Stream Cipher (流密码)

Sheng Zhong Yuan Zhang

Computer Science and Technology Department
Nanjing University

Outline

Stream ciphers (流密码)

What is a stream cipher? What is it used for?

- A deterministic algorithm that extends a short random seed to a stream of “random-looking” bits.
- Used as a cryptographic primitive, to instantiate PRGs, encrypt messages, ...
- Pro: good efficiency, **CON**: has no rigorous security proof

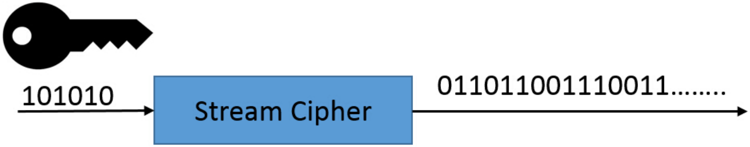


图 1: A stream cipher

Formal definition of stream ciphers

A **stream cipher** is a pair of deterministic algorithms:

- **Init**: $st_0 := Init(s, [IV])$.
- **GetBits**: $(y_i, st_i) := GetBits(st_{i-1})$ for $i = 1, 2, \ldots$

where

1. $st_0, st_1, \ldots$ are state information;
2. *Init* is an initialization algorithm that takes as input a seed s and an optional initialization vector IV, and outputs an initial state st_0;
3. *GetBits* algorithm takes as input a state st_{i-1} and outputs a bit y_i and a new state st_i.
4. the bit stream output $y_1, y_2, \ldots$ is often called the keystream (since it is generally XORed with the plaintext to generate the ciphertext).
5. In practice, y_i is often a block of bits instead of one bit.

Instantiate PRGs with stream ciphers

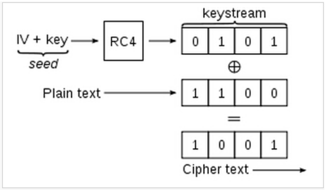
Given a steam cipher and any desired expansion factor l, we can implement a PRG using the following algorithm:

Algorithm 1 Instantiate PRGs with stream ciphers

```
Input: Seed $s$ and optional initialization vector $IV$
Output: $y_1, \ldots, y_l$
1: $st_0 := Init(s, IV)$
2: for $i = 1$ to $l$ do;
3: $(y_i, st_i) := GetBits(st_{i-1})$
4: return $y_1, \ldots, y_l$
```

Encryption using stream ciphers

- Encryption is done via **XORing the plaintext and the keystream**.
- Pros: highly efficient, simple implementation in hardware, easy to handle messages of arbitrary length,...
- Cons:
  - cannot use the same key or keystream to encrypt different plaintexts, otherwise correlations in plaintexts can be easily spotted in corresponding ciphertexts.
  - vulnerable to bit-flipping attacks, thus message integrity validation is often needed (will talk about this soon).



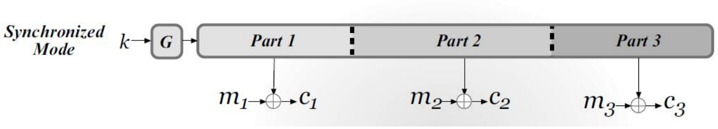
Basic WEP encryption: RC4 keystream XORed with plaintext

图 2: An example: WEP for WIFI network uses a stream cipher called RC4 to encrypt messages (Pic from wiki)

Stream-cipher modes of operation

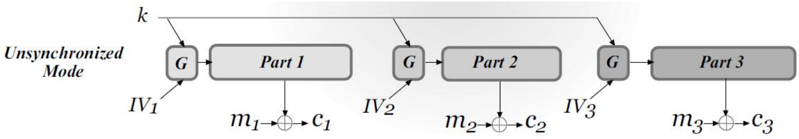
In practice, we have two “modes of operation” （工作模式） for encrypting with stream ciphers:

- **Synchronized mode**



Synchronized Mode
k
G
Part 1
Part 2
Part 3
$m_1 \rightarrow \oplus \rightarrow c_1$
$m_2 \rightarrow \oplus \rightarrow c_2$
$m_3 \rightarrow \oplus \rightarrow c_3$

