Stream Ciphers

Gianluca Dini
Dept. of Ingegneria dell'Informazione
University of Pisa
Email: gianluca.dini@unipi.it
Last version: 2024-02-28

1

Can we modify OTP to make it practical?

Stream Ciphers

STREAM CIPHERS

2

Making OTP practical (1/3)

UNIVERSITÀ DI PISA

Instead of generating the key I could have a pseudo random generation

- Idea: replace the random key stream by a pseudo-random key stream
- Pseudo Random Generator **G** is an efficient and deterministic function

$$G : \{0,1\}^s \rightarrow \{0,1\}^n \quad , n >> s$$

Seed space **Key-stream space**

The key stream is computed from a seed

I have a small seed and function G gives me a long key as needed, called key stream.

3

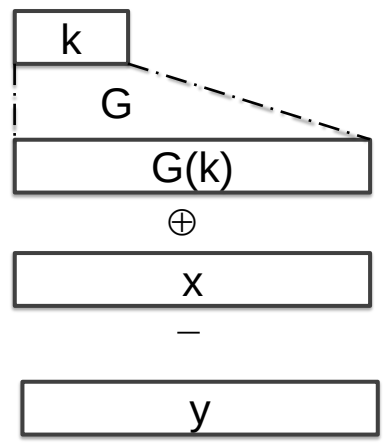
Intuition: for different messages I use different parts of the stream.

Making OTP practical (2/3)

UNIVERSITÀ DI PISA

Encryption: y = G(k) ⊕ x
Decryption: x = G(k) ⊕ y

- Key *k* is a small secret (e.g., 100 bits)
- G is pseudo-random so *sndr* & *rcvr* generate the same key stream



4

Making OTP practical (3/3)

- Is OTP-modified (stream cipher) still perfect?
  - NO!
    - #keys < #msg ➔ Shannon's theorem is violated
  - We need a new definition of security!
- Security will depend on the specific PRG
  - PRG must look random, i.e., indistinguishable from a TRG for a limited adversary [we don't want impossible, just difficult]
    - It must be computationally unfeasible to distinguish PRNG output from a TRG output
  - Computational security (a new definition of security)

This will be considered secure (from a different pov) provided that the generation looks random. We need an output that cannot be distinguished by a true random output

5

from your standpoint

Computational security

- A new definition of security
- A cipher is computationally (practically) secure if the *perceived* level of computation required to defeat it, using the *best attack known*, exceeds, by a comfortable margin, the computation resources of the *hypothesized adversary* [LIMITED ADVERSARY] [WE ASSUME TO KNOW THE BEST KOWN ATTACK]
- Now, the adversary is assumed to have a limited computation power

6

Computational security

UNIVERSITÀ DI PISA

- What is the *best known attack*?
- Even if a lower bound on the complexity of one attack is known, we don't know whether any other, more powerful attacks, are possible
- The best we can do is to design cryptosystem for which it is *assumed* that they are computationally secure

Feb-24 Stream Ciphers 8

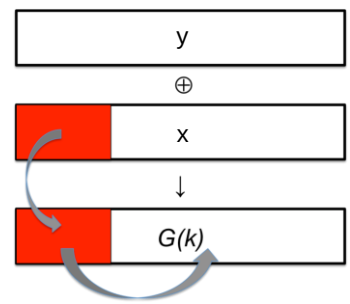
8

PROPERTIES A PRG MUST FOLLOW:
- THEY NEED VERY GOOD STATISTIC: they should produce a sequence of bits in which the # of 0s and 1s is approx the same, same for 00, 01, 10, 11 etc.
- THEY NEED TO BE UNPREDICTABLE
THIS IS WHAT A PRG NEEDS TO LOOK RANDOM

PRG must be unpredictable

UNIVERSITÀ DI PISA

- Not only a PRG must have good statistics, it must be also unpredictable
- If PRG is predictable, a stream cipher is not secure!
  - Assume an adversary can determine a prefix of x then
  - Then, (s)he can compute a prefix of the key stream
  - If G is predictable, (s)he can compute the rest of the key stream and thus decrypt y



Feb-24 Stream Ciphers 9

9

Cyph ⊕ plaint. = prefix of G(k). If x is predictable, you can get bits of G(k).

