Stream Ciphers

Gianluca Dini
Dept. of Ingegneria dell'Informazione
University of Pisa
Email: gianluca.dini@unipi.it
Last version: 2021-03-02

Stream Ciphers

ONE-TIME PAD

One Time Pad
(Vernam cipher, 1917)

- Definition
  - Assumptions
    - Let *x* be a t-bit message, i.e., $x \in \{0,1\}^t$
    - Let *k* be a t-bit key stream, $k \in \{0, 1\}^t$, where each bit is truly random chosen
    - The key is only known to the legitimate communicating partners and is used just once
  - Encryption
    - $y_i = m_i \oplus k_i$ i.e., $y_i = m_i + k_i \bmod 2$
  - Decryption
    - $x_i = c_i \oplus k_i$, i.e., $x_i = y_i + k_i \bmod 2$
  - Consistency property can be easily proven

CONSISTENCY PROPERTY

Dalla tabella di verità di xor
ci xor ki = mi xor ki xor ki = mi

Xor = (. + .) mod 2
Ci + ki mod 2 = ((mi + ki) mod 2) + ki = mi + ki + ki = mi + 2ki mod2 = mi

EXAMPLE
m = 01010101, *k* = 01001110, *c* = 00011011
m is periodic but *c* is not!

QUIZ. IS IT POSSIBLE TO COMPUTE THE OTP KEY FROM M AND C?
Yes, it is possible. Furthermore, the key that maps *m* into *c* is unique.

Why ⊕ is a good encryption function?

- **Theorem**.
  - Let X be a random variable on $\{0, 1\}^n$, and K an independent uniform variable on $\{0,1\}^n$.
  - Then, $Y = X \oplus K$ is uniform on $\{0,1\}^n$.
  - **Proof**. (for n = 1)

The theorem explains why ⊕ is so frequently used in cryptography.

PROOF (FOR n = 1)
- Let X = 0 with X0 and X = 1 with X1, s.t., X0 + X1 = 1
- As K is uniform, i.e., K0 = K1 = K_ = 0.5
- Let's now compute Y0 and Y1, i.e., distribution of Y = X xor K
  - Y0 = Pr[(X = 0 and K = 0) or (X = 1 and K = 1)]
  - Y1 = Pr[(X = 0 and K = 1) or (X = 1 and K = 0)]
- Since K is independent
  - Y0 = X0 K0 + X1 K1 = (X0 + X1) K_ = 0,5
  - Y1 = X0 Y1 + X1 K0 = (X0 + X1) K_ = 0,5

QED

OTP has perfect secrecy

- **THM.** OTP has perfect secrecy iff

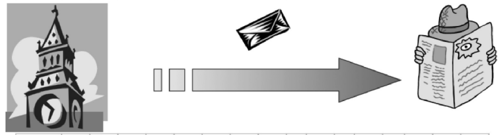
1. The key stream k_i is truly random
2. The key stream k_i is only known to the communication parties
3. Every key stream k_i is used just once

Requirement 1 requires the use of a truly random generator.
Requirement 2 requires that Alice and Bob exchage a possibly large key, as large as the message. Why not exchange the message directly, then.
Requirement 3 is the most unpractical. Key stream cannot be reused.

OTP has perfect secrecy: intuition

- **c[i] = m[i] + k[i] mod 26**
- **m = "SUPPORT JAMES BOND"**

m =	S	U	P	P	O	R	T	J	A	M	E	S	B	O	N	D
k =	W	C	L	N	B	T	D	E	F	J	A	Z	G	U	I	R
c =	O	W	A	C	P	K	W	N	F	V	E	R	H	I	V	U



c =	O	W	A	C	P	K	W	N	F	V	E	R	H	I	V	U
k' =	M	W	L	J	V	T	S	E	F	J	A	Z	G	U	I	R
m =	C	A	P	T	U	R	E	J	A	M	E	S	B	O	N	D

- From a given CT we can generate any PT. Equivalently, we may say that #CT generated from all plaintexts by means of all possible keys, #CT = #PT × #K is much greater than 26^t, i.e., the number of keys on *t*-letters ($\#CT_{t\text{-letter}}$).
- Given a plaintext PT and a ciphertext CT, there exists only one key K which encrypts PT into CT. Such a key is K = CT ⊕ PT.
- In our example, these imply that given ciphertext *c*, we can generate any possible plaintext of the same size. This certainly implies the original message, i.e., *p* = "Support James Bond," but also any other message of the same size including p' = "Capture James Bond". Only if you know the right key k, you can obtain the original message p.

