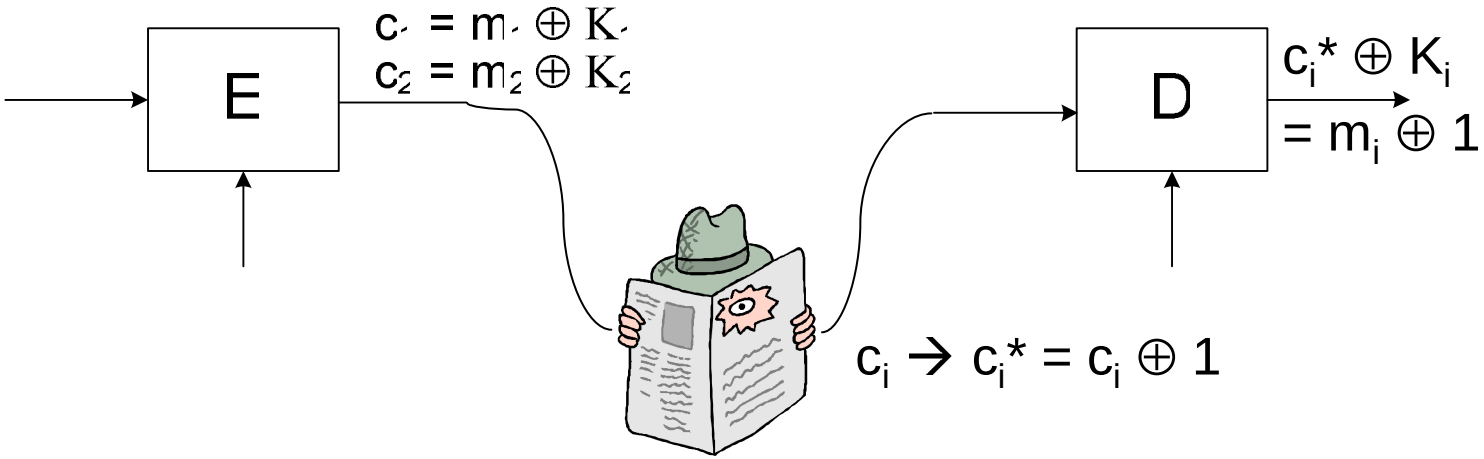
Attacks on OTP and Stream Ciphers

- OTP highly malleable:
  - $m_i \in \{0,1\}$
  - $K_i \in \{0,1\}$
  - $c_i = E(K_i,m_i) = K_i \oplus m_i$
- Assume that $m_i=0$ with prob. > 80% (e.g., in voting)



Attacks on OTP and Stream Ciphers

- OTP highly malleable:
  - $m_i \in \{0,1\}$
  - $K_i \in \{0,1\}$
  - $c_i = E(K_i,m_i) = K_i \oplus m_i$
- Assume that $m_i=0$ with prob. > 80% (e.g., in voting)