When we say that a **pseudorandom number generator (PRNG) must have good statistics**, we mean that the numbers it produces should **appear truly random** when analyzed using various statistical tests. Since PRNGs are deterministic algorithms, their output is not truly random, but it should be **indistinguishable from real randomness** for most practical purposes.

Here's what "good statistics" typically involves:

1. Uniform Distribution

Each possible number (or sequence of numbers) should appear with roughly equal probability over a large enough sample. If a PRNG generates numbers between 0 and 255, for example, each number should appear approximately **1/256 of the time**.

2. Independence

Unpredicatbility

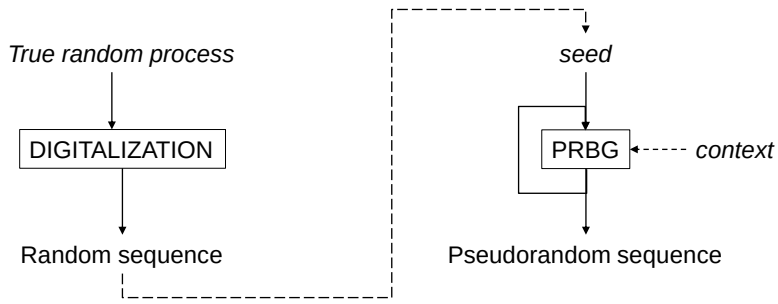
UNIVERSITÀ DI PISA

- Forward unpredicatability
  - If the seed is not known, the next output bit of a sequence must be unpredictable regardless of knowledge of any prefix of the sequence
- Backward unpredicatability
  - It must not be possible to determine the seed from the knowledge of any generated sequence
- If a sequence is/appears random it is not possible to predict either the next bit(s) or the seed

10

TRBG and PRBG

UNIVERSITÀ DI PISA



Idea as: I use seed with a PRBG, but how can I generate the seed?
I use a true random generator.
NOTE: adversary can always bruteforce the possible seeds. So seeds need to be so large to discourage bruteforce attacks.

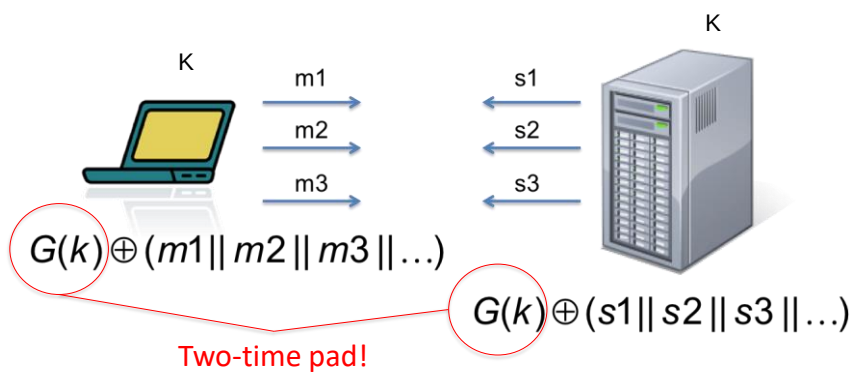
11

Stream ciphers

STATE OF THE ART AND CASE STUDIES

12

MS-PPTP (Windows NT)



13

MS-PPTP (Windows NT)

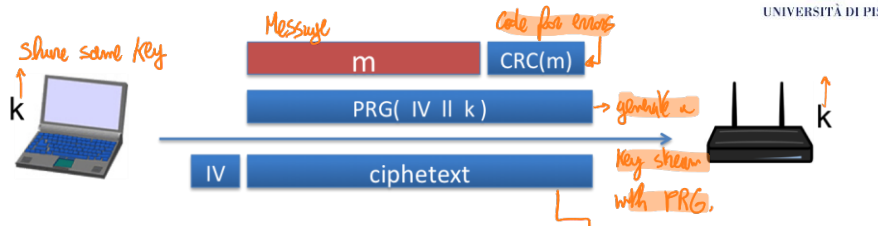
UNIVERSITÀ DI PISA

- The correct way to proceed is $K = (K_{cs}, K_{sc})$
- $Z_{cs} = G(K_{cs})$, key stream for encryption client → server
- $Z_{sc} = G(K_{sc})$, key stream for encryption server → client