Pros and Cons of OTP - Pros

UNIVERSITÀ DI PISA

- Unconditionally secure
  - A cryptosystem is unconditionally or information-theoretically secure if it cannot be broken even with infinite computational resources
- Very fast enc/dec
- Only one key maps m into c

Pros and Cons of OTP - Cons

UNIVERSITÀ DI PISA

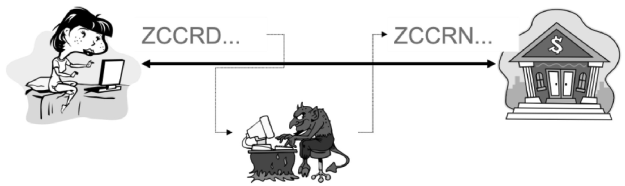
- Long keys: unpractical!
  - Key len == msg len
- Keys must be used once: avoid two-time pad!
  - Let C1 = M1 xor K and C2 = M2 xor K => C1 xor C2 = M1 xor M2 => Redundancies of M1, M2 can be exploited (e.g., English and ASCII)
- A Known-PlainText attack breaks OTP
  - Given (m, c) => k = m xor c
- OTP is malleable
  - Modifications to cipher-text are undetected and have predictable impact on plain-text

- LONG KEYS. Alice has to securely transfer the key to Bob. The key is as long as the message. If Alice has a means to securely transfer the key, than she can use the same mechanism to transfer the message.
- LONG KEYS, NO 2-TIME PAD. OTP is not used for commercial applications
- KPT-attack + MALLEABILITY. No CT-only attacks are possible but other attacks are possible. So, OTP is perfectly secure but not so secure in practice.

OTP is malleable

UNIVERSITÀ DI PISA

m =	D	A	R	E	C	E	N	T	O	E	U	R	O	A	B	O	B
k =	W	C	L	N	B	T	D	E	F	J	A	Z	G	U	I	R	X
c =	Z	C	C	R	D	X	Q	X	T	N	U	Q	U	U	J	F	Y



c' =	Z	C	C	R	N	B	O	P	J	N	U	Q	U	U	J	F	Y
k =	W	C	L	N	B	T	D	E	F	J	A	Z	G	U	I	R	X
m =	D	A	R	E	M	I	L	L	E	E	U	R	O	A	B	O	B

Malleability

UNIVERSITÀ DI PISA

- Malleability
  - A crypto scheme is said to be malleable if the attacker is capable of transforming the ciphertext into another ciphertext which leads to a known transformation of the plaintext
    - The attacker does not decrypt the ciphertext but (s)he is able to manipulate the plaintext in a predictable manner

On OTP malleability

- Attack against integrity
  - Alice sends Bob: $c = p \oplus k$
  - The adversary
    - intercepts c and
    - transmits Bob $c' = c \oplus r$, with r called perturbation
  - Bob
    - receives c' and
    - Computes $p' = c' \oplus k = c \oplus r \oplus k = p \oplus k \oplus r \oplus k$ so obtaining $p' = p \oplus r$
    - The perturbation goes undetected
    - The perturbation has a predictable impact on the plaintext

Example
Let us suppose that the adversary intercepts an encrypted email. The adversary does not know anything about the email but Bob is the sender. Furthermore, since the message comes from Bob, then the adversary knows that the first line of the message is "from: Bob".

The adversary wants to make the message to appear as coming from Eve.