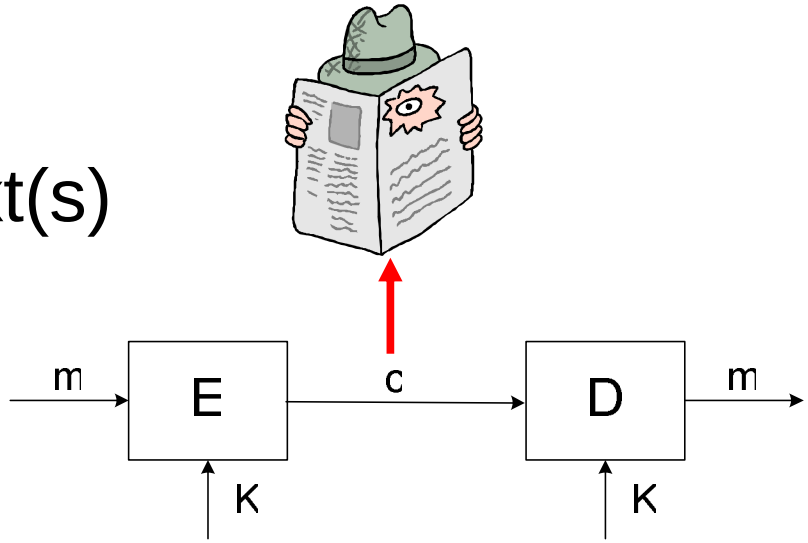


Attacks on OTP and Stream Ciphers

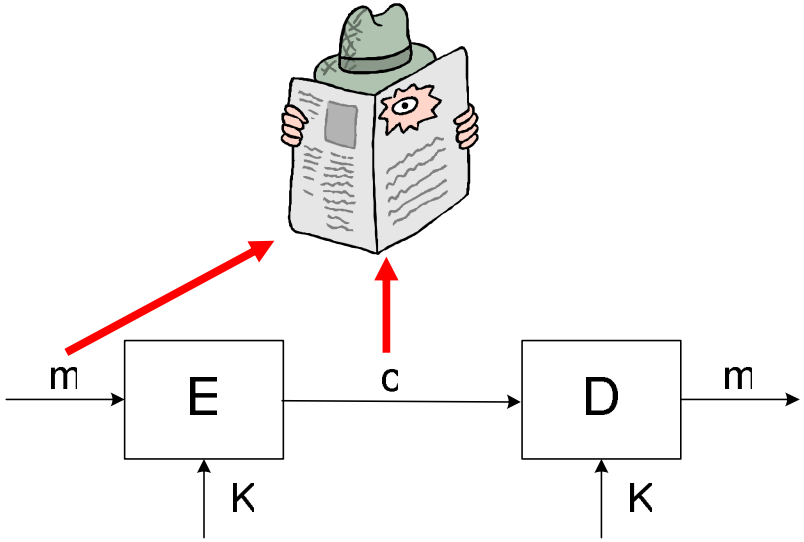
- OTP and stream ciphers are vulnerable to active attacks (chosen message, chosen ciphertext attacks)
- Defense: Use integrity mechanism (MACs)

On Attacks Models (for ciphers)

- Ciphertext-only attack: observation of ciphertext(s) (passive)

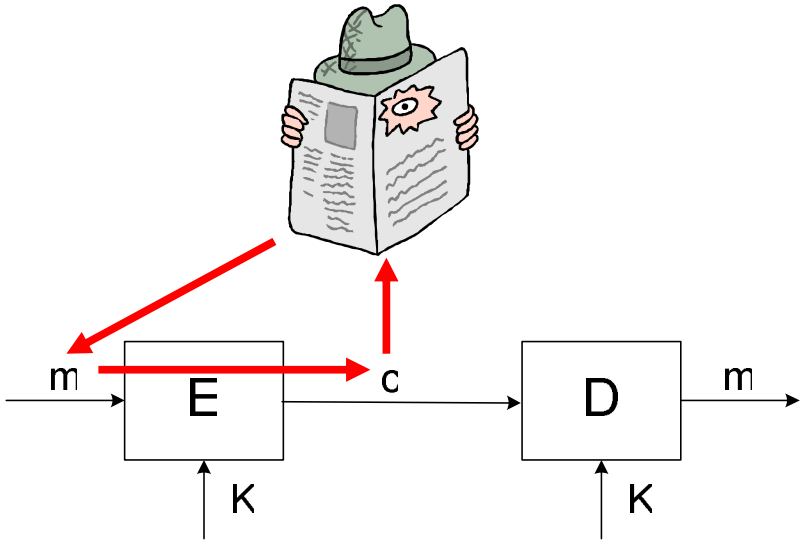


- Known plaintext attack: observation of plain- and ciphertexts (passive)

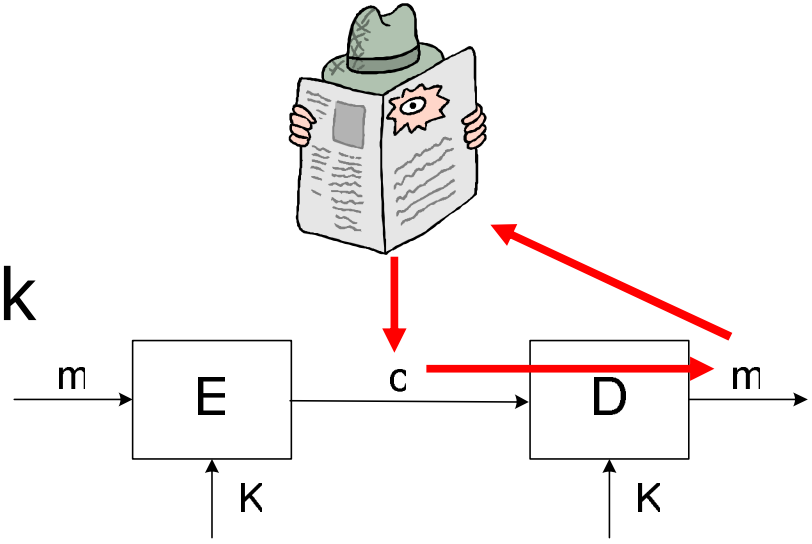


On Attacks Models (cont'd)