- **Unsynchronized mode**



k
Unsynchronized Mode
G
Part 1
G
Part 2
G
Part 3
IV_1
IV_2
IV_3
$m_1 \rightarrow \oplus \rightarrow c_1$
$m_2 \rightarrow \oplus \rightarrow c_2$
$m_3 \rightarrow \oplus \rightarrow c_3$

Linear-feedback shift registers (线性反馈移位寄存器)

An (degree-n) **linear-feedback shift register** (LFSR) can be defined by:

- An array of n 1-bit registers: $s_{n-1}, \ldots, s_0$;
- n feedback coefficients: $c_{n-1}, \ldots, c_0$,

such that
its (current) state is the set of bits contained in all registers;
and the next state after time t is determined as:

$$\begin{aligned} \text{Shift: } s_i^{(t+1)} &:= s_{i+1}^{(t)}, & i = 0, \ldots, n-2 \\ \text{Linear-feedback: } s_{n-1}^{(t+1)} &:= \bigoplus_{i=0}^{n-1} c_i s_i^{(t)}; \end{aligned}$$

its output is the bit sequence $s_0^{(0)}, s_0^{(1)}, \ldots$.

Example: A degree-4 LFSR

Figure 3 shows a degree-4 LFSR with feedback coefficients equaling 0,1,0,1:

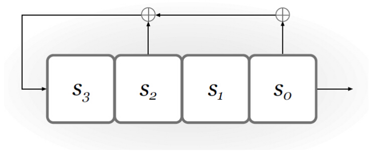


图 3: A degree-4 LFSR with feedback coefficients 0,1,0,1

Given the initial state $(0,0,1,1)$, then next states would be $(1,0,0,1)$, $(1,1,0,0)$, $(1,1,1,0)$, $(1,1,1,1)$, $(0,1,1,1)$, $(0,0,1,1)$, $(1,0,0,1)$, ...
Its output would be: 1, 1, 0, 0, 1, 1, 1, 1, ...

A few security analyses (1)

- **Q**: Can the degree-4 LFSR output an unlimited-length stream of “random bits”?
  A: No, the stream always repeats itself after outputting at most 2^4 bits.

A few security analysis (2)

- **Q**: What about a degree-n LFSR?
  A: Start to generate repeated bits after outputting at most 2^n bits.

A few security analysis (3)

- **Q**: When n is large, say $n = 128$, is the degree-n LFSR secure for cryptographic usage?
  A: No, even if we know how to make it a maximal-length LFSR of degree n, i.e. the LFSR that cycles through all 2^n states before generating repeated bits.

Reconstruction attacks on LFSR

- In general, LFSR is not suitable for cryptographic usages.
- Due to its internal linearity, the LFSR is vulnerable to reconstruction attacks: an adversary can reconstruct the entire states of a degree-n LFSR, after observing $2n$ output bits.

A reconstruction attack example

Example: an adversary observes a degree-4 LFSR with the 8 consecutive output bits:$1, 1, 0, 0, 1, 1, 1, 1$.
To reconstruct the LFSR, the adversary needs to know $(s_3^{(0)}, s_2^{(0)}, s_1^{(0)}, s_0^{(0)})$ and (c_3, c_2, c_1, c_0).

- The adversary knows $(s_3^{(0)}, s_2^{(0)}, s_1^{(0)}, s_0^{(0)}) = (0, 0, 1, 1)$.
- The adversary can compute (c_3, c_2, c_1, c_0) by solving 4 linear equations with 4 unknowns:

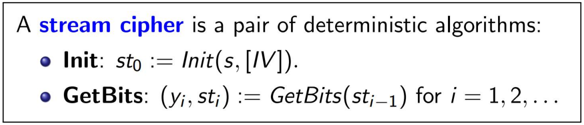
$$\begin{aligned} 1 = \quad & c_3 \cdot 0 \oplus c_2 \cdot 0 \oplus c_1 \cdot 1 \oplus c_0 \cdot 1 && (1) \\ 1 = \quad & c_3 \cdot 1 \oplus c_2 \cdot 0 \oplus c_1 \cdot 0 \oplus c_0 \cdot 1 && (2) \\ 1 = \quad & c_3 \cdot 1 \oplus c_2 \cdot 1 \oplus c_1 \cdot 0 \oplus c_0 \cdot 0 && (3) \\ 1 = \quad & c_3 \cdot 1 \oplus c_2 \cdot 1 \oplus c_1 \cdot 1 \oplus c_0 \cdot 0 && (4) \end{aligned}$$