The adversary has only to apply a change to bytes 7-9 and transform the from 'B' 'o' 'b' to 'E' 'v' 'e'. This is quite simple:

'B' 'o' 'b' xor X = 'E' 'v' 'e'

If we consider the Ascii codes
B o b -> 42 6F 62
E v e -> 45 76 65

X = Bob xor Eve (byte per byte) = 07 19 07

Stream Ciphers

STREAM CIPHERS

Making OTP practical (1/3)

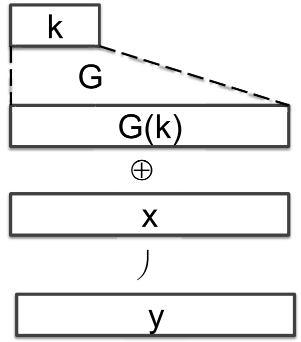
- Idea: replace the random key stream by a pseudo-random key stream
- Pseudo Random Generator G is an efficient and deterministic function

$$G : \underbrace{\{0,1\}^s}_{\text{Seed space}} \rightarrow \underbrace{\{0,1\}^n}_{\text{Key-stream space}}, n >> s$$

The key stream is computed from a seed

Making OTP practical (2/3)

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

Making OTP practical (3/3)

- Is OTP-modified (stream cipher) still perfect?
  - NO! #keys < #msg => Shannon's theorem 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
    - It must be computationally unfeasible to distinguish PRNG output from a TRG output
  - A new definition of security is necessary: computational security

Is a stream-cipher perfectly secure? NO, because the *k* is smaller than messages

A more exact **definition of unpredictable pseudo-random bit generator** is that given *n* consecutive bits of the key stream, $k_1, k_2,…,k_n$, there is no polynomial time algorithm that can predict the next bit, kn+1, with better than 50% chance of success.

Computational security

- Definition
  - A cryptosystem is computationally secure if the best known algorithm for breaking it requires at least *t* operations
  - Cons
    - What is the best known attack?
    - The best we can do it to design cryptosystem for which it is *assumed* that they are computationally secure

Computational security

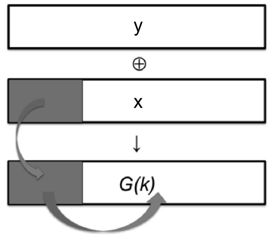
- Cons
  A. What is the best known attack?
  B. Even if a lower bound on the complexity of one attack is known, we don't know whether any other, more powerful attacks, are possible
- The best we can do it to design cryptosystem for which it is *assumed* that they are computationally secure

A. Consider RSA which can be broken by factoring large integers. Even though many factoring algorithms are known, we do not know whether there exist any better ones.

B. Consider the mono-alphabetic substitution algorithm. We know the exact computational complexity of exhaustive key search. However, a more powerful attack exists.

Why we need predictability

- If PRG is predictable, a stream cipher is not secure!
  – Assume an adversary is able to 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



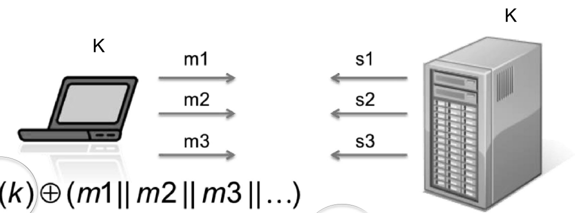
If PRG is predictable, then a stream cipher is not secure.
- Let us suppose that the adversary has intercepted a given ciphertext y.
- Let us suppose that the adversary, by prior knowledge, knows that the beginning of cleartext x has some know value (Kerchoff's principle). For example, in the case of SMTP, the standard for email, you know that every emails begin with "from: ".
- It follows that the adversary can compute a prefix of the keystream by xoring the prefix of the cleartext and the prefix of the ciphertext.
- Then, if the PRG is predictable, given a prefix, then the adversary gets able to predict the remaining of the keystream and thus decrypt the ciphertext.

Exercise
Suppose $G{:}K \longrightarrow \{0,1\}^n$ is such that for all k: XOR(G(k)) = 1. Is G unpredictable?
It is not. If you know a prefix composed of the first n-1 bits, then you can compute the n-th.

Stream ciphers

STATE OF THE ART AND CASE STUDIES

MS-PPTP (Windows NT)



$G(k) \oplus (m1 \| m2 \| m3 \| \ldots)$

$G(k) \oplus (s1 \| s2 \| s3 \| \ldots)$

Two-time pad!

