

Stream Ciphers

Martin Stanek

Department of Computer Science
Comenius University
`stanek@dcs.fmph.uniba.sk`

Cryptology 1 (2022/23)

Content

Introduction

- idea, general properties

Examples of stream ciphers

- RC4

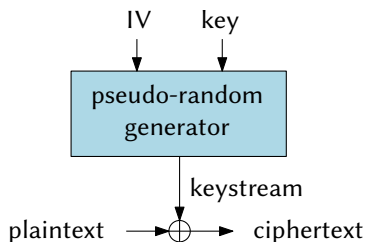
- ChaCha20

- Snow 3G

Introduction

- ▶ Vernam cipher (one-time pad)
 - ▶ perfect secrecy
 - ▶ impractical – long key that cannot be reused
- ▶ (some) stream ciphers examples:
 - ▶ RC4 – old software and protocols, e.g. WEP, SSL/TLS etc.
 - ▶ E0 – Bluetooth (BR/EDR – basic rate/enhanced data rate)
remark: Bluetooth Low Energy uses AES-CCM
 - ▶ ChaCha20 – TLS (RFC 7905)
- ▶ basic types of stream ciphers: synchronous and self-synchronizing

Synchronous stream ciphers

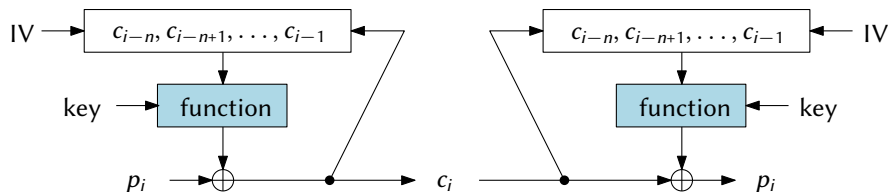


- ▶ the most common stream ciphers used in practice
- ▶ encryption and decryption are the same
- ▶ keystream does not depend on plaintext
- ▶ usually binary additive stream ciphers (XOR of plaintext and keystream)

Synchronous stream ciphers 2

- ▶ periodic
- ▶ require synchronization
 - ▶ decryption breaks after losing some bits of ciphertext
- ▶ vulnerable to active attacks
 - ▶ e.g. changing bits in ciphertext results in change of corresponding plaintext bits
- ▶ errors are not propagated
- ▶ IV and key must not repeat (otherwise ...two-time pad)
 - ▶ be careful of possible keystreams overlaps

Self-synchronizing stream ciphers



- ▶ keystream depends on ciphertext (and therefore on plaintext)
- ▶ ability to self-synchronize after the loss of some ciphertext
- ▶ aperiodic
- ▶ hard to analyze, hard to guarantee security properties

Remarks

- ▶ stream ciphers can be constructed from block ciphers
- ▶ specific modes of operation:
 - ▶ synchronous: OFB, CTR
 - ▶ self-synchronizing: CFB
- ▶ Why stream ciphers at all?
 - ▶ speed
 - ▶ simplicity (HW implementation, constrained environment)
- ▶ requirements (preliminary observations):
 - ▶ long period
...How do you attack stream cipher with short period?
 - ▶ good statistical properties
...statistical tests of randomness are not sufficient
 - ▶ keystream should be *unpredictable* (*indistinguishable* from a random sequence)
...KPA \Rightarrow knowing some part of the keystream

RC4

- ▶ Ron Rivest, 1987
- ▶ trade secret; posted anonymously to a mailing list in 1994
- ▶ internal state $S[0 \dots 255]$ – permutation $\{0, \dots, 255\}$
- ▶ key $K[0 \dots k]$ – array of bytes (16 for 128-bit key)
- ▶ initialization:

```
for  $i = 0, \dots, 255$ :  $S[i] = i$ ;  
 $j = 0$ ;  
for  $i = 0, \dots, 255$ :  
     $j = (j + S[i] + K[i \bmod k]) \bmod 256$ ;  
    swap( $S[i], S[j]$ );
```

RC4 (2)

- ▶ generating keystream:

```
 $i = 0; j = 0;$   
while (is needed):  
     $i = (i + 1) \bmod 256;$   
     $j = (j + S[i]) \bmod 256;$   
    swap( $S[i], S[j]$ );  
    output  $S[(S[i] + S[j]) \bmod 256];$ 
```

- ▶ additive cipher, the output is XOR-ed with plaintext bytes
- ▶ first bytes of keystream leak information about key
 - ▶ WEP attack (key and IV used as RC4 key)
 - ▶ drop some keystream prefix / different construction of the key

Klein's attack on WEP 1

- ▶ WEP (Wired Equivalent Privacy) – security for 802.11 WiFi networks
 - ▶ superseded by WPA2 (WiFi Protected Access)
- ▶ data frame:

$$\underbrace{\text{IV, padding, ID}_{\text{Rk}}}_{\text{plaintext}}, \underbrace{\text{data, ICV}}_{\text{encrypted}}$$