STANDARD

802.11b WEP

UNIVERSITÀ DI PISA



- A new IV for each new message
  - Key is fixed (104-bits)
  - IV avoids 2TP
- Length of IV: 24 bits (in the standard!)
  - Repeated IV after $2^{24} \approx 16M$ frames
  - On some 802.11 cards IV resets to 0 after power cycle

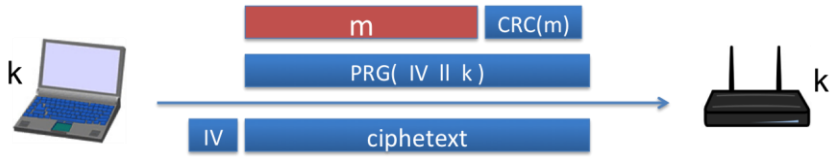
An application of stream cyphers is in Wifi WEP.
How to generate seed? Made by 2 quantities: IV (initialization vector) | key
24bit
Conc.
104bit → 128bits for seed.

* To avoid 2 TIME PAD: we don't want to reuse twice the same key stream

* I have 2^{24} possible IVs, so 16 million frames. But this is a small traffic. You can obtain 16M frames even in a network of medium size. Consider the key is shared in the whole network.
When I run out of IVs I fall into 2 time pads.

Δ IV has to be new for each message. So they implemented it by means of a counter.

802.11b WEP



Key for frame #1: 1||k
Key for frame #2: 2||k
Key for frame #3: 3||k
…

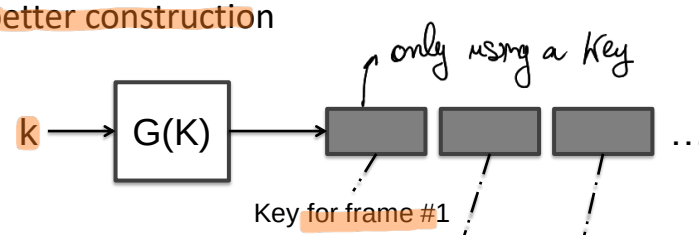
- Related keys, not random
- FMS 2001 attack can recover K in 10^6 frames (now 40 Kframes)
- **Avoid related keys!**

1M frames optimized later

16

802.11b: WEP

- A better construction



- Each frame has its own key
- Keys are pseudo-random

But here you need a large key.

17

- For PRG they used a generator called RC4. Considered quite good but it has some defects. If you use related keys, it becomes predictable.
- Another mistake is using CRC, which is linear with respect to XOR. Cryptosystem is malleable, I could apply modifications on plaintext and CRC to make them match. This was used to modify the destination address of the packet.

RC4

UNIVERSITÀ DI PISA

- RC4 (1987)
  - Used in HTTPS and WEP
  - Variable seed; output: 1 byte
- Weaknesses
  - Bias
    - Pr[2nd byte = 0] = 2/256 (twice as random)
      - Other bytes are biased too (e.g., 1st,3rd)
      - It is recommended that the first 256 byes are ignored
    - Pr[00] = $1/256^2 + 1/256^3$
      - Bias starts after several gigabytes but it is still a distinguisher
  - Related keys
- It is recommended not to use RC4 but modern CSPRNG

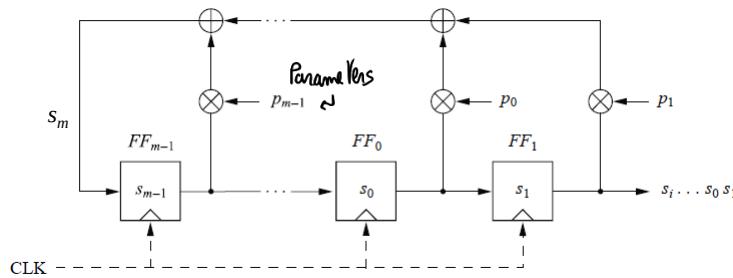
2nd byte of the sequence is not completely random

Way to make pseudo random # gen

Linear Feedback Shift Register

UNIVERSITÀ DI PISA

- p_i = feedback coefficient (If p_i == 1, the feedback is active; otherwise it is not)



$$s_m \equiv p_{m-1}s_{m-1} + \cdots + p_1 s_1 + p_0 s_0 \bmod 2$$

$$s_{m+1} \equiv p_{m-1}s_m + \cdots + p_1 s_2 + p_0 s_1 \bmod 2$$

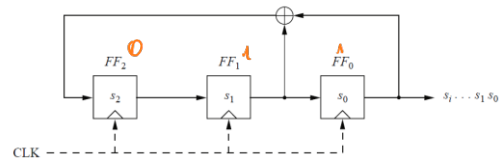
$$s_{i+m} \equiv \sum_{j=0}^{m-1} p_j \cdot s_{i+j} \bmod 2, s_i, p_j \in \{0,1\}, i = 0, 1, 2, \cdots$$