In **Microsoft PPTP (Point-To-Point Tunneling Protocol)** the entire interaction, from the client to the server, is considered as one stream. In other words, messages m1, and m2 and m3, are viewed as one long stream that is encrypted using the stream cipher with key K. So that's perfectly fine. There's nothing wrong with that.

The problem is the same thing is happening also on the server side. In other words, all the messages from the server – s1, s2, s3 – are also treated as one long stream. The problem is that these stream is encrypted **using, unfortunately, the same pseudo-random seed, i.e., using the same stream cipher key**. So basically, two time pad is taking place.

In fact, what you need to do is to have one key for interaction between the client and the server (K_{cs}) and one key for interaction between the server and the client (K_{sc}). In practice, this means that the **shared key k is actually a pair of keys**.

MS-PPTP (Windows NT)

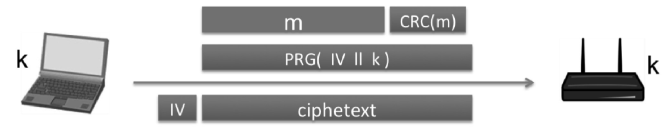
- The correct way to proceed is $K = (K_{cs}, K_{sc})$
- $Z_{cs} = G(K_{cs})$, key stream for encryption client $\rightarrow$ server
- $Z_{sc} = G(K_{sc})$, key stream for encryption server $\rightarrow$ client

In **Microsoft PPTP (Point-To-Point Tunneling Protocol)** the entire interaction, from the client to the server, is considered as one stream. In other words, messages m1, and m2 and m3, are viewed as one long stream that is encrypted using the stream cipher with key K. So that's perfectly fine. There's nothing wrong with that.

The problem is the same thing is happening also on the server side. In other words, all the messages from the server – s1, s2, s3 – are also treated as one long stream. The problem is that these stream is encrypted **using, unfortunately, the same pseudo-random seed, i.e., using the same stream cipher key**. So basically, two time pad is taking place.

In fact, what you need to do is to have one key for interaction between the client and the server (K_{cs}) and one key for interaction between the server and the client (K_{sc}). In practice, this means that the **shared key k is actually a pair of keys**.

802.11b WEP



- 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

The client and the access point (AP) share a secret key whose size is 128 bits. The key is divided in two components: a 24-bit Initialization Vector (IV) and a 104-bit long term key K.

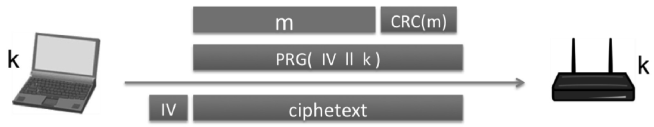
Assume the client wants to send AP a frame containing the plain text M. C appends some sort of check sum (CRC(M)) to the plain tex. The check sum is not important at this point. The resulting plaintext gets encrypted using a stream cypher where the stream cypher key is IV || K, i.e. the concatenation of a value of IV and the long term key K. To fix ideas, you can imagine that IV starts from zero and is incremented by one for every packet (IV can be randomly generate for each new packet). The rationale behind this is to avoid two-time pad: IV must be different for each packet (IV must be "fresh"). IV is also sent in the clear along with the cipher text to let the receiver to generate the key stream for decryption. Actually, the recipient knows the key K, obtains IV from the frame and thus can obtain the key stream from the PRG by concatenating IV and K.

The problem with this of course is the IV is only 24 bits long. Which means that there are only two to the 2^{24} possible IV's. Which means that after sixteen million frames are transmitted essentially the IV has to cycle. And once it cycles after 16 million frames essentially we get a two-time pad. The key K never changes, it's long-term key and as a result, the same key, namely the IV concatenated to K, would be used to encrypt two different frames. The attacker can easily figure this out by inspecting the plain text of both frames.

The problem gets worst many 802.11 cards reset IV to zero when you power-cycle the card. As a result, every time you power cycle the card, essentially, you'll be encrypting the next payload using zero concatenated K. So, after every power cycle, you'll be using the zero concatenated K key to encrypt many, many, many times the same packets. So, the same pad could be used to encrypt many different messages as soon as the IV is repeated. There is nothing to prevent the IV from repeating after a power-cycle, i.e., after every sixteen million frames, which aren't that many frames in a busy network.