Solving (1-4), we know:

$$(c_3, c_2, c_1, c_0) = (0, 1, 0, 1).$$

Adding nonlinearity

- To thwart reconstruction attacks, nonlinearity can be introduced in the **GetBits** function.
- In particular, nonlinearity can be added into 1) the state updating procedure, or 2) bit-generating procedure, or 3) both.
- Examples include RC4, Trivium, and so on.



A **stream cipher** is a pair of deterministic algorithms:

- **Init**: $st_0 := Init(s, [IV])$.
- **GetBits**: $(y_i, st_i) := GetBits(st_{i-1})$ for $i = 1, 2, \ldots$

图 4: The **GenBits** algorithm of the stream cipher is responsible for 1) updating states and 2) generating bit outputs from the current state.

Rivest Cipher 4

RC4 (Rivest Cipher 4 also known ARC4 or ARCFOUR meaning Alleged RC4) is a prominent stream cipher.

- designed by Ron Rivest in 1987.
- efficient for both hardware implementation and software implementation.
- widely used today. e.g. in Wired Equivalent Privacy (WEP) for WIFI networks, and in TLS/SSL protocols.
- Recent researches have shown serious cryptographic weakness (i.e. statistical biases) and it should NO LONGER be used.[4]

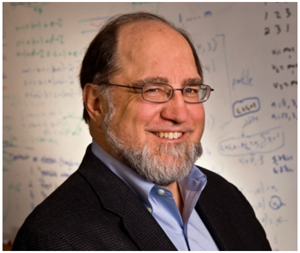


图 5: R. Rivest, co-inventor of the RSA algorithm, Turing award winner (2002), photo downloaded from `https://www.soldierx.com/`

The Init algorithm of RC4

- Init algorithm outputs a permuted array of $S = (1, 2, \ldots, 255)$.

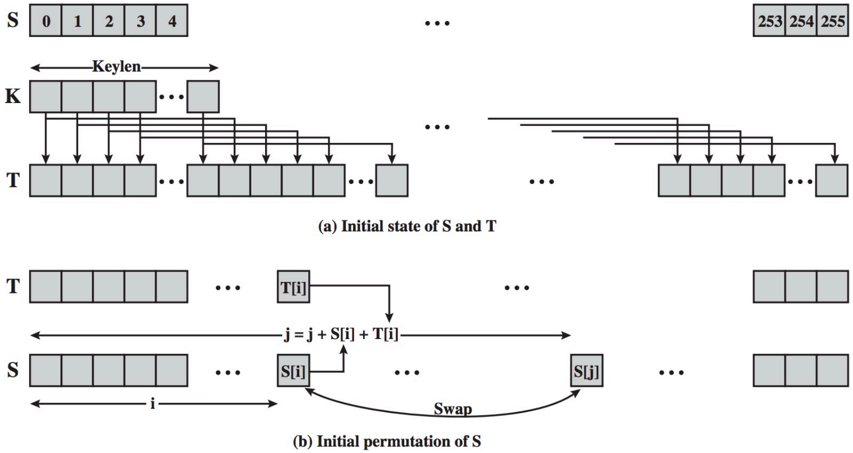


图 6: The Init algorithm of RC4 (pic from W. Stallings “Cryptography and Network Security”)

The Init algorithm of RC4

Algorithm 2 Init algorithm for RC4 (All addition is done modulo 256)

```
Input: 16-byte key $k$
Output: Initial state $(S, i, j)$
  for $i = 0$ to 255:
  for $S[i] := i$
  for $T[i] := k[i \bmod 16]$
  $j := 0$
  for $i = 0$ to 255:
  for j:=j+S[i]+T[i]
  for Swap S[i] and S[j]
  $i := 0$, $j := 0$
  return $(S, i, j)$
```

The GetBits algorithm of RC4

- At every clock tick, GetBits algorithm outputs an element of the permuted array of $S = (1, 2, \ldots, 255)$, and re-permutes the array by swapping two elements in the array.