- Chosen plaintext attack:
  Plaintexts selectable
  (active)



- Chosen ciphertext attack
  Ciphertexts selectable
  (active)

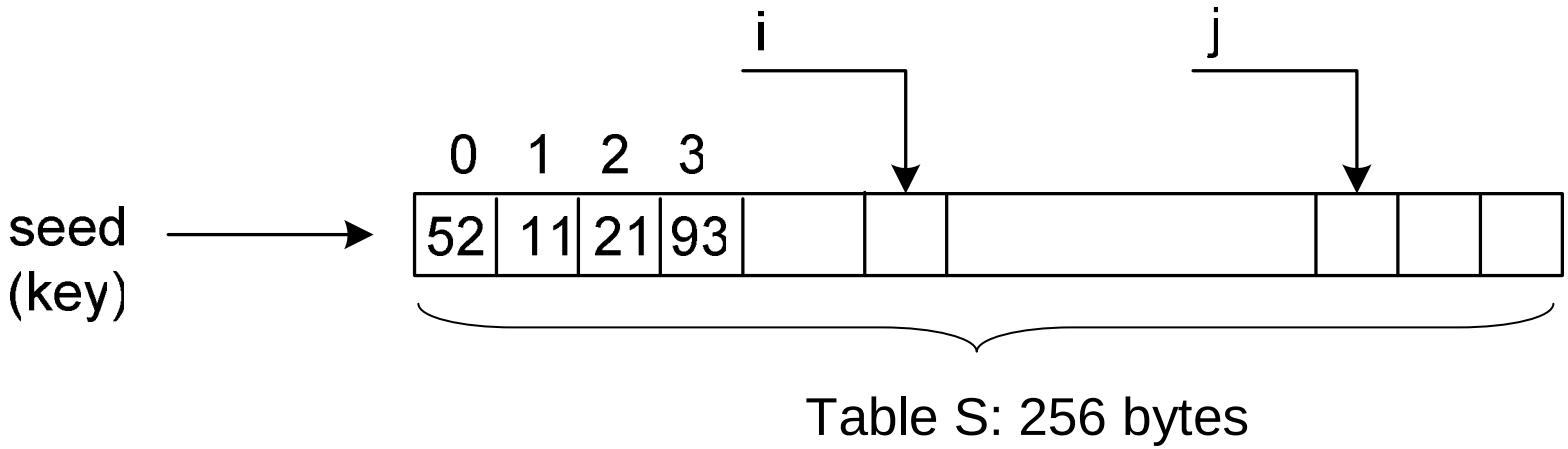


Example of PRGs

- Fast ones from practice: RC4, LFSRs,
- Slow but provably secure ones:
  - Blum-Blum-Shub (BBS): based on factorization
  - Blum-Micali: based on discrete logorithm.

Example PRG: RC4

- RC4 invented by Rivest ‘87
- Starts with a seed, keeps a table S



- S stores permutation of {0,…,255}
- Initialization: $\mathcal{K}$ → Perm({0,…,255})

RC4 Program and Initialization

- RC4 Program:
  - i = j = 0
  - i = i + 1 (mod 256)
  - j = i + S[i] (mod 256)
  - Swap S[i] and S[j]
  - t = S[i] + S[j] (mod 256)
  - Output S[t]
- Init (given key K):
  - For (i=0, i++, i < 256)
  - j = 0, S[i] = i
  - For (i=0, i++, i < 256)
  - j = j + S[i] + K[i mod keylen] (key could be short)
  - Swap S[i] and S[j]

Attacks on RC4

- Widely used: SSL, TLS, 802.11b WEP
- Some successful (weak) attacks on RC4:
  - Problems with Init: Second byte is twice as likely to be 0 as it should (2/256 instead of 1/256):

    → Drop first 256 bytes of output
  - Statistical attacks: Probability of seeing (0,0) is $1/256^2 + 1/256^3$ (i.e. biased by $1/256^3$)

    → After $(256^2)^3 = 2^{48} \approx 10^{16}$ bytes, one can distinguish RC4 output from random with probability > 99%.