Unfortunately this cannot be changed as the size of IV is written in the standard.

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!**

RC4, the PRG used in WEP, was not designed to be secure when you use **related keys**, i.e., that are closely related.

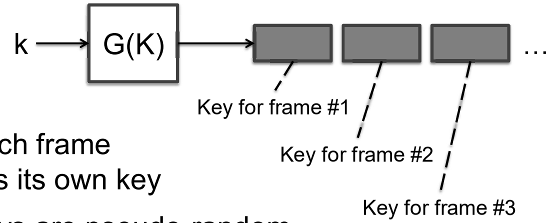
The problem with WEP is that keys are actually related: (1||K), (2||K), … **FMS 2001 attack** exploits this property.

A better approach would be to consider the interaction between the client and the APP as a sequence of messages and generate a single key stream from k to encrypt the stream of messages. That is, consider [m1, m2, m3,…] as a single stream and thus encrypt it as G(k) **xor** [m1 || m2 || m3 || …]

Alternatively, if you just want to use a different key for a different frame (as WEP designers wished to although they failed), as shown in the next slide.

802.11b: WEP

- A better construction



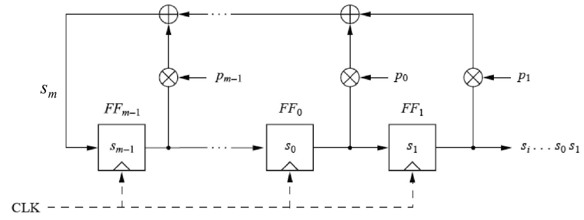
- Each frame has its own key
- Keys are pseudo-random

RC4

- 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

Linear Feedback Shift Register

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



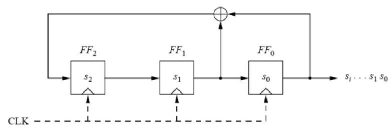
$s_m \equiv p_{m-1}s_{m-1} + \cdots + p_1 s_1 + p_0 s_0$ mod 2

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

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

LFSR is periodical

- LFSR
  - Degree: 3
- Sequence of states



clk	FF_2	FF_1	$FF_0 = s_i$	
0	1	0	0	← The initial state (*seed*)
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	← The sequence of states is *periodical*
8	0	1	0	

LFSR - Properties

- 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

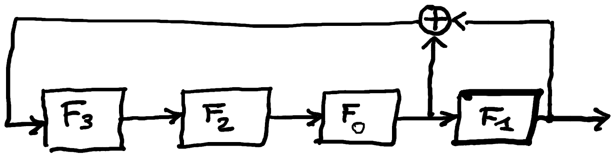
It is easy to show that the Theorem holds. The output of the LSFR depends on its state. As soon as LFSR assume a previous state, it starts to repeat. Since the number of nonzero states is at most 2^m -1, the maximum length before repetition is $2^m - 1$.

The length of an LFSR depends on the feedback coefficients.

An LFSR can be described by a polynomial: $P(x) = x^m + p_{m-1} x^{m-1} + \dots + p_1 x + p_0$
Maximum length LSFR have primitive polynomial. Primitive polynomials are a type of irreducible polynomials. An irreducible polynomial is a sort of prime number that is, its factors are 1 and the polynomial itself. Primitive polynomials can relatively easily be computed. Hence, maximum-length LSFR can easily be found.

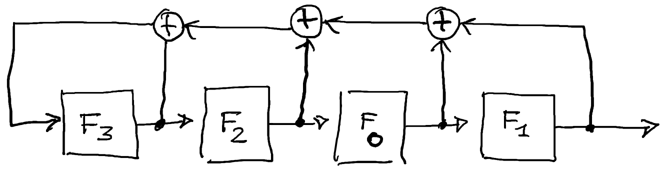
LFSR – example #1

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



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



LFSRs are not good for crypto

- Pros:
  - LFSRs have good statistical properties
- Cons
  - Periodical
  - Linear

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

LSFRs are not good for crypto

UNIVERSITÀ DI PISA

- 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)

Trivium is quite a new stream cipher. Even though there are no known attacks at the time of writing, one should keep in mind that Trivium is a relatively new cipher and attacks in the future are certainly a possibility.