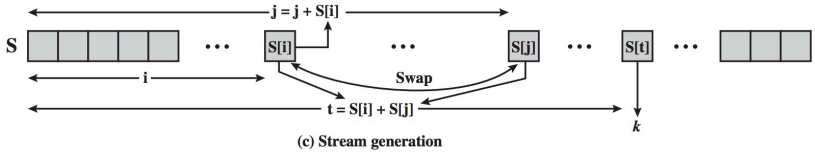


图 7: The GetBits algorithm of RC4 (pic from W. Stallings "Cryptography and Network Security")

The GetBits algorithm of RC4

Algorithm 3 GetBits algorithm for RC4 (All addition is done modulo 256)

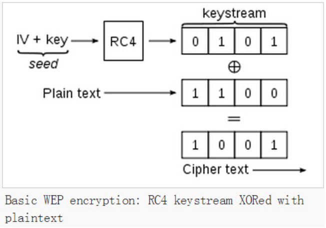
Input: Current state (S, i, j)

Output: Output byte y; updated state (S, i, j)

```
  $i := i + 1$
  $j := j + S[i]$
  Swap $S[i]$ and $S[j]$
  $t := S[i] + S[j]$
  $y := S[t]$
  return $(S, i, j), y$
```

WEP in WIFI

- **Wired Equivalent Privacy (WEP)** protocol for WIFI networks adopts RC4 for message confidentiality, and CRC-32 for correctness validation.
- Standard 64-bit WEP uses a 40-bit key (10 hexadecimal character $0-9$,$A-F$) and a 24-bit IV.
- Two WEP users firstly share the same key, and then use different IVs to encrypt different messages to avoid using same keystreams in encryptions.



Basic WEP encryption: RC4 keystream XORed with plaintext

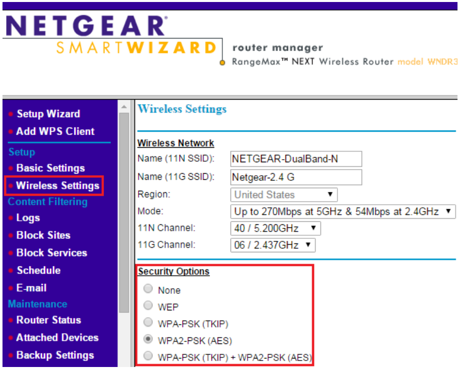
图 8: Pic from WEP on Wikipedia

The short IV flaw of WEP

- There are only 2^{24} possible IVs in total.
- All IVs would be exhausted by a busy access point (e.g. constantly sends 1500-byte packets at 11Mbps) after 5 hrs approximately ($(2^{24} * 1500 * 8/(11 * 10^{60})$ ≈ 5 hrs).
- After that, repeated keystreams start to exist.
- Users have been advised to never use WEP (can be broken in minutes).
- Although **Wi-Fi Protected Access (WPA)** uses RC4 in a more secure way, users are now advised to abandon WPA too since RC4 has been shown to be insecure.

WEP, WPA, WPA2, WPA3

- For WIFI security, users are suggested to **WPA2 or WPA3** now.
- WPA2 and WPA3 use the block cipher AES to encrypt, and other security measures.



Wireless Settings

Wireless Network

Name (11N SSID):	NETGEAR-DualBand-N
Name (11G SSID):	Netgear-2.4 G
Region:	United States
Mode:	Up to 270Mbps at 5GHz & 54Mbps at 2.4GHz
11N Channel:	40 / 5.200GHz
11G Channel:	06 / 2.437GHz