- ▶ IV – initialization vector (3B)
 - ▶ ID_{Rk} – Rk's identifier (2 bits)
 - ▶ ICV – integrity check value (CRC32)
- ▶ RC4 with key $K = \text{IV} || \text{Rk}$ (Rk – root key)
- ▶ Notation:
 - ▶ S_i – internal permutation after i -th round ($i \leq 256$ corresponds to initialization)
 - ▶ j_i – internal variable j after i -th round
 - ▶ X – keystream (obtained by XORing ciphertext and known plaintext data)

Klein's attack on WEP 2

- ▶ Klein proved the following property of RC4 ($n = 256$):

$$\Pr[K[i \bmod k] = S_i^{-1}[i - X[i - 1]] - (S_i[i] + j_i)] \approx \frac{1.36}{n}$$

instead of desired $1/n$.

- ▶ $IV = K[0], K[1], K[2]$ is known $\Rightarrow S_3$ and j_3 can be computed
- ▶ the value $w = S_3^{-1}[3 - X[2]] - (S_3[3] + j_3)$ is $K[3]$ with probability $\approx \frac{1.36}{n}$
- ▶ attacker observes many frames (fixed R_k and different IV) ... correct value of $K[3]$ (the first byte of R_k) revealed by statistics
- ▶ knowing $K[3] \Rightarrow$ next RC4 round computation: $S_4, j_4 \dots$ etc.
- ▶ improvements for WEP, e.g. PTW attack (2007)
- ▶ attack on RC4 in TLS: AlFardan et al. (2013)

ChaCha20

- ▶ high-speed ARX cipher (add-rotate-xor)
- ▶ designed by D.J. Bernstein (2008)
- ▶ details described e.g. in RFC 8439
- ▶ ChaCha20 – specific instance of ChaCha with 20 rounds
- ▶ state: 4×4 matrix, elements are 32-bit words
- ▶ inputs:
 - ▶ key: 256 bits (8 words)
 - ▶ nonce (IV): 96 bits (3 words)
 - ▶ counter: 32 bits (1 word) \Rightarrow max. 256 GB
- ▶ output: 512 bits (64 bytes, 16 words)
- ▶ different nonce/counter lengths possible (we follow RFC 8439)

ChaCha20 – initialization and quarter-round

const ⁰	const ¹	const ²	const ³
key ⁴	key ⁵	key ⁶	key ⁷
key ⁸	key ⁹	key ¹⁰	key ¹¹
cnt ¹²	nonce ¹³	nonce ¹⁴	nonce ¹⁵

QuarterRound(a, b, c, d) :

a += b; d ^= a; d <<<= 16;

c += d; b ^= c; b <<<= 12;

a += b; d ^= a; d <<<= 8;

c += d; b ^= c; b <<<= 7;

ChaCha20 – block function

- ▶ iterate 10 times following two rounds:

QuarterRound(0, 4, 8, 12)

QuarterRound(1, 5, 9, 13)

QuarterRound(2, 6, 10, 14)

QuarterRound(3, 7, 11, 15)

QuarterRound(0, 5, 10, 15)

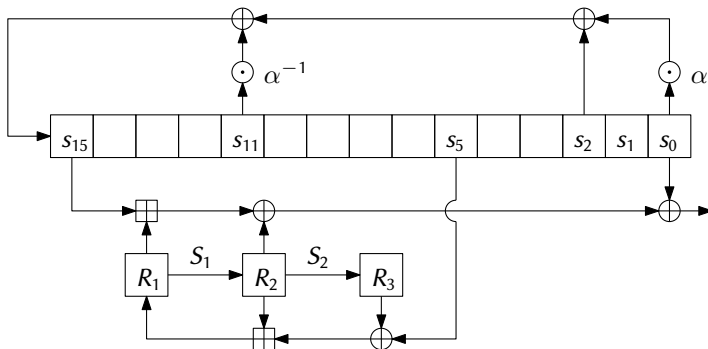
QuarterRound(1, 6, 11, 12)

QuarterRound(2, 7, 8, 13)

QuarterRound(3, 4, 9, 14)

- ▶ the output state is added (word by word) to the input state \mapsto keystream block
- ▶ the output state is used again as an input to the block function

Snow 3G – keystream generator



- ▶ SNOW 3G is the base of confidentiality and integrity algorithms UEA2 and UIA2 (for LTE)
- ▶ LSFR: 16 32-bit words; S_1 , S_2 – s-boxes
- ▶ FSM (finite state machine): R_1 , R_2 , R_3 – 32-bit values
- ▶ α is the root of some fixed polynomial