LFSRs are used by CSS, GSM (algorithm A5/1 and A5/2) and Bluetooth (algorithm E0).

State of the art

UNIVERSITÀ DI PISA

- 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

State of the art

- eSTREAM Project
  - ECRYPT NoE
    - Call for stream ciphers; 34 candidates
  - Profile 1. Stream ciphers for software applications with high throughput requirements
    - HC-128, Rabbit, Salsa20/12, SOSEMANUK
  - Profile 2. Stream ciphers for hardware applications with restricted resources
    - Grain v1, MICKEY v2, Trivium

eSTREAM performance

- RC4 126 Mb/s (*)
- Salsa 20/12 643 Mb/s
- Sosemanuk 727 Mb/s
- (*) AMD Opteron 2.2. GHz (Linux)

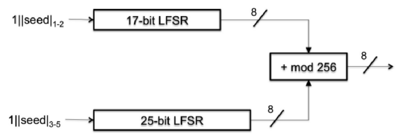
Stream Ciphers

CONTENT SCRAMBLING SYSTEM (CSS)

Content Scrambling System

UNIVERSITÀ DI PISA

- Seed (key)
  - initial states of the LFSRs 5 bytes (80 bit)
- Each round
  - 8 CLK cycles
  - Each LFSR produces 8 bits
  - LFSR's outputs are added mod 256(*) so producing the key stream
    - (*) neglect carry bit for simplicity



Stream cipher to encrypt DVD movies.

The CSS stream cipher is fast and cheap but, unfortunately, very easy to break. It is based on LFSR.

The seed (the key) is 5 bytes and defines the initial state of LFSRs and constitutes the key.

The key is 5-bytes, 40-bits. It's short! Due to USA regulations on crypto (not valid anymore)

Content Scrambling System

UNIVERSITÀ DI PISA

- Easy to break in 2^{17} steps (<< 2^{40})
- Known-plaintext attack
  - A prefix$|_{1\text{-}20}$ of the (cleartext) movie is known => a prefix of the keystream$|_{1\text{-}20}$ can be computed
    - E.g., 20 initial bytes in mpeg
- For details
  - https://www.cs.cmu.edu/~dst/DeCSS/Kesden/

Stream cipher to encrypt DVD movies.

The CSS stream cipher is fast and cheap but, unfortunately, very easy to break. It is based on LFSR.

The seed is 5 bytes, defines the initial state of LFSRs and constitutes the key.

The key is 5-bytes, 40-bits. It's short! Due to USA regulations on crypto (not valid anymore)

Content Scrambling System

UNIVERSITÀ DI PISA

- Attack algorithm
  - For all possible initial setting of LFSR-17 (2^{17})
    1. Run LFSR-17 to get 20 bytes of output
    2. Subtract LFSR-17$|_{1\text{-}20}$ from keystream$|_{1\text{-}20}$ and obtain a candidate output of LFSR-25$|_{1\text{-}20}$
    3. Check whether LFSR-25$|_{1\text{-}20}$ is consistent with LSFR-25
       a. If it is consistent then we have found correct initial setting of both and the algorithm is finished!
       b. Otherwise, go to 1 and test the next LFSR-17 initial setting
  - Using key, generate entire CSS output
  - Complexity
    - At most, the attack need to try all the possible initial setting of LFSR-17 (2^{17})

- A prefix of the movie is known (e.g., 20 bytes in mpeg)
- Then a prefix of CSS$|_{1\text{-}20}$ can be computed
- For all possible initial setting of LFSR-17
  - Run LFSR-17 to get 20 bytes of output → LFSR-17$|_{1\text{-}20}$
  - Subtract LFSR-17$|_{1\text{-}20}$ from CSS$|_{1\text{-}20}$ prefix and obtain candidates 20 bytes output of LFSR-25 → LFSR-25$|_{1\text{-}20}$
  - Check if LFSR-25$|_{1\text{-}20}$ is consistent with LFSR-25
  - If so, find the initial setting of LFSR-25 and terminate the algorithm
- At the end of the algorithm the adversary has got the initial setting of both LFSR, namely the seed!