From a security pov it is linear, so when you have enough points you can interpolate

LFSR is periodical

UNIVERSITÀ DI PISA

- LFSR
  - Degree: 3
- Sequence of states



clk	FF_2	FF_1	$FF_0 = s_i$
0	1	0	0
1	0	1	0
2	1	0	1
3	1	1	0
4	1	1	1
5	0	1	1
6	0	0	1
7	1	0	0
8	0	1	0

← The initial state (*seed*)

← The sequence of states is *periodical*

20

LFSR - Properties

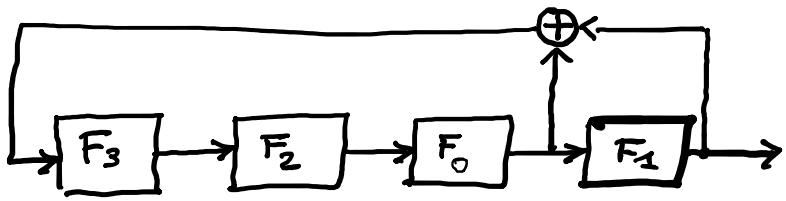
UNIVERSITÀ DI PISA

- Properties
  - Seed = initial state of the register
    - All 0's state must be avoided
  - Degree = number of storage units
    - Degree = 8
  - Periodic
- Maximum-length LSFR
  - Theorem
    - The maximum sequence length generated by an LFSR of degree m is $2^m - 1$
  - Maximum-length LSFR can be easily found

21

LFSR – example #1

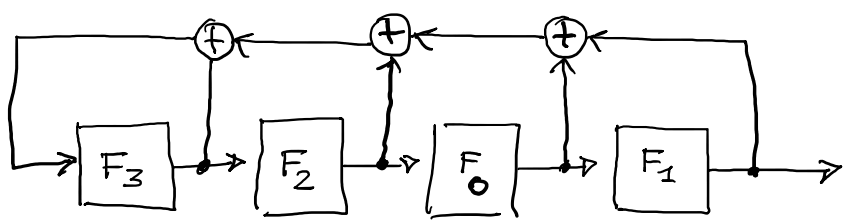
- LFSR with maximum output sequence
  - Degree m = 4
  - Coefficients: $p_3 = 0$, $p_2 = 0$, $p_1 = 1$, $p_0 = 0$
  - Period = $2^m - 1 = 15$



22

LFSR – example #2

- LFSR with non-maximum output sequence
  - Degree m = 4
  - Coefficients: $p_3 = 1$, $p_2 = 1$, $p_1 = 1$, $p_0 = 1$
  - Period = 5



23

LFSRs are not good for crypto

- Pros:
  - LFSRs have good statistical properties
- Cons
  - Periodical (but any PRNG is periodical)
  - Linear

24

LFSRs are not good for crypto

- Known-Plaintext attack against LFSR
  1. Given 2m pairs (pt, ct), the adversary determines a prefix of the sequence s_i
  2. Then, the adversary determines *feedback coefficients* by solving a system of m linear equations in m unknowns
  3. Finally, the adversary can "build" the LFSR and produce the entire sequence

25

LSFRs are not good for crypto

- Have LSFRs to be thrown away?
  - Use a non-linear combination of several LFSRs to build strong cryptosystems
    - E.g., use AND
  - E.g.: Trivium (2003)

Ex: remove XOR and add more known ops like NAND, NOR etc...

This is good when you don't need unpredictability but random bits. If you know a prefix of adequate length you can find the generator.

26

State of the art

- Software-oriented
  - RC4 and SEAL
    - Very well-investigated; secure
- Hardware-oriented
  - LFSR-based
    - Many have been broken
  - GSM A5/1 and A5/2
    - A5/1 used to be secret but was reverse-engineered
    - A5/2 has serious flaws
    - Neither of them is recommended nowadays
    - A5/3 (KASUMI) is used but it is a block cipher

27