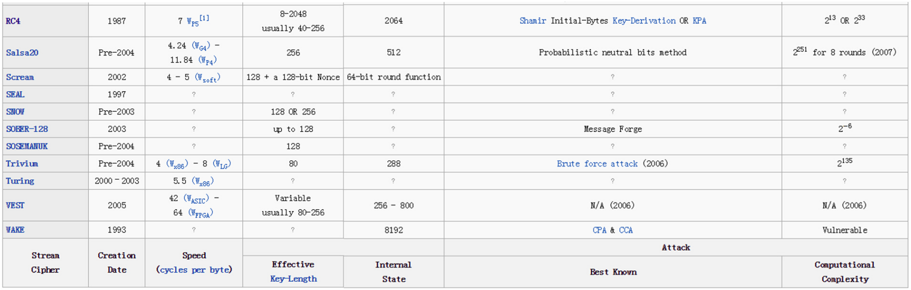
Security Options

- ○ None
- ○ WEP
- ○ WPA-PSK (TKIP)
- ● WPA2-PSK (AES)
- ○ WPA-PSK (TKIP) + WPA2-PSK (AES)

图 9: Security options in a router (pic from `http://kbnetgearrouter.net`)

Tip 1

- **Stop using unsafe stream ciphers.**



RC4	1987	7 W_{P5}[1]	8–2048 usually 40–256	2064	Shamir Initial-Bytes Key-Derivation OR KPA	2^{13} OR 2^{33}
Salsa20	Pre-2004	4.24 (W_{G4}) – 11.84 (W_{P4})	256	512	Probabilistic neutral bits method	2^{251} for 8 rounds (2007)
Scream	2002	4 – 5 (W_{soft})	128 + a 128-bit Nonce	64-bit round function	?	?
SEAL	1997	?	?	?	?	?
SNOW	Pre-2003	?	128 OR 256	?	?	?
SOBER-128	2003	?	up to 128	?	Message Forge	2^{-6}
SOSEMANUK	Pre-2004	?	128	?	?	?
Trivium	Pre-2004	4 (W_{x86}) – 8 (W_{LG})	80	288	Brute force attack (2006)	2^{135}
Turing	2000–2003	5.5 (W_{x86})	?	?	?	?
VEST	2005	42 (W_{ASIC}) – 64 (W_{FPGA})	Variable usually 80–256	256 – 800	N/A (2006)	N/A (2006)
WAKE	1993	?	?	8192	CPA & CCA	Vulnerable
Stream Cipher	**Creation Date**	**Speed (cycles per byte)**			**Attack**	
			Effective Key-Length	**Internal State**	**Best Known**	**Computational Complexity**

图 10: Comparison of streaming ciphers (Tbl from `https://en.wikipedia.org/wiki/Stream_cipher`)

Tip 1

- **Stop using unsafe stream ciphers.**
- a few recommended ones include:
  - Zuc 128/256 stream cipher (“**祖冲之流密码**”) by **国家密码局** (2012∼now).
  - Enocoro-80/128, Trivium-80 by ISO/IEC 29192 (2012∼now).
  - Grain128-AEAD is the stream cipher in NIST’s Lightweight Crypto Competition’s finalist (still under review, 2021∼now).[1]

[1] Read `https://billatnapier.medium.com/after-aes-nist-is-defining-the-next-great-standard-for-encryption-light-we` if interested.

Tip 2

- In private-key encryptions, for better security, **block ciphers are generally preferable to stream ciphers**

References I

Katz, J. and Lindell, Y..
Chapter 3,6 of "Introduction to modern crytography" (2nd ed).
Chapman & Hall/CRC, 2014

Stallings, W.
"Cryptography and Network Security-Principles and Practices" (4th ed).
Prentice Hall, 2005

R.Rivest's photo is downloaded from
https://www.soldierx.com/system/files/hdb/Ronald_Rivest.jpg

Vanhoef, M. and Piessens F..
"All Your Biases Belong to Us: Breaking RC4 in WPA-TKIP and TLS"
https://www.rc4nomore.com/

The short IV flaw example is from
http://security.blogoverflow.com/2013/08/
wifi-security-history-of-insecurities-in-wep-wpa-and-wpa2/

References II

Turan, M.
"On the NIST Lightweight Cryptography Standardization"
https://csrc.nist.gov/CSRC/media/Presentations/on-the-nist-lwc-standardization/images-media/Talk-Elliptic-Curve-Crypto-Meltem_Dec2019.pdf