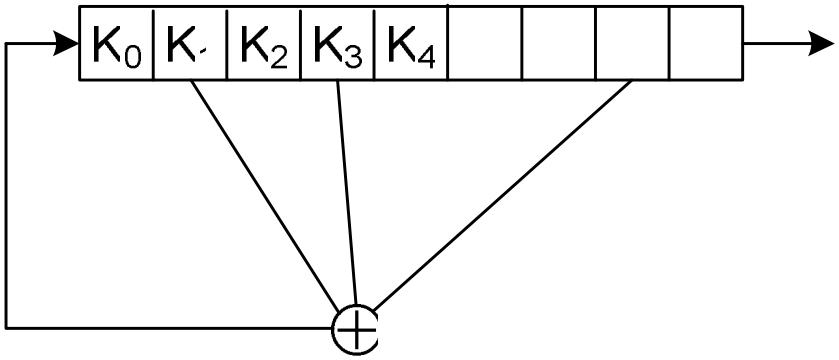
→ RC4 not provably secure but works pretty well in practice (and used in lots of products such as web browsers)

Brief Excerpt: Provably Secure PRGs

- Blum-Blum-Shub, 1986
- Setup:
  1. Choose two primes p, q random, large primes
  2. Set n = pq.
  3. Seed s_0 = random string in of length n which is invertible mod n (element of Z^*_n)
- Generation of pseudorandom string (i-th bit)
  - State update: $s_i = (s_{i-1})^2 \bmod n$
  - Next bit: $b_i = s_i \bmod 2$
- Provably secure under computational assumption that factoring is hard (more later in the course)
- Style of proof: “If prob. poly-time A breaks a scheme with prob. ε, then a prob. poly-time A’ exists that solves another problem that is assumed intractable with a similar prob.”

Hardware PRG

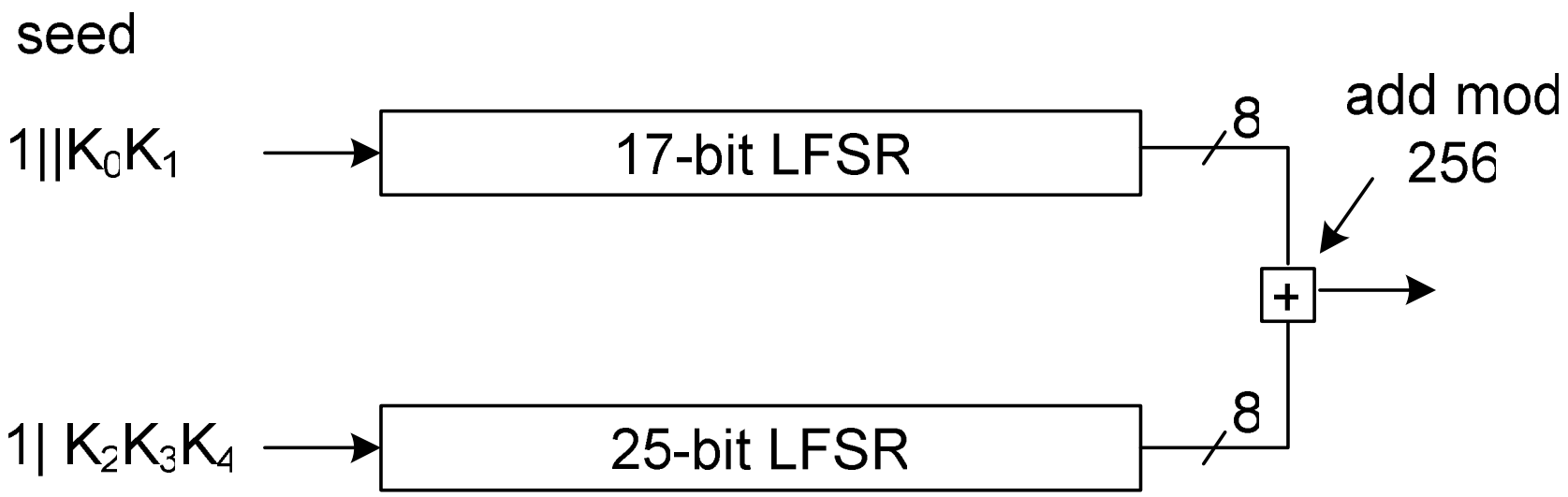
- LFSR (Linear Feedback Shift Register)
  - used in CSS, GSM
  - Standard solution for doing cheap hardware encryption (as a stream cipher)



- Seed: Initial value of the register
- On their own not usable: first bits output are key bits

CSS

- Content Scrambling System (CSS)
- Key = 40 bits = 5 bytes ($K_0K_1K_2K_3K_4$)



- Easy to break in time ca. 2^{20} (2^{40} with